Université Paris Diderot (Paris 7) – Sorbonne Paris Cité
École doctorale 386 – Science Mathématiques de Paris Centre
Institut de Recherche en Informatique Fondamentale

and

Università degli studi di Milano
Dipartimento di Informatica

Bruno Guillon

Two-wayness: Automata and Transducers

PhD Thesis

supervised by Christian Choffrut *and* Giovanni Pighizzini,

defended on May 30, 2016,
in front of the jury composed of

Alessandra Cherubini	examinator
Christian Choffrut	supervisor
Martin Kutrib	reviewer
Sylvain Lombardy	reviewer
Giovanni Pighizzini	supervisor
Jean-Marc Talbot	reviewer

Contents

Chapter 1

Introduction

1.1 Computational model

Computer science is born from the question: what does "compute" mean? The first researchers who have mathematically formalized the concept shared the same natural idea (still shared by the entire community of Computer scientists): a computation is the execution of a sequence of successive "elementary operations". What are the elementary operations? A well-established fact is that the complexity of computations does not come from the choice of the elementary operations, but rather from the entire sequence of these operations. In other words, a set of very simple operations suffices to build complex computations. Programmers can confirm this assertion. Indeed, though a single line of code could easily be understood, the meaning of the entire program is significantly harder to catch.

Let us give a simple example due to the mathematician Lothar Collatz. Choose a positive integer and compute the following: if it is even, divide it by 2 (applying the elementary function $x \mapsto \frac{x}{2}$), otherwise multiply it by 3 and add 1 (thus applying the simple function $x \mapsto 3x + 1$). Repeat the preceding operation, but each time starting from the resulting integer, until you reach the value 1. Will you eventually halts no matter which integer you initially chose? In spite of the very simplicity of the operations used, the answer to this question raised almost one century ago, remains one of the most challenging open problem in discrete mathematics. This perfectly illustrate how the simplicity of the elementary operations may contrast with the high complexity of computations they can generate.

Informally, a *computational model* is the choice of a set of elementary operations. Fixing them defines a notion of computability, *i.e.*, which problems could be solved using the elementary operations. As said before, for very small sets of very simple operations

(*e.g.*, addition and multiplication), the model reaches the universal computability level, *i.e.*, those specified by Alan Turing in the 20's which corresponds to the level reached by the majority of the programming languages when ignoring the space and time constraints. This high level of computability has a cost. Indeed, the dynamics of the models (*i.e.*, the sequences of operations) are difficult to grasp and major problems are open (*e.g.*, the Collatz problem detailed above), undecidable (*i.e.*, there is no method for solving these problems), or untractable (that is, the methods which can solve the problem are unrealizable in practice).

During this PhD, we have studied lower (*i.e.*, non-universal) computational models, where the elementary operations are very restricted. This approach has two objectives. On one hand, choosing weak models that are restrictions of some universal one (mainly, they are restrictions of the well-known Turing machine) allows us to prove properties on their computations which are more involved in the general model. This kind of results shades some light on the general model and on the natural complexity of some problems. On the other hand, such models are more realistic than the universal one and hence their concrete applications are usable in practice.

Defining a computational model raises questions. What kind of problems can be solved or computed, *i.e.*, which notion of computability is defined by the model? Does it specify a known sub-family of problems? Can we characterize it logically or algebraically? Is the model equivalent to another, *i.e.*, can they mutually simulate? What is the "cost" of such an equivalence? Which problems are "simpler" to solve with a certain model than with another?

In this manuscript we study different computational models which we consider under three distinct motivations. First, when the models are equivalent or at least comparable, we examine the cost of the simulations between them. More precisely, we ask the question of the optimal size of a program (*i.e.*, the description of a method) expressed in the first model relative to the size expressed in the second model. This is a typical concern of the theory of descriptional complexity. Second, we generalize some known models and we try to specify the computational power so reached. This kind of considerations is connected to the fields of algorithmic complexity and descriptive complexity. The former aims to give the best algorithms (*i.e.*, methods) solving a given problem, for some meaning of "best" (time complexity, space complexity *etc...*). The latter attempt to characterize some families of problems by defining a computational model which captures exactly the family. Third, starting from a given model, we consider the iteration of the operations it defines. In other words, we study the meta-model obtained by taking each operation computable in the original model as elementary operations. This approach quickly reaches the frontier of decidability. However, by drastically restricting the original model, we obtain interesting

results that limit the computational power of the meta-model so-obtained. Thus, a wide gap between low-level computational model is exhibited.

1.2 From Turing machines to finite automata

1.2.1 Turing machines

In the early 20th century, pioneers of Computer science such as Alan Turing and Alonzo Church started to formalize the notion of computation. They defined an abstract model, namely the Turing Machine (TM), which captures the notion of computation. Therefore, a problem is said *computable* (or *solvable*) if and only if it may be solved by a TM. This universality has a cost: TMs may use an infinite amount of memory. Indeed, a TM has a finite set of control states which guide its behavior according to a table of rules (the *transitions*), but it has access, through a read & write (RW) head, to an infinite tape. This model has two unsatisfactory aspects. The first one concerns its applications: TMs are unrealizable in practice. The second one is theoretical: due to the unboundedness of the space, the dynamics of TMs are of high complexity and therefore many problems are unsolvable (*e.g.*, the halting problem) or untractable (*e.g.*, the famous $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ question).

The theoretical limitation leads two families of questions. The first is on the dichotomy solvable VERSUS unsolvable. Some questions of this type are treated in Chapter 6, when considering the *iteration*, *i.e.*, the transitive closure, of binary word relations. The second is the famous opposition: determinism VERSUS nondeterminism. This kind of question is explicitly studied in Chapter 3. Also, the results in Chapter 5 shows a gap between the nondeterministic (our results) and the deterministic (known results) cases, when considering a particular computational model, namely the *unary two-way transducers*.

1.2.2 Finite automata

The high complexity of TMs has led researchers to consider restrictions of the model. The probably most known but also most drastic such restriction is that of Finite Automata (FA)[1] which was intensively studied since the 50's. In a FA, the tape is bounded by the length of the input, the head is made read-only (RO) and its moves are restricted to one direction only, say from left to right. The model so obtained is much less powerful than the general TM. In particular since no output can be written, FAs are acceptors, *i.e.*, the output is reduced to the binary accepted/rejected. However, thanks to these restrictions,

[1] The terminology chosen in Chapter 2 refers to the model by *Classical Finite Automata* (CFA), in order to save the name FA for a more general variant, namely the two-way nondeterministic FA.

the two limitations of TMs vanish in the case of FAs. On the one hand FAs can be and are really implemented in the industry (*e.g.*, in electronic circuits, in parser specifications, in treatment of natural languages). On the other hand their dynamics are much more simple to study.

Actually, many properties have been investigated at the beginning of the theory (*e.g.*, by Stephen Cole Kleene, Michael Rabin, Dana Scott). Ever since, the theory has developed tremendously and many connections with other fields have been established. In language theory, the sets accepted by FAs happen to be exactly the languages generated by left (or right) linear grammar introduced by Noam Chomsky. The well-celebrated algebraic characterization of Kleene states that these languages are precisely the rational subsets of a free monoid. Furthermore, the Büchi-Elgot-Trakhtenbrot Theorem shows their equivalence with the sets definable in Monadic Second Order (MSO) Logic. Also FAs enjoy a large amount of properties (*e.g.*, the effective closure under Boolean operations) and powerful mathematical tools (*e.g.*, the determinization procedure for FAs, the pumping Lemma).

1.3 Variants of finite automata

In this manuscript we study other abstract computational models which can be seen both as restrictions of TMs and as extensions of FAs. These variants are obtained by controlling four parameters: (1) the two-wayness (*e.g.*, one-way, rotating, sweeping and two-way devices), (2) the nondetermism (*e.g.*, deterministic, unambiguous, outer-nondeterministic and nondeterministic devices), (3) the size of the alphabets (unary (*i.e.*, single-letter) alphabet or not) and (4) the output production ability (*e.g.*, recognizers (FA), transducers, weighted FAs, *etc...*).

We are particularly interested in the two-way extension of FAs (2FAs), *i.e.*, in devices obtained from classical FAs by enabling the head to move to the left, to the right or to stay in position at each step, according to the transition applied. In other words, these 2FAs are read-only Turing Machines. Actually they were considered at the very beginning of the Theory by Michael Rabin and Dana Scott in [63]. The authors proved that even the nondeterministic variant (2NFA) is equivalent to one-way deterministic FAs (1DFA) as long as recognition power is considered. However, when considering the size of the equivalent devices, an exponential blow up can be observed. Technically, there exists a two-way deterministic FA (2DFA) with n states that has no equivalent 1DFA with less than 2^n states. This typically falls within the area of descriptional complexity.

1.3.1 Descriptional complexity

The descriptional complexity area aims to detail the reductions between computational models. Given a problem which is solvable in two computational models, the question is to determine what is the relative size of the best algorithm in each of these models. In particular we are interested in the case where the two models are equivalent or where one is a restriction of the other. The results are mainly of two kinds: upper and lower bounds. Upper bounds are obtained by controlling the cost of simulation algorithms between models, while lower bounds are obtained by exhibiting families of instances which require a minimum size in one model relatively to the other. The second kind of problems is often more difficult to obtain since proving that a certain size is required could be tricky. Combining the two kinds of results, we may obtain tight simulations which are thus optimal.

Starting from the work of Albert R. Meyer and Michael J. Fischer [57], the domain has been intensively studied in the context of automata and language theories. Indeed, the simulations between different variants of FAs, mainly involving the two-wayness and the nondeterminism, has been a particular source of interest in the 70's and 80's and more recently from the beginning of the century. The two main questions, raised in 1978 by William J. Sakoda and Michael Sipser [68], are the following.

1. What is the minimal size, in terms of states, of a 2DFA equivalent to a one-way nondeterministic FA (1NFA)?
2. What is the minimal size of a 2DFA equivalent to a nondeterministic 2FA (2NFA)?

The authors conjectured an exponential blow up for both questions. They can be reformulated as asking the relative contribution of two-wayness and nondeterminism. For example, in the first case nondeterminism is traded for two-wayness.

Let us focus on this second question which we study in Chapter 3. Despite all efforts, exponential gaps were proved only between 2NFAs and some restricted *weaker versions of* 2DFA*s*. In 1980, Micheal Sipser proved that if the resulting machine is required to be *sweeping* (reversing the direction of its input head only at the *endmarkers*, two special symbols used to mark the left and right ends of the input), the simulation of a 2NFA is indeed exponential [74]. However, Piotr Berman and Silvio Micali [6, 58] proved independently that this does not solve the general problem: in fact the simulation of unrestricted 2DFAs by sweeping 2DFAs also requires an exponential number of states. Sipser's result was generalized by Juraj Hromkovič and Georg Schnitger [39], who considered *oblivious machines* (following the same trajectory of input head movements along all inputs of equal length) and by Christos A. Kapoutsis [45], considering 2DFAs with the number of input head reversals that is sublinear in the length of the input. However, even the last condition gives a machine provably less succinct than unrestricted 2DFAs, and hence the

general problem remains open.

Starting from 2003 with a paper by Viliam Geffert *et. al.* [32], a different kind of restriction has been investigated in this context: the subclass or regular languages using *a single-letter input alphabet*. Even under this restriction, the problem of Sakoda and Sipser looks difficult, since it is connected with $\mathrm{L} \stackrel{?}{=} \mathrm{NL}$, an open question in complexity theory. (L and NL denote the respective classes of languages accepted in deterministic and nondeterministic logarithmic space.) In [32], a new normal form was obtained for unary automata, *i.e.*, for automata with a single-letter input alphabet. In this form all nondeterministic choices and input head reversals take place at the endmarkers only. Moreover, the state-size cost of the conversion into this normal form is only *linear*. This normal form is a starting point for several other properties of unary 2NFAs. First, in the same paper, each n-state unary 2NFA is simulated by an equivalent 2DFA with $O(n^{\lceil \log_2(n+1)+3 \rceil})$ states, which gives a subexponential but still superpolynomial upper bound. It is not known whether this simulation is tight. However, a positive answer would imply the separation between the classes L and NL. In fact, under assumption that L = NL, each unary 2NFA with n states can be simulated by a 2DFA with a number of states polynomial in n [34]. A minor modification of the proof (without the necessity of assuming L = NL) gives that each unary 2NFA can be made unambiguous, keeping the number of states polynomial. (For further connections between two-way automata and logarithmic space, we address the reader to [7, 44, 46].)

Along these lines of investigation, in [33], the problem of the complementation for unary 2NFAs has been considered, by proving that for each n-state 2NFA accepting a unary language L there exists a 2NFA with $O(n^8)$ states accepting the complement of L. The proof combines the above normal form for unary 2NFAs with inductive counting arguments.

Christos A. Kapoutsis [42] considered the complementation in the case of general input alphabets, but restricting the input head reversals. He showed that the complementation of *sweeping* 2NFAs requires exponentially many states, thus emphasizing a relevant difference with the unary case.

In this present work, we use a different approach. Instead of *restricting the power of* 2DFA*s* to the degree for which it is already possible to derive an exponential gap between the weaker model and the standard 2NFAs, we *increase the power of* 2DFA*s*, towards 2NFAs, to the degree for which it is still possible to obtain a subexponential conversion from the stronger model to the standard 2DFAs. Such new stronger model then clearly shows that, in order to prove an exponential gap between 2NFAs and 2DFAs, one must use capabilities not allowed in the proposed new model.

1.3.2 Finite automata with outputs

Though the main limitation of finite automata is the finiteness of their memory, another is that FAs are recognizers while TMs are computing machines. Indeed, the former model is limited to accept or reject an element (hence has a binary output), while the latter may generate an output depending on the run on the input. Extending the model of FA in order to compute more general objects, such as relations or functions on words or formal series on a semiring, is hence a natural progression in the theory of computational complexity.

In 1959 Michael Rabin and Dana Scott introduced the model of *multitape finite automata* [63]. In this model, the device is provided with several tapes scanned from left to right by independent heads, one for each tape, that may move at different speeds. The device hence recognizes tuples of words, in which each component plays the same role. In fact, the model remains a recognizer because no output is produced, but it may accept more general mathematical objects (namely the n-ary relations on words, for a n-tape FA) than classical FAs.

Another idea was followed by George H. Mealy in 1955 [56] and, independently, by Edward F. Moore in 1956 [59]. They both provided the deterministic FAs with a production function, respectively mapping the transitions used or the state visited in a run into a finite output alphabet, hence producing an output word of the same length as the input. Later, the models were generalized by Calvin C. Elgot and Jorge E. Mezei leading to the model of *transducer* [23]. A transducer is a NFA which may perform stationary moves on the input and produces at each step a word which is appended on an additional write-only output tape. In this way it defines a binary word relation which is called a *transduction*. Transducers are equivalent to 2-tape FAs, as far as recognition power of the nondeterministic version is taken into consideration. Technically, the only difference between transducers and 2-tape FAs is that in the latter case the "output" preexists the computation.

Some results on FAs extend to transducers. The probably most typical one is the well-known characterization of transductions as rational subsets of the Cartesian product of two free monoids (see for instance [9, Theorem III 6.1]). However, a lot of other results on FAs do not extend: *e.g.*, transductions are not closed under complementation hence not under intersection, the deterministic and nondeterministic variants are not equivalent and the equivalence of transducers is undecidable.

It is convenient to view transductions as special cases of *formal power series* with coefficients in a semiring $\mathbb{K}$, *i.e.*, of functions of the free monoid Σ^* into $\mathbb{K}$ or equivalently of formal sums of words in Σ^* with coefficients in $\mathbb{K}$ [11, 67]. We now quickly justify this viewpoint. To begin, some formalism is required (it may help the reader to have

the nonnegative integers in mind when reading the next lines). A *semiring* is a domain along with two operations + and ·, call them "addition" and "multiplication" to fix ideas, which satisfy simple properties: + is commutative and associative and possesses a unit 0, · is associative, distributes over + and possesses a unit 1 and 0 is an absorbing element for ·. The operations may be interpreted in many different domains. For example, with the family of subsets of the free monoid Σ^*, the addition may be interpreted as the set union and the multiplication as the set concatenation and under this interpretation the properties are clearly satisfied. The same observation holds for the family $\text{RAT}(\Sigma^*)$ of rational languages in Σ^*. *Polynomials* are formal series which have only finitely many nonzero coefficients.

The semiring structure of $\mathbb{K}$ can be extended to the formal series. Two series σ, τ can be added $\sigma + \tau$ and multiplied $\sigma\tau$ applying the same rules as for polynomials (*i.e.*, the Cauchy product). The set of *rational series* $\text{RAT}(\mathbb{K}\langle\langle\Sigma^*\rangle\rangle)$ is the family of series obtained from the polynomials by applying the operations of sum, product and the so-called Kleene star an arbitrary number of times.

In order to study effective families of series, the notion of *weighted automata* ($\mathbb{K}$-FA) was introduced. They are defined in the same way as transducers except that the output associated with a transition belongs to $\mathbb{K}$ [70, 71, 10, 21]. They specify a formal power series which associates to every input w in Σ^* an element of $\mathbb{K}$, which is the sum over all successful runs labeled by w of the product along the run of the outputs associated to the transitions. The fundamental theorem of Kleene-Schützenberger claims that they are precisely the rational series.

Weighted automata have been studied in many different contexts. For instance, on the tropical semiring $\langle\mathbb{N} \cup \{\infty\}, \min, +\rangle$ they have led to the notion of *distance automata*, a topic that was intensively studied [37, 53, 50, 54, 48]. Though they are not considered in these terms, probabilistic automata may be viewed as particular $\mathbb{K}$-FAs (at each step the weights of the possible transitions sum up to 1) on the semiring of nonnegative real numbers ($\mathbb{R}_+$), see [61, 62]. In this manuscript, we will be more particularly interested in the case where we can identify power series with transductions, *i.e.*, $\mathbb{K} = \text{RAT}(\Sigma^*)$. However, for the sake of generality, many preliminary results are proved for more general semirings (Chapters 2 and 4).

Two-way $\mathbb{K}$-FAs and two-way transducers

Unlike the case where $\mathbb{K}$ is the Boolean semiring with two values 0 and 1, the properties of the automaton underlying a $\mathbb{K}$-FA for a general $\mathbb{K}$ impact the output produced. Hence it is no longer superfluous to assume that the automaton is one- or two-way, deterministic or nondeterministic *etc...* This is the reason why we investigate which of the restrictions

are equivalent and which are different. Though the literature on (one-way) transducer and $\mathbb{K}$-FAs is quite rich, it is no longer the case when considering the two-way variants. As far as we can reconstruct the history, the first indirect approach was in the context of probabilistic devices. Indeed, two-way probabilistic automata work in $\mathbb{R}_+$ and in that respect, the works of Janis Kaneps and Rusins Freivalds [41, 27], or Marcella Anselmo and Alberto Bertoni [3], fall into the category of two-way weighted automata but they were not investigated as such. With the only exception of [2], it is only recently that they were considered as a field of its own, see [12, 13, 55]. Concerning the special case of two-way transducers, the research dates back to the seventies but focused on the functional case [20, 18]. As a major result, Joost Engelfriet and Hendrik Hoogeboom proved that two-way functional transductions can be recognized by deterministic two-way tansducers. Furthermore they characterized them in terms of MSO logic [24]. With the result of Eitan M. Gurari this implies that the equivalence of two-way functional transducers is decidable [36]. Let us also mention that it is decidable whether a two-way [26] or sweeping [5] functional transducer admits an equivalent one-way transducer.

In this manuscript we follow another idea. Instead of restricting the capabilities of the transducer, we choose to restrict the input and/or output alphabets which allows us to obtain an algebraic characterization for the general case. This characterization is obtained by introducing new algebraic operators on formal series, that better capture the behavior of two-way $\mathbb{K}$-FAs than the traditional rational operations.

Indeed, given to series σ and τ accepted by two one-way $\mathbb{K}$-FAs, it is a simple exercise to build $\mathbb{K}$-FAs accepting $\sigma+\tau$, $\sigma\tau$ or σ^*. This is no longer the case for the two-way $\mathbb{K}$-FAs. However, we exhibit three possible abilities of two-way $\mathbb{K}$-FAs: (1) the ability to perform two successive computations on the same input separated by a rewind of the input head; (2) the ability to repeat an arbitrary number of times a computation on the same input; (3) the ability to scan the input word from right to left. These three abilities can be described by algebraic operations: (1) corresponds to the *Hadamard product* of series ($\mathbb{K}$-FA) where the coefficients associated to the same input in two series are multiplied (see for instance [66, Definition III 1.2]); (2) can be described by a new operator of arity 1, called *Hadamard star*, where the coefficient associated to an input in the Hadamard star of a series σ is the infinite sum of the successive powers of the corresponding coefficient in σ; (3) is captured by the *mirror* operation, consisting of reversing the input.

It happens that these three operations characterize precisely the series realized by sweeping $\mathbb{K}$-FAs (the head may change direction at the two boundaries of the input tape only). Indeed, these series are exactly the *Mirror-Hadamard series* (MHAD), *i.e.*, the series which can be obtained from rational series by applying the operations of sum, Hadamard product, Hadamard star and mirror an arbitrary number of times (observe the analogy with the family of rational series). In the case of transducers and under

the assumption that both the input and output alphabets are reduced to a single letter, we prove that the characterization extends to unrestricted two-way transducers (Theorem 16), showing in particular that unary two-way transducers can be made sweeping. This no longer holds when at least one alphabet is non-unary (Theorems 19 and 20).

These results contrast with the deterministic (or functional) case. Indeed, it can be deduced from [2] (and recovered in our construction, see Corollary 19) that functional two-way transducer with a unary output alphabets are equivalent to one-way transducer. Because the family of Mirror-Hadamard series strictly extends those of rational series even in the unary case, our results show how two-wayness and nondeterminism are crucial.

1.4 Iteration of binary relations on words

Since the rational relations considered in this manuscript are binary, there remains to study a natural operation, namely the composition of relations with the usual meaning of the word. It is well-known that composing two rational relations yields a rational relation. We consider the transitive closure of rational relations under composition, *i.e.*, $R \cup R \circ R \cup R \circ R \circ R \cup \ldots$ for some rational relation R. Since, by the high complexity of this "operation" we are led to study subfamilies of rational relations, it is more convenient to use the equivalent definition of rational relations as subsets of pairs of words recognized by 2-tape 1FA. This field has been studied either directly ([75, 52]) or indirectly ([17, 51]). A transition of a Turing machine, *i.e.*, one step computation, defines a binary relation on the set of configurations. Properly encoded this is a rational relation over a finite alphabet. Actually since two successive configurations differ only locally, the relation thus defined belongs to the subfamily of *synchronous relations*, *i.e.*, relations accepted by 2-tape 1FA with both heads moving at the same speed until one arrives at the right border of the corresponding tape. The reachability problem of Turing machines can thus be viewed as a problem on iteration of a rational relation. Consequently, the infinite iterate of a rational relation can be a nonrecursive set. We focus on low levels of relations, because even with the simple subfamily of synchronous relations, the transitive closure is Turing complete. Also on one-letter alphabets, undecidability can be obtained by iterating rational relations. This result is due to John H. Conway by elaborating on the famous Collatz problem. Instead of arbitrary linear functions on the integers such as in Collatz problem, we consider unary synchronous relations on a unary alphabet. These relations, when viewed as relations on $\mathbb{N}$ in the natural identification of the free monoid generated by unique generator with the set of nonnegative integers, are simple binary relations one form of which is the set of pairs $(Mn + i, Mm + j)$ for some fixed integers $0 \leq i, j < M$ where $n \leq m$ are arbitrary integers. In that case, not only the infinite iterate of such relations is a recursive set, but the family of unary synchronous relations

is closed under transitive closure of the composition of relations. Intuitively this is due to the fact that we cannot multiply or divide an integer by some integer greater than 1 and thus the model cannot simulate a Minsky machine as done in Conway's construction.

1.5 Outline

We now briefly present the contents of the different chapters of the manuscript in the order of appearance, and highlight the main contributions.

Preliminaries (Chapter 2)

We gather all prerequisites of the investigations of this manuscript. The different versions of finite automata are recalled, starting from the simplest one and proceeding to the more general ones. We show several ways of providing automata with different types of outputs in order to convert an accepting into a producing device. We first define the weighted automaton model and then, as a particular case, the model of transducers. For the convenience of the reader we the basics and elementary properties are recall but we refer to different textbooks on the topic for further details, *e.g.* [8, 21, 66].

Outer-nondeterministic finite automata (Chapter 3)

Different issues of descriptional complexity on two-way nondeterministic finite automata (2NFAs) are addressed. We consider the famous problem of the cost, in terms of size, of the simulation of a 2NFA by its deterministic counterpart (2DFA). The question raised in 1978 by Sakoda and Sipser [68] who conjectured that it is exponential, remains open in spite of all attempts to solve it.

We focus on a weaker question: the simulation by 2DFAs, of *two-way outer-nondeterministic finite automata* (2ONFA), that are defined as 2NFAs with nondeterministic choices restricted to the endmarkers [31]. The approach is quite new since we do not restrict the simulating machine but rather the simulated one, contrary to numerous works in this area (*e.g.*, [74, 39, 45]). We prove several results, including the complementation of 2ONFA with polynomial cost (Theorem 8) and a subexponential simulation between the two devices (Theorem 9), which turns out to be polynomial under the assumption L = NL (Theorem 10). Hence, a superpolynomial lower bound for the simulation of 2ONFA by 2DFA would imply the separation of the two classes. This last result can be adapted in order to convert a 2ONFA into an unambiguous one, with still a polynomial cost (Theorem 11). Finally, the alternating case is considered (see [43, 30]). We define *two-way outer-alternating finite automata* (2OAFAs) similarly as 2ONFAs, thus restricting both

universal and existential choices to occur at the endmarkers only. Under the assumption L = P (*resp.* NL = P) (where P denote the class of languages accepted in deterministic polynomial time), we prove that each 2OAFA is equivalent to some 2DFA (*resp.* 2ONFA) with a polynomial number of states (Theorems 12 and 13). These results involve techniques developed in [73] and adapted to sweeping 2ONFA in [33] when considering the Sakoda and Sipser's problem restricted to the unary case, *i.e.*, when the alphabet is restricted to a single letter.

Super- and sub-classes of rational series (Chapter 4)

We introduced in Chapter 2 the basics on formal series on a semiring $\mathbb{K}$. Here we study further operations on these objects. With the identification of a relation in $\Sigma^* \times \Delta^*$ with a function of Σ^* into the powerset of Δ^*, *i.e.*, with a series in the semiring $\mathbb{K} = 2^{\Delta^*}$, these operations make sense for relations in $\Sigma^* \times \Delta^*$.

As seen in the preliminaries, the notion of rational relations or series requires the use of union (or sum), product and Kleene star. We introduced in a previous publication [15] a new operation which we called Hadamard star because it shares with the usual Hadamard product the fact that it is defined pointwise on power series. Provided the coefficients in $\mathbb{K}$ have a Kleene star, the coefficient of the Hadamard star of a series is the Kleene star of the coefficient of the series (the infinite sum $1 + k + k^2 + \ldots$ where $k \in \mathbb{K}$ is well-defined).

Some operations are specific to word relations while some others are meaningful for general semirings. *E.g.*, exchanging the two components of a binary relation does not make sense for all semirings. On the contrary the mirror operation is essentially meaningful for series seen as a function of the free monoid into a semiring since it takes the mirror image of the argument. For binary relations the mirror image can be taken independently or simultaneously for both components. Some closure issues under these operations are solved and others are left as interesting themes to be investigated. By doing this, we reveal the contribution of some operations. For instance, we show in Proposition 12 that the mirror of a rational relation cannot be obtained from rational relations using the Hadamard operations and the sum only, thus showing that the mirror is crucial for the study of two-way transducers. We also give some clue on the problem of rational relations whose mirror are also rational.

The purpose of introducing new operators is to possibly capture certain aspects of the behavior of a two-way transducer. Intuitively, the Hadamard star reflects the fact that the transducer can repeatedly execute several computations on the same input but always in the same direction. The possibility for a two-way transducer to read backwards is reflected in the mirror operator. Is there a collection of operators that define two-way transductions the way the rational operators define the rational relations? All we can say

is that in the case of sweeping transducers the operations of sum, Hadamard product, Hadamard star and mirror are precisely those operations that simulate their dynamics (Corollary 4).

Unary two-way transducers (Chapter 5)

We focus on relations in $\Sigma^* \times \Delta^*$ defined by two-way transducers where Σ or Δ or both are unary. We obtain a complete algebraical characterization when both alphabets are unary (Theorem 16): they are precisely what we call Hadamard relations which are the closure of the rational relations under finite union, Hadamard product and Hadamard star (operations defined in Chapter 4). By passing we show that these relations have a very simple expression in which Hadamard product and Hadamard star have depth less than or equal to 1 (Proposition 32). This is reminiscent of the rational relations in $\mathbb{N} \times \mathbb{N}$ which have very simple rational expressions (technically they are semilinear subsets). This implies in particular that any two-way transducer with unary input and output alphabets is equivalent to a sweeping two-way transducer that can be effectively constructed. A natural question is to ask whether or not the result applies to relations defined by more general two-way transducers. A careful consideration on the images of each input allows us to prove that when the input alphabet is no longer unary, even in the case of a unary output alphabet, the Hadamard relations are strictly included in the two-way transductions (Theorem 19). An adhoc argument shows that even when assuming the input alphabet is unary, general two-way transductions strictly include the relations defined by sweeping transducers (Theorem 20).

The central construction developed for the characterization of Theorem 16 allows us to prove additional results in the case where the output alphabet only is unary. Firstly, if the initial two-way transducer is *loop-free*, *i.e.*, if every successful run is linearly bounded in the length of the input[2], we can recover Anselmo's result [2] stating in a more general framework the effective equivalence between such transducers and one-way transducer. Hence, in that case (*e.g.*, functional or unambiguous), the transductions accepted are rational (Corollary 19). This has to be confronted with Theorem 19 mentioned above, which shows that nonfunctional transductions with unary outputs are not even Mirror-Hadamard (the closure of rational transductions under sum, Hadamard product, Hadamard star and mirror). Secondly, by simply ignoring the possible loops, we may tackle the *uniformization problem* of two-way transductions. Given a two-way transduction, the problem consists of asking whether or not it contains a functional two-way transduction of the same domain. Indeed, we prove a strong positive answer: two-way

[2]In fact, it suffices to ask the device to have no "useful" loops, where "useful" means reachable in a successful run and producing a nonempty output.

transductions with unary outputs can be effectively uniformized by one-way (not only two-way) transducers (Corollary 20).

Iteration of binary relations on words (Chapter 6)

We study the composition of binary word relations. More precisely, we consider the *iterated*, *i.e.*, transitive closure, of these relations. For some subfamilies of rational relations we recover the known fact that these transitive closures are non-recursive. In particular, since the computational steps of a Turing machine can be encoded in a synchronous relation, such a relation could have a non-recursive iterated. Moreover, it follows from the work of John H. Conway that the same holds for rational unary functions (Theorem 29).

Working on weaker subfamilies we prove after David Simplot and Alain Terlutte [75] that the iterated of length-preserving rational relations is NL-complete, *i.e.*, it can be nondeterministically recognized in logarithmic space (Theorem 26) and conversely it captures any computations of a logarithmic space bounded Turing machine (Theorem 27). Considering the unary case, by stating a normal form for unary synchronous relations (Theorem 24), we are able to prove that this family is closed under iteration (Theorem 28). This last contribution contrasts with the previously mentioned result where unary rational functions were proved to have a non-recursive transitive closure.

Chapter 2

Preliminaries

In this chapter we collect all the material necessary for the exposition of the rest of the manuscript. With very few exceptions, we adopt the well-established definitions in the literature and we recall the most important results. Some constructions can be considered as routine, but we detail them for the convenience of the reader.

The first section is standard. In the second section, we consistently adopt the formalism of power series in order to study binary relations on the product of free monoids. Finally, in the third section, we introduce different kind of finite state machines, from the most famous one to the more general one. Some basics or known results are given.

2.1 Basic definitions and notations

2.1.1 Sets and monoids

We denote sets by capital letters and the empty set is denoted $\varnothing$. The *powerset* of a X is denoted 2^X. The *cardinality* of X, denoted $|X|$, is the number of elements in X if it is finite, and ∞ otherwise. If $|X|$ is equal to 1, then X is a *singleton*.

A monoid is a structure $\langle M, \cdot, 1\rangle$ where M is a set and $\cdot$ is an associative binary operation on M, admitting a neutral element 1. When the context is clear, we may simply denote it by M. The *closure* of a subset X of M by $\cdot$ is the smallest monoid containing X denoted X^*. Equivalently, it is the smallest set Y such that:

- $X \cup \{1\} \subseteq Y$ and
- $x, x' \in Y \implies x \cdot x' \in Y$

If $\langle M, \cdot \rangle$ is equal to X^* for some $X \subset M$, then M is *generated* by X. We are particularly interested in finitely generated monoids, *i.e.* monoid X^* with X finite. When the elements of $M = X^*$ are the finite sequences of symbols in X, that is, when the $\cdot$ operation is the string concatenation, we say that M is *the free monoid* on (or generated by) X.

When M is a *commutative monoid*, we prefer to use the symbol + instead of $\cdot$ and 0 instead of 1.

This document deals mainly with families of subsets of monoids, essentially free monoids and direct product of free monoids. We give the two most famous families.

Definition 1 (Rational subsets)**.** *Given a monoid M, the family* $\textsc{Rat}(M)$ *of* rational subsets *of M is the least family which contains the singletons and closed under set union,* set product *($(X,Y) \mapsto X \cdot Y := \{x \cdot y \mid x \in X,\ y \in Y\}$) and* Kleene star *($X \mapsto X^* := \{x_1 \cdot \ldots \cdot x_n \mid n \geq 0$ and $x_i \in X\}$, with the convention $x_1 \cdot \ldots \cdot x_n = 1$ if $n = 0$).*

Definition 2 (Recognizable subsets)**.** *Given a monoid M, the family* $\textsc{Rec}(M)$ *of* recognizable subsets *of M is the family of subsets $X \subseteq M$ for which there exists a morphism $\phi : M \to F$ where F is a finite monoid, such that $X = \phi^{-1}(\phi(X))$.*

The family of recognizable subsets of direct product of two monoids $M \times N$ has a simple characterization in terms of the recognizable subsets of each monoids. This important result will be applied where M and N are two free monoids. It extends also to direct product of finitely many monoids, but we will not need this extension.

Theorem 1 (Elgot and Mezei [23])**.** *A subset of the direct product of two monoids M and N is recognizable if and only if it is a finite union of the direct product of a recognizable subset of M and a recognizable subset of N.*

2.1.2 Alphabets, words and languages

An *alphabet* Σ is, in our context, a finite set whose elements are the *symbols*. We denote by Σ^* the free monoid generated by Σ. An element u of Σ^* is a *word*. Its *length* is denoted $|u|$. The *empty word*, denoted ϵ, has length 0. The *concatenation* of two words u, v in Σ^* is denoted uv. The element ϵ is the unit of the concatenation.

A word u is a *factor of a word* v if there exist $u', u'' \in \Sigma^*$, such that $v = u'uu''$. If moreover $u' = \epsilon$ (*resp.* $u'' = \epsilon$) then u is a *prefix* (*resp.* a *suffix*) of v. A factor u of v is *proper* if $u \neq v$.

A *language* on Σ is a set of finite words on Σ *i.e.*, a subset of Σ^*. The concatenation operator extends to languages as follows:

$$L \cdot L' := \{uu' \mid u \in L \text{ and } u' \in L'\}$$

When $L = \{u\}$ (*resp.* $L' = \{v\}$), we simply write $u \cdot L'$ (*resp.* $L' \cdot v$). The structure $\left\langle 2^{\Sigma^*}, \cdot, \{\epsilon\} \right\rangle$ is a monoid. Observe that $\varnothing$ is an absorbing element for the language concatenation.

From language concatenation, it is possible to inductively define *the powers of a language*:

$$L^{(0)} := \{\epsilon\}$$
$$L^{(i+1)} := L^{(i)} \cdot L$$

For a word u and an integer k, we denote u^k the unique element of the singleton $\{u\}^k$. The *Kleene star* of a language is defined as:

$$L^* := \bigcup_{i \geq 0} L^{(i)}$$

In other words, L^* is the closure by $\cdot$ of the subset $L \cup \{\epsilon\}$. We denote by u^* the language $\{u\}^*$ for any word u. Observe that, considering Σ as the language of all words of length 1, the language Σ^* is the set of all finite words. Hence the notation * makes sense.

In the present document, we mainly deal with finite words. However, because it is convenient to view runs in a computation as words, we are led, further in this manuscript, to consider *infinite words* in order to describe non halting computations. In particular, given a finite word u, we denote u^ω the infinite sequence composed of infinitely many successive occurrences of u.

The mirror of a word $u = u_1 \cdots u_n$, where the u_i's are symbols, is the word $\overline{u} = u_n \cdots u_1$. We pose $\overline{\epsilon} = \epsilon$. The mirror is an involution, *i.e.*, $\overline{\overline{u}} = u$ for any u. We may extend the mirror to *the mirror of a language* L, denoted $\overline{L}$, which is the language $\overline{L} = \{\overline{u} \mid u \in L\}$. By involution, for any L we have $\overline{\overline{L}} = L$.

2.1.3 Relations and functions

We denote $X \times Y$ the Cartesian product of X and Y, *i.e.*, the set $\{(x, y) \mid x \in X,\ y \in Y\}$. A subset $R \subseteq X \times Y$ is a *relation* on $X \times Y$. The *domain* of R, denoted $\textsc{Dom}(R)$, is the set:

$$\textsc{Dom}(R) := \{x \mid \exists y \in Y,\ (x, y) \in R\}$$

It is empty if and only if R is empty. If $\textsc{Dom}(R) = X$, the relation R is *total*, otherwise it is *partial*. Given an element $x \in X$, the *image of x by R* is the set:

$$R(x) := \{y \mid (x, y) \in R\}$$

and it is empty if and only if $x \notin \textsc{Dom}(R)$. The *image of R*, denoted $\textsc{Img}(R)$, is the union over X of all images by R, *i.e.*,

$$\textsc{Img}(R) := \bigcup_{x \in X} R(x) = \{y \mid \exists x \in X,\ (x, y) \in R\}$$

It is empty if and only if R is empty. If $\textsc{Img}(R) = Y$, the relation R is said *surjective*. Given an element $y \in \textsc{Img}(R)$, the *pre-image of y by R*, denoted $R^{-1}(y)$, is the set:

$$R^{-1}(y) := \{x \mid y \in R(x)\} = \{x \mid (x, y) \in R\}$$

The relation R is *injective* if for each $y \in \textsc{Img}(R)$, the pre-image of y by R is a singleton. It is *bijective* if it is both injective and surjective.

If for each $x \in \textsc{Dom}(R)$, $R(x)$ is a singleton, R is *functional*. In this case, it is convenient to identify $R(x)$ with its unique element, when the context is clear. Observe that, with our conventions, there are non-functional bijective relations.

Let R be a relation on $X \times Y$. Then, R may be seen as the functional relation $\widetilde{R}$ on $X \times 2^Y$ defined by $\widetilde{R}(x) := \{R(x)\}$ for each $x \in X$. Observe that there exists non-injective relations whose associated functional relation is injective.

Restrictions: Given $X' \subseteq X$ and $Y' \subseteq Y$, the *restriction* of R to $X' \times Y'$, denoted $R_{|X' \times Y'}$, is the intersection of R with $X' \times Y'$, *i.e.*,

$$R_{|X' \times Y'} := R \cap X' \times Y'$$

In particular, the restriction of R to $\textsc{Dom}(R) \times \textsc{Img}(R)$ is a total surjective relation. When $Y' = Y$, we simply write $R_{|X'}$ rather than $R_{|X' \times Y}$.

2.2 Formal power series

A semiring is a structure $\langle \mathbb{K}, +, \cdot, 1, 0 \rangle$ where $+$ and $\cdot$ are two operations usually called *addition* and *multiplication*. The addition provides $\mathbb{K}$ with a structure of a commutative monoid with 0 as neutral element, and the multiplication provides $\mathbb{K}$ with a structure of a monoid with 1 as neutral element. The element 0 is absorbing for the multiplication. Furthermore, the multiplication distributes over the addition. A typical example is the set of non-negative integers $\mathbb{N}$ with the usual operations of addition and multiplication. A *$*$-semiring* is a semiring provided with an internal operator star of arity 1.

2.2.1 General definitions

Let $\langle \mathbb{K}, +, \cdot, 1, 0 \rangle$ be a semiring and let Σ be an alphabet. A *formal power series* σ on Σ^* *with coefficient* in $\mathbb{K}$ is a mapping from Σ^* to $\mathbb{K}$. Formally, we denote σ as an infinite sum:

$$\sigma = \sum_{w \in \Sigma^*} \langle \sigma, w \rangle w$$

where $\langle \sigma, w \rangle$ is *the image of* w, also called the *coefficient* of w, in σ. The series is *constant* if all its images are equal. The *support of* σ, denoted $\textsc{Supp}(\sigma)$, is the language of words with non-zero image in σ, *i.e.*:

$$\textsc{Supp}(\sigma) := \left\{ w \in \Sigma^* \mid \langle \sigma, w \rangle \neq 0 \right\}$$

The set of all series on $\mathbb{K}$ with coefficient in Σ^* is denoted $\mathbb{K}\langle\langle \Sigma^* \rangle\rangle$.

Definition 3. *A* polynomial *is a series with finite support. The family of polynomials is usually denoted* $\mathbb{K}\langle \Sigma^* \rangle$*, but we prefer the more suggestive notation* $\textsc{Pol}(\mathbb{K}\langle\langle \Sigma^* \rangle\rangle)$*, or simply* Pol *when* $\mathbb{K}$ *and* Σ *are understood.*

Definition 4. *A series is* proper *if the coefficient associated to the empty word (the constant term) is* 0 i.e.*, if* $\langle \sigma, \epsilon \rangle = 0$*. The family of proper series is denoted* $\textsc{Prop}(\mathbb{K}\langle\langle \Sigma^* \rangle\rangle)$ *or simply* Prop *when* $\mathbb{K}$ *and* Σ *are understood.*

2.2.2 Rational operations on series

Now, we recall the main operations on series, namely the rational operations.

The sum

The sum over $\mathbb{K}$ extends in a natural way to the *sum of series* as:

$$\sigma + \tau := \sum_{w \in \Sigma^*} \left(\langle \sigma, w \rangle + \langle \tau, w \rangle \right) w$$

Trivially, the *zero series* $0_+ := \sum_{w \in \Sigma^*} 0w$ is the neutral element for the sum. Observe that the sum of two polynomials (*resp.* a proper series) is a polynomial (*resp.* proper series). Hence, both Pol and Prop are closed under sum.

The Cauchy product of series

We define the *Cauchy product of σ and τ* as:

$$\sigma \cdot \tau := \sum_{w \in \Sigma^*} \left(\sum_{uv=w} \langle \sigma, u \rangle \cdot \langle \sigma, v \rangle w \right)$$

The Cauchy product admits a neutral element, the polynomial $1_\odot$ which maps the empty word ϵ to 1 and other words to 0, *i.e.*, the series defined by:

$$\langle 1_\odot, w \rangle := \begin{cases} 1 & \text{if } w = \epsilon; \\ 0 & \text{otherwise.} \end{cases}$$

The families Pol and Prop are closed under Cauchy product.

We can inductively define the powers of σ under the *Cauchy product*:

$$\sigma^k := \begin{cases} 1_\odot & \text{if } k = 0; \\ \sigma \cdot \sigma^{k-1} & \text{otherwise.} \end{cases}$$

In particular, for any σ we have $\sigma^0 = 1_\odot$ and $\sigma^1 = \sigma$.

The Kleene star of a series

The *Kleene star of σ* is:

$$\sigma^* := \sum_{k \in \mathbb{N}} \sigma^k = \sum_{w \in \Sigma^*} \left(\sum_{\substack{u_1 \cdots u_k = w \\ k \in \mathbb{N}}} \langle \sigma, u_1 \rangle \cdots \langle \sigma, u_k \rangle \right) w$$

when the infinite sum of coefficients inside the parenthesis is well-defined.

Observe that when σ is proper, σ^* is always defined since every word has finitely many decompositions. In that case, the coefficient of w in σ^* is equal to:

$$\langle \sigma^*, w \rangle = \left\langle \left(\sum_{0 \leq k \leq |w|} \sigma^k \right), w \right\rangle$$

Notice that the resulting σ^* is not proper since $\langle \sigma^*, \epsilon \rangle = 1$.

Rationally additive semirings

The Kleene star in $\mathbb{K}\langle\langle\Sigma^*\rangle\rangle$ is only a partial operator because it is not always defined for non-proper series (for example, consider the semiring of non-negative integers). There exist different assumptions under which it is defined. Such a condition is that $\mathbb{K}$ satisfies different properties that make it a so-called *rationally additive semiring*. We refer to [25, p.3] for a precise definition. Suffice it to say that for every x, the infinite sum $1+x+x^2+\ldots$, denoted x^*, is well-defined and that further axioms, such as distributivity over infinite sums and associativity extended to infinite sums, are satisfied.

A first consequence is that the Kleene star in $\mathbb{K}\langle\langle\Sigma^*\rangle\rangle$ is always defined. A second one is that the star operator can be extended to square matrices X in $\mathbb{K}^{n\times n}$, in such a way that X^* is the infinite sum of the successive powers of X (the product of matrices is the usual product with the two operations of $\mathbb{K}$, and the (i,j)-entry of X^* is the infinite sum of the (i,j)-entries of the successive powers of X). Formally,

Theorem 2 ([25, Theorem 9])**.** *If $\mathbb{K}$ is rationally additive, then so is $\mathbb{K}^{n\times n}$ for $n>0$.*

Scalar product

Given an element k of $\mathbb{K}$ and a series σ in $\mathbb{K}\langle\langle\Sigma^*\rangle\rangle$, we define the *left* (*resp. right*) *scalar product* of σ by k as follows:

$$\text{(left)}\quad k\cdot\sigma := \sum_{w\in\Sigma^*}\big(k\cdot\langle\sigma,w\rangle\big)w \qquad\qquad \text{(right)}\quad \sigma\cdot k := \sum_{w\in\Sigma^*}\big(\langle\sigma,w\rangle\cdot k\big)w$$

Considering the series $k\epsilon$, which associates k to the empty word and 0 to the other words, we have: $k\epsilon = k\cdot 1_\odot$ where $1_\odot$ corresponds to the series 1ϵ as defined in Section 2.2.2. Trivially, for any k and any σ we have $k\cdot\sigma = (k\epsilon)\cdot\sigma$ and $\sigma\cdot k = \sigma\cdot(k\epsilon)$. Thus, the scalar product may be considered as syntactic sugar referring to a particular Cauchy product.

2.2.3 Rational series

The sum, the Cauchy product and the Kleene star are called *rational operations*. Provided that the semiring $\mathbb{K}$ has the property that the series σ^* is well defined for every σ, we have:

Definition 5. *The family of* rational series*, denoted* $\textsc{Rat}(\mathbb{K}\langle\langle\Sigma^*\rangle\rangle)$ *or simply* RAT*, is the closure of* $\textsc{Pol}(\mathbb{K}\langle\langle\Sigma^*\rangle\rangle)$ *under rational operations.*

Precisely, it is the least family F of series containing POL *and satisfying the three following conditions:*

$$\sigma,\tau\in F\Rightarrow\sigma+\tau\in F \qquad\qquad \sigma,\tau\in F\Rightarrow\sigma\cdot\tau\in F \qquad\qquad \sigma\in F\Rightarrow\sigma^*\in F$$

2.2.4 Restriction to a recognizable language

We introduce a special operation, which consists in restricting the support of a series to a given language. Given a language L and series σ, the *restriction* of σ to L is the series:

$$\sigma_{|L} := \sum_{w \in L} \langle \sigma, w \rangle$$

We are mainly interested in the case where the language L is recognizable (see Definition 2).

Proposition 1 (Support restriction to recognizable language)**.** *Let σ be a series in $\mathbb{K}\langle\langle \Sigma^* \rangle\rangle$ and let L be a recognizable language on Σ. If σ belongs to* Pol, Prop *or* Rat*, then so does $\sigma_{|L}$.*

Proof. We anticipate on the presentation of $\mathbb{K}$-fas (see Definition 11). The Theorem of Kleene-Schützenberger (see, for instance [66, Theorem III.3.5]) establishes the equality between $\mathrm{Rat}(\mathbb{K}\langle\langle \Sigma^* \rangle\rangle)$ and the set of series recognized by 1-way $\mathbb{K}$-fa (see Section 5). It suffices to observe that $\sigma_{|L}$ is accepted by the product of the $\mathbb{K}$-fa accepting σ with the ordinary finite automaton recognizing L. □

2.3 Finite automata

2.3.1 Classical finite automaton

In the case of $M = \Sigma^*$, the well-celebrated Theorem of Kleene gives an alternative definition of the family $\mathrm{Rat}(M)$ (see Definition 1), in terms of *finite automata*. We briefly recall the definition in this paragraph.

There are different versions of finite automata in the literature. We start with the most basic one, to which we refer as the *classical finite automaton* or simply the *classical automaton*, in order to distinguish it from the more general one given later on (see Definition 7). This device is composed in one read-only tape scanned in one direction by an input head, and a finite control, storing at each instant a current state.

Definition 6. *A* Classical finite automaton *(*cfa*) is a tuple $\mathcal{A} = (Q, \Sigma, I, F, \delta)$ where:*

- *Q is a finite set, whose elements are* states*;*
- *Σ is the* input alphabet*;*
- *I and F are two subsets of Q, whose elements are respectively the* initial *and the* accepting *states;*

- δ *is a subset of* $Q \times (\Sigma \cup \{\epsilon\}) \times Q$*, whose elements are the* transitions *of the classical automaton. We assume that* $\delta \cap (F \times \{\epsilon\} \times Q)$ *is empty.*

The behavior of a cFA $\mathcal{A}$ is as expected: an input word $u = u_1 u_2 \cdots u_n \in \Sigma^*$ is written on the tape, one symbol per cell. Initially, the input head is positioned on the leftmost cell, *i.e.*, scanning u_1, and the finite control is in a state in I. At each step, the automaton chooses a transition (q, c, q') in δ such that q is the current state and c is either ϵ or the symbol currently scanned by the input head. Then the automaton enters the state q', and if $c \neq \epsilon$, the input head is moved one cell to the right. The automaton *accepts the word* u if it eventually enters some state in F with the input head positioned on the first empty cell to the right of the input word. Observe that, due to the restriction on δ, no more transition can be applied, and hence the automaton halts. The set of all words accepted by a $\mathcal{A}$ is *the language accepted*, denoted $\|\mathcal{A}\|$. Two automata are *equivalent* if they accept the same language.

Graph representation

The *graph representation* of a classical finite automaton (Q, Σ, I, F) is the edge-labeled graph (Q, E, ℓ) where E is the set of edges $e = (q, q')$ labelled by $\ell(e) = c \in \Sigma \cup \{\epsilon\}$ such that (q, c, q') belongs to δ. Observe that multi-edges are allowed, since two transitions (q, a, q') and (q, b, q') with $a \neq b$ are possible. A computation of the automaton corresponds to a path in the graph. The label of a path is the concatenation of the labels of the edges traversed by the path. In particular, a word is accepted by the automaton if it is the label of a path starting from a vertex in I and ending in a vertex in F.

Matrix representation

Another representation of classical finite automata is algebraic. Consider the $Q \times Q$-matrix M whose coefficient $M_{q,q'}$ is equal to $\{c \in \Sigma \cup \{\epsilon\} \mid (q, c, q') \in \delta\}$. Then, considering the matrix product over the semiring $\langle 2^{\Sigma^*}, \cup, \cdot \rangle$ allows us to define the successive powers of M. The Kleene star of the matrix M is then defined as

$$M^* := \bigcup_{i \geq 0} M^i$$

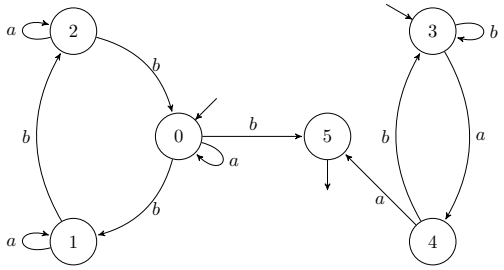
with the convention that M^0 is the $Q \times Q$ matrix with all the coefficients equal to $\{\epsilon\}$ on the diagonal and $\varnothing$ otherwise. The language accepted by the automaton is the finite union:

$$L := \bigcup_{q \in I, q' \in F} M^*_{q,q'}$$

Example 1. *The automaton* $(\{0,1,2,3,4,5\},\{a,b\},\{0,3\},\{5\},\delta)$ *with:*

$$\delta := \left\{ \begin{array}{lllll} (0,a,0) & (1,a,1) & (2,a,2) & (3,a,4) & (4,a,5) \\ (0,b,5) & (0,b,1) & (1,b,2) & (2,b,0) & (3,b,3) \quad (4,b,3) \end{array} \right\}$$

accepts the language of words which either have $3k+1$ *occurrences of* b *for some* $k \in \mathbb{N}$*, or admits a unique factor* aa *which is a suffix.*



$$M := \left(\begin{array}{cccccc} \{\mathbf{a}\} & \{\mathbf{b}\} & \{\epsilon\} & \{\epsilon\} & \{\epsilon\} & \{\epsilon\} \\ \{\epsilon\} & \{\mathbf{a}\} & \{\mathbf{b}\} & \{\epsilon\} & \{\epsilon\} & \{\epsilon\} \\ \{\mathbf{b}\} & \{\epsilon\} & \{\mathbf{a}\} & \{\epsilon\} & \{\epsilon\} & \{\epsilon\} \\ \{\epsilon\} & \{\epsilon\} & \{\epsilon\} & \{\mathbf{b}\} & \{\mathbf{a}\} & \{\epsilon\} \\ \{\epsilon\} & \{\epsilon\} & \{\epsilon\} & \{\mathbf{b}\} & \{\epsilon\} & \{\mathbf{a}\} \\ \{\epsilon\} & \{\epsilon\} & \{\epsilon\} & \{\epsilon\} & \{\epsilon\} & \{\epsilon\} \end{array} \right)$$

Restrictions

A transition in $Q \times \{\epsilon\} \times Q$ is called an *ϵ-transition*. If $\delta \subseteq Q \times \Sigma \times Q$, *i.e.*, if δ contains no ϵ-transition, the automaton is said *ϵ-free*. If I is a singleton, the automaton is said *initial*. It is *deterministic* if it is initial and for each $q \in Q$ and each $c \in \Sigma$, there exists at most one transition in $\{q\} \times \{c,\epsilon\} \times Q$. It is *complete* if it is ϵ-free and for each $q \in Q$ and $c \in \Sigma$, there exists at least one q' such that $(q,c,q') \in \delta$. The following results are well-known and we omit the proofs.

Proposition 2.

- *Every* n*-state* cFA *is equivalent to an initial* ϵ*-free* $(n+1)$*-state* cFA*.*
- *Every* n*-state* ϵ*-free* cFA *is equivalent to a complete* $(n+1)$*-state* cFA*. The construction preserves determinism and the property of being initial.*
- *Every* n*-state* cFA *is equivalent to a deterministic* cFA *with at most* 2^n*-state.*

The bounds are tight and each statement is effective.

Characterization

As mentioned above, the family of languages accepted by cFAs is exactly the family $\textsc{Rat}(M)$. This is the famous Theorem of Kleene.

Theorem 3 (Kleene)**.** *A language is accepted by a* cFA *if and only if, it is rational.*

2.3.2 Two-way finite automata

Here we introduce the computational model which is the basis of the investigations in this manuscript. The definition we choose for *finite automaton* is general, in the sense that it refers to the largest commonly agreed extension of cFA that does not increase the computational power of the device. More precisely, *finite automaton* refers to the model known as "two-way nondeterministic finite automaton" (see for example [38]). Other simpler versions, such as "one-way deterministic finite automaton", are obtained by restricting this general model.

Model

The notion of 2-way finite automaton was introduced at the very beginning of the Theory. Rabin and Scott [63], and independently Shepherdson [72], proved it to be equivalent to the cFA model. The main difference with cFAs is that the head may move in both directions or stay in place. Two special symbols are required to mark the left and right borders of the input and to prevent the computation to leave the input space.

Definition 7. *A* finite automaton *(*FA*) is a tuple* $\mathcal{A} = (Q, \Sigma, \triangleright, \triangleleft, I, F, \delta)$ *where:*

- Q *is a finite set, whose elements are the* states *of the finite automaton;*
- $I \subseteq Q$ *is the set of* initial states*;*
- $F \subseteq Q$ *is the set of* accepting states*;*
- $\triangleright$ *and* $\triangleleft$ *are two special symbols not belonging to* Σ*, respectively called* left *and* right endmarkers*. The set* $\Sigma \cup \{\triangleright, \triangleleft\}$ *is denoted* $\Sigma_{\triangleright\triangleleft}$*.*
- δ *is a subset of* $Q \times \Sigma_{\triangleright\triangleleft} \times \{-1, 0, 1\} \times Q$ *whose elements are the* transitions *of the automaton. It satisfies the following three restrictions:*

  1. $(q, \triangleright, d, q') \in \delta \Rightarrow d \in \{0, +1\}$*;*
  2. $(q, \triangleleft, d, q') \in \delta \Rightarrow d \in \{-1, 0\}$*;*
  3. $(q, \triangleleft, d, q') \in \delta \Rightarrow q \notin F$*;*

The *direction* of a transition is its $\{-1, 0, 1\}$-component. A transition is *stationary* if $d = 0$, it is *restless* otherwise.

We informally describe the dynamics of the device. Given an input word $u = u_1 \cdots u_n$ on Σ we augment it to $\tilde{u} = u_0 \cdot u_1 \cdots u_n \cdot u_{n+1}$ where $u_0 = \triangleright$ and $u_{n+1} = \triangleleft$. The automaton starts the computation with the word $\tilde{u}$ written on the tape, the input head positioned on

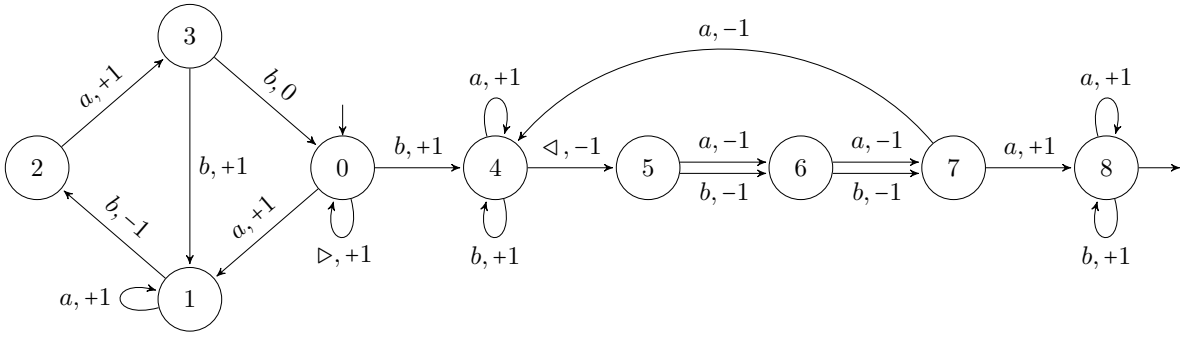


FIGURE 2.1: The graphical representation of the FA $\mathcal{A}$ given in Example 2. The state of the FA are the vertices while each transition (q, a, d, q') is depicted as an edge from q to q' labelled by a, d. The initial and accepting states are denoted by an incoming and an outgoing arrow respectively.

the leftmost cell scanning u_0, and in some state $q_- \in I$. At each step, the automaton reads the input symbol $a \in \Sigma_{\triangleright\triangleleft}$ scanned by the head, and according to its current state q chooses a direction d and a state q' with $(q, a, d, q') \in \delta$. Then it enters the state q' and moves its head one cell to the left or to the right, according to whether or not d is equal to -1 or $+1$. If $d = 0$ the head does not move. The FA *accepts* the input word u if it eventually enters an accepting state at the rightmost position, u_{n+1}. Observe that the purpose of the restriction 3 is to enforce the computation to halt in this case. Moreover, because of the restrictions 1 and 2, the input head cannot move out of $\tilde{u}$ during the computation. The set of all words accepted by a FA $\mathcal{A}$ is the *language accepted*, denoted $\|\mathcal{A}\|$. Two FAs are *equivalent* if they accept the same language.

Example 2. *We give a non-trivial example of a 2-way finite automaton $\mathcal{A}$, which is depicted in Figure 2.1. Formally $\mathcal{A} = (Q, \Sigma, \triangleright, \triangleleft, I, F, \delta)$, where $Q = \{0,1,2,3,4,5,6,7,8\}$, $\Sigma = \{a, b\}$, $I = \{0\}$, $F = \{8\}$ and the transition set δ is:*

$$\delta = \left\{ \begin{array}{llll} (0, \triangleright, +1, 0) & (1, a, +1, 1) & & (5, a, -1, 6) \quad (7, a, +1, 8) \\ (0, a, +1, 1) & (1, b, -1, 2) & (4, \triangleleft, -1, 5) & (5, b, -1, 6) \quad (7, a, -1, 4) \\ (0, b, +1, 4) & & (4, a, +1, 4) & \\ & (3, b, 0, 0) & (4, b, +1, 4) & (6, a, -1, 7) \quad (8, a, +1, 8) \\ (2, a, +1, 3) & (3, b, +1, 0) & & (6, b, -1, 7) \quad (8, b, +1, 8) \end{array} \right\}$$

The automaton accepts a word w if and only if it contains at least one occurrence of b and has an occurrence of a in position $|w| - 2$ (in particular, $|w| \geq 3$). The language accepted is thus $\|\mathcal{A}\| = \Sigma^* a \Sigma^2 \cap \Sigma^* b \Sigma^*$.

Configurations, runs, traces

Time, when dealing with cFA, is an implicit variable. With 2-way automata, some caution is required, since it has to be explicitly incorporated in the proofs. The notions of *configuration* and *run* are standard, since they are adaptations to automata of the same notions for Turing Machines. The concept of *trace* is more specific and will prove useful when extending FAs to *2-way* $\mathbb{K}$-FA*s* (see Section 2.3.3) or to *2-way transducers* (see Section 2.3.4).

The description of the system at a fixed time is given by the current state and the input head position: a *configuration* of $\mathcal{A}$ on $u = u_1 \cdots u_n$ is a pair (q, p) where q is a state and p is a *position* of $\tilde{u}$, *i.e.*, an integer such that $0 \leq p \leq n+1$. A configuration is *initial* (*resp. accepting*) if it is of the form $(q, 0)$ with $q \in I$ (*resp.* $(q, n+1)$ with $q \in F$). We call *border configuration*, any configuration whose position is equal to 0 (*left border*) or $n+1$ (*right border*). A non-border configuration is *central*.

The successor relation on configurations can be defined from the transition set. A pair of configurations $((q, p), (q', p'))$ belongs to the *successor relation*, written $(q, p) \to (q', p')$, if the automaton may enter (q', p') from (q, p) in one step, that is if $(q, u_p, (p' - p), q')$ belongs to δ. In particular $(p' - p)$ has to be equal to -1, 0 or 1. Observe that the relation depends on the input word u. A configuration that has no successor is said *halting*. For instance, the configuration $(2, 2)$ of the automaton given in Example 2 on the word *abbaaa* is halting, since no transition can be performed from state 2 when reading the symbol b. The restriction 3 on δ in Definition 7 enforces every accepting configuration to be halting.

With a fixed input $u \in \Sigma^*$ of length n, the set $Q \times \{0, \ldots, n+1\}$ is finite. It is convenient to consider it as an alphabet and sequences of configurations as possibly infinite words.

Definition 8. *A* run *of* $\mathcal{A}$ *on* u *is a possibly infinite and non-empty word* $\mathbf{r} = c_0 c_1 \cdots$ *on* $(Q \times \{0, \ldots, |u|+1\})$ *such that each consecutive symbols in* $\mathbf{r}$ *are successive configurations,* i.e., *for each* i, *we have* $c_i \to c_{i+1}$.

A run is:

- trivial *if it is reduced to a single configuration,* i.e., $|\mathbf{r}| = 1$;
- initial *if its first configuration is initial;*
- halting *if it is finite and its last configuration is halting;*
- successful *if it is initial, finite and its last configuration is accepting.*

Example 3. *We consider the automaton* $\mathcal{A}$ *given in Example 2 and depicted in Figure 2.1.*

On the input word abbaab, it admits the following four runs:

$\mathbf{r}_1 := (0,0)(0,1)(1,2)(2,1)(3,2)(0,2)$ (*in black on Figure 2.2a and 2.2b*)
$\mathbf{r}_2 := (4,3)(4,4)(4,5)(4,6)(4,7)(5,6)(6,5)(7,4)$ (*in blue on Figure 2.2b*)
$\mathbf{r}_3 := (8,5)(8,6)(8,7)$ (*in green on Figure 2.2a*)
$\mathbf{r}_4 := (0,0)(0,1)(1,2)(2,1)(3,2)(1,3)(2,2)$ (*in red on Figure 2.2c*)

Observe that $\mathbf{r}_1 \cdot \mathbf{r}_2 \cdot \mathbf{r}_3$ *is a successful run (see Figure 2.2a). Obviously,* $\mathbf{r}_1 \cdot \mathbf{r}_2^\omega$ *is an initial infinite run (Figure 2.2b) and* $\mathbf{r}_1 \cdot \mathbf{r}_4$ *is a non-successful halting run, since the configuration* $(2,2)$ *is halting but not accepting (Figure 2.2c).*

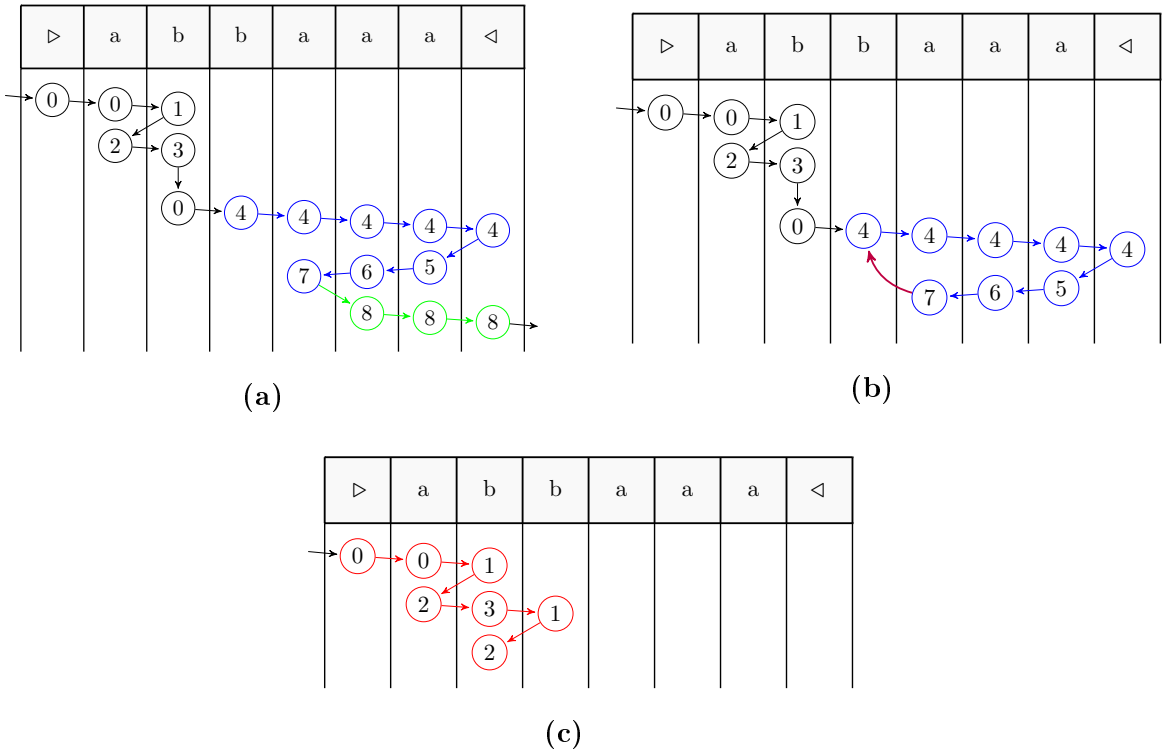


FIGURE 2.2: The three examples of runs given in Example 2.2 (see the automaton in Figure 2.1): (a) is the successful run $\mathbf{r}_1 \cdot \mathbf{r}_2 \cdot \mathbf{r}_3$, (b) is the initial infinite run $\mathbf{r}_1 \cdot \mathbf{r}_2^\omega$ and (c) is the non-successful halting run $\mathbf{r}_4$. The run $\mathbf{r}_1$ is in black, $\mathbf{r}_2$ is in blue, $\mathbf{r}_3$ is in green and $\mathbf{r}_4$ is in red. (Initial and accepting configurations are indicated by incoming and outgoing arrows respectively.)

By definition, an input word u is accepted by an automaton $\mathcal{A}$ if there exists a successful run of $\mathcal{A}$ on u.

Trace: The following notion is probably superfluous when dealing with automata but it is instrumental when working with transducers (see Section 2.3.4) or $\mathbb{K}$-automata (see Section 2.3.3). In the definition below, we identify the transition set δ with an alphabet.

Definition 9. *The trace of a run* $\mathbf{r} = (q_0, p_0)(q_1, p_1)\cdots$ *of* $\mathcal{A}$ *on* u *is the (possibly infinite) word* $\mathbf{t_r} = t_1 t_2 \cdots$ *in* δ^* *such that for each* $0 < i < |\mathbf{r}|$, t_i *is the witness of* $(q_{i-1}, p_{i-1}) \to (q_i, p_i)$ i.e., $t_i = (q_{i-1}, u_{p_{i-1}}, p_i - p_{i-1}, q_i)$.

Example 4. *The trace of the run* $\mathbf{r}_4$ *introduced in Example 3 and depicted in Figure 2.2c, is the word:*

$$(0, \triangleright, +1, 0)(0, a, +1, 1)(1, b, -1, 2)(2, a, +1, 3)(3, b, +1, 1)(1, b, -1, 2)$$

Notations: Considering runs as words allows us to reuse the classic tools, introduced in Section 2.1.2, such as language, factor, concatenation. However, runs are really particular words: they are non-empty and locally consistent, *i.e.*, two consecutive letters represent two successive configurations. For convenience and in order to better specify the context, we change the notation introduced in Section 2.1.2 when speaking of runs. We denote runs (words) with bold letters, and we index their configurations (symbols) from 0 on, *e.g.* $\mathbf{r} = c_0 c_1 \cdots$. So $\mathbf{r}_0$ refers to the first configuration of the run $\mathbf{r}$. Observe that if $\mathbf{r}$ is finite of length n, its last configuration is c_{n-1}, and its trace has length $n-1$.

Run composition: Since the concatenation of two runs is not necessarily a run, we introduce a new controlled concatenation operator, called *run composition* and denoted @. This operator is partial. For two runs $\mathbf{r}$ and $\mathbf{s}$ with $\mathbf{r}$ finite of length n, the (controlled) composition of $\mathbf{r}$ and $\mathbf{s}$ is defined when $\mathbf{r}_{n-1} = \mathbf{s}_0$ by $\mathbf{r}@\mathbf{s} = \mathbf{r}_0 \cdots \mathbf{r}_{n-1} \mathbf{s}_1 \cdots$. It is left undefined otherwise. Run composition differs from concatenation, since one common symbol ($\mathbf{s}_0$) at the interface of both factors is deleted. When $\mathbf{r}@\mathbf{s}$ is defined, $\mathbf{r}$ is said to be *composable with* $\mathbf{s}$.

Loops: A finite run $\mathbf{r} = \mathbf{r}_0 \cdots \mathbf{r}_{n-1}$ is a *loop* if $\mathbf{r}_0 = \mathbf{r}_{n-1}$ *i.e.*, if it is self-composable. Moreover, if none of its proper factors is a loop, $\mathbf{r}$ is a *simple loop*. When $\mathbf{r}_0 = \mathbf{r}_{n-1} = c$ is known, we say that $\mathbf{r}$ is a *c-loop*. Trivially, if $\mathbf{r}$ and $\mathbf{s}$ are two c-loops for some configuration c, $\mathbf{r}@\mathbf{s}$ exists and is a c-loop. In particular, for any c-loop $\mathbf{r}$, $\mathbf{r}@\mathbf{r}$ is a c-loop. This leads to the inductive definition of powers of a c-loop: $\mathbf{r}^{@0} = c$ and $\mathbf{r}^{@k+1} = \mathbf{r}^{@k}@\mathbf{r}$. The set of all $\mathbf{r}^{@k}$

is denoted $\mathbf{r}^{@*}$. The run $\mathbf{r}@\mathbf{r}@\cdots$ is denoted $\mathbf{r}^{@\omega}$ and is infinite if $\mathbf{r}$ is non-trivial. A run is said *loop-free* if none of its factor is a non-trivial loop, *i.e.*, none of its configurations occurs twice.

Example 5. *We consider the runs introduced in Example 3. The run* $\mathbf{r}_2 \cdot (4,3)$ *(see the blue and purple path in Figure 2.2b) is a loop. However, each run* $\mathbf{r}_1$, $\mathbf{r}_2$, $\mathbf{r}_3$, *and* $\mathbf{r}_4$ *is loop-free. The run* $\mathbf{r}_1 \cdot \mathbf{r}_2 \cdot \mathbf{r}_3$ *(see Figure 2.2a) is a successful loop-free run on* $u = abbaaa$. *Since* $\mathbf{r}_2 \cdot (4,3)$ *is a loop, we may generate an infinite family of successful run* $(\mathbf{s}_k)_{0<k}$ *as follows:*

$$\mathbf{s}_k = \mathbf{r}_1 \cdot (\mathbf{r}_2 \cdot (4,3))^{@k} \cdot \mathbf{r}_2 \cdot \mathbf{r}_3$$

Due to the existence of loops, finite automata may admit infinite and unbounded finite runs. The following lemma states the more or less trivial fact that eliminating loops in a finite run leads to a loop-free run. Of course, there are many different ways of eliminating these loops, resulting in different loop-free runs (see Figure 2.3).

Lemma 1. *For every* $u \in \Sigma^*$ *and every finite run* $\mathbf{r}$ *on* u, $\mathbf{r}$ *can be factored into:*

$$\mathbf{r} = \lambda(c_0)\lambda(c_1)\cdots\lambda(c_\ell)$$

such that $c_0c_1\cdots c_\ell$ *is a loop-free run on* u *and each* $\lambda(c_i)$ *is a* c_i*-loop.*

Proof. Given $\mathbf{r} = \mathbf{r}_0\cdots\mathbf{r}_{n-1}$, we proceed by induction on n. If $n = 1$, then we choose $c_0 = \mathbf{r}_0$ and we set $\lambda(c_0)$ to be the trivial c_0-loop c_0. Let $n > 1$. By induction, there exists a loop-free run $c_1\cdots c_{\ell-1}$ and for each $1 \leq i < \ell$, there exists a c_i-loop $\lambda(c_i)$ such that $\mathbf{r}_1\cdots\mathbf{r}_{\ell-1} = \lambda(c_1)\cdots\lambda(c_{\ell-1})$. Furthermore, we have $\mathbf{r}_0 = \lambda(c_0)$ as in the case $n = 1$.

Thus we can write $\mathbf{r} = \lambda(c_0)\lambda(c_1)\cdots\lambda(c_{\ell-1})$. If c_0 is different from all the c_is with $i > 0$, then we are done. Otherwise, for some i, $c_0 = c_i$. We set $\lambda'(c_0) = \lambda(c_0)\lambda(c_1)\cdots\lambda(c_i)$ which is a c_0-loop. Then we get $\mathbf{r} = \lambda'(c_0)\lambda(c_{i+1})\cdots\lambda(c_{\ell-1})$. □

A direct consequence is that every accepted word is accepted through a loop-free run.

Corollary 1. *If* u *is accepted by* $\mathcal{A}$, *then there exists a loop-free successful run of* $\mathcal{A}$ *on* u.

Border and central runs, hits: We now define other particular types of runs. A run is *border* if its first configuration is border and if its last configuration, when it exists, is border too. A run is *central* if all its configurations are central. In particular, these definitions apply to loops. Observe that there exist runs which are neither border nor central.

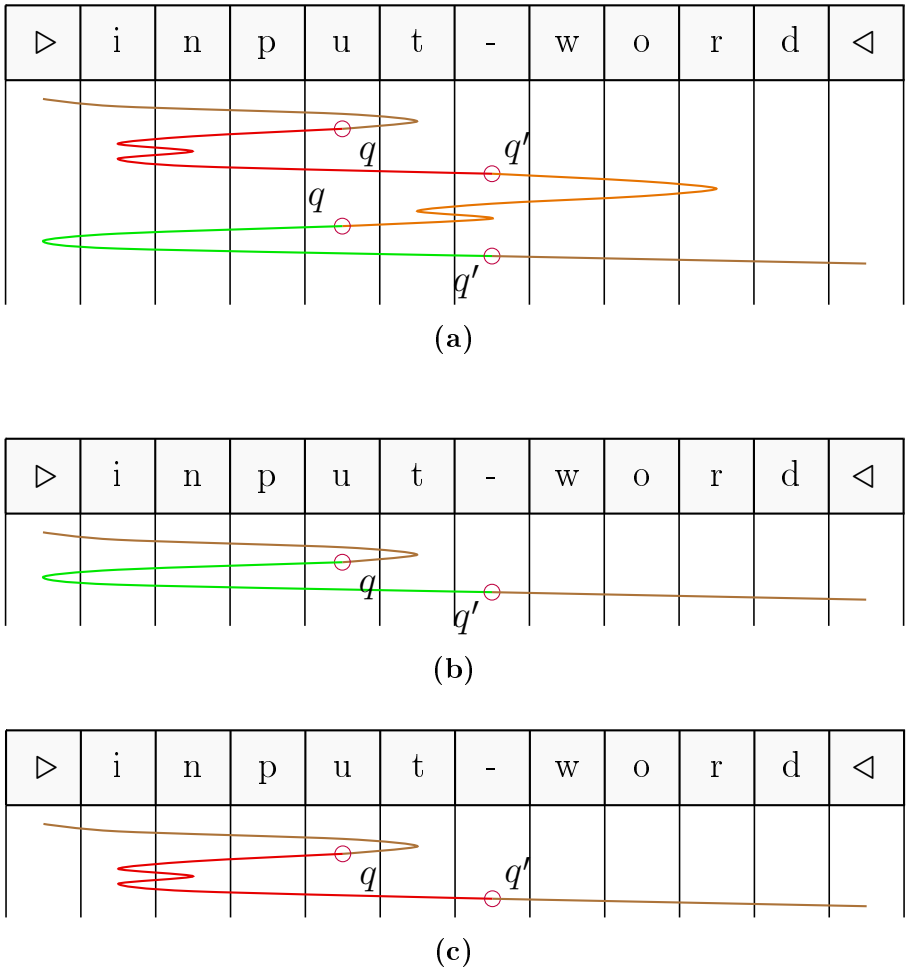


FIGURE 2.3: Two loop-free runs (b) and (c) obtained from a run (a). The run (a) has two loops: a q-loop (`red@orange`) and a q'-loop (`orange@green`). Cutting one or the other leads to two different loop-free runs (b) and (c).

A run on u is a *hit*, if it is border, finite and if its only border configurations are its first and last configurations. We may also determine a hit by a pair of *border points* which are of the form $(q, \triangleright)$ or $(q, \triangleleft)$ for some $q \in Q$ with the natural meaning. For two border points b_0 and b_1, we speak of a b_0 *to* b_1 *hit* on u. A border configuration (q, p) *satisfies* a border point (q', b) if $q = q'$ and $u_p = b$.

Because initial and accepting configurations are border, every successful run is a finite composition of hits.

Lemma 2. *Any successful run* $\mathbf{r}$ *can be uniquely factored as:*

$$\mathbf{r} = \mathbf{h_1} @ \mathbf{h_2} @ \dots @ \mathbf{h}_\ell$$

where each $\mathbf{h_i}$ *is an non-trivial hit.*

This factorization is called the *hit factorization* of the successful run.

Example 6. *For example, the successful run* $\mathbf{r}_1 \cdot \mathbf{r}_2 \cdot \mathbf{r}_3$ *(see Figure 2.2a and Example 3) contains two hits:*

$$\mathbf{h}_1 := \mathbf{r}_1 \cdot (4,3)(4,4)(4,5)(4,6)(4,7) \qquad \textit{and} \qquad \mathbf{h}_2 := (4,7)(5,6)(6,5)(7,4) \cdot \mathbf{r}_3$$

More precisely, $\mathbf{h}_1$ *is a* $(0, \triangleright)$ *to* $(4, \triangleleft)$ *hit and* $\mathbf{h}_2$ *is a* $(4, \triangleleft)$ *to* $(8, \triangleleft)$ *hit. The hit factorization of* $\mathbf{r}_1 \cdot \mathbf{r}_2 \cdot \mathbf{r}_3$ *is* $\mathbf{h}_1 @ \mathbf{h}_2$.

One-directed runs: A run $\mathbf{r}$ is *one-directed* if the successive positions of its configurations are monotone. Formally, this holds if its trace either belongs to ${\delta_{|\{0,1\}}}^*$ or to ${\delta_{|\{-1,0\}}}^*$. Since every step is one-directed, every run may be factored into one-directed factors.

The *number of reversals of a run* is the minimal number of factors in a decomposition into one-directed factors minus 1.

For instance, the number of reversals of the run $\mathbf{r}_1 \cdot \mathbf{r}_2 \cdot \mathbf{r}_3$ of Example 3 (see Figure 2.2a) is equal to 4. A successful run has an even number of reversals.

Restricted FAs

As for cFAs, the model admits restricted versions. Some of them are defined by a property that the structure of the automaton has to satisfy, while others are properties of the dynamic of the device *i.e.*, properties on the runs. We recall below that every variant is in fact equivalent to the most constrained one. This known result leads to some normal forms of FAs or to some restrictions. We consider a fixed FA $\mathcal{A} = (Q, \Sigma, \triangleright, \triangleleft, I, F, \delta)$.

Structural restrictions: The FA $\mathcal{A}$ is said *initial* (*resp. final*) if I (*resp.* F) is a singleton. It is *deterministic* if it is initial and for any state q and any symbol c in $\Sigma_{\triangleright\triangleleft}$, there exists at most one $q' \in Q$ and $d \in \{-1, 0, 1\}$, such that $(q, c, d, q') \in \delta$. It is *complete* if for each pair (q, c), there exists at least one pair (q', d) such that $(q, c, d, q') \in \delta$. It is *outer-nondeterministic* if nondeterministic choices may happen at the endmarkers only *i.e.*, if for each q and $c \notin \{\triangleright, \triangleleft\}$, there exists at most one pair (q', d) such that $(q, c, d, q') \in \delta$.

We say that $\mathcal{A}$ is *1-way* (*resp. restless*) if $(q, c, d, q') \in \delta$ implies $d \neq -1$ (*resp.* $d \neq 0$). It is *sweeping* if the input head may change its direction when scanning an endmarker only and it is *rotating* if moreover it does not change state on its way back to the left endmarker. Formally:

Definition 10. *An automaton* $(Q, \Sigma, \triangleright, \triangleleft, I, F, \delta)$ *is* sweeping *if there exists a partition* $Q_{-1} \cup Q_{+1}$ *of* Q*, such that* δ *is included in the union of the following sets:*

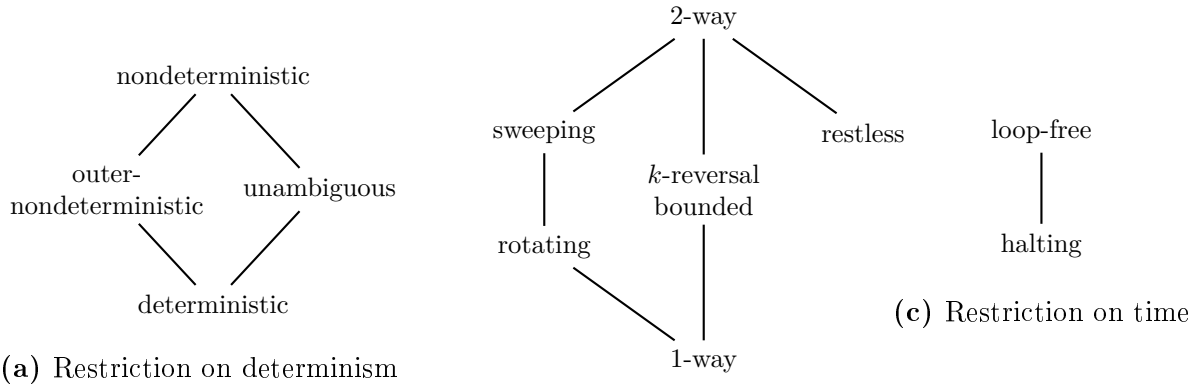
- $Q \times \{\triangleright\} \times \{0, 1\} \times Q_{+1}$
- $Q \times \{\triangleleft\} \times \{-1, 0\} \times Q_{-1}$
- $Q_d \times \Sigma \times \{0, d\} \times Q_d$ *for each* $d = -1, 1$.

If moreover each transition in $Q_{-1} \times \Sigma \times \{0, d\} \times Q_{-1}$ *is of the form* $(q, a, -1, q)$*, then the automaton is said* rotating.

The automaton $\mathcal{A}$ is *unary* if Σ is unary.

Dynamical restrictions: We say that $\mathcal{A}$ if *unambiguous*, if for every input u there exists at most one successful run of $\mathcal{A}$ on u. If every successful (*resp.* initial) run of $\mathcal{A}$ is loop-free, then $\mathcal{A}$ is said *loop-free* (*resp. halting*). In particular, observe that halting implies loop-free. If the number of reversals is bounded by k in every successful run, the automaton is said *k-reversal bounded*.

Summary of the restrictions: We have split the restrictions of FAs in two categories: structural and dynamical, but we can classify them based on their logical implications. Indeed, some of the restrictions act on the determinism of the device (outer-nondeterminism, unambiguousity and determinism). Others act on the head moves (k-reversal boundedness, sweepingness, rotatingness, one-wayness and restlessness). The third type acts on the time (loop-freeness and haltingness). The three classes of restriction are depicted in Figure 2.4, where a edge means that the lower implies the upper one.



(a) Restriction on determinism

(b) Restriction on head moves

(c) Restriction on time

FIGURE 2.4: The main restrictions of FAs, by type. An edge stands for implication of the upper restriction by the lower one.

Equivalences and normal forms

All restricted versions of FA are known to be equivalent to the most constrained one, namely the 1-way restless deterministic FA, which is itself equivalent to cFA. In this section, we present some classical constructions that detail the simulation between different variants. Some kind of normal forms for restricted FAs follows from these constructions. Additionally, we estimate the cost, in terms of states, of each construction.

In the same way as for cFAs (Proposition 2), every FA can be made initial.

Proposition 3. *Every n-state* FA *is effectively equivalent to an initial $(n+1)$-state* FA. *Moreover, the construction preserves all the restrictions introduced in the previous paragraph.*

Proof. Let $\mathcal{A} = (Q, \Sigma, \triangleright, \triangleleft, I, F, \delta)$ be a FA and let q_- be a fresh state *i.e.*, $q_- \notin Q$. We define the set $\delta_- := \{(q_-, \triangleright, d, q') \mid (q, \triangleright, d, q') \in \delta \text{ and } q \in I\}$. Observe that the automaton $\mathcal{A}_- := ((Q \cup \{q_-\}), \Sigma, \triangleright, \triangleleft, \{q_-\}, F, (\delta_- \cup \delta))$ is equivalent to $\mathcal{A}$. Moreover, the construction cannot alter the structural and dynamical restrictions. □

It is well-known that the most restricted variant of FA is equivalent to the most general one (see for instance [38]).

Theorem 4. *Every* FA *is effectively equivalent to a 1-way restless deterministic* FA.

2.3.3 Two-way weighted-automata

A 2-way finite automaton accepts a language. That is, it scans an input word and either accept or reject (by halting in a non-accepting state, or by never halting). The "output" of a 2-way finite automaton is thus a Boolean value. We now introduce *weighted automata* ($\mathbb{K}$-*automata*), as an extension of FAs, which is provided with the ability to associate with each run, an *output*. The definition is generic, since the output space is an arbitrary semiring $\mathbb{K}$.

Definition 11 ($\mathbb{K}$-automaton)**.** *Given a semiring $\mathbb{K}$, a* $\mathbb{K}$-Automaton *($\mathbb{K}$-FA) is a pair $\mathcal{K} = (\mathcal{A}, \phi)$ where $\mathcal{A}$ is a finite automaton with transition set δ, and ϕ, called* production function*, is a mapping of δ into $\mathbb{K}$. The image of a transition by ϕ is called* the multiplicity *of the transition.*

A pair $(u, k) \in \Sigma^* \times \mathbb{K}$ is recognized *by $\mathcal{K}$ if there exists a successful run of $\mathcal{A}$ on u, with trace $t_1 \cdots t_n$ such that $k = \phi(t_1) \cdot \ldots \cdot \phi(t_n)$. The image of u by $\mathcal{K}$, denoted $\langle \mathcal{K}, u \rangle$, is the sum of such k, if it is defined and 0 otherwise. In particular, if u is not accepted by $\mathcal{A}$, $\langle \mathcal{K}, u \rangle = 0$. The series* recognized *by $\mathcal{K}$ is the series:*

$$||\mathcal{K}|| := \sum_{u \in \Sigma^*} \langle \mathcal{K}, u \rangle u$$

Two $\mathbb{K}$-automata are *equivalent* if they accept the same series. Since a $\mathbb{K}$-FA has an underlying FA, we extend in a natural way, the notions of configurations, runs and traces to $\mathbb{K}$-FA. We may hence speak of hits, loops or successful runs of a $\mathbb{K}$-FA.

Two-way $\mathbb{K}$-FA on the Boolean semiring

If $\mathbb{K} = \langle \mathbb{B}, \vee, \wedge, \text{TRUE}, \text{FALSE} \rangle$, a $\mathbb{K}$-FA $\mathcal{K} = (\mathcal{A}, \phi)$ is no more than an FA. Indeed, supposing its production function never maps a transition to 0, its recognized series is the characteristic series of the language accepted by its underlying automaton, *i.e.*,

$$||\mathcal{K}|| = \sum_{w \in ||\mathcal{A}||} \text{TRUE} \quad w \tag{2.1}$$

The support of a $\mathbb{K}$-FA is the support of its recognized series. The following trivial result shows that we can always suppose that ϕ maps δ into $\mathbb{K} \setminus \{0\}$.

Proposition 4. *Given a semiring $\mathbb{K}$ of zero 0, every $\mathbb{K}$-FA is equivalent to another $\mathbb{K}$-FA $(\mathcal{A}, \phi)$ such that $\phi(t) \neq 0$ for any transition t of $\mathcal{A}$.*

Proof. Observe that if for some transition t we have $\phi(t) = 0$, then, every run which uses the t are mapped to 0, by absorption. Hence, restricting ϕ and δ to the set of transition which have non-0 images leads to an equivalent $\mathbb{K}$-FA. □

Restricted version

As for FA, we define a few restricted variants. Most of them are simply a restriction on the underlying automaton.

Given a $\mathbb{K}$-FA $\mathcal{K} = (\mathcal{A}, \phi)$, we say that it is respectively *deterministic*, *outer-nondeterministic*, *1-way*, *restless*, *sweeping*, *loop-free*, *halting*, *k-reversal bounded* for some $k > 0$, depending on which properties are satisfied by $\mathcal{A}$. It is *rotating* if $\mathcal{A}$ is rotating and ϕ maps every backward transition, *i.e.*, in $Q \times \Sigma_{\triangleright\triangleleft} \times \{-1\} \times 1$ to 1.

It is well-known that the family of series recognized by 1-way $\mathbb{K}$-FA is the family of rational series in $\mathbb{K}\langle\langle \Sigma^* \rangle\rangle$, *e.g.*, [21, Theorem VII.5.1].

Theorem 5 (Kleene-Schützenberger Theorem)**.** *One-way $\mathbb{K}$-FA's recognize exactly the family of rational series.*

Basic equivalences

When studying the dynamic of a 2-way device, we often require it to be restless. This restriction does not alter the computational power. The following result states the above mentioned equivalence, taking into consideration the size of the equivalent device.

Proposition 5. *Given an n-state $\mathbb{K}$-FA $\mathcal{K}$, we can effectively build a $3n$-state restless $\mathbb{K}$-FA $\mathcal{K}'$, preserving determinism, outer-nondeterminism, unambiguity, loop-freeness and haltingness.*

Proof. Let $\mathcal{K} = (\mathcal{A}, \phi)$ be a fixed $\mathbb{K}$-FA of transition set δ.

The construction is based on a really simple idea: we build a $\mathbb{K}$-FA $\mathcal{K}'$ which simulates each step $(q,p)(q',p')$ of $\mathcal{K}$ either directly, if the step is restless (that is $|p'-p| = 1$), or, if $p = p'$, by a two-step restless run $(q,p)(\overline{q'}, p+d)(q',p)$, where $\overline{q'}$ is a fresh copy of q' storing some $d \in \{-1,1\}$. Due to the restriction when scanning the endmarkers, we need to use both $d = -1$ and $d = 1$. Formally, we create two fresh copies of Q: $\overleftarrow{Q}$ and $\overrightarrow{Q}$ and we define the sets:

$$\delta_{\text{ZIG}} := \left\{(q, c, 1, \overleftarrow{q'}) \mid (q, c, 0, q') \in \delta \text{ and } c \neq \triangleleft\right\} \cup \left\{(q, \triangleleft, -1, \overrightarrow{q'}) \mid (q, \triangleleft, 0, q') \in \delta\right\}$$

and:

$$\delta_{\text{ZAG}} := \{(\overleftarrow{q}, c, -1, q) \mid c \in \Sigma_{\triangleright\triangleleft}\} \cup \{(\overrightarrow{q}, c, 1, q) \mid c \in \Sigma_{\triangleright\triangleleft}\}$$

Denoting $\delta_{|\{-1,1\}}$ the set of restless transitions of $\mathcal{A}$, we define the finite automaton:

$$\mathcal{A}' := \left(\left(Q \cup \overleftarrow{Q} \cup \overrightarrow{Q}\right), \Sigma, \triangleright, \triangleleft, I, F, \left(\delta_{|\{-1,1\}} \cup \delta_{\text{ZIG}} \cup \delta_{\text{ZAG}}\right)\right)$$

By construction it is restless. Moreover, there is a simple one-to-one correspondence that maps the trace of each run of $\mathcal{A}$ into the trace of a run of $\mathcal{A}'$. Thus, the construction preserves determinism, outer-nondeterminism, unambiguity, loop-freeness and haltingness. This strong correlation between the simulated and simulating runs also allows us to define the production function ϕ' of $\mathcal{K}'$ as follows:

$$\text{for each transition } t' \text{ of } \mathcal{A}' \qquad \phi'(t') := \begin{cases} \phi(t') & \text{if } t' \text{ belongs to } \delta_{|\{-1,+1\}} \\ \phi((q,a,0,q')) & \text{if } t' = \left(q,a,1,\overleftarrow{q'}\right) \\ \phi((q,\lhd,0,q')) & \text{if } t' = \left(q,\lhd,-1,\overrightarrow{q'}\right) \\ 1 & \text{if } t' \in \delta_{\text{ZAG}} \end{cases}$$

Hence, $\mathcal{K}' = (\mathcal{A}', \phi')$ is equivalent to $\mathcal{K}$. □

2.3.4 Two-way transducers

We now introduce the model of transducers. In the same spirit as in Definition 7 and Definition 11, we define it as the most general version, *i.e.*, the version known as "two-way nondeterministic transducer" (see [35]). Moreover, our transducers may produce any word of a recognizable language at each step.

Informally, a transducer is a finite automaton provided with a second tape, on which a write-only head, called *output head*, may append some words at each step. This second head is 1-way, that is, a written symbol is written forever. However, the underlying finite automaton may reject the computation (either by entering a non-accepting halting configuration, or by never halt), rejecting at the same time the current output produced.

A pair of words (u, v) is accepted if the automaton may perform a successful computation u in which v is written on the output tape in the meaning time. Hence, transducers accept relations in $\Sigma^* \times \Delta^*$, where Σ and Δ are respectively *the input* and *the output alphabet*. Without loss of generality, we may suppose Σ and Δ disjoint (as in Chapter 5) or equal (as in Chapter 6), depending on what is studied. A relation accepted by a transducer is called *a transduction*.

Now, we formally introduce the model of *transducer* as an extension of FA, but we immediately observe that it is a particular $\mathbb{K}$-FA, in which $\mathbb{K}$ is the semiring of rational languages on Δ.

Definition 12. *A* transducer *is a pair $\mathcal{T} = (\mathcal{A}, \phi)$ where $\mathcal{A}$ is a* FA *with transition set δ and where ϕ is a* production function *mapping δ into the set of nonempty rational subsets of Δ.*

Let u be a word in Σ^* and let $\mathbf{r}$ be a run on u of trace $t_1 \cdots t_\ell$. The word $v \in \Delta^*$ is an output word produced *by $\mathbf{r}$ if it belongs to the subset $\phi(t_1) \cdots \phi(t_\ell)$. We will also use the notation $\Phi_{\mathcal{T}}(\mathbf{r}) = \phi(t_1) \cdots \phi(t_\ell)$ or simply $\Phi(\mathbf{r})$ when the transducer $\mathcal{T}$ is understood.*

A pair $(u, v) \in \Sigma^* \times \Delta^*$ *is* accepted by the transducer *if* v *is produced by an accepting run on* u. *The* relation accepted *by* $\mathcal{T}$ *is the set of all such* (u, v), *denoted* $\|\mathcal{T}\|$.

In other words, $\mathcal{T}$ is a $\mathbb{K}$-FA $\mathcal{K}$, with $\mathbb{K} = \langle \mathrm{RAT}(\Delta^*), \cup, \cdot, \{\epsilon\}, \varnothing \rangle$. The relation $\|\mathcal{T}\|$ corresponds to the series $\|\mathcal{K}\|$.

Restricted versions and equivalences

We adapt the restrictions of $\mathbb{K}$-FA to transducers. Observe the similarity with the restrictions introduced in Section 2.3.3. However, when considering the restrictions on nondeterminism, some important differences appears.

A transducer $(\mathcal{A}, \phi)$ is respectively *1-way*, *restless*, *sweeping*, *loop-free*, *halting*, *k-reversal bounded* for some $k > 0$, if $\mathcal{A}$ is. It is *rotating* if it is a rotating $\mathbb{K}$-FA with $\mathbb{K} = \mathrm{RAT}(\Delta^*)$. This holds if $\mathcal{A}$ is rotating and each backward transition, *i.e.*, the transitions in the set Q_{-1} as defined in Section 2.3.2, is mapped to $\{\epsilon\}$ by ϕ.

Given a relation, we speak of *1-way*, *rotating*, *sweeping* or *2-way transductions*, depending on by which variant it is accepted.

It is well-known that the family of relations accepted by 1-way transducers is the family of rational relations, e.g., [8, Theorem III. 6.1] [21, 66].

Theorem 6. *One-way transducers accept exactly the family of rational relations.*

Example 7. *We fix* $\Sigma = \{a, b\}$ *and* $\Delta = \{c, d\}$ *and we consider the following relation in* $\Sigma^* \times \Delta^*$, *whose domain is* $\Sigma^* b$ *and which for all maximal factors* w *of the form* a^*b, *substitutes* c^n *to* w *if* $w = a^{2n}b$ *and* d^n *if* $w = a^{2n+1}b$:

$$R := \Big(\ (a^2, c)^*(b, \epsilon) \ \cup \ (a^2, d)^*(ab, \epsilon) \ \Big)^*$$

A 1-way transducer accepting R *is depicted in Figure 2.5.*

The notion of deterministic transducer is different from the corresponding notion for $\mathbb{K}$-FA, that is, a deterministic $\mathbb{K}$-FA with $\mathbb{K} = \mathrm{RAT}(\Delta^*)$ is not necessary a deterministic transducer. Indeed, since at each step a transducer may produce an entire word from a recognizable language associated to the transition, some kind of nondeterminism is hidden in the production function. In order to detail the complexity of the production function, we put some restrictions on the productions.

A transition t is *single-valued* if $|\phi(t)| = 1$. The transducer $\mathcal{T}$ is *single-valued* if all its transitions are single-valued. If moreover $\mathcal{A}$ is deterministic, then $(\mathcal{A}, \phi)$ is *deterministic*. It is *outer-nondeterministic* if $\mathcal{A}$ is outer-nondeterministic and each transition $t = (q, c, d, q')$ with $c \notin \{ \rhd, \lhd \}$ is single-valued. If for each transition t of $\mathcal{A}$, and each production $w \in \phi(t)$ we have $|w| \leq 1$, the transducer is said *elementary*.

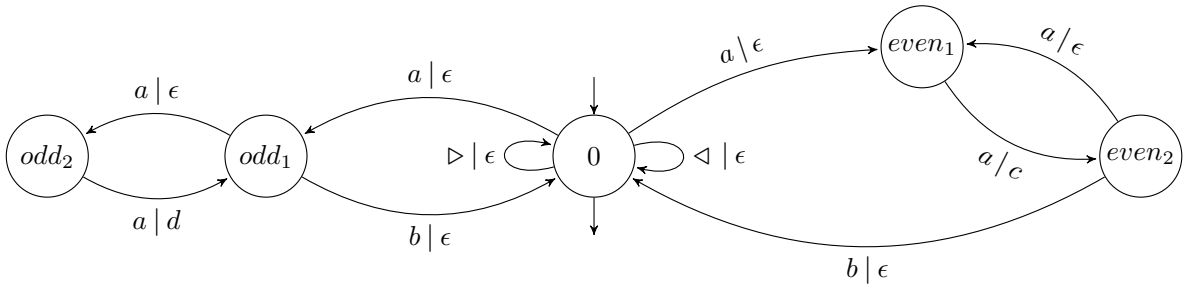


FIGURE 2.5: A 1-way transducer accepting the relation R defined in Example 7. An edge from q to q' is labeled by $x \mid y$, if the transducer has a transition $(q, x, +1, q')$ with associated output $\{y\}$. In particular, the transducer is *single-valued* and *elementary*.

Making single-valued and elementary: Each transducer admits an equivalent transducer which is single-valued and elementary, *i.e.*, with production function ϕ mapping δ into $\Delta \cup \{\epsilon\}$[1].

Proposition 6. *Every transducer $\mathcal{T}$ is effectively equivalent to an elementary single-valued transducer $\mathcal{T}'$. Moreover, if $\mathcal{T}$ is deterministic, outer-nondeterministic, rotating, sweeping, k-reversal bounded, or 1-way, so is $\mathcal{T}'$.*

However, restlessness, loop-freeness and halting property are not preserved.

Proof. Let $\mathcal{T} = (\mathcal{A}, \phi)$ be a transducer. We first construct a transducer $\mathcal{T}'$ which is elementary, but not necessarily single-valued. An easy second step will make it single-valued.

For each transition t of $\mathcal{A}$, $\phi(t)$ is rational. Hence, by Theorem 3, it is accepted by a CFA $\mathcal{A}_t$ of state set Q_t, that we suppose initial by Proposition 2. Without loss of generality, we suppose that all different Q_ts are pairwise disjoint and disjoint from Q.

We build a simulating transducer $\mathcal{T}' = (\mathcal{A}', \phi')$ which works as follows: it simulates one step of $\mathcal{T}$ of trace $t = (q, a, d, q')$ by entering three consecutive phases. All along the simulation, the transducer makes no move except at the last step.

INITIALIZING it enters the initial state s of $\mathcal{A}_t$ without moving, producing an empty output, *i.e.*, it performs a transition $(q, a, 0, s)$ with associated output ϵ.

[1] This is actually an abusive formulation, since ϕ is a mapping from δ into 2^{Δ^*}. The correct formulation should be, the production of a single-valued elementary transducer is a mapping from δ into the singletons of $2^{\Delta \cup \{\epsilon\}}$.

SIMULATING it simulates the automaton $\mathcal{A}_t$ by nondeterministically choosing a symbol in Δ, writing it on the output tape and entering the next state in Q_t. That is, $\mathcal{T}'$ performs a transition of the form $(s, a, 0, s')$ with associated output b such that (s, b, s') is a transition of $\mathcal{A}_t$. This mode is repeated until some nondeterministically chosen point, where the current state is accepting.

FINALIZING finally, from this accepting state s', it enters the state q' and moves its head according to d producing an empty output, *i.e.*, it performs a transition (s', a, d, q') with associated output ϵ.

Observe that nondeterministic choices may occur in the simulating procedure at three points only: (1) in the choice of t, (2) in the SIMULATING procedure when choosing the simulated transition in δ_t (including the choice of the symbol b) and (3) when halting the SIMULATING mode in some accepting state q_t^+. Obviously if $\mathcal{A}$ is deterministic, the first choice is deterministic. Moreover, if ϕ is single-valued, then we may suppose that, for each state q_t of $\mathcal{A}_t$ there exists at most one symbol $b \in \Delta$ and one state q_t' such that (q_t, b, q_t') is a transition of $\mathcal{A}_t$. Thus, the choices (2) becomes deterministic. Finally, by halting the SIMULATING at the first occurrence of an accepting state, the choice (3) is made deterministic. Hence, if $\mathcal{T}$ is deterministic, so is the resulting $\mathcal{T}'$ implementing the procedure.

We built an elementary transducer $\mathcal{T}'$ equivalent to $\mathcal{T}$. It is now easy to make it single-valued. Indeed, it suffices to store in the finite control, the last output set, which can be done because there are at most $|\Delta| + 1$ possible outputs. □

Making restless: Observe that thanks to Proposition 5, every transducer can be made restless. Moreover, it is straightforward to see that the constructions preserve the restrictions on the production function, namely single-valuation and elementarity. However, as observed before, the construction does not preserve some important properties of the device, such as 1-way, rotating or sweeping. We give here two other constructions, in the case of transducers. These constructions are generalizable for the case of $\mathbb{K}$-FA with a rationally additive semiring $\mathbb{K}$ (see Section 2.2.2), but we are essentially interested in the case where $\mathbb{K} = \text{RAT}(\Delta^*)$, *i.e.*, the case of transducers. The two new constructions, which are essentially based on a same idea, do not preserve the two restrictions on the production function.

Proposition 7. *Given an n-state transducer, we can effectively build*

1. *a $3n$-state restless transducer, preserving determinism, outer-nondeterminism, unambiguity, loop-freeness, haltingness, single-valuation and elementarity;*

2. *a $(n+2)$-state restless transducer, preserving determinism, outer-nondeterminism, unambiguity, loop-freeness, haltingness, sweepingness and rotatingness.*

3. *a $(n+1)$-state restless transducer, preserving unambiguity, loop-freeness, haltingness, one-wayness, sweepingness and rotatingness;*

Proof. The first statement is a direct consequence of Proposition 5, observing that single-valuation and elementarity are preserved.

The two other statements are obtained by two constructions, which share the same basic idea but differ on a final part. Thus we start by detail the shared part of the construction, which is usual. We fix a transducer $\mathcal{T} = (\mathcal{A}, \phi)$ of transition set δ.

We say that a run is 1-*move-at-end* if it is of the form $(q_0, p)(q_1, p)\cdots(q_\ell, p)(q_{\ell+1}, p+d)$ for some $\ell \geq 0$ and some direction $d \in \{-1, +1\}$. The main idea of the proof is to simulate every such 1-move-at-end run by a single restless step $(q_0, p)(q_{\ell+1}, p+d)$. The question of the output generated by this single step is treated in as follows.

We consider 0-*move runs*, *i.e.* runs whose configurations have all the same position component. Because only one input tape cell is visited, the run does not really depend on the input nor the position but on the current scanned symbol only. The set of 0-move runs which start in state q scanning symbol a and end in state q' (still scanning a) is denoted $\mathbf{R_{q,q',a}}$. The language $L_{q,q',a}$ is defined as the union of the production associated to the runs in $\mathbf{R_{q,q',a}}$, *i.e.*,

$$L_{q,q',a} = \bigcup_{\mathbf{r} \in \mathbf{R_{q,q',a}}} \Phi(\mathbf{r})$$

We prove that $L_{q,q',a}$ is rational. Observe that we can easily obtain a 1-way transducer $\mathcal{T}_{q,q',a}$ from $\mathcal{T}$ such that $\mathcal{T}_{q,q',a}$ accepts the relation $\{(a, v) \mid v \in L_{q,q',a}\}$. By Theorem 6, the relation is rational and hence, the language $L_{q,q',a}$ is rational by projection.

We now define a new set of transitions δ_1 and its associated production function ϕ_1 (see Figure 2.6a). A transition $t = (q, a, d, q'')$ belongs to δ_1 if and only if $d \neq 0$ and there exists a state q' such that $\mathbf{R_{q,q',a}}$ is not empty and (q', a, d, q'') belongs to δ. The rational image of t by ϕ_1 is given by:

$$\phi_1(t) = \bigcup_{q'} (L_{q,q',a} \cdot \phi(q', a, d, q''))$$

Observe that a restless transition t of $\mathcal{A}$ belongs to δ_1 and $\phi(t)$ is a subset of $\phi_1(t)$.

Every accepting run $\mathbf{r}$ of $\mathcal{T}$ can be factorized in

$$\mathbf{r} = \mathbf{r_0} @ \mathbf{r_1} @ \ldots @ \mathbf{r_{k-1}} @ \mathbf{r_k} \tag{2.2}$$

where $\mathbf{r_i}$ is a 1-move-at-end run for each $0 \leq i < k$ and $\mathbf{r_k}$ is a 0-move run occurring at the rightmost position (scanning the right endmarker). Using δ_1 and ϕ_1, we may simulate $\mathbf{r_0}@, \ldots, @\mathbf{r_{k-1}}$ by a restless run $\mathbf{r}'$.

The last factor $\mathbf{r_k}$ is problematic since it may be a non trivial 0-move run, which is not followed by a restless step. We propose two ways to deal with this factor, leading to the two last statements of the Proposition.

The simplest idea is to avoid such non-trivial factors, that is, to enforce an accepting state to be entered with non-stationary transitions only. This can be easily done by a slight pre-modification of $(\mathcal{A}, \phi)$, that preserves rotatingness but not one-wayness. Suppose $\mathcal{A} = (Q, \Sigma, \rhd, \lhd, I, F, \delta)$. Using two fresh states $\leftarrow$ and $\rightarrow$, we define a transition set $\overleftrightarrow{\delta}$ that enforces $\mathcal{A}$ to perform a right-to-left followed by a left-to-right traversal of the input before accepting. Formally:

$$\overleftrightarrow{\delta} := \left\{ \begin{array}{ll} (\leftarrow, a, -1, \leftarrow) & \mid a \in \Sigma \\ (\rightarrow, a, +1, \rightarrow) & \mid a \in \Sigma \\ (\leftarrow, \rhd, +1, \rightarrow) & \\ (q, \lhd, -1, \leftarrow) & \mid q \in F \end{array} \right\}$$

Then we define the associated production function $\overleftrightarrow{\phi}$ as the function which maps every transition in $\overleftrightarrow{\delta}$ to $\{\epsilon\}$. Finally, we claim that $\mathcal{T}$ is equivalent to $\mathcal{T}' = (\mathcal{A}', \phi')$, where:

$$\mathcal{A}' = \left((Q \cup \{\leftarrow, \rightarrow\}), \Sigma, \rhd, \lhd, I, \{\rightarrow\}, \delta \cup \overleftrightarrow{\delta}\right) \qquad \text{and} \qquad \phi' = \phi \cup \overleftrightarrow{\phi}$$

By construction, no run of $\mathcal{A}'$ admits a decomposition as Equation (2.2) with a non-trivial $\mathbf{r}_k$. Thus, the transducer $\mathcal{T}'' = (\mathcal{A}'', \phi' \cup \phi_1)$, where

$$\mathcal{A}'' = \left((Q \cup \{\leftarrow, \rightarrow\}), \Sigma, \rhd, \lhd, I, \{\rightarrow\}, \left(\delta_1 \cup \overleftrightarrow{\delta}\right)\right) \qquad \text{and} \qquad \phi'' = \phi_1 \cup \overleftrightarrow{\phi}$$

is a restless transducer equivalent to $\mathcal{T}$. Observe that if $\mathcal{T}$ is rotating or sweeping, so is $\mathcal{T}''$. This concludes the proof of the statement (2).

The second way of dealing with non trivial factor $\mathbf{r}_k$ in Equation (2.2), is to nondeterministically guess the right boundary one position to the left of the endmarker and to produce the outputs in $L_{q,q',\lhd}$ for any accepting state q'.

To this end, we fix a fresh halting state $q_\lhd$ and we define a transition set $\delta_\lhd$ and its associated production function $\phi_\lhd$ (see Figure 2.6b). A transition $t = (q, a, d, q_\lhd)$ belongs to $\delta_\lhd$ if and only if $d = +1$ and there exist two states q' and q'' such that $\mathbf{R}_{\mathbf{q'},\mathbf{q''},\lhd}$ is not

empty, q'' is an accepting state of $\mathcal{T}$ and (q, a, d, q') belongs to δ_1. The image of t by $\phi_\triangleleft$ is given by:

$$\phi_\triangleleft(q, a, d, q_\triangleleft) = \bigcup_{\substack{q', q'' \\ q'' \text{ is accepting}}} (\phi_1(q, a, d, q') \cdot L_{q', q'', \triangleleft})$$

which is rational.

Our restless transducer simulating $\mathcal{T} = (\mathcal{A}, \phi)$ is defined as $\mathcal{T}' = (\mathcal{A}', \phi_1 \cup \phi_\triangleleft)$ where $\mathcal{A}' = (Q \cup \{q_\triangleleft\}, \Sigma, \triangleright, \triangleleft, I, \{q_\triangleleft\}, \delta_1 \cup \delta_\triangleleft)$, where Q and I denote respectively the state set and the initial state set of $\mathcal{A}$. From any run of $\mathcal{T}'$ producing a word v, we can find a run of $\mathcal{T}$ producing v and reciprocally. Since the transitions of δ_1 and $\delta_\triangleleft$ have the same directions as the corresponding transitions (or 1-move-at-end runs) of $\mathcal{T}$, the construction preserves one-wayness and sweepingness. This concludes the proof of Statement (3). □

Functional transducers: A transducer $\mathcal{T}$ is *functional* (*resp. k-functional*) if $\|\mathcal{T}\|$ is a function (*resp.* a *k-function*), *i.e.*, if for each input u there exists at most 1 output (*resp.* k outputs) v such that (u, v) is accepted by $\mathcal{T}$. By characterizing the family of functions accepted by 2-way transducers, Engelfriet and Hoogeboom proved that they are exactly the relations accepted by deterministic (2-way) transducers [24].

Theorem 7 (Engelfriet and Hoogeboom – 2001)**.** *A relation R accepted by a (2-way) transducer is a function, if and only if it is accepted by some deterministic (2-way) transducer.*

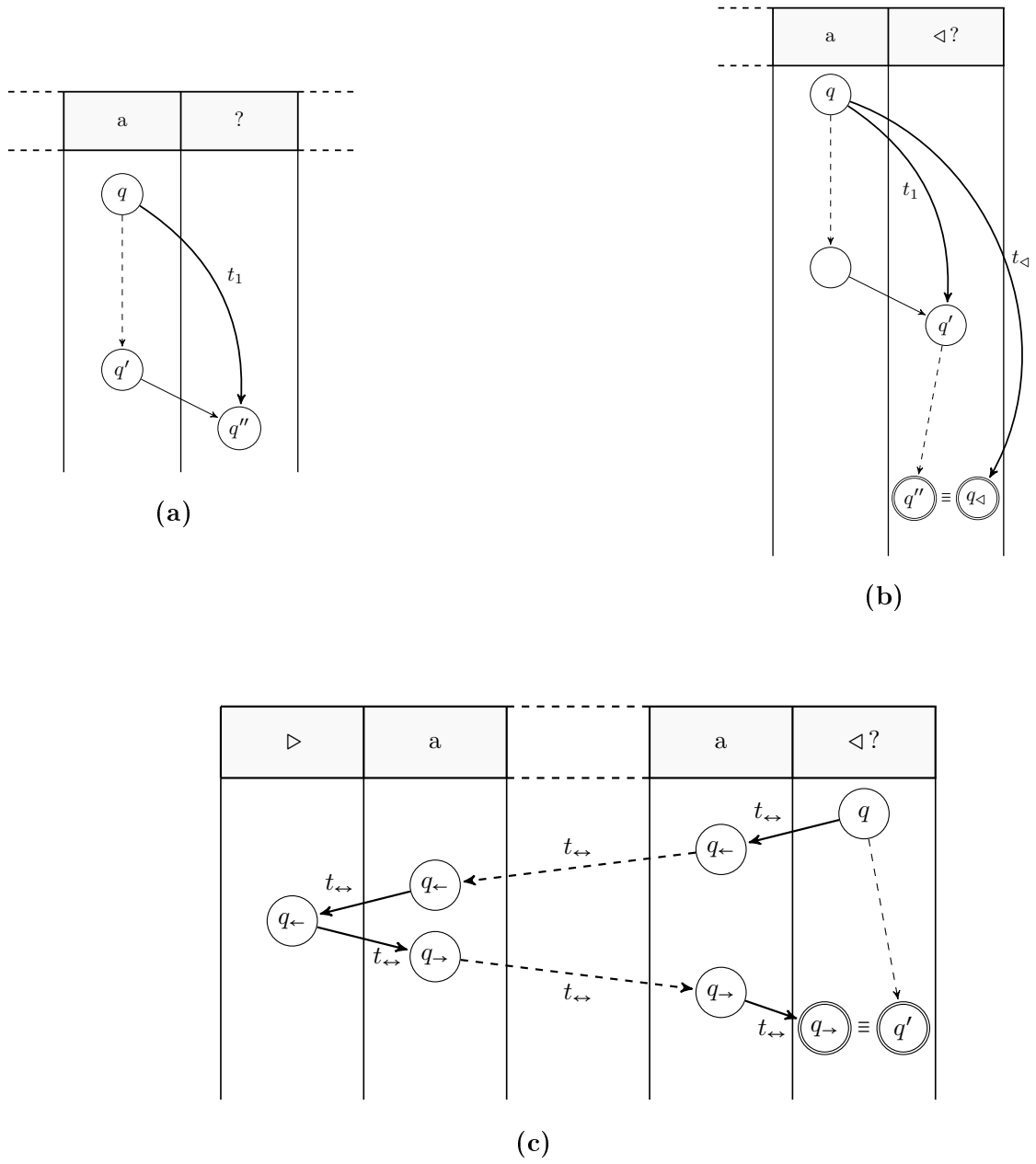


FIGURE 2.6: Transition in δ_1 (a), $\delta_\triangleleft$ (b) and $\delta_\leftrightarrow$ (c), for restless simulation of 2-way transducers.

Chapter 3

Outer-nondeterministic Finite Automata

3.1 Introduction

In this chapter, we investigate the question of cost of the simulation of 2NFAs by 2DFAs. This known open problem were raised in 1978 by Sakoda and Sipser who conjectured an exponential cost [68]. Contrary to various contributions to the problem (*e.g.*, [74, 39, 45]), we do not attack the problem by restricting the simulating machine but rather the simulated one. Indeed, we consider the weaker problem of simulating 2ONFAs by 2DFAs. We also consider the alternating version of 2ONFAs, namely the 2AFAs, in which both universal and existential choices may occur at the endmarker only (see the details in Section 3.7). Our work extends some of the results obtained in the case of unary alphabets [32, 33, 34] to the case of 2ONFAs and 2OAFAs on arbitrary alphabets. In particular, we prove the following:

- Each n-state 2ONFA can be simulated by a *halting* 2-way *self-verifying* finite automaton (2SVFA) [19] with $O(n^8)$ many states. (Self-verifying automata are nondeterministic automata with a restricted kind of nondeterminism — symmetric with respect to accepted languages and their complements. A more detailed description is given in Section 3.2.3.) This fact has two important implications:
  - The complementation of 2ONFAs can be done by using a polynomial number of states. Note the contrast with the above mentioned case of sweeping 2NFAs, studied in [42].
  - Each 2ONFA can be simulated by a *halting* 2ONFA using a polynomial number of states.

- Each n-state 2ONFA can be simulated by a 2DFA with $O(n^{\log_2(n)+7})$ states.

- If L = NL, then each n-state 2ONFA can be simulated by a 2DFA with a number of states polynomial in n. Hence, a superpolynomial lower bound for the simulation of 2ONFAs by 2DFAs would imply L ≠ NL. (Unlike in [7], there are no restrictions on the length of potential witness inputs.)

- Each n-state 2ONFA can be simulated by an *unambiguous* 2ONFA with a polynomial number of states.

- If L = P, then each n-state 2OAFA can be simulated by a 2DFA with a number of states polynomial in n, with the same consequences as presented for L $\stackrel{?}{=}$ NL.

- Similarly, if NL = P, we get the corresponding polynomial conversion from 2OAFAs to 2NFAs.

These results are obtained by generalizing the constructions given originally for the unary case. However, here we do not have a normal form that simplifies automata by restricting input head reversals to the endmarkers. Our generalization relies on a different tool, presented in the first part of the chapter. Basically, we extend some techniques developed originally for deterministic devices [73, 33] to machines with nondeterminism at the endmarkers. This permits us to check the existence of certain computation paths, including infinite loops, by the use of a linear number of states.

The chapter is organized as follows. In Section 3.2, we recall basic definitions and preliminary results required later. In Section 3.3, we develop a fundamental tool that will be used several times, namely, a deterministic procedure that allows us to check the existence of computation paths between two given states in the given 2ONFA, starting and ending at the left endmarker and not visiting the left endmarker in the meantime. This procedure is also useful to make all computations halting. The next sections are devoted to our main results. In Section 3.4, we present the polynomial simulation of 2ONFAs by 2SVFAs and its consequences. In Section 3.5, we show the subexponential simulation of 2ONFAs by 2DFAs. Then, in Section 3.6, under the assumption L = NL, such simulation is made polynomial in the number of states. Moreover, we show how to simulate a 2ONFA by an unambiguous 2NFA using a polynomial number of states. Finally, in Section 3.7, we present the corresponding results for the alternating case. Some concluding remarks are briefly discussed in Section 3.8.

3.2 Preliminaries

In this section we establish some preliminary results concerning the computational model, namely the 2-way outer-nondeterministic finite automata, investigated in the chapter. We provide a simplification of 2ONFA with respect to the definition. This lead us to introduce the *computational segments*, which are particular compositions of hit. After that, we introduce a new kind of 2-way nondeterministic automaton, namely the *self-verifying* automaton, which has the ability to recognize either a language or its complement.

3.2.1 Normal form for 2ONFAs

Recall that a 2ONFA is a 2NFA $\mathcal{A} = (Q, \Sigma, \triangleright, \triangleleft, \delta, q_-, F)$ that can take nondeterministic decisions only when the input head is scanning one of the two endmarkers, *i.e.*, for each $q \in Q$ and $a \notin \{\triangleright, \triangleleft\}$, we have at most one d and one p such that (q, a, d, p) belongs to δ (see definition in Section 2.3.2). Actually, with a linear increase in the number of states, we can restrict the use of the nondeterminism to the left endmarker only. We also obtain some other restrictions, such as restlessness, which will be useful to simplify several other proofs. All these improvements are given the next lemma.

Lemma 3. *For each* 2ONFA $\mathcal{A} = (Q, \Sigma, \triangleright, \triangleleft, \delta, I, F)$ *with* n *states, there exists an equivalent* 2ONFA $\mathcal{A}'$ *with no more than* $3n + 3$ *states that satisfies the following properties:*

1. *$\mathcal{A}'$ is initial and restless (let us call q_- its unique initial state);*
2. *nondeterministic choices are taken only when the input head is scanning the left endmarker;*
3. *there is a special state q_+ which is the unique accepting state assumed for the first time when the head is positioned on the left endmarker. From then on, the automaton does not change state and performs right move until it reaches the right endmarker.*

Proof. Points 1 and 3. Making $\mathcal{A}$ initial is simply done by Proposition 3, using a new initial state q_-. By adapting Proposition 7 Statement 2 to FAs, we can transform $\mathcal{A}$ into a restless and still outer-nondeterministic FA. Moreover, as described in the proof, the resulting automaton uses two special states $\leftarrow$ and $\rightarrow$, where $\rightarrow$ is as in the third statement of the Lemma. Hence, we set $q_+ = \rightarrow$.

Point 2. Assuming that $\mathcal{A}$ satisfies Points 1 and 3, we build an automaton $\mathcal{A}'$ making nondeterministic choices only on the left endmarker. The only point to settle is to eliminate the nondeterministic choice at the right endmarker. Each time $\mathcal{A}$ is about to

make a nondeterministic choice at the right endmarker, in a state q, $\mathcal{A}'$ traverses the input from right to left, using $\overleftarrow{q}$, a copy of q, to reach the *left* endmarker, where it simulates a single transition from q at the right endmarker, *i.e.*, it chooses a state p such that $(q, \triangleleft, -1, p) \in \delta$. To do that, $\mathcal{A}'$ enters a state $\overrightarrow{p}$, a copy of p, and, remaining in this state, traverses the input back to the right endmarker. Then $\mathcal{A}'$ moves its head one position to the left entering the original state p. From this configuration, $\mathcal{A}'$ resumes the ordinary simulation of $\mathcal{A}$.

Consequently, it can be easily seen that the resulting automaton $\mathcal{A}'$ is equivalent to the original 2ONFA $\mathcal{A}$ and satisfies the properties listed in the statement of the lemma. The set of states consists of three copies of the set Q, plus the states q_-, $\leftarrow$ and $q_+ = \rightarrow$. Thus the total number of states of $\mathcal{A}'$ does not exceed $3n + 3$. □

3.2.2 Computational segments

In Section 2.3.2, we introduced the hits, as being particular runs that start and end in some border configurations, without visiting any other border configuration in the meaning time. These particular runs are natural when dealing with unary inputs alphabets. However, in the case of outer-nondeterministic FAs and due to Lemma 3, it is more convenient to work with another kind of run, that starts from and ends to the left endmarker, without visiting it in the meaning time. A *(computational) segment* is a run $\mathbf{r} = (q_0, p_0) \cdots (q_\ell, p_\ell)$ such that $p_0 = p_\ell = 0$ and for each $0 < i < \ell$, we have $p_i > 0$. When q_0 and q_ℓ are given, $\mathbf{r}$ is a (computational) segment *connecting* q_0 and q_ℓ. The definition is analog to that of hit, however, there are not directly connected: a segment may contain an unbounded number of hits, because the head might rebound several times on the right endmarker before returning back to the left endmarker. A sequence of segments connecting some states $q_0, q_1, \ldots, q_t$ (that is, the ith segment is from p_{i-1} to p_i, for $i = 1, \ldots, t$, with $t \geq 0$) is called a *sequence of t segments* from q_0 to q_t.

Since we are working with FA, *i.e.*, acceptors, we are mainly interested in loop-free runs, as shown with Lemma 4. It is easy to see that every loop-free segment is a finite composition of loop-free hits. The following is routine:

Lemma 4. *Let $\mathcal{A}$ be a* 2NFA *with n states. Then, for each $w \in \|\mathcal{A}\|$, $\mathcal{A}$ has at least one accepting computation path that does not visit the left (right) endmarker more than n times.*

3.2.3 Self-verifying automata

Here we introduce a variant of nondeterministic finite automata, which is able to recognize both its accepted language and its complement.

A 2-way *self-verifying* automaton (2SVFA) $\mathcal{A}$ is a 2NFA which, besides the set of accepting states $F \subseteq Q$, is equipped also with a disjoint set of *rejecting* states $F^{\mathrm{r}} \subseteq Q$. Any rejecting state is supposed to be halting and an initial finite run which halts in a rejecting state is said *rejecting* (see Section 2.3.2). For each input $w \in \|\mathcal{A}\|$, there exists at least one successful run, and no rejecting run. Conversely, for $w \notin \|\mathcal{A}\|$, there exists at least one rejecting run and no successful run. Note that some runs of a 2SVFA may end with a "don't know" answer, by halting in a state not belonging to $F \cup F^{\mathrm{r}}$, or executing an infinite loop.

3.3 The subroutine REACH

This section is devoted to develop a tool which will be fundamental in the proof of our results. Let $\mathcal{A}$ be a 2ONFA with n states, in normal form given by Lemma 3. Then, for a fixed input string w, we shall test reachability by computational segments, among configurations scanning the left endmarker. (Clearly, w is accepted if and only if there exists a sequence of computational segments connecting the unique initial configuration $(q_-, 0)$ with the unique final configuration $(q_+, 0)$.) Note that the size of a reachability graph, in which edges represent computational segments, does not grow in the length of the input, but rather in n, the number of states in $\mathcal{A}$.

3.3.1 Description of the subroutine REACH

We are now ready to design a subroutine REACH$(q_{\mathrm{S}}, q_{\mathrm{T}})$ that receives two states $q_{\mathrm{S}}, q_{\mathrm{T}}$ of $\mathcal{A}$ as parameters and, by examining the string w on the input tape, decides whether $\mathcal{A}$ has a computational segment from q_{S} to q_{T} on w. Depending on the outcome, the procedure returns the corresponding Boolean value TRUE or FALSE. We will show that this subroutine can be implemented by the use of a 2DFA with $O(n)$ "internal" states and with a "read-only" access to q_{S} and q_{T}.

A first idea would be to try to compute REACH$(q_{\mathrm{S}}, q_{\mathrm{T}})$ by initializing the automaton $\mathcal{A}$ in the state q_{S} with the input head at the left endmarker and by stopping its computation as soon as it reaches the left endmarker again, then testing whether the state so reached is q_{T}. However, this approach runs into two problems: first, the original automaton $\mathcal{A}$ could get into an infinite loop, never coming back to the left endmarker; second, the first move from the state q_{S} on the left endmarker could be nondeterministic.

To solve these problems, we adapt the construction given in [33] (where it was used for different purposes, to make a 2DFA halting, with $4n$ states) which, in turn, was a refinement of the corresponding Sipser's construction [73] for space bounded Turing machines. Here we give a brief outline of this construction. For each $w \in \Sigma^*$, a deterministic

machine accepts w if and only if there is a "backward" path, tracing back the history of the computation, from the unique accepting configuration c_+ to the unique initial configuration c_-.

Consider the graph in which nodes represent configurations and edges single computation steps. If the machine under consideration is deterministic, the component of the graph containing the accepting configuration c_+ is a tree rooted at this configuration, with backward paths branching to all possible predecessors of c_+. In addition, since the accepting configuration is also halting, no backward path starting from c_+ can cycle (hence, it is of finite length). Thus, it is sufficient to perform a depth-first search of this tree in order to detect whether the initial configuration c_- belongs to the predecessors of c_+. If this is the case, the simulator accepts. On the other hand, if the entire tree is examined without reaching c_-, there is no path from c_- to c_+ and so w is not in the language. Hence, the simulator rejects.

We adapt this procedure by choosing $c_- = (q_S, 0)$ and $c_+ = (q_T, 0)$, where q_S and q_T are the two parameters, since we are interested in detecting the existence of *just one computational segment*, from q_S to q_T. This is possible, because our machine $\mathcal{A}$ is in normal form given by Lemma 3, and hence it does not make nondeterministic decisions when the input head is not scanning the left endmarker, *i.e.*, $|\delta \cap (\{q\} \times \{a\} \times \{-1, +1\} \times Q)| \leq 1$ for each $q \in Q$ and $a \neq \triangleright$.

The only problem is that both q_S and q_T are located at the left endmarker, where nondeterministic branching is allowed. This is resolved as follows. For the purposes of walking along the tree of the backward depth-first search, we ignore the transitions on the left endmarker, *as if such transitions did not exist*, *i.e.*, as if $\delta \cap (Q \times \{\triangleright\} \times \{-1, +1\} \times Q) = \varnothing$. Hence, the backward search starts from $c_+ = (q_T, 0)$, which now behaves as a root of a tree with backward paths leaving the left endmarker and branching to possible predecessors of q_T. However, the transitions at the left endmarker are missing. Therefore, a backward path can never reach the left endmarker again. Instead, such path ends up one backward step earlier, in a configuration $c = (q, 1)$, with the input head placed one position to the right of the left endmarker, even though, in reality, using a transition $(p, \triangleright, +1, q) \in \delta$, the original machine could get to c from a configuration $c' = (p, 0)$. Hence, there exists a computational segment connecting $c_- = (q_S, 0)$ with $c_+ = (q_T, 0)$ if and only if, for some configuration $c = (q, 1)$ placed one position to the right of the left endmarker, there exists a backward path from c_+ to c visiting the left endmarker only in c_+ and, moreover, $(q_S, \triangleright, +1, q) \in \delta$.

Thus, we stop the depth-first search and return TRUE at the moment when we reach a configuration c with the above properties. If the entire tree has been visited without reaching any such c, we stop and return FALSE.

In the course of the depth-first search, our procedure needs to detect when the input

head of the original 2ONFA $\mathcal{A}$ is placed exactly one position to the right of the left endmarker. By a closer look to the simulation in [33], one can observe that, for each configuration $c = (q, i)$, all the "left" predecessors $c' = (p, i{-}1)$ are examined in $q_{\nwarrow}$, a copy of the state q, with the input head shifted one position to the left of the actual input head position, *i.e.*, in the position $i{-}1$, while the "right" predecessors $c' = (p, i{+}1)$ are examined in $q_{\nearrow}$, another copy of q, with the input head shifted one position to the right.[1] (For completeness, the simulation uses two more copies for each state q, namely, $q_{\downarrow_1}$ and $q_{\downarrow_2}$, with the input head placed exactly at the actual input head position i, at the moment when, respectively, all the *left*, or *both* all the left and all the right predecessors have already been visited.) Hence, when the procedure reaches a state $q_{\nwarrow}$ with the input head scanning the left endmarker and, moreover, $(q_{\mathrm{S}}, \rhd, +1, q) \in \delta$, we can stop the computation and return TRUE.

There are only three points which depend on the states q_{S} and q_{T}: the choice of the root for the depth-first search, *i.e.*, of the configuration $c_+ = (q_{\mathrm{T}}, 0)$, the detection of exit nodes in this tree, depending on transitions in $\delta \cap (\{q_{\mathrm{S}}\} \times \{\rhd\} \times \{+1\} \times Q)$, and finally handling some trivial cases, namely, if $q_{\mathrm{S}} = q_{\mathrm{T}}$ or if $q_{\mathrm{T}} = q_+$.

More details are presented in Section 3.3.2. Hence, with this strategy, we obtain the following result, required later:

Lemma 5. *Let $\mathcal{A}$ be a* 2ONFA *with a state set Q, in the form of Lemma 3, and let* REACH$(q_{\mathrm{S}}, q_{\mathrm{T}})$ *be a procedure returning* TRUE *or* FALSE *depending on whether $\mathcal{A}$ has a computational segment connecting two states $q_{\mathrm{S}}, q_{\mathrm{T}}$ on a given input w. Then the truth of* REACH$(q_{\mathrm{S}}, q_{\mathrm{T}})$ *can be computed by a* 2DFA *$\mathcal{A}''$ using two read-only variables containing $q_{\mathrm{S}}, q_{\mathrm{T}} \in Q$ and one working variable to store one of $O(|Q|)$ internal states.*

More precisely, there exists a 2DFA *$\mathcal{A}''$ such that, starting at the left endmarker of the input w in a state $[q_{\mathrm{START}}, q_{\mathrm{S}}, q_{\mathrm{T}}]$, $\mathcal{A}''$ will stop in $[q_{\mathrm{TRUE}}, q_{\mathrm{S}}, q_{\mathrm{T}}]$ or $[q_{\mathrm{FALSE}}, q_{\mathrm{S}}, q_{\mathrm{T}}]$, depending on the truth of the statement* REACH$(q_{\mathrm{S}}, q_{\mathrm{T}})$. *This holds for each $q_{\mathrm{S}}, q_{\mathrm{T}} \in Q$ and each input w. The state set of $\mathcal{A}''$ is $Q'' = Q' \times Q \times Q$, with $|Q'| \leq 4|Q|{-}1$. Thus, $\mathcal{A}''$ uses $4{\cdot}|Q|{-}1$ internal states; the second and third components in Q'', containing $q_{\mathrm{S}}, q_{\mathrm{T}} \in Q$, are never modified and hence used in a read-only way.*

3.3.2 Implementation details for the subroutine REACH

Here we present a more detailed description of the subroutine REACH, introduced in Section 3.3.1. A reader not interested in such low-level implementation details may skip the rest of this section.

[1] There are no predecessors of type $c' = (p, i)$, since the machine $\mathcal{A}$ does not use stationary moves, except for transitions used to halt and accept.

Recall that, for the given 2ONFA $\mathcal{A} = (Q, \Sigma, \delta, q_-, F)$ with n states, in the form of Lemma 3, we need a 2DFA $\mathcal{A}''$ capable of testing, for any given pair of states $q_S, q_T \in Q$ (passed to $\mathcal{A}''$ as two read-only parameters, in the finite state control) and any given w (presented on the 2-way read-only input tape), whether $\mathcal{A}$ has a computational segment from q_S to q_T on the input w. This implements the Boolean procedure $\textsc{Reach}(q_S, q_T)$, specified in Lemma 5.

Consider first the standard case, in which $q_S \neq q_T$ and $q_T \neq q_+$. Basically, the standard case uses the depth-first search described already in Section 3.3, obtained by modifying the construction given in [33]. For these reasons, the presentation will be given along the same lines as in [33], keeping, as much as possible, the same notation.

As already told in Section 3.3, we ignore the transitions on the left endmarker in the course of the depth-first search, which completely excludes the final state q_+ from further considerations, as an isolated singleton. (Recall that, by Lemma 3, q_+ is a halting state that can be reached only from the left endmarker. We shall return to such special cases later.)

Second, before proceeding further, we need to fix a linear order on Q, the state set of the original automaton $\mathcal{A}$. As usual, the symbols "<" and ">" will denote the ordering relation.

Now, in the course of the depth-first search, our implementation examines each visited configuration $c = (q, i)$ in two modes:

Mode 1: Examination of the "left" predecessors of $c = (q, i)$. A left predecessor of c is a configuration in the form $(p, i-1)$, such that $(p, w_{i-1}, +1, q) \in \delta$. (Here w_i denotes the ith symbol on the input tape.)

Mode 2: Examination of the "right" predecessors, namely, configurations in the form $(p, i+1)$, such that $(p, w_{i+1}, -1, q) \in \delta$.

For each $q \in Q$ and both modes, we introduce a starting and a finishing state. Taking into account that the computation depends also on the parameters q_S and q_T, we thus define the state set Q'' for our 2DFA $\mathcal{A}''$ as follows:

$$Q' = \{q_{\nwarrow}, q_{\downarrow 1}, q_{\nearrow}, q_{\downarrow 2}, q : q \in Q \setminus \{q_+\}\} \cup \tag{3.1}$$

$$\{q_{\text{START}}, q_{\text{TRUE}}, q_{\text{FALSE}}\}, \tag{3.2}$$

$$Q'' = Q' \times Q \times Q\,. \tag{3.3}$$

However, the transition function $\delta'' : Q'' \times (\Sigma \cup \{\rhd, \lhd\}) \to Q'' \times \{-1, 0, +1\}$ never modifies the parameters q_S, q_T. That is, for each $[r, q_S, q_T] \in Q' \times Q \times Q$ and $a \in \Sigma \cup \{\rhd, \lhd\}$, we always have $\delta''([r, q_S, q_T], a) = ([r', q_S, q_T], d)$, for some $r' \in Q'$ and $d \in \{-1, 0, +1\}$. Thus,

$q_{\mathrm{S}}, q_{\mathrm{T}}$ are always used in a "read only" way. For this reason, $q_{\mathrm{S}}, q_{\mathrm{T}}$ do not contribute to the total number of states used by the subroutine REACH. (They only contribute to the number of states used by any other routine *calling* REACH.) Moreover, the only parts of the computation that actually depend on q_{S} and q_{T} are transitions starting in q_{START} or stopping in q_{TRUE}. Therefore, for simplicity, we describe the behavior of $\mathcal{A}''$ by filling the entries in the transition table for a function $\delta' : Q' \times (\Sigma \cup \{ \rhd, \lhd \}) \to Q' \times \{-1, 0, +1\}$, but indicating explicitly where the next move does really depend on q_{S} or q_{T} (very few cases). A final expansion of the transition table for δ' to a table for δ'' will be then quite straightforward.

Now, the states $r \in Q'$ are interpreted as follows:

$q_{\nwarrow}$ Starting state for the Mode 1: examination of left predecessors for the configuration (q, i). Left predecessors will be examined one after another, according to the linear order induced by the relation "<". To inspect the content of the input square $i-1$, the simulator (if it is in the state $q_{\nwarrow}$) has its input head shifted one position to the left of the actual position of the original machine $\mathcal{A}$.

$q_{\downarrow_1}$ Finishing state for the Mode 1. All the left predecessors of (q, i) have been examined, but we still have to examine the right predecessors of (q, i). In the state $q_{\downarrow_1}$, the input head of the simulator is in the actual position i.

$q_{\nearrow}$ Starting state for the Mode 2: examination of right predecessors for (q, i), when the left predecessors have been finished. The right predecessors will also be examined in the linear order induced by "<". In the state $q_{\nearrow}$, the simulator has its input head shifted one position to the right of the actual position, to inspect the content in the input square $i+1$.

$q_{\downarrow_2}$ Finishing state for the Mode 2. Both the left and the right predecessors of (q, i) have been examined. In the state $q_{\downarrow_2}$, the input head of the simulator is in the actual position, *i.e.*, the position i.

q Actually not utilized by the procedure REACH, reserved for future use, by variable r in Algorithm 5 of Section 3.4, and in the subroutine nREACH of Section 3.4.1. (Here we list the complete set of values in Q', as used also by other subroutines that will be considered later.)

$q_{\mathrm{START}}, q_{\mathrm{TRUE}}, q_{\mathrm{FALSE}}$ The respective starting and halting states for the depth-first search, viewed also as entry and exit points to/from the Boolean procedure REACH.

Let us now describe the transition function δ' implementing this strategy. For each (type of) nonhalting state $r \in Q'$ and each symbol $a \in \Sigma \cup \{ \rhd, \lhd \}$, we display a procedure

that assigns a value of $\delta'(r,a) \in Q' \times \{-1,0,+1\}$ to the transition table and present an explanation for this procedure. Note that $\mathcal{A}''$ will use stationary moves. The reader should also keep in mind that the procedures (called macros here) displayed below are *not* executed by the machine $\mathcal{A}''$ but, rather, they are used to fill in the entries in the transition table for $\mathcal{A}''$.

Let us begin with the transition table of $\mathcal{A}''$ for a state $q_{\nwarrow}$, which is presented as Macro 1. (This case is most different from the corresponding one in [33].) Recall that

```
1  if  a = ◁ then
2  |   δ'(q↖, a) := undefined
3  else if  a = ▷ then
4  |   if (q_S, ▷, +1, q) ∈ δ then (q↖, a, 0, q_TRUE) ∈ δ'        // — return TRUE
5  |   else δ'(q↖, a) := (q↓1, +1)
6  else if there is no p ∈ Q : δ(p, a) = {(q, +1)} then
7  |   δ'(q↖, a) := (q↓1, +1)
8  else
9  |   p̃ := min {p ∈ Q : δ(p, a) = {(q, +1)} }
10 |   δ'(q↖, a) := (p̃↖, −1)
```

Macro 1: *Transition of Type* $\delta'(q_{\nwarrow}, a)$

$\mathcal{A}''$ gets to the state $q_{\nwarrow}$ when, for some i, it starts the examination of the left predecessors of the configuration $c = (q,i)$. By definition of $q_{\nwarrow}$, $\mathcal{A}''$ has its input head already at the position $i-1$. The procedure considers four cases:

- $a = \triangleleft$ (line 1): actually this case is unreachable, it is given just for completeness, to fill in all entries in the transition table for δ'.
- $a = \triangleright$ (from line 3): in this case, the input head of the original 2ONFA $\mathcal{A}$ is placed exactly one position to the right of the left endmarker. *Depending on the first parameter* q_S, there are two possibilities. If $(q_S, \triangleright, +1, q) \in \delta$, the backward simulation has been successfully completed, the remaining part of the backward tree can be ignored. Hence, the machine stops immediately in the state q_{TRUE} (line 4). Otherwise, the configuration $c = (q,1)$ has no left predecessors, since there are no transitions in the depth-first search tree at the left endmarker. Hence, the machine terminates Mode 1 and moves its head to the real input position (line 5).
- In the middle of the input, and there are no left predecessors of (q,i) (line 6): the machine ends Mode 1 and moves its head to the real input position. (Recall also that all transitions in the middle of the input are deterministic.)

- In the middle of the input, and the configuration (q, i) has at least one left predecessor (lines 8–9). We select the first left predecessor in the linear order and start to examine this configuration with the same method. To this aim, we switch the state to $\tilde{p}_{\nwarrow}$, and move the head one position to the left of $i-1$.

Consider now transitions for a state $q_{\downarrow_1}$, presented as Macro 2. (This case is exactly the same as in [33].) In this state, the examination of the left predecessors of (q, i) has been

```
11 if a ≠ ◁ then δ'(q↓1, a) := (q↗, +1)
12 else δ'(q↓1, a) := (q↓2, 0)
```

Macro 2: *Transition of Type* $\delta'(q_{\downarrow_1}, a)$

completed. Hence, the search continues with the examination of the right predecessors in Mode 2 (line 11), by switching to the state $q_{\nearrow}$ and moving the head to the position $i+1$. However, if the input head is on the right endmarker, *i.e.*, $a = \triangleleft$, then the configuration (q, i) does not have any right predecessors (line 12). Hence, by switching to $q_{\downarrow_2}$, we finish Mode 2 immediately, as if all predecessors to the right had been searched.

Next, consider transitions for $q_{\nearrow}$, displayed as Macro 3. (Even this case is the same as in [33].) In the state $q_{\nearrow}$, $\mathcal{A}''$ starts to examine the right predecessors of (q, i). $\mathcal{A}''$ has

```
13 if a = ▷ then δ'(q↗, a) := undefined
14 else if there is no p ∈ Q : δ(p, a) = {(q, −1)} then δ'(q↗, a) := (q↓2, −1)
15 else
16 |  p̃ := min {p ∈ Q : δ(p, a) = {(q, −1)} }
17 |  δ'(q↗, a) := (p̃↖, −1)
```

Macro 3: *Transition of Type* $\delta'(q_{\nearrow}, a)$

its head already at the position $i+1$. In a right predecessor of a configuration, the head cannot scan the left endmarker, and hence all transitions from the right predecessors are deterministic. There are three main cases:

- $a = \triangleright$ (line 13): unreachable case, given for completeness.
- There are no right predecessors (line 14): we finish Mode 2 immediately, which completes the search for (q, i).
- Otherwise (lines 16–17) we select $(\tilde{p}, i+1)$, the first right predecessor of (q, i), and start to examine it with the same method. (Among others, the left predecessors of $(\tilde{p}, i+1)$ are going to be examined.) To this aim, we switch to $\tilde{p}_{\nwarrow}$, and move the head one position to the left of $i+1$.

```
18 if a = ▷ then  δ′(q↓2, a) := (q_FALSE, 0)                    // — return FALSE
19 else if δ(q, a) = ∅ then δ′(q↓2, a) := undefined
20 else
21     (q̃, d) := unique element of δ(q, a)
22     if there is no p ∈ Q : p > q and δ(p, a) = {(q̃, d)}  then
23         if d = +1 then δ′(q↓2, a) := (q̃↓1, +1)
24         else δ′(q↓2, a) := (q̃↓2, −1)
25     else
26         p̃ := min {p ∈ Q : p > q and δ(p, a) = {(q̃, d)}}
27         δ′(q↓2, a) := (p̃↖, −1)
```

Macro 4 *Transition of Type* $\delta'(q_{\downarrow 2}, a)$

Finally, consider transitions for a state $q_{\downarrow 2}$, displayed as Macro 4. This state concludes the examination of the configuration (q, i), and all configurations in the subtree rooted at (q, i). The machine $\mathcal{A}''$ has its head at the position i, the actual position of the head of the simulated machine $\mathcal{A}$. There are three main cases:

- The head is scanning the left endmarker (line 18): since our backward depth-first search ignores the transitions from the left endmarker (see line 5 in Macro 1), the only configuration of the form $(q, 0)$ that can be reached is $c_+ = (q_{\mathrm{T}}, 0)$, the root for the depth-first search. This means that we have examined the entire tree rooted at $(q_{\mathrm{T}}, 0)$ and hence a computational segment from $(q_{\mathrm{S}}, 0)$ to $(q_{\mathrm{T}}, 0)$ on the given input w does not exist. Therefore, $\mathcal{A}''$ stops the computation in the state q_{FALSE}.

- The configuration does not have any successor (line 19): such configuration is never reached in the backward search. This case is included only for completeness.

- Otherwise (from line 20): since the input head is away from the left endmarker, the configuration (q, i) has as a unique successor, which can be obtained by the use of the transition function of $\mathcal{A}$ (line 21). Namely, if $\delta(q, a) = \{(\tilde{q}, d)\}$, the unique successor is $(\tilde{q}, i+d)$. Depending on the value of d, we have to consider either left predecessors ($d = +1$) or right predecessors ($d = -1$) of $(\tilde{q}, i+d)$. (We call them here "d-predecessors", for short.) First, we try to find a state p greater than q such that (p, i) is a d-predecessor. If such state does not exist, then (q, i) is the last d-predecessor of $(\tilde{q}, i+d)$. Hence, depending on d, we complete Mode 1 or Mode 2 for $(\tilde{q}, i+d)$ (lines 23–24). Otherwise (lines 26–27), we start to examine, in Mode 1 and with the same method, the next d-predecessor of $(\tilde{q}, i+d)$.

To complete the description, we have to specify how to start the depth-first search. *This initialization depends on the state* q_{T}. Recall that we want to find a segment from

$(q_S, 0)$ to $(q_T, 0)$ on the input w. Taking into account that the configuration $(q_T, 0)$ does not have left predecessors, we can start the depth-first search in the state $q_{T\downarrow_1}$ with the head at the left endmarker. If there exists the segment we are looking for, then the computation stops, as explained above, in the state q_{TRUE} (Macro 1, line 4). Otherwise, it stops in q_{FALSE} (Macro 4, line 18) after traversing the entire subtree rooted at $(q_T, 0)$.

Finally, let us examine some trivial cases, *dependent on the states* q_S, q_T:

- If $q_S = q_T$, then $\mathcal{A}''$ stops immediately in q_{TRUE}, not starting the depth-first search at all.
- If $(q_S, \triangleright, 0, q_T) \in \delta$, then $\mathcal{A}''$ stops immediately in q_{TRUE}. (For machines in the normal form of Lemma 3, this condition ensures, automatically, that $q_T = q_+$.)
- If $q_S \neq q_T$, $(q_S, \triangleright, 0, q_T) \notin \delta$, but $q_T = q_+$, then $\mathcal{A}''$ stops immediately in q_{FALSE}.

We are now ready to expand the transition table for δ', manipulating with the states in Q', to a table for δ'', manipulating with $Q'' = Q' \times Q \times Q$. Basically, if $\delta'(r, a) = (r', d)$, we define $\delta''([r, q_S, q_T], a) = ([r', q_S, q_T], d)$, for each $q_S, q_T \in Q$, with the following exceptions:

Handling trivial cases: $\delta''([q_{\text{START}}, q_S, q_T], \triangleright) = ([q_{\text{TRUE}}, q_S, q_T], 0)$, for each q_S, q_T satisfying $q_S = q_T$ or $(q_S, \triangleright, 0, q_T) \in \delta$, but

$\delta''([q_{\text{START}}, q_S, q_T], \triangleright) = ([q_{\text{FALSE}}, q_S, q_T], 0)$, for each q_S, q_T satisfying $q_S \neq q_T$, $\delta(q_S, \triangleright) \not\ni (q_T, 0)$, and $q_T = q_+$.

Initialization: $\delta''([q_{\text{START}}, q_S, q_T], \triangleright) = ([q_{T\downarrow_1}, q_S, q_T], 0)$, for each q_S, q_T satisfying $q_S \neq q_T$, $(q_S, \triangleright, 0, q_T) \notin \delta$, and $q_T \neq q_+$.

Transitions returning TRUE: $\delta''([q_\nwarrow, q_S, q_T], \triangleright) = ([q_{\text{TRUE}}, q_S, q_T], 0)$, for each q, q_S, q_T satisfying $(q_S, \triangleright, +1, q) \in \delta$, but

$\delta''([q_\nwarrow, q_S, q_T], \triangleright) = ([q_{\downarrow_1}, q_S, q_T], +1)$, for each q, q_S, q_T not satisfying this condition.

(This situation has already appeared in Macro 1, lines 4–5.)

On the other hand, transitions returning FALSE (Macro 4, line 18) are handled in the standard way: $\delta''([q_{\downarrow_2}, q_S, q_T], \triangleright) = ([q_{\text{FALSE}}, q_S, q_T], 0)$, for each $q_S, q_T \in Q$. The same holds for stopping states: both $\delta''([q_{\text{TRUE}}, q_S, q_T], \triangleright)$ and $\delta''([q_{\text{FALSE}}, q_S, q_T], \triangleright)$ are left undefined here, for each q_S, q_T. (Actually, the control returns to the main program.)

Summing up, we have shown that, for a given n-state 2ONFA $\mathcal{A}$ in the form of Lemma 3, there exists a 2DFA $\mathcal{A}''$ such that, starting in the state $[q_{\text{START}}, q_S, q_T]$ at the left endmarker of the input w, $\mathcal{A}''$ will stop in $[q_{\text{TRUE}}, q_S, q_T]$ or $[q_{\text{FALSE}}, q_S, q_T]$, depending on whether

the automaton $\mathcal{A}$ has a computational segment from q_S to q_T on w. This holds for each $q_S, q_T \in Q$ and each input w. The state set of $\mathcal{A}''$ is $Q'' = Q' \times Q \times Q$, in which $|Q'| \leq 4n-1$.[2] Thus, $\mathcal{A}''$ uses $4n-1$ internal states; the second and third components, containing $q_S, q_T \in Q$, are used in a read-only way. Clearly, this gives an implementation for the Boolean procedure REACH(q_S, q_T) of Lemma 5.

3.4 Simulation by halting self-verifying automata

In this section we prove that each n-state 2ONFA $\mathcal{A}$ accepting a language L can be replaced by an equivalent halting 2SVFA with a polynomial number of states and making nondeterministic choices only when the input head is scanning the left endmarker. As a consequence, we can derive halting 2ONFAs with polynomial many states that accept L and the complement of L.

Our starting point is the construction given in [33] for the unary case, based on the well known *inductive counting*. The simulation inductively counts, for $t = 0, 1, 2, \ldots$, how many states are reachable at the endmarkers from the initial state in t hits along the input tape. As a side effect, this generates also a list of such states. When all such states have been listed, we can decide whether the original machine accepts the input. However, there are deep differences from our case. In particular, [33] uses a normal form for unary 2NFAs in which, besides all nondeterministic decisions, also all reversals in the input head movement must take place at the endmarkers. Consequently, a computation cannot make a *U-turn* that starts and ends at the same endmarker without visiting the opposite one in the meantime, or execute an *infinite loop* in the middle of the input.

In our case, we do not have a normal form of this kind. Hence, besides hits across the entire input, a computation can present U-turns, or run into an infinite loop not visiting the endmarkers any more. To overcome the first problem, our procedure considers *computational segments* instead of hits: for $t = 0, 1, 2, \ldots$, we count how many states are reachable from the initial state by computations consisting of t segments, *i.e.*, visiting the left endmarker $t+1$ times. At the same time, we generate the complete list of these states. As a direct consequence of Corollary 1, each accepted input admits an accepting run visiting the left endmarker at most n times, and hence it is enough to consider runs consisting of at most $n-1$ segments. This also avoids infinite loops involving the endmarkers. We shall later discuss how to deal with infinite loops taking place in the middle of the input.

[2] Here we count only the states used to implement the procedure REACH. In (3.1), among the states of Q', we introduced an unmarked copy of each $q \in Q \setminus \{q_+\}$, not used by the procedure REACH, but reserved for future use. These copies will be counted later, when they will be used.

In some situations, our simulation can end up in a "don't know" state, denoted here by $q_?$. This new state is halting: when it is reached, even in the code of a subroutine, the *entire* simulation will be aborted.

To simulate individual computational segments, we make use of the subroutine REACH(q_S, q_T), with parameters $q_S, q_T \in Q$, discussed in Lemma 5. Recall that REACH uses also a global working variable $r \in Q'$, where it temporarily stores one of $|Q'| \leq 4 \cdot |Q| - 1$ internal states in the course of the computation. (More details can be found in Section 3.3.)

We shall also need another Boolean subroutine tREACH(t, q_S), with $t \in \{0, \ldots, |Q|-2\}$ and the same parameter $q_S \in Q$ as in REACH, which verifies whether there exists a sequence of t segments starting from the initial state q_- and leading to q_S, on the input under consideration. In the positive case, the Boolean subroutine tREACH returns TRUE in a standard way, *i.e.*, nothing "special" happens. (Both t and q_S are used in a read-only way. However, as a side effect, tREACH can temporarily store some internal state in the variable r, used as a temporary storage also by REACH, so the original value of r is "lost".) Conversely, if such a sequence does not exist, the subroutine tREACH aborts the computation in the state $q_?$. As will be seen later, this subroutine is nondeterministic. Hence, it may abort the computation in $q_?$ also due to a wrong sequence of nondeterministic guesses. However, tREACH *can never get into an infinite cycle*. We are now ready for a more detailed implementation, displayed as Algorithm 5.

```
28 m̃ := 1
29 for t := 0 to |Q|−2 do
30 |  m := m̃;  m̃ := 0
31 |  foreach q_T ∈ Q do
32 |  |  for i := 1 to m do
33 |  |  |  r := a nondeterministically chosen state, from Q ∖ {q_+}
34 |  |  |  if i > 1 and r ≤ q_S then halt in q_?
35 |  |  |  q_S := r
36 |  |  |  if tREACH(t, q_S) and REACH(q_S, q_T) then
   |  |  |  |                    // — side effects: both tREACH and REACH modify r
   |  |  |  |                                  // — tREACH may abort by halting in q_?
37 |  |  |  |  if q_T = q_+ then halt in q_ACCEPT
38 |  |  |  |  m̃ := m̃+1
39 |  |  |  |  break i                        // — start the next iteration for q_T
40 halt in q_REJECT
```

Algorithm 5: *Simulation of* 2ONFA*s by Halting* 2SVFA*s*

The algorithm proceeds by counting, for $t = 0, \ldots, |Q|-2$, the exact number of all states

reachable by $\mathcal{A}$ at the left endmarker by all computation paths starting from the initial configuration and consisting of exactly $t+1$ segments (loop from line 29). During this process, the algorithm also generates all such states, and hence it can correctly decide whether to accept or reject the given input.

At the beginning of the t-th iteration of this loop (line 29), a variable $\tilde{m}$ contains the number of states reachable at the left endmarker by all computation paths with exactly t segments. (In line 28, we prepare $\tilde{m} = 1$ for $t = 0$, the only state reachable by zero segments is the initial state q_{-}.) In line 30, we save the "old" value of $\tilde{m}$ in the variable m, and clear $\tilde{m}$ for counting the number of states reachable upon completing one more segment, *i.e.*, with exactly $t+1$ segments.

The value of $\tilde{m}$ is computed in the loop from line 31: for each state $q_{\mathrm{T}} \in Q$, we test whether or not it is reachable by a path with exactly $t+1$ segments. If it is, we shall increment the value of $\tilde{m}$ (line 38).

To decide whether q_{T} can be reached by exactly $t+1$ segments, we generate in a nondeterministic way, one after another, using the variable q_{S}, all m states that are reachable at the left endmarker by all computation paths with exactly t segments and we verify if q_{T} can be reached from any of these states by a single segment. This is carried out by the innermost loop (from line 32).

Within this loop, running for $i = 1, \ldots, m$, we verify that *(i)* the sequence of m nondeterministically chosen states is generated in increasing order, and that *(ii)* each of them is indeed reachable with exactly t segments. (This ensures that we are using only the "proper" states in q_{S}, and that we obtain the complete sequence: such m states are all different and hence none of the m "proper" states is skipped.)

The first condition is verified by temporarily storing each generated state in a separate variable r (line 33) and by comparing this state with the old value of q_{S}, saved in the previous iteration of the loop. If the nondeterministically generated sequence does not respect the fixed order on Q, the computation is aborted in $q_?$ (line 34). If the state in r passes this test, it is saved in q_{S} (line 35). Note also that the temporary variable r will not be required until the subsequent iteration of the loop, and hence it is now free for any other purposes.

The second condition is verified by calling the procedure tREACH(t, q_{S}). As a side effect of this call (line 36), the computation is aborted in $q_?$ if at least one of the m states generated in q_{S} is *not* reachable from the initial state by a computation path with exactly t segments. Hence, the only computation which "survives" testing for these two conditions is the one generating, in the sorted order, all m states reachable in exactly t segments.[3]

[3]The subroutine tREACH modifies the variable r, using it as a temporary storage. Besides that, tREACH uses another temporary variable $\tilde{t} \in \{0, \ldots, |Q|-2\}$. Further details about this subroutine, which

Now, for each q_S among these m states, the algorithm tests whether, by using one more computational segment starting from q_S, it can reach the state q_T under examination. This is carried out by calling REACH(q_S, q_T) on line 36.[4]

If the result of this test is positive, the variable $\tilde{m}$ is incremented (line 38). At this point (line 39), we must abort the innermost loop, iterated for $i = 1, \ldots, m$, in order to avoid counting the state q_T twice, in case it is reachable from the initial state by several different paths. Thus, we continue with the next iteration, if any, of the loop examining all states $q_T \in Q$.

However, before doing so (line 37), we halt the entire computation in the accepting state q_{ACCEPT}, if we discover that the final state was reached at the left endmarker.

On the other hand, if the iteration of the outermost loop has been completed for each $t = 0, \ldots, |Q|-2$, never reaching q_+ at the left endmarker, then the input is not accepted by the original automaton. (Otherwise, the search would have stopped already, in line 37.) Therefore, in line 40, we stop in the rejecting state q_{REJECT}.

It is not hard to see that: *(i)* if the input is accepted by $\mathcal{A}$, at least one computation path halts in the state q_{ACCEPT} and no path halts in q_{REJECT}, *(ii)* if the input is rejected, at least one path halts in q_{REJECT} and no path halts in q_{ACCEPT}. *(iii)* Due to wrong sequences of nondeterministic guesses, some computation paths halt in $q_?$, but no path can get into an infinite loop.

It only remains to present a possible implementation of the subroutine tREACH. First, we can modify the backward search described in Section 3.3, to obtain an auxiliary subroutine nREACH which works with the same global variable r as does the subroutine REACH, but does not use any parameters. Starting with the head at the left endmarker and an initial value $q'_{\downarrow 1}$ stored in the variable r, for some $q' \in Q \setminus \{q_+\}$, the subroutine stops at the left endmarker after leaving, in the same variable r, a nondeterministically chosen state $q \in Q \setminus \{q_+\}$ such that $\mathcal{A}$ has a computational segment from q to q' on the given input w. If the subroutine is not able to find such a state q, it aborts the entire computation by halting in the state $q_?$. (However, it can never get into an infinite cycle.) That is, nREACH simulates nondeterministically $\mathcal{A}$ one segment backward. In the course of the computation, the variable r contains one of $4n-4$ internal states, of type $q_\nwarrow, q_{\downarrow 1}, q_\nearrow, q_{\downarrow 2}$ (the same values were used also by the subroutine REACH) but, upon exit, one of $n-1$ states $q \in Q \setminus \{q_+\}$. (For further details, see Section 3.4.1.)

Now, the implementation of tREACH(t, q_S) is simple, as shown in Algorithm 6. Initially, it assigns $q_{S\downarrow 1}$ to the variable r and then it runs t iterations of nREACH, properly

is also nondeterministic, are given below.

[4] Also the subroutine REACH uses r as a temporary storage, keeping there one of the values from Q' in the course of the backward depth-first search starting from q_T. See Section 3.3 and (3.1) in Section 3.3.

re-initializing r in between each two iterations, by the execution of $r := r_{\downarrow 1}$. Hence, at the end, if r contains the initial state then the search was successful and the Boolean subroutine tREACH(t, q_S) returns TRUE. Otherwise it aborts the entire computation by halting in $q_?$. For $t = 0$, the subroutine nREACH is not called at all, and tREACH$(0, q_S)$ returns TRUE only if $q_S = q_-$.[5]

```
41 r := q_S;  t̃ := t
42 while t̃ > 0  do
43 |   r := r_{↓1};  t̃ := t̃−1
44 |   nREACH                          // — step variable r one segment backward
45 if r ≠ q_− then halt in q_?
46 return TRUE
```

Algorithm 6: *Boolean Function* tREACH(t, q_S)

Finally, consider the cost of our implementation, in the number of states. The number of possible values for each one of the 8 variables $m, \tilde{m}, t, \tilde{t}, i, q_S, q_T, r$ is bounded by $n+1$, except for r, storing one of $5n-2$ possible values. (Cf. definition of Q', presented by (3.1) in Section 3.3.) Hence, our simulation can be carried out by using $O(n^8)$ states.

Observe also that, in the main algorithm and in the subroutines, all nondeterministic choices are taken when the input head is scanning the left endmarker.

By summarizing, we have proved the following:

Theorem 8. *Each n-state* 2ONFA $\mathcal{A}$ *can be replaced by an equivalent halting* 2SVFA $\mathcal{A}'$ *with $O(n^8)$ states making nondeterministic choices only when the input head is scanning the left endmarker.*

Corollary 2. *For each n-state* 2ONFA $\mathcal{A}$*, there exist an equivalent halting* 2ONFA $\mathcal{A}'$ *with $O(n^8)$ states and a* 2ONFA $\mathcal{A}''$ *with $O(n^8)$ states accepting the complement of $\|\mathcal{A}\|$.*

3.4.1 Implementation details for the subroutine nREACH

This section is devoted to low-level details in the implementation of the subroutine nREACH, obtained by a slight modification of REACH, described in Section 3.3. The reader not interested in such details may safely skip this section.

[5] Instead of this approach, we could generate all states reachable in exactly t segments by a direct nondeterministic simulation of $\mathcal{A}$, as in [33, subroutine *simulation*]. This will also produce a correct inductive counting algorithm. However, since $\mathcal{A}$ can run into infinite loops not visiting the endmarkers, the resulting algorithm could also enter some loops. As shown here, our implementation produces *halting* automata with the same upper bound on the number of states as given — for the unary case only — in [33].

Recall that, starting with the head at the left endmarker and an initial value $q'_{\downarrow_1}$ stored in the variable r, for some $q' \in Q \setminus \{q_+\}$, the subroutine nREACH should stop at the left endmarker again, after leaving, in r, a nondeterministically chosen state $q \in Q \setminus \{q_+\}$ such that there exists a computational segment from q to q' on the given input string. If the subroutine does not find a segment connecting some q with the initially given q', due to the fact that such a state q does not exist or due to wrong nondeterministic guessing, the subroutine will halt in the state $q_?$, aborting the entire computation.

This task can be carried out by a modified version of REACH. In particular, nREACH uses the same backward search, described in Section 3.3, starting with r containing the initially given $q'_{\downarrow_1}$ at the left endmarker, corresponding to the root configuration $c_+ = (q', 0)$. When, in the course of the backward search, we reach some state $\tilde{q}_\nwarrow$ with the input head scanning the left endmarker again, corresponding to a configuration $c = (\tilde{q}, 1)$ with the input head placed exactly one position to the right of the left endmarker, the subroutine nREACH proceeds as follows (cf. Macro 1, lines 4 and 5):

- First, the old value $\tilde{q}_\nwarrow$ is replaced in the variable r with a nondeterministically chosen value from the set $\{q \in Q : \delta(q, \rhd) \ni (\tilde{q}, +1)\} \cup \{\tilde{q}_{\downarrow_1}\}$.

- Now, if the chosen value is a state q such that $(q, \rhd, +1, \tilde{q}) \in \delta$, we stop the backward search, leaving the head at the left endmarker. This means that, in the variable r, the subroutine returns a state q such that $(q, 0)$ is a left predecessor of $(\tilde{q}, 1)$ and so there is a computational segment from $(q, 0)$ to the starting root configuration $c_+ = (q', 0)$.

- If the chosen value is $\tilde{q}_{\downarrow_1}$, we move the head to the right, ignore the left predecessors of $(\tilde{q}, 1)$, and carry on the backward search. Hence, the machine tries to find another segment, corresponding to another path in the search tree rooted at c_+.

Thus, the actions implemented on lines 4 and 5 in Macro 1 now correspond to $\delta'(\tilde{q}_\nwarrow, a) := \{(q, 0) : (q, \rhd, +1, \tilde{q}) \in \delta\} \cup \{(\tilde{q}_{\downarrow_1}, +1)\}$.

The computation can also traverse the entire subtree rooted at c_+ without returning any state q. This can happen either because a segment ending in c_+ does not exist at all or, because of a wrong sequence of nondeterministic choices along the backward search, we have ignored all suitable candidates. In this case we have to halt in the state $q_?$ and abort the entire computation. (Cf. Macro 4, line 18.) Such implementation of nREACH uses the same state variable r as does REACH.

3.5 Subexponential deterministic simulation

In this section, we prove that each 2ONFA with n states can be simulated by an equivalent 2DFA with $O(n^{\log_2(n)+7})$ states, *i.e.*, with a subexponential, but still superpolynomial, number of states. In the authors' knowledge this is the first case of a model using nondeterminism, an unrestricted alphabet, and having a subexponential simulation by 2DFAs.

This result generalizes a result proved for the unary case in [32]. Actually, even the proof is very similar: the new "ingredient" in our version is the subroutine REACH presented in Section 3.3. So we give a very short presentation, addressing the reader to [32] for further details.

Let $\mathcal{A}$ be a 2ONFA with n states in the form of Lemma 3. The 2DFA simulating $\mathcal{A}$ implements a recursive function, called REACHABLE(q,p,t), with parameters $q,p \in Q$ and $t \in \{0,\ldots,|Q|-1\}$. For these parameters, the function returns a Boolean value TRUE or FALSE, depending on whether $\mathcal{A}$ has a sequence of at most t segments from the state q to the state p, on the input w under consideration. The function is based on the well known divide-and-conquer technique, presented here as Algorithm 7.

```
47 if t ≤ 1 then return REACH(q,p)
48 else
49     foreach r ∈ Q do
50         if REACHABLE(q,r,⌊t/2⌋) then
51             if REACHABLE(r,p,⌈t/2⌉) then return TRUE
52     return FALSE
```

Algorithm 7: *Recursive Boolean Function* REACHABLE(q,p,t)

Hence, according to Corollary 1, to decide whether w is accepted by the machine $\mathcal{A}$, we call REACHABLE$(q_-,q_+,|Q|-1)$. We point out that for the base of the recursion, $t \leq 1$, we have to verify the existence of a sequence of at most one segment from q to p, by the use of the subroutine REACH. (A sequence with zero segments is possible only if $q = p$.) Now, we can prove the following.

Theorem 9. *Each n-state* 2ONFA $\mathcal{A}$ *can be replaced by an equivalent* 2DFA $\mathcal{A}'$ *with* $O(n^{\log_2(n)+7})$ *states.*

Proof. The implementation of REACHABLE and its complexity analysis are very close to those given in [32] for the unary case. We just outline a rough estimation of the state upper bound.

First, we suppose that the given 2ONFA $\mathcal{A}$ is in the form given in Lemma 3. The implementation of the function REACHABLE can be done using a constant height stack, as in [32], with few differences.

- The height of the stack is $\lceil \log_2(n-1) \rceil \leq \log_2(n)+1$. For each level of the recursion, we remember one of $2n$ possible values, namely, the state r plus one bit, indicating which of the two recursive calls of REACHABLE was activated — line 50/51. (In [32], the stack height was $\lceil \log_2(n+1) \rceil$.)
- At the base level of the recursion, the subroutine REACH uses $4n-1 < 4n$ states. (The corresponding subroutine in [32] was implemented with n^2+3 states.)

This gives that the number of different stack configurations can be bounded by $4n \cdot (2n)^{\log_2(n)+1}$. If the automaton $\mathcal{A}$ is not in the form of Lemma 3, we need to convert it, using $3n$ states. Hence, by replacing n by $3n$ in the formula, we obtain the following rough upper bound:

$$\begin{aligned} 4 \cdot 3n \cdot (2 \cdot 3n)^{\log_2(3n)+1} &= 12n \cdot (6n)^{\log_2(6n)} = 12n \cdot 6^{\log_2(6n)} \cdot n^{\log_2(6n)} \\ &= 12 \cdot n \cdot (6n)^{\log_2 6} \cdot n^{\log_2(n)+\log_2 6} \\ &= 12 \cdot 6^{\log_2 6} \cdot n^{\log_2(n)+1+2 \cdot \log_2 6}. \end{aligned}$$

One can easily verify that $1 + 2 \cdot \log_2 6 < 7$. □

3.6 Conditional and unambiguous simulations

It is quite natural to doubt whether the upper bound for making 2ONFAs deterministic, presented in Theorem 9, is optimal. We remind the reader that, so far, the best known gap between a 2NFA and an equivalent 2DFA is only n versus $\Omega(n^2)$ states [16]. In this section we shall show that the optimality of the upper bound in Theorem 9, or any other superpolynomial state lower bound for converting 2ONFAs to 2DFAs, would imply the separation between deterministic and nondeterministic logarithmic space, hence solving a longstanding open problem in structural complexity.

This is a direct consequence of the following statement: if L = NL, then each n-state 2ONFA can be simulated by a 2DFA with a number of states polynomial in n. After a slight modification, without using such additional assumptions, we shall also prove that each 2ONFA can be made *unambiguous* with a polynomial increase in the number of the states. An extension of these results to alternating 2-way automata will be discussed in Section 3.7.

The key to these results is a reduction of a language accepted by any given 2ONFA to the *graph accessibility problem* (GAP), *i.e.*, to the problem of deciding whether a

given directed graph $G = (V, E)$ contains a path connecting two designated vertices. This problem is well known to be complete for NL, the class of languages accepted by nondeterministic $O(\log n)$ space bounded machines [69].

Let us present our reduction. As for the results in Sections 3.4 and 3.5, it is obtained by combining a technique developed for the unary case [34] with the use of the subroutine REACH presented in Section 3.3.

Consider now an arbitrary 2ONFA $\mathcal{A}$ with n states, in the "normal" form of Lemma 3. In this machine, let us fix the state set to $Q = \{q_1, \ldots, q_n\}$, with $q_- = q_1$ and $q_+ = q_n$. Now, with each input string $w \in \Sigma^*$, we can associate a directed graph $G(w) = (Q, E(w))$, where

$$E(w) \quad = \quad \{(q_i, q_j) \in Q \times Q : \textsc{Reach}(q_i, q_j) = \textsc{true}\} .$$

That is, $E(w)$ is the set of state pairs (q_i, q_j) such that $\mathcal{A}$ has a segment from q_i to q_j on input w. These edges can be presented on an input tape of a Turing machine in the form of a binary *adjacency matrix*, written row by row, in which the bit at the position $(i-1)\cdot n + j$ is equal to 0 or 1 depending on whether $(q_i, q_j) \in E(w)$. Clearly, the length of this representation is $n \times n = n^2$ bits.

It should also be clear that w is accepted by $\mathcal{A}$ if and only if the graph $G(w)$ contains a path from vertex $q_- = q_1$, the initial state, to vertex $q_+ = q_n$, the accepting state. Hence, the mapping $G : w \to G(w)$ defines a reduction from the language accepted by $\mathcal{A}$ to GAP. As mentioned already, GAP is a complete language for NL under logarithmic space reductions [69]. Hence, GAP $\in$ L if and only if L = NL. This permits us to prove the following:[6]

Theorem 10. *If* L = NL*, then each* 2ONFA $\mathcal{A}$ *with* n *states can be replaced by an equivalent* 2DFA $\mathcal{A}'$ *with a number of states polynomial in* n*.*

Proof. Let D_{GAP} be a deterministic Turing machine which solves GAP in logarithmic space. Under the hypothesis that L = NL, such a machine must exist.

Now, by composing the above reduction $G : w \to G(w)$ with the machine D_{GAP}, (see Figure 3.1), we can build a 2DFA $\mathcal{A}'$ deciding membership in $\|\mathcal{A}\|$ as follows.

For the given input w, the machine $\mathcal{A}'$ simulates D_{GAP}, pretending that the input is $E(w)$, written on the tape as the corresponding adjacency matrix. Since the length of this representation is n^2 bits, D_{GAP} uses $O(\log n)$ space on its worktape. Recall that

[6] For the restricted case of unary input alphabet, a result similar to Theorem 10 was shown in [34, Lem. 4.1]. The following stronger result (without the unary restriction) is presented in [46] in a different context: if L/poly $\supseteq$ NL then each 2ONFA $\mathcal{A}$ with n states can be replaced by an equivalent 2DFA $\mathcal{A}'$ with a number of states polynomial in n. L/poly denotes the class of languages accepted by deterministic Turing machines in logarithmic space, with the additional help of an advice of polynomial length.

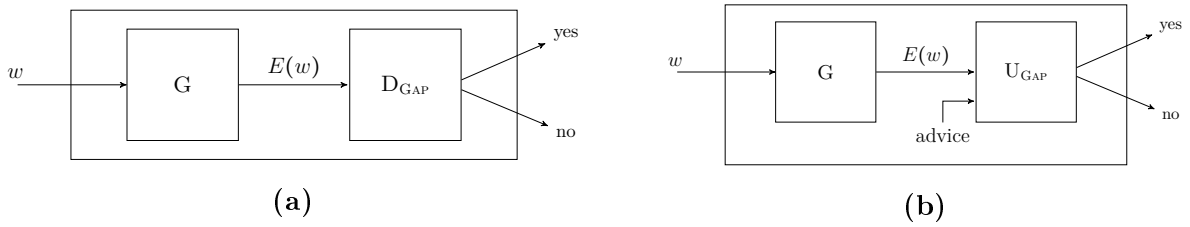


FIGURE 3.1: The machines $\mathcal{A}'$ (a) of Theorem 10 and $\mathcal{A}''$ (b) of Theorem 11.

the automaton $\mathcal{A}$ is fixed, and hence n does not depend on $|w|$, the length of the real input. Therefore, a worktape of size $O(\log n)$ can be represented in a finite state control, with a number of states polynomial in n. The same holds for the finite state control of D_{GAP}, as well as for the position of its virtual input head, with values ranging between 0 and n^2+1. Depending on whether D_{GAP} accepts or rejects $E(w)$, the machine $\mathcal{A}'$ accepts or rejects w, respectively.

The only problem is that the adjacency matrix for $E(w)$ cannot be stored in the finite control of $\mathcal{A}'$, because this would require at least $2^{n \cdot n}$ states. Hence, each time D_{GAP} needs to access one symbol from its input, such a symbol is computed "on the fly". More precisely, if the bit at a position $(i{-}1){\cdot}n+j$ is required, for some $i, j \in \{1, \ldots, n\}$, the simulation of D_{GAP} is temporarily interrupted and $\mathcal{A}'$ calls the subroutine REACH(q_i, q_j), presented in Section 3.3, to test whether the original machine $\mathcal{A}$ has a segment from q_i to q_j on the original input w. This subroutine uses $4n{-}1$ internal states. After obtaining this bit of information, $\mathcal{A}'$ can resume the simulation of D_{GAP}. Each time the virtual input head of D_{GAP} reaches the position 0 or n^2+1, $\mathcal{A}'$ imitates the presence of the left or right endmarker, respectively, without calling REACH. □

While the deterministic simulation in Theorem 10 stays polynomial under the assumption that L = NL, the next simulation by unambiguous machines does not require any extra assumption:

Theorem 11. *Each* 2ONFA $\mathcal{A}$ *with* n *states can be replaced by an equivalent* unambiguous 2ONFA $\mathcal{A}''$ *with a number of states polynomial in* n.

Proof. The simulation of $\mathcal{A}$ presented here[7] is similar to that in Theorem 10 but, instead of the (unproven) assumption L = NL, we shall utilize the following (consequence[8] of a)

[7] In [34, Thm. 5.2], an unambiguous simulation was presented for the restricted case of unary input alphabet.

[8] Actually, the corresponding statement in [65] is much more general: with an additional help of an advice of polynomial length, *any* $O(\log n)$ space bounded nondeterministic Turing machine (not only a machine for GAP) can be made unambiguous and still working in logarithmic space.

result published in [65]: There exists U_{GAP}, a nondeterministic Turing machine working in logarithmic space and never using more than one accepting path on any input, and $\{\alpha_n\}_{n\geq 0}$, a fixed sequence of binary strings with lengths bounded by a polynomial in n, such that, for any graph $G = (V, E)$ with n vertices, $G \in \mathrm{GAP}$ if and only if U_{GAP} accepts the input $E \sharp \alpha_{n^2}$. That is, using $\{\alpha_n\}_{n\geq 0}$ as an assisting *advice of polynomial length* [47], we can accept GAP by an unambiguous machine U_{GAP} in logarithmic space.

Now, for the given a 2ONFA $\mathcal{A}$ with n states, we apply the construction used in Theorem 10 but, instead of the machine D_{GAP} on the virtual input $E(w)$, we simulate the unambiguous machine U_{GAP}, pretending that the input tape contains $E(w) \sharp \alpha_{n^2}$. (See also Figure 3.1.) Clearly, $w \in \|\mathcal{A}\|$ if and only if $G(w) \in \mathrm{GAP}$, which in turn holds if and only if U_{GAP} accepts $E(w) \sharp \alpha_{n^2}$.

Note that the length of $E(w) \sharp \alpha_{n^2}$ is polynomial in n, and does not depend on $|w|$. Thus, by the same reasoning as in Theorem 10, U_{GAP} uses $O(\log n)$ space on its worktape, and hence all data required during the simulation can be represented in a finite state control, with a number of states polynomial in n.

However, there are few differences. First, the position of the virtual input head is now in the range $0, \ldots, n^2+1+|\alpha_{n^2}|+1$. Second, the n^2 bits of $E(w)$ are computed "on the fly", by calling the subroutine $\textsc{Reach}(q_i, q_j)$, but we cannot access this way the second part of the virtual input — the advice string α_{n^2}. However, the advice depends only on the size of the graph $G(w)$ (but not on the graph $G(w)$ itself), which in turn depends only on the number of states in $\mathcal{A}$ (but not on the real input w). So, for the given 2ONFA $\mathcal{A}$ with n states, the advice α_{n^2} is fixed, and hence it can be encoded in the "hardware", *i.e.*, in the transition table for our new machine $\mathcal{A}''$.

Finally, observe that $\mathcal{A}''$ accesses its input tape only to compute the bits of the adjacency matrix $E(w)$, by calling the subroutine REACH. This deterministic subroutine starts and ends its computation with the head at the left endmarker. Thus, $\mathcal{A}''$ takes all nondeterministic decisions with the head scanning the left endmarker, to simulate the unambiguous machine U_{GAP}. Hence, $\mathcal{A}''$ is an unambiguous 2ONFA. □

3.7 The alternating case

In this section we briefly discuss an extension of the techniques used in Section 3.6, to the case of automata with *alternations* [14], considered in [43, 30]. Such automata combine the power of nondeterminism with parallelism.

A *2-way alternating automaton* (2AFA, for short) is defined in the same way as a 2NFA, but now the set of states Q is partitioned in two disjoint sets $Q_\exists$ and $Q_\forall$, the sets of *existential* and *universal* states, respectively.

The acceptance of an input string w by a 2AFA $\mathcal{A}$ is witnessed as follows. Consider the tree of all possible computation paths, starting from the initial configuration c_-. In this tree, a configuration $c = (q, i)$ is declared to be *successful*, if *(i)* the state q is accepting, or *(ii)* q is existential and at least one successor of $c = (q, i)$ is successful, or *(iii)* q is universal and all successors of $c = (q, i)$ are successful. The input w is accepted, if the initial configuration c_- becomes successful in this way. Notice that nondeterministic automata are just alternating automata without universal states.

Even for 2AFAs, we can restrict the use of (both existential and universal) choices as we did for 2NFAs, considering *2-way outer-alternating finite automata* (2OAFAs). In this model, the choices can be taken only when the head is scanning one of the endmarkers; a configuration positioned away from the endmarkers can have at most one successor.

Actually, using arguments very similar to those of Lemma 3, we can restrict the use of choices to the left endmarker:[9]

Lemma 6. *For each* 2OAFA $\mathcal{A} = (Q_\exists, Q_\forall, \Sigma, \delta, q_-, F)$ *with* n *states, there exists an equivalent* 2OAFA $\mathcal{A}'$ *with no more than* $3n$ *states that satisfies the following properties:*

- *both existential and universal choices are taken only when the input head is scanning the left endmarker,*
- *there is a unique accepting state* q_+ *and this state is also halting,*
- q_+ *is reachable only at the left endmarker, by stationary moves,*
- *stationary moves are used only at the left endmarker.*

Now, consider the *alternating graph accessibility problem* (AGAP, for short), an alternating version of GAP. The instance of the problem is an *alternating* directed graph, *i.e.*, a graph $G = (V_\exists, V_\forall, E)$ with a partition of $V = \{v_1, \ldots, v_n\}$ in two disjoint sets $V_\exists$ and $V_\forall$. The question is if the predicate APATH(v_1, v_n) is true, where, for $v_i, v_k \in V$, APATH(v_i, v_k) = TRUE if and only if:

- $v_i = v_k$, or
- $v_i \in V_\exists$ and, for some edge $(v_i, v_j) \in E$, APATH(v_j, v_k) = TRUE, or
- $v_i \in V_\forall$ and, for each edge $(v_i, v_j) \in E$, APATH(v_j, v_k) = TRUE.

[9] Notice a small difference with respect to Lemma 3, in which the stationary moves were allowed only to halt in q_+ at the left endmarker. However, here they can be used also for other purposes, but still only at the left endmarker. Due to the presence of both universal and existential states, a complete removal of stationary moves would require a more complicated argument than the simple one used to prove Lemma 3. However, this is not necessary for our purposes.

This problem is known to be complete for the class P, with respect to logarithmic space reductions [40].

As is Section 3.6, the edges in E can be represented by a binary adjacency matrix, of length $n \times n = n^2$ bits, but here we also need to specify partitioning of vertices into $V_\exists$ and $V_\forall$. This can be given as a string consisting of n additional bits, denoted here by $V_{\exists/\forall}$, in which the bit at the position i is equal to 0 or 1 depending on whether $v_i \in V_\exists$ or $v_i \in V_\forall$. Therefore, a binary represented instance of the problem is in the form $E \sharp V_{\exists/\forall}$, using n^2+n bits.

Now we can reduce the language accepted by a given 2OAFA $\mathcal{A}$, in normal form of Lemma 6, to AGAP: with each input string w, we associate the alternating graph $G(w) = (Q_\exists, Q_\forall, E(w))$, binary encoded by $E(w) \sharp Q_{\exists/\forall}$. Here we assume, without loss of generality, that $q_- = q_1$ and $q_+ = q_n$. The edges are defined in the usual way: $(q_i, q_j) \in E(w)$ if and only if $\mathcal{A}$ has a computational segment from q_i to q_j on the input w. (The extension of the notion of computational segments to 2AFA is obvious.) Since $\mathcal{A}$ makes both existential and universal choices only at the left endmarker, $w \in \|\mathcal{A}\|$ if and only if $G(w) \in$ AGAP.

Since the deterministic subroutine REACH presented in Section 3.3 depends only on the transition function of the given automaton $\mathcal{A}$ but not on the acceptance condition, we can use it to detect segments even in the case of outer 2AFAs.[10] This permits us to prove the following result:

Theorem 12. *If* L = P*, then each* 2OAFA $\mathcal{A}$ *with* n *states can be replaced by an equivalent* 2DFA $\mathcal{A}'$ *with a polynomial number of states.*

Proof. First, without loss of generality, we can assume that $\mathcal{A}$ is in normal form given by Lemma 6. Second, under the assumption that L = P, using also the fact that AGAP is complete for P, there must exist a deterministic Turing machine D_{AGAP} that solves AGAP in logarithmic space.

Now, for the given n-state 2OAFA $\mathcal{A}$, we apply the construction of Theorem 10 but, instead of the machine D_{GAP} on the virtual input $E(w)$, we simulate D_{AGAP} on the virtual input $E(w) \sharp Q_{\exists/\forall}$. Clearly, for each input string w, $w \in \|\mathcal{A}\|$ if and only if $G(w) \in$ AGAP, that is, if and only if D_{AGAP} accepts $E(w) \sharp Q_{\exists/\forall}$.

During the simulation, the first n^2 bits forming the string $E(w)$ are tested by calling the subroutine REACH with appropriate parameters, in the same way as used in the proof of Theorem 10. On the other hand, the last n bits that represent $Q_{\exists/\forall}$, partitioning Q

[10] As mentioned in the previous footnote, we can have stationary moves at the left endmarker. The subroutine REACH$(q_{\mathrm{S}}, q_{\mathrm{T}})$ has been implemented considering this possibility. In particular, it returns TRUE for each $q_{\mathrm{S}}, q_{\mathrm{T}}$ satisfying $q_{\mathrm{S}} = q_{\mathrm{T}}$ or $(q_{\mathrm{S}}, \triangleright, 0, q_{\mathrm{T}}) \in \delta$. (See item "Handling trivial cases" in the implementation of 2DFA $\mathcal{A}''$, at the end of Section 3.3.)

into existential and universal nodes, do not depend on w, but only on $Q_\exists$ and $Q_\forall$. So, for the given 2OAFA $\mathcal{A}$, this information is fixed, and hence it can be encoded in the transition table for our new machine $\mathcal{A}'$. (In the proof of Theorem 11, we encoded this way the advice string α_{n^2}.)

By the same reasoning as in the proof of Theorem 10, we also get that the new machine $\mathcal{A}'$ is a finite-state device, namely, a 2DFA with a polynomial number of states, accepting the same language as does $\mathcal{A}$. □

In a similar way, we can prove the following:

Theorem 13. *If* NL = P*, then each* 2OAFA $\mathcal{A}$ *with* n *states can be replaced by an equivalent* 2ONFA $\mathcal{A}'$ *with a polynomial number of states.*

Proof. Under the hypothesis NL = P, there exists a nondeterministic Turing machine N_{AGAP} that solves AGAP in logarithmic space.

The rest of the argument is the same as in the proof of Theorem 12, replacing the machine D_{AGAP} by the machine N_{AGAP}. The only difference from Theorem 12 is that now the resulting 2-way machine $\mathcal{A}'$ is nondeterministic.

Note also that nondeterministic choices are always made with the input head at the left endmarker, since $\mathcal{A}'$ accesses its input tape only by calling the deterministic subroutine REACH. Hence, $\mathcal{A}'$ is actually 2ONFA. □

3.8 Concluding remarks

In a unified framework, we have generalized some results from unary 2NFAs to machines with arbitrary input alphabets, but making nondeterministic choices only at the input tape endmarkers. Among others, we have shown that any superpolynomial lower bound for the conversion of such machines to standard 2DFAs would imply L ≠ NL. (Unlike in [7], there are no restrictions on the length of potential witness inputs.) In Section 3.7, we also related the alternating version of such machines to L $\stackrel{?}{=}$ NL $\stackrel{?}{=}$ P, the classical computational complexity open problems.

Comparing our results with those obtained for other restricted models of 2-way automata, we observe that:

- Actually, unary 2NFAs can use only a restricted form of nondeterminism. In fact, we can restrict their nondeterminism to the endmarkers without increasing significantly their size [32]. (A similar phenomenon has been observed by Chrobak in the case of unary *1-way* automata [16].)

- In the general case, the possibility of reversing the input head movement at any input position does not seem as powerful as the possibility of making nondeterministic decisions at any input position. (Compare our polynomial upper bound for the complementation of 2ONFAs with the exponential lower bound for the complementation of sweeping 2NFAs in [42].)
- However, in the deterministic case, the possibility of reversing the input head at any input position can make automata exponentially smaller than machines reversing the input head only at the endmarkers [6, 58].

It would be interesting to see if the results proved in this chapter could not be extended to a model using nondeterminism in a less restricted way than the one considered here.

Chapter 4

Super- and sub-classes of rational series

We recall that we made the convention of identifying binary relations of the direct product $\Sigma^* \times \Delta^*$ as formal series in $\mathbb{K}\langle\langle \Sigma^* \rangle\rangle$ where $\mathbb{K} = 2^{\Delta^*}$. We pursue our study of formal power series over an arbitrary semiring (see Section 2.2), by introducing some new operations with a view to applying our results to 2-way transductions (see Section 2.3.4).

4.1 Further operations on series

We introduce the Hadamard product and the Hadamard star of series. Their relevance to 2-way $\mathbb{K}$ automata, and more precisely, with sweeping and rotating $\mathbb{K}$ automata, is highlighted. The Hadamard product is usual, while the Hadamard star appears to be a new operation, which will make sense for rationally additive semirings such as the semiring $\text{RAT}(\Delta^*)$ (see Section 2.2.2 in Chapter 2).

Notation Given a sequence of operators $\omega_1, \omega_2 \ldots$ in $\mathbb{K}\langle\langle \Sigma^* \rangle\rangle$ and a family F of $\mathbb{K}$-series we denote by $[F, \omega_1, \omega_2, \ldots]$ the least family, when it exists, containing F and closed under $\omega_1, \omega_2, \ldots$ For example, the family of rational series is $[\text{POL}, +, \cdot, ^*]$.

4.1.1 Hadamard operations

The *Hadamard product* of two series σ and τ is defined as:

$$\sigma \text{Ⓗ} \tau \coloneqq \sum_{w \in \Sigma^*} \langle \sigma, w \rangle \cdot \langle \tau, w \rangle w$$

In particular, the support of $\sigma \text{Ⓗ} \tau$ is the intersection of both supports of σ and τ. Hence, if one of σ or τ is a polynomial (*resp.* a proper series), so is $\sigma \text{Ⓗ} \tau$. The Hadamard product

admits a neutral element: the series $1_{\text{Ⓗ}}$ that maps any word to 1 *i.e.*, the series defined as:

$$1_{\text{Ⓗ}} := \sum_{w \in \Sigma^*} 1w$$

Example 8. *We fix* Σ *and* $\mathbb{K} = \text{RAT}(\Sigma^*)$. *Consider the series* ID, *which maps every word to (the singleton containing) itself,* i.e.,

$$\text{ID} := \sum_{w \in \Sigma^*} \{w\}w$$

It is rational because it is equal to the polynomial series mapping each one-symbol word on Σ *into itself. Now, consider the Hadamard product of* ID *by itself:*

$$\text{SQUARE} := \sum_{w \in \Sigma^*} \{ww\}w$$

By the projection on the second component of SQUARE *is not rational, providing that* Σ *has at least two symbols.*

Observe that when $\mathbb{K}$ is commutative, the Hadamard product commutes, *i.e.*, $\sigma \text{Ⓗ} \tau = \tau \text{Ⓗ} \sigma$. Furthermore, it is proved in [66, Theorem III. 3.1] that in this case, the family RAT is closed under Hadamard product.

Theorem 14. *If* $\mathbb{K}$ *is commutative then the family* RAT *is closed under Hadamard product.*

As usual, we define the successive powers of a series under Hadamard product:

$$\sigma^{\text{Ⓗ}k} := \begin{cases} 1_{\text{Ⓗ}} & \text{if } k = 0; \\ \sigma \text{Ⓗ} \sigma^{\text{Ⓗ}(k-1)} & \text{otherwise.} \end{cases}$$

As expected, for any σ we have $\sigma^{\text{Ⓗ}0} = 1_{\text{Ⓗ}}$ and $\sigma^{\text{Ⓗ}1} = \sigma$. Then, we can define the *Hadamard star* of σ as:

$$\sigma^{\circledast} := \sum_{k \in \mathbb{N}} \sigma^{\text{Ⓗ}k} = \sum_{w \in \Sigma^*} \left(\sum_{k \in \mathbb{N}} \langle \sigma^{\text{Ⓗ}k}, w \rangle w \right)$$

when the infinite sum is defined.

Over rationally additive semirings (Section 2.2.2) the Hadamard star is always defined and it takes a simpler expression.

Proposition 8. *If* $\mathbb{K}$ *is a rationally additive semiring and* Σ *is an alphabet, then for every series* σ *in* $\mathbb{K}\langle\langle \Sigma^* \rangle\rangle$ *we have:*

$$\sigma^{\circledast} = \sum_{w \in \Sigma^*} \langle \sigma, w \rangle^* w$$

The families POL and PROP are not closed under Hadamard star as $0^{\circledast} = 1_{\text{Ⓗ}}$. The Hadamard star is idempotent, *i.e.*, $\left(\sigma^{\circledast}\right)^{\circledast} = \sigma^{\circledast}$.

Example 9. *Working on series in* $\mathrm{RAT}(a^*)\langle\langle a^*\rangle\rangle$*, we define the series* UMULT *as being the Hadamard star of* ID *(see Example 8):*

$$\mathrm{UMULT} := \mathrm{ID}^{\circledast} = \sum_{n\in\mathbb{N}} \left\{a^{kn} \mid k \in \mathbb{N}\right\} a^n$$

By identifying the monoids $a^* \times a^*$ *with* $\mathbb{N}\times\mathbb{N}$*, this series defines the relation "being a multiple of". However rational series of* $\mathbb{N}$ *are first-order definable in Presburger arithmetics,* i.e.*, arithmetics with addition only. Hence,* UMULT *is not rational. That implies that the family of rational series is not closed under Hadamard star, even in the* $\mathbb{K}$ *commutative and* Σ *unary case (observe the contrast with Theorem 14).*

The Hadamard operations were introduced because they are well-suited to the behavior of 2-way $\mathbb{K}$-FAs. Indeed, this family of series is closed under Hadamard operations.

Proposition 9. *If* σ *and* τ *are series respectively accepted by 2-way, sweeping, rotating* $\mathbb{K}$-FA*s, so are the series the* $\sigma \text{Ⓗ} \tau$ *and* $\sigma^{\circledast}$*. Moreover, if the* $\mathbb{K}$-FA*s are deterministic, so is the resulting* $\mathbb{K}$-FA *accepting* $\sigma \text{Ⓗ} \tau$*.*

Proof. The formal proof of the construction is left to the reader. The ideas are given in Figure 4.1, where $\mathcal{K}_\sigma$ and $\mathcal{K}_\tau$ boxes stand for direct simulations of the $\mathbb{K}$-FA accepting σ and τ respectively and where $-$ and $+$ are new states that are used to cross the entire input up to the left and the right endmarker respectively. Observe that sweepingness and rotatingness are preserved. For the Hadamard product, even determinism is preserved. □

Example 10. *A direct application of the previous proposition, is that the series* UMULT *defined in Example 9 is accepted by a rotating transducer. We give one in Figure 4.2 (observe the similarity with the construction presented in Figure 4.1b).*

The Hadamard product and the Hadamard star are called *Hadamard operations.* As shown in Example 8 and Example 9, the following defines a super family of the rational series.

Definition 13. *The family of* Hadamard series*, denoted* $\mathrm{HAD}(\mathbb{K}\langle\langle\Sigma^*\rangle\rangle)$ *or simply* HAD *if* $\mathbb{K}$ *and* Σ *are understood, is the closure of* $\mathrm{RAT}(\mathbb{K}\langle\langle\Sigma^*\rangle\rangle)$ *under Hadamard operations and sum,* i.e.*:*

$$\mathrm{HAD} = [\mathrm{RAT}, +, \text{Ⓗ}, \circledast]$$

Natural examples of Hadamard relations are SQUARE (Example 8) and UMULT (Example 9).

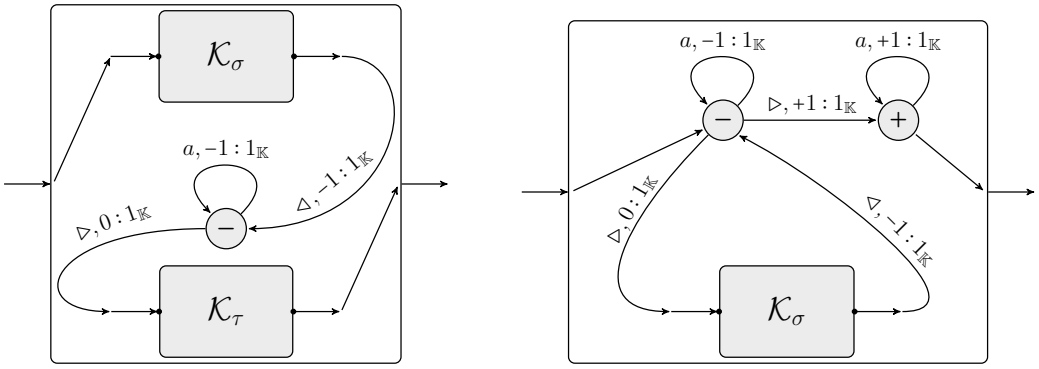


FIGURE 4.1: Construction of transducers accepting $\sigma \mathbin{Ⓗ} \tau$ (a) and $\sigma^{⊛}$ (b) respectively.

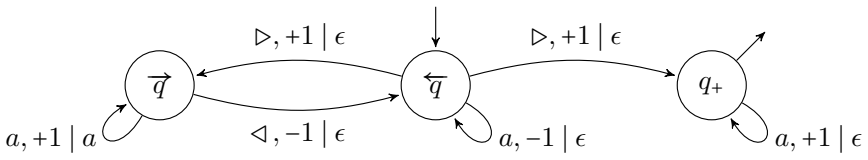


FIGURE 4.2: A rotating transducer accepting the relation UMULT (an edge (q, q') is labeled $(s, d \mid w)$ if ϕ maps the transition (q, s, d, q') to w).

We are able to prove that, under the assumption that $\mathbb{K}$ is rationally additive (see Section 2.2.2), the family HAD with sum, Hadamard product and Hadamard star, is also a rationally additive semiring.

Proposition 10. *If $\mathbb{K}$ is rationally additive, then $\left\langle \text{HAD}(\mathbb{K}\langle\langle \Sigma^* \rangle\rangle), +, Ⓗ, ^{⊛}, 0, 1_{Ⓗ} \right\rangle$ is a rationally additive semiring.*

Proof. This is a direct consequence of the general following observation: consider a class Γ of functions from a set X to a set Y provided with a some internal operations $\omega_1, \omega_2, \ldots$ of any arity. Extend each operation ω of arity r to Γ by setting, for $f_1, \ldots f_r \in \Gamma$ and $x \in X$, $\Omega(f_1, \ldots, f_r)(x) = \omega(f_1(x), \ldots, f_r(x))$. Every identity on X transfers to Γ. □

Since $\langle \text{HAD}, +, Ⓗ, 0, 1_{Ⓗ} \rangle$ is a semiring, we can define the matrix product with the specific operations of sum and Hadamard product. Therefore we can inductively define the successive powers of a matrix M, by setting $M^{Ⓗ 0}$ is equal to the matrix which has coefficients $1_{Ⓗ}$ on the diagonal and 0 elsewhere, and $M^{Ⓗ i+1} = M \mathbin{Ⓗ} M^{Ⓗ i}$ for each $i \geq 0$.

The transitive closure can be interpreted in the present case: let $\sigma_{i,j}$ the coefficient in position (i,j) of the matrix M, the entry (k,ℓ) of M^* is the power series σ defined as follows:

$$\langle \sigma, w\rangle = \left\{ \langle \sigma_{i_1,i_2}, w\rangle \cdot \langle \sigma_{i_2,i_3}, w\rangle \cdot \ldots \cdot \langle \sigma_{i_{r-1},i_r}, w\rangle \mid i_1, \ldots, i_r \in \{1,\ldots,n\},\ i_1 = k,\ i_r = \ell,\ r \geq 0 \right\}$$

The following extends the Kleene-Scützenberger Theorem (Theorem 5) to rotating $\mathbb{K}$-FAs.

Corollary 3. *The family of series recognized by rotating $\mathbb{K}$-FAs is equal to* HAD.

Proof. The inclusion of HAD in the family of series accepted by rotating $\mathbb{K}$-FAs is a consequence of Theorem 5 and Proposition 9.

Now we prove the opposite direction. Let $(\mathcal{A},\phi)$ be a rotating $\mathbb{K}$-FA with $\mathcal{A} = (Q,\Sigma,\rhd,\lhd,I,F,\delta)$. With the notation of Definition 10, let δ_{+1} be the intersection $\delta \cap (Q \times \Sigma_{\rhd\lhd} \times \{0,1\} \times Q_{+1})$. Observe that, for $q,q' \in Q$ the $\mathbb{K}$-FA

$$\mathcal{K}_{q,q'} = ((Q,\Sigma,\rhd,\lhd,\{q\},\{q'\},\delta_{+1}),\phi)$$

recognizes a rational series thanks to Theorem 5.

Now, we define for each pair (q,q'') the rational series $\sigma_{q,q''}$ as the sum of the series $\|\mathcal{K}_{q,q'}\|$ such that $(q',\lhd,-1,q'') \in \delta$.

Consider the matrix $M \in \mathrm{RAT}^{Q\times Q}$ whose (q,q')-entry is the series $\sigma_{q,q'}$. It describes the behavior of $(\mathcal{A},\phi)$ between two visits of the left endmarker. Since the semiring is rationally additive, we may consider the matrix M^*, which describes the behavior of the automaton in an arbitrary number of hits, starting and ending at the left endmarker. The series recognized by $(\mathcal{A},\phi)$ is thus

$$\sigma = \bigcup_{q\in I,\ q'\in Q,\ q''\in F} \left(M^*_{q,q'} \mathbin{Ⓗ} \|\mathcal{K}_{q',q''}\| \right)$$

□

4.1.2 Mirror operation

The last operator on series introduced here is the mirror, denoted MIR. The *mirror* of a series σ, is the series $\overline{\sigma}$ defined as:

$$\overline{\sigma} := \sum_{w\in\Sigma^*} \langle \sigma, w\rangle \overline{w}$$

Recall that $\overline{w}$ denotes the mirror of the word w as introduced in Section 2.1.2. Since the mirror on words is an involution, so is the mirror of series, *i.e.*, for any σ we have $\overline{\overline{\sigma}} = \sigma$.

By definition, we have: $\text{SUPP}(\overline{\sigma}) = \overline{\text{SUPP}(\sigma)}$. Thus, the families POL and PROP are both closed under mirror. However, it can be shown that, in general, neither RAT nor HAD are closed under mirror. Thus, the next definitions introduce new families of series.

Definition 14. *The family of* Mirror-rational series*, denoted* $\text{MRAT}(\mathbb{K}\langle\langle\Sigma^*\rangle\rangle)$ *or simply* MRAT *when the context is clear, is the closure of* $\text{POL}(\mathbb{K}\langle\langle\Sigma^*\rangle\rangle)$ *under rational operations and mirror* i.e.*:*

$$\text{MRAT} := [\text{POL}, \text{MIR}, +, \cdot, *]$$

The family of Mirror-Hadamard series*, denoted* $\text{MHAD}(\mathbb{K}\langle\langle\Sigma^*\rangle\rangle)$ *or simply* MHAD *when the context is clear, is the closure of* $\text{RAT}(\mathbb{K}\langle\langle\Sigma^*\rangle\rangle)$ *under Hadamard operations and mirror* i.e.*:*

$$\text{MHAD} := \left[\text{RAT}, \text{MIR}, +, \text{Ⓗ}, ^{\circledast}\right]$$

Example 11. *Let* $\Sigma = \{a, b\}$ *and* $\mathbb{K} = \text{RAT}(\Sigma^*)$*. From the series* ID *defined in Example 8, we define the series* REV *which maps every word to its mirror,* i.e.*,*

$$\text{REV} := \overline{\text{ID}} = \sum_{w \in \Sigma^*} \{\overline{w}\} w$$

The series ID *is rational and hence* REV *is the mirror of a rational series.*

The series $\text{ID} \cdot \text{REV}$ *belongs to the family* MRAT*. It maps every word* w *into the set of words* $u\overline{v}$ *such that* $w = uv$*.*

The series $\text{ID} \text{Ⓗ} \text{REV}$ *belongs to the family* MHAD*. It maps every word* w *into the singleton* $\{w\overline{w}\}$*.*

As said previously, the mirror operator is an involution on Σ^*. It is a really natural operator, especially when considering 2-way or sweeping $\mathbb{K}$-FA which may scan their input word letter by letter from right to left.

In fact, as an extension of Corollary 3, the family MHAD characterizes the series accepted by sweeping $\mathbb{K}$-FAs.

Corollary 4. *A series is accepted by a sweeping* $\mathbb{K}$-FA *if and only if it belongs to* MHAD*.*

Proof. The first direction is a simple consequence of Theorem 5, Proposition 9 and Proposition 13, that we prove later. Indeed, it is easy to recognize the mirror of a rational series σ by a sweeping $\mathbb{K}$-FA, by simply scanning the input from right to left, while simulating a 1-way $\mathbb{K}$-FA accepting σ.

The other direction is proved similarly as for Corollary 3, considering left-to-right and right-to-left hits. The formal adaptation of the former proof is left to the reader. □

In the restricted case of unary alphabet, *i.e.* when Σ has cardinality 1, the mirror is trivially the identity mapping: $\overline{\sigma} = \sigma$. For more general cases, even when restricting the input to some recognizable sets such as *bounded languages*, *i.e.*, finite concatenations of powers of different symbols, the mirror performs a new operation.

In other to better describe the contribution provided by the mirror operator, we introduce new families of series. For a family $\mathcal{X}$, we denote by $\overline{\mathcal{X}}$ the family of series $\overline{\sigma}$ with $\sigma \in \mathcal{X}$. In particular:

$$\overline{\text{Rat}} := \{\overline{\sigma} \mid \sigma \in \text{Rat}\}$$
$$\overline{\text{Had}} := \{\overline{\sigma} \mid \sigma \in \text{Had}\}$$

The family Rat and $\overline{\text{Rat}}$ are incomparable.

Proposition 11. *The families* Rat *and* $\overline{\text{Rat}}$ *are incomparable.*

Proof. By involution, it suffices to prove $\overline{\text{Rat}} \not\subseteq \text{Rat}$. By Proposition 42 (later, in Chapter 6), if $\sigma \in \text{Rat}(\text{Rat}(\Sigma^*)\langle\langle\Sigma^*\rangle\rangle)$ then the language $L_\sigma = \left\{w\#\overline{\langle\sigma, w\rangle} \mid w \in \text{Supp}(\sigma)\right\}$ is context-free. The language $L_{\text{Rev}} = \{w\#w\}$ is not context-free. So Rev belongs to $\overline{\text{Rat}} \setminus \text{Rat}$ and hence Rat and $\overline{\text{Rat}}$ are incomparable. □

We can even more prove a stronger result, separating the families Rat and $\overline{\text{Had}}$.

Proposition 12. *The families* $\overline{\text{Rat}}$ *(*resp. Rat*) and* Had *(*resp. $\overline{\text{Had}}$*) are incomparable. The relation* Rev *is a natural separator, in* $\overline{\text{Rat}} \setminus \text{Had}$.

Proof.[1] By involution and since Rev trivially belongs to $\overline{\text{Rat}}$, it suffices to prove that Rev does not belong to $\overline{\text{Had}}$ or equivalently that it could not be accepted by a rotating transducer. We proceed by contradiction. Suppose that there exists a rotating transducer $\mathcal{T} = (\mathcal{A}, \phi)$ accepting Rev and let s denote its number of states. By Proposition 6 we may suppose that the production function ϕ is elementary and single-valued, and that the automaton $\mathcal{A}$ has a unique initial state q_- and a unique accepting state q_+. Because $\mathcal{T}$ is rotating and functional we can also suppose that it is loop-free. Moreover, we suppose without loss of generality that at each step exactly one of the two heads moves, *i.e.*, for each transition $t = (q, a, d, q')$, $\phi(t) = \epsilon$ if and only if $d \neq 0$.

[1] The main idea of this proof has been found with the help of Rémi De Joannis de Verclos during the summer school *EJC IM 2016* organized by the *GDR IM*. Many thanks to him.

Let w be a word in Σ^* and let m denote its length. We fix a successful run $\mathbf{r}$ of $\mathcal{T}$ on w. Since $\mathcal{T}$ is rotating and loop-free, the hit factorization of $\mathbf{r}$ is composed in $2k+1$ hits $\mathbf{h}_1, \ldots, \mathbf{h}_{2k+1}$ for some $k < s$, *i.e.*,

$$\mathbf{r} = \mathbf{h}_1 @ \mathbf{h}_2 @ \ldots @ \mathbf{h}_{2k+1} \qquad k < s \tag{4.1}$$

where for each $1 \leq i \leq 2k+1$, depending on whether i is odd or even, $\mathbf{h}_i$ is a left to right hit or a right to left rewind. In particular, $\phi(\mathbf{r}) = \phi(\mathbf{h}_1)\phi(\mathbf{h}_3)\cdots\phi(\mathbf{h}_{2k+1})$.

Since the input and the output have the same length, we can imagine the two tapes one on top of the other. For convenience, we suppose that the write head moves from right to left, starting from the rightmost position. Due to the assumption made on the transitions of $\mathcal{T}$, at each step of a left to right hit, exactly one of the two heads moves one cell to the right or to the left accordingly. Therefore, for each left to right hit $\mathbf{h}_{2i-1}$ with $1 \leq i \leq k+1$, there exists exactly one position p_i on which the two heads coincide. Let q_i be the state of the machine when this event occurs and observe that the sequence of p_i is decreasing. These $k+1$ positions define a factorization $w_{k+1}\cdots w_0$ of the input word, where, for each $0 \leq i \leq k+1$, the length of w_i is equal to $p_i - p_{i+1}$ with the convention $p_0 = m$ and $p_{k+2} = 0$ (see Figure 4.3). Based on this observation, we can find a second decomposition of $\mathbf{r}$ into $k+2$ runs: $\mathbf{r} = \mathbf{t}_0 @ \ldots @ \mathbf{t}_{k+1}$ where:

- $\mathbf{t}_0$ is the initial sub-run from the initial configuration $(q_-, 0)$ to the first time the read and the write heads reach the same position, *i.e.*, to the configuration (q_1, p_1);
- for each $0 < i \leq k$, $\mathbf{t}_i$ is the sub-run between the i-th and $(i+1)$-th time the positions of the two heads coincide, *i.e.*, from the configuration (q_i, p_i) to the configuration (q_{i+1}, p_{i+1});
- $\mathbf{t}_{k+1}$ is the final sub-run from the last time this event occurs to the accepting configuration, *i.e.*, from the configuration (q_{k+1}, p_{k+1}) to the final configuration $(q_+, m+1)$.

Let us enter into more details and consider one factor $\mathbf{t}_i$ for some $0 < i \leq k$. During that sub-run, the transducer performs a three step run $\mathbf{r}_{\text{suffix}} @ \mathbf{r}_{\text{rewind}} @ \mathbf{r}_{\text{prefix}}$ (see Figure 4.4), where:

1. $\mathbf{r}_{\text{suffix}}$ is a left to right run starting from the configuration (q_i, p_i) and ending at the right endmarker, thus scanning a suffix $c \cdot z$ of w where $z := w_{i-1}w_{i-2}\cdots w_0$ and c is the last letter of w_i;
2. $\mathbf{r}_{\text{rewind}}$ is a mute and blind one-way right to left rewind on w;
3. $\mathbf{r}_{\text{prefix}}$ is a left to right run starting from the left endmarker and ending in the configuration (q_{i+1}, p_{i+1}), thus scanning a prefix $x := w_{k+1}w_k\cdots w_{i+1}$ of w.

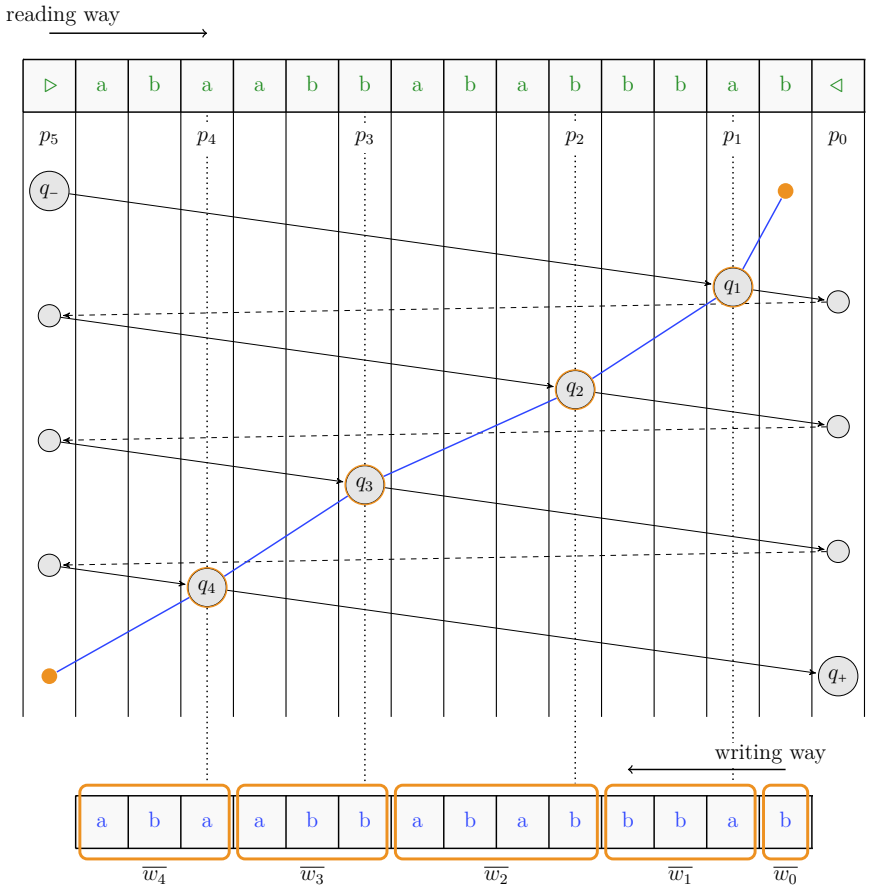


FIGURE 4.3: The factorization of $\mathbf{r}$ according to the times at which the input and output heads coincide. (The output tape should be considered from right to left.)

It remains to settle the other two cases: if $i = 0$ (*resp.* $i = k+1$) then the run $\mathbf{t}_i$ is reduced to the third (*resp.* the first) step. Denote by (q,p) and (q',p') the first and the last configuration of $\mathbf{t}_i$ respectively. In all cases the word produced, which is unique since $\mathcal{T}$ is single-valued, is exactly the word $y := \overline{w_i}$. Moreover, if there exists another run starting from the configuration (q,p) and ending in the configuration (q',p'), then its associated image by ϕ should be equal to y because $\mathcal{T}$ is functional. Hence y depends on four paramaters only: the states q and q' and the factors x and $c \cdot z$. Indeed, given q, q', x and $c \cdot z$, it is easy to compute y by simply simulating the runs $\mathbf{r}_{\text{suffix}}$ and $\mathbf{r}_{\text{prefix}}$. Therefore, since $w = x \cdot \overline{y} \cdot z$, we can recover w.

Now, we reconsider the choice of i. Indeed, because the output associated to the

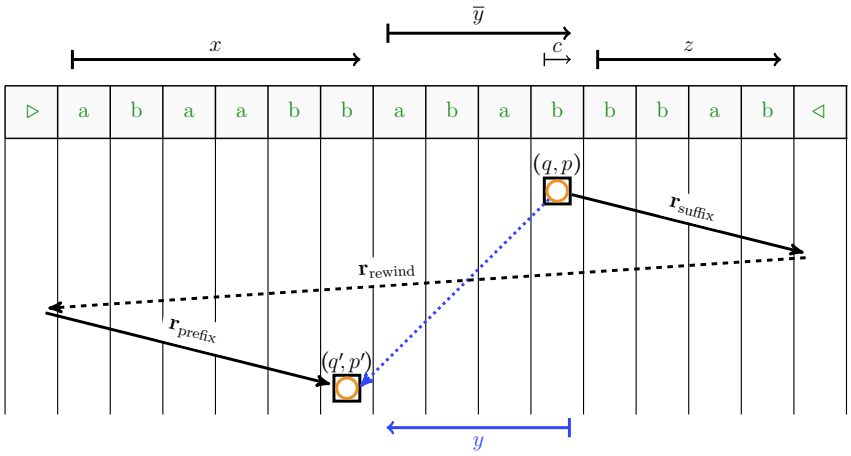


FIGURE 4.4: Decomposition in three steps of a sub-run $\mathbf{t}_i$.

run $\mathbf{r}$ has length $m = |w|$ and since we have decomposed w in at most $s+1$ factors, there exists an i such that $\phi(\mathbf{t}_i) = w_i$ has length at least $\frac{m}{s+1}$. In this way, the prefix x and the suffix $c \cdot z$ associated to the sub-run $\mathbf{t}_i$ satisfy $|x| + |c \cdot z| = |x \cdot c \cdot z| \leq \frac{sm}{s+1} + 1$.

Consequently, for each word w of length m there exists a quadruple (q, q', x, z) such that $|xz| \leq \frac{sm}{s+1}+1$ and w is computable from (q, q', x, z). So when m is fixed, we can deduce from the first statement an upper bound on the number of possible such quadruples:

$$s^2 \times \text{Card}\left(\left\{(x,z) \in \Sigma^* \times \Sigma^* \mid |xz| \leq \frac{sm}{s+1} + 1\right\}\right) \leq s^2 \times m \times |\Sigma|^{\frac{sm}{s+1}+2}$$

The second statement shows that distinct words cannot be associated to the same quadruple. Hence, because the accepted function is total, the number of such quadruples should be at least equal to the number of words of length m. For sufficiently large enough m, the previous upper bound is less than $|\Sigma|^m$. This is a contradiction. □

Combining mirror and Hadamard operations

The mirror commutes with the Hadamard operations:

Proposition 13. *The mirror commutes with the sum, the Hadamard product and the Hadamard star.*

Proof.

- sum:

$$\overline{\sigma+\tau} = \sum_{w\in\Sigma^*} \langle \sigma+\tau, w\rangle \overline{w} = \sum_{w\in\Sigma^*} \Big(\langle \sigma, w\rangle + \langle \tau, w\rangle\Big)\overline{w}$$
$$= \left(\sum_{w\in\Sigma^*} \langle \sigma, w\rangle \overline{w}\right) + \left(\sum_{w\in\Sigma^*} \langle \tau, w\rangle \overline{w}\right) = \overline{\sigma} + \overline{\tau}$$

- Hadamard product:

$$\overline{\sigma \,\text{Ⓗ}\, \tau} = \sum_{w\in\Sigma^*} \langle \sigma \,\text{Ⓗ}\, \tau, w\rangle \overline{w} = \sum_{w\in\Sigma^*} \Big(\langle \sigma, w\rangle \cdot \langle \tau, w\rangle\Big)\overline{w}$$
$$= \left(\sum_{w\in\Sigma^*} \langle \sigma, w\rangle \overline{w}\right) \text{Ⓗ} \left(\sum_{w\in\Sigma^*} \langle \tau, w\rangle \overline{w}\right) = \overline{\sigma} \,\text{Ⓗ}\, \overline{\tau}$$

- Hadamard star:

$$\overline{\sigma^{\circledast}} = \sum_{w\in\Sigma^*} \left\langle \sigma^{\circledast}, w\right\rangle \overline{w} = \sum_{w\in\Sigma^*} \langle \sigma, w\rangle^* \overline{w} = \sum_{w\in\Sigma^*} \langle \sigma, \overline{w}\rangle^* w = \sum_{w\in\Sigma^*} \langle \overline{\sigma}, w\rangle^* w = \overline{\sigma}^{\circledast}$$

□

Hence, the family $\overline{\text{HAD}}$ is the closure under Hadamard operation of the family $\overline{\text{RAT}}$.

Corollary 5. $\overline{\text{HAD}} = \left[\overline{\text{RAT}}, +, \text{Ⓗ}, \circledast\right]$

Considering now the family MHAD, by Proposition 13 we have:

Corollary 6. $\text{MHAD} = \left[\text{RAT} \cup \overline{\text{RAT}}, +, \text{Ⓗ}, \circledast\right]$

There exists a series in $\text{MHAD} \setminus \left(\text{HAD} \cup \overline{\text{HAD}}\right)$:

Example 12. *We fix the alphabet* Σ *and we work in* $2^{\Sigma^*}\langle\langle\Sigma^*\rangle\rangle$. *Let* PAL_2 *be the series defined as:* $\text{PAL}_2 := \text{ID}\,\text{Ⓗ}\,\text{REV}$. *Observe that the set of images of* PAL_2 *is exactly the set of palindromes of even length. It belongs to* MHAD *by definition, but it belongs neither to* HAD *nor to* $\overline{\text{HAD}}$ *(see Proposition 21).*

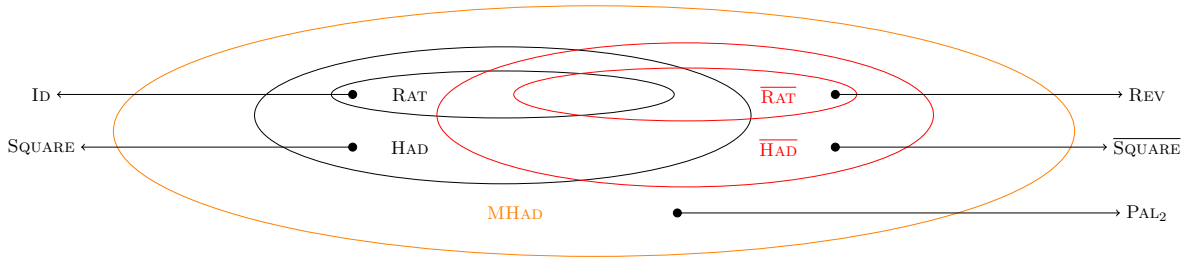


FIGURE 4.5: The families RAT, $\overline{\text{RAT}}$, HAD, $\overline{\text{HAD}}$ and MHAD.

Combining mirror and rational operations on commutative semirings

As seen before with the series ID and REV, the mirror of a rational series is, in general, not rational. That is, RAT and $\overline{\text{RAT}}$ are incomparable. However, this does not hold anymore, when the semiring $\mathbb{K}$ is commutative. In this case we are able to prove some properties of the mirror operation:

Proposition 14. *If $\mathbb{K}$ is a commutative semiring, then the mirror anticommutes with the Cauchy product and commutes with the Kleene star.*

Proof. For w in Σ^*.

- Cauchy product:

$$\begin{aligned}
\langle \overline{\sigma \cdot \tau}, w\rangle &= \langle \sigma \cdot \tau, \overline{w}\rangle \\
&= \sum_{uv=\overline{w}} \langle \sigma, u\rangle \cdot \langle \tau, v\rangle \\
&= \sum_{\overline{v}\cdot\overline{u}=w} \langle \tau, v\rangle \cdot \langle \sigma, u\rangle && \text{— by commutativity of } \mathbb{K} \\
&= \sum_{\overline{v}\cdot\overline{u}=w} \langle \overline{\tau}, \overline{v}\rangle \cdot \langle \overline{\sigma}, \overline{u}\rangle && \text{— by idempotency of the mirror} \\
&= \langle \overline{\tau} \cdot \overline{\sigma}, w\rangle
\end{aligned}$$

- Kleene star:

$$\left\langle \overline{\sigma^*}, w\right\rangle = \langle \sigma^*, \overline{w}\rangle = \sum_{u_0\cdots u_k=\overline{w}} \langle \sigma, u_0\rangle\cdots\langle \sigma, u_k\rangle = \sum_{\overline{u_k}\cdots\overline{u_0}=w} \langle \overline{\sigma}, \overline{u_k}\rangle\cdots\langle \overline{\sigma}, \overline{u_0}\rangle = \langle \overline{\sigma}^*, w\rangle$$

□

Thus, when $\mathbb{K}$ is commutative, the family RAT is closed under mirror.

Corollary 7. *If* $\mathbb{K}$ *is commutative, then* MRat = Rat.

Proof. As claimed in Section 4.1.2, the mirror of a polynomial is trivially a polynomial. Thus, it directly follows from Proposition 14 that the family of rational series over commutative semiring is closed under mirror. □

If $\mathbb{K}$ is commutative, the family MHad is equal to Had.

Corollary 8. *If* $\mathbb{K}$ *is commutative then* MHad = Had.

Proof. By Corollary 6, we have $\mathrm{MHad} = \left[\mathrm{Rat} \cup \overline{\mathrm{Rat}}, +, ⓗ, \circledast\right]$. It follows from Corollary 7, that $\overline{\mathrm{Rat}} = \mathrm{Rat}$. Hence, $\mathrm{MHad} = \left[\mathrm{Rat}, +, ⓗ, \circledast\right] = \mathrm{Had}$. □

4.1.3 On the scalar product

Observe that any constant series is of the form $k \cdot 1_{ⓗ}$, where $1_{ⓗ}$ is the series with all the coefficients equal to 1, as defined in Section 4.1.1. We thus denote such a series by $k_{ⓗ}$.

In fact, it is also possible to consider it as a particular Hadamard product. Indeed, for each k and each σ we have $k \cdot \sigma = k_{ⓗ} ⓗ \sigma$ and $\sigma \cdot k = \sigma ⓗ k_{ⓗ}$. Through these expressions as Cauchy or Hadamard product, the scalar products inherits all the basic properties satisfied by the two former products. In particular, all the families of series defined until now are closed under scalar product.

Proposition 15. *The families* Pol, Prop, Rat, $\overline{\mathrm{Rat}}$, Had, $\overline{\mathrm{Had}}$, MRat *and* MHad *are all closed under scalar product.*

4.1.4 Restriction to a recognizable support

We show that the restriction to a recognizable support (see Section 2.2.4) enjoys nice commutativity properties with all the Hadamard operations and the mirror.

Proposition 16. *Given two series* σ *and* τ *in* $\mathbb{K}\langle\langle\Sigma^*\rangle\rangle$ *and a language* L *on* Σ*, we have:*

$$(\sigma + \tau)_{|L} = \sigma_{|L} + \tau_{|L} \qquad (\sigma ⓗ \tau)_{|L} = \sigma_{|L} ⓗ \tau_{|L} \qquad \left(\sigma^{\circledast}\right)_{|L} = \left(\sigma_{|L}\right)^{\circledast} \qquad \overline{\sigma_{|L}} = \overline{\sigma}_{|\overline{L}}$$

Proof. The proofs are immediate. □

A direct corollary of Proposition 1 and Proposition 16 is the stability of all the families of series considered until now, by the operation of restriction to a recognizable language.

Corollary 9 (Support restriction to recognizable language)**.** *Let σ be a series in $\mathbb{K}\langle\langle\Sigma^*\rangle\rangle$ and let L be a recognizable language on Σ. If σ belongs respectively to* Pol, Prop, Rat, Had, MRat, MHad, $\overline{\text{Rat}}$, *or* $\overline{\text{Had}}$, *then so does $\sigma_{|L}$.*

Proof. The case Pol, Prop or Rat is exactly Proposition 1. The other stabilities are obtain by the commutations of Proposition 16. □

4.2 Hierarchy

Here we compare a few families obtained via subsets of operations defined in Section 4.1 and in Section 2.2. Though restricted, we start by defining the following family, which is very natural.

4.2.1 Recognizable series

A word of caution: Eilenberg uses a notion of recognizable series which is shown to be equivalent to rational series [21]. Our definition is much more restrictive (see Proposition 19). It is an adaptation to arbitrary semirings, of the notion of recognizable subsets of $\Sigma^* \times \Gamma^*$ when binary relations are viewed as series in $2^{\Gamma^*}\langle\langle\Sigma^*\rangle\rangle$ (see Section 2.1.3).

Definition 15. *A series σ is* recognizable *if there exists a finite family of recognizable languages $L_1, \ldots, L_n$ on Σ, and constants $k_1, \ldots, k_n$ such that, denoting σ_i the characteristic series of L_i (that is, the series $1_{@_{|L_i}}$),*

$$\sigma := \sum_{0<i\leq n} k_i\sigma_i = \sum_{0<i\leq n}\left(\sum_{w\in L_i} k_i w\right)$$

The family of recognizable series is denoted $\text{Rec}(\mathbb{K}\langle\langle\Sigma^*\rangle\rangle)$ *or simply* Rec *when $\mathbb{K}$ and Σ are understood.*

By refining the sets $L_1, \ldots, L_n$, we may suppose that they form a partition of Σ^*.

Proposition 17. *Given a recognizable series σ, there exists a partition $P_1, \ldots, P_n$ of Σ^* with each P_i recognizable, and elements $k_1, \ldots, k_n$ in $\mathbb{K}$ such that, denoting σ_i the characteristic series of P_i,*

$$\sigma = \sum_{0<i\leq n} k_i\sigma_i = \sum_{0<i\leq n}\left(\sum_{w\in P_i} k_i w\right)$$

Proof. Suppose $\sigma = \sum_{0<i\leq n} k_i\sigma_i$ as in Definition 15. For every $s \in 2^{\{1,\dots,n\}}$ we define a language P_s and the element k'_s as follows:

$$P_s := \{w \in \Sigma^* \mid \forall i \in \{1,\dots,n\},\ w \in L_i \Leftrightarrow i \in s\} \qquad k'_s := \sum_{i\in s} k_i$$

The family $(P_s)_{s\subseteq\{1,\dots,n\}}$ is known as the coarsest refinement of the family $(L_i)_{1\leq i\leq n}$, which is a partition of Σ^*. Moreover, each P_s is a recognizable language, since it is a finite boolean combination of recognizable languages. We conclude the proof by observing that, denoting τ_i the characteristic series of P_i:

$$\sigma = \sum_{s\subseteq\{1,\dots,n\}} k'_s\tau_i$$

□

The family Rec is closed under all operations defined until now, except the Kleene star.

Proposition 18. *The family* Rec *is closed under sum, Cauchy product, Hadamard operations, mirror and restriction to a recognizable language.*

It is not closed under Kleene star, even when Σ is unary and $\mathbb{K}$ is commutative.

Proof. Let $\sigma = \sum_{0<i\leq n}\left(\sum_{w\in R_i}\langle\sigma_i, w\rangle\right)$ and $\tau = \sum_{0<j\leq m}\left(\sum_{w\in S_j}\langle\tau_j, w\rangle\right)$ be two recognizable series decomposed as in Definition 15, and let L be a recognizable language on Σ. Then:

$$\sigma + \tau = \sum_{\substack{0<i\leq n\\0<j\leq m}}\left(\sum_{w\in R_i\cap S_j}\big(\langle\sigma_i, w\rangle + \langle\tau_j, w\rangle\big)\,w\right) \quad \sigma \,Ⓗ\, \tau = \sum_{\substack{0<i\leq n\\0<j\leq m}}\left(\sum_{w\in R_i\cap S_j}\big(\langle\sigma_i, w\rangle \cdot \langle\tau_j, w\rangle\big)\,w\right)$$

$$\sigma^{⊛} = \sum_{0<i\leq n}\left(\sum_{w\in R_i}\langle\sigma_i, w\rangle^* w\right) \quad \overline{\sigma} = \sum_{0<i\leq n}\left(\sum_{w\in \overline{R_i}}\langle\overline{\sigma_i}, w\rangle w\right) \quad \sigma_{|L} = \sum_{0<i\leq n}\left(\sum_{w\in R_i\cap L}\langle\sigma_i, w\rangle w\right)$$

Observe that each $R_i \cap L$ is recognizable. Thus we proved that Rec is closed under sum, Hadamard product, Hadamard star, mirror and restriction to a recognizable language.

We now prove that Rec is not closed under Kleene star. The series uUnit $:= \{\epsilon\}\epsilon + \{a\}a$ in $2^{a^*}\langle\langle a^*\rangle\rangle$ is trivially recognizable since it is a polynomial. However, its Kleene star uId := uUnit$^* = \sum_{w\in a^*}\{w\}w$ is not recognizable, since it has an infinite amount of distinct coefficients. □

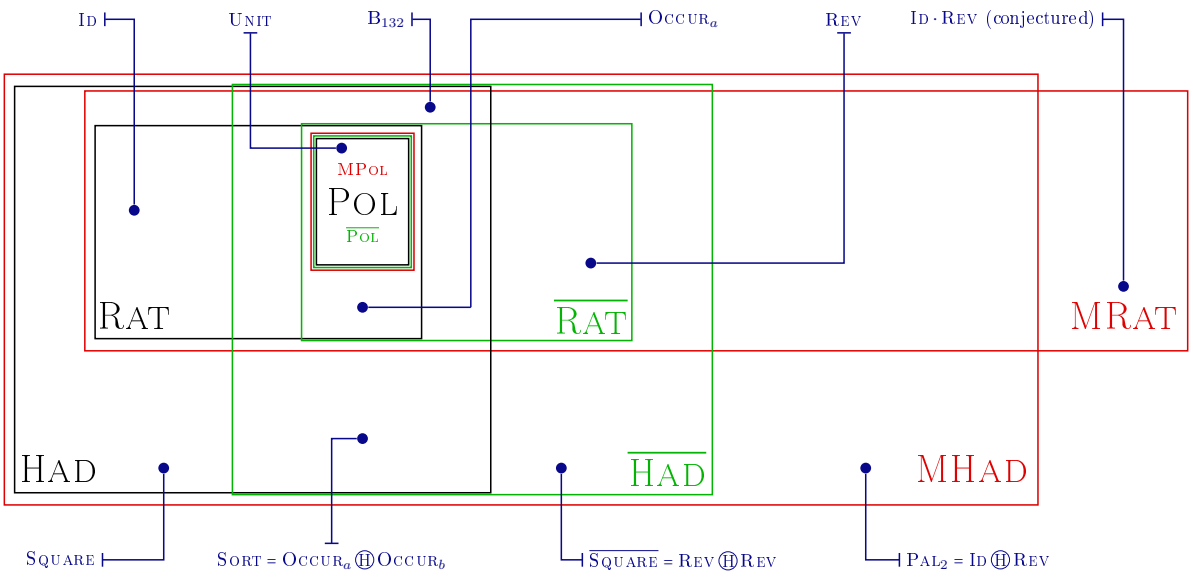
4.2.2 Comparison of families

In this section, we prove the following hierarchy, which is depicted in Figure 4.6:

$$\textsc{Pol} \subset \textsc{Rec} \subset \textsc{Rat} \subset \textsc{Had} \subset \textsc{MHad} \qquad \textsc{Rat} \notin \overline{\textsc{Had}} \notin \textsc{MRat} \qquad \textsc{Rat} \cup \overline{\textsc{Rat}} \subset \textsc{MRat}$$
$$\textsc{Rec} \subset \overline{\textsc{Rat}} \subset \overline{\textsc{Had}} \subset \textsc{MHad} \qquad \overline{\textsc{Rat}} \notin \textsc{Had} \notin \textsc{MRat} \qquad \textsc{MHad} \notin \textsc{MRat} \notin \textsc{MHad}$$

The results are of two types: inclusion and separation. A majority of the inclusion

FIGURE 4.6: The Hadamard hierarchy



presented above are direct consequences of the definitions. Some separating results have already be presented in Examples 8 or in Example 12.

First of all, we consider the family REC. Contrary to the families RAT, HAD, MRAT, or MHAD, it has not been introduced as the closure of some sub-family under some operations. It can be shown that this family strictly includes POL but is strictly included in $\textsc{Rat} \cap \overline{\textsc{Rat}}$.

Proposition 19. *The following inclusions hold:* $\textsc{Pol} \subset \textsc{Rec} \subset \left(\textsc{Rat} \cap \overline{\textsc{Rat}}\right)$.

Proof. The inclusion $\textsc{Pol} \subseteq \textsc{Rec}$ is trivial and the series $1_{\textcircled{H}}$ is clearly a non-polynomial recognizable series. Thus, $\textsc{Pol} \subset \textsc{Rec}$.

We now prove $\text{REC} \subseteq \text{RAT} \cap \overline{\text{RAT}}$. Since by definition RAT and $\overline{\text{RAT}}$ are closed under sum, it is sufficient to prove that for any constant series $k_{\circledH}$ and any recognizable language L, the series $k_{\circledH/L}$ belongs to $\text{RAT} \cap \overline{\text{RAT}}$. Thanks to the observations made in Section 4.1.3, this is a direct consequence of Proposition 18.

Finally, consider the series UID introduced in the proof of Proposition 18. It was proved that it does not belong to REC. However, it obviously belongs to $\text{RAT} = \overline{\text{RAT}}$. This concludes the proof. □

We now focus on the family MRAT. We introduce a new example of series, that belongs to HAD but not to MRAT. This also implies that its mirror belongs to $\overline{\text{HAD}}$ but not to MRAT, and hence $\text{HAD} \cup \overline{\text{HAD}} \notin \text{MRAT}$.

We consider the series SQUARE (Example 8). Observe that is is named after its image, which is the language of all square words on Σ, *i.e.*, words ww for $w \in \Sigma^*$. We first prove an instrumental lemma, stating a property satisfied by all the series in MRAT when $\mathbb{K}$ is the semiring of rational languages, *i.e.*, $\mathbb{K} = \text{RAT}(\Sigma^*)$.

Lemma 7. *Let Σ be an alphabet and let $\mathbb{K} = \text{RAT}(\Sigma^*)$. If σ belongs to $\text{MRAT}(\mathbb{K}\langle\langle\Sigma^*\rangle\rangle)$ then $\text{IMG}(\sigma)$ is a rational language on Σ.*

Proof. We proceed by structural induction. The result is trivially true for polynomials. Observe that we have:

$$\text{IMG}(\sigma + \tau) = \text{IMG}(\sigma) + \text{IMG}(\tau) \qquad \text{IMG}(\sigma^*) = \text{IMG}(\sigma)^*$$
$$\text{IMG}(\sigma \cdot \tau) = \text{IMG}(\sigma) \cdot \text{IMG}(\tau) \qquad \text{IMG}(\overline{\sigma}) = \overline{\text{IMG}(\sigma)}$$

Thus, by induction and using the closure of the class of rational languages under mirror, the image of a series in MRAT is a rational language on Σ. □

Proposition 20. $\text{HAD} \notin \text{MRAT}$ *and* $\overline{\text{HAD}} \notin \text{MRAT}$.

Proof. Observe first that the two statements above are equivalent, by involution of the mirror. Hence, it is sufficient to prove $\text{HAD} \notin \text{MRAT}$. By definition, SQUARE belongs to HAD. The image of SQUARE is:

$$\text{IMG}(\text{SQUARE}) = \{ww \mid w\Sigma^*\}$$

This language is known to be non-rational, which can be proved by a simple pumping argument. We conclude with Lemma 7. □

We can prove that PAL_2 (see Example 12) belongs to none of MRAT, $\overline{\text{HAD}}$ and HAD. The result uses a proposition that will be proved later, in Section 4.3. This implies that: $\left(\text{HAD} \cup \overline{\text{HAD}}\right) \subset \text{MHAD} \notin \text{MRAT}$.

Corollary 10. $\mathrm{MHad} \notin \mathrm{MRat}$

Proof. It is in fact a direct consequence of Proposition 20 together with the natural inclusion $\mathrm{Had} \cup \overline{\mathrm{Had}} \subseteq \mathrm{MHad}$. Nevertheless, observe that the same argument, namely Lemma 7, can be used in order to prove that $\mathrm{Pal}_2 \notin \mathrm{MRat}$. □

Proposition 21. $\left(\mathrm{Had} \cup \overline{\mathrm{Had}}\right) \subset \mathrm{MHad}$

Proof. By definition, Pal_2 belongs to MHad. It suffices to prove that Pal_2 does not belong to $\mathrm{Had} \cup \overline{\mathrm{Had}}$. To this aim, we consider a slightly different series, using a symbol $\# \notin \Sigma$:

$$\sigma := \mathrm{Id} \,Ⓗ\, 1_{\#} \,Ⓗ\, \mathrm{Rev} \qquad \text{with} \qquad 1_{\#} := \sum_{w \in \Sigma^*} \{\#\}w$$

In other words, σ is the series that maps every word w into the singleton $\{w\#\overline{w}\}$. Observe that it still belongs to MHad. By contradiction, suppose $\sigma \in \mathrm{Had}$. By Corollary 3, there exists a rotating $\mathbb{K}$-fa $\mathcal{K}$, with $\mathbb{K} = 2^{\Sigma^*}$, accepting it. It is easy to build a rotating $\mathbb{K}$-fa $\mathcal{K}'$ from $\mathcal{K}$, which recognizes Rev. This is a contradiction to Proposition 12. Similarly σ does not belong to $\overline{\mathrm{Had}}$. □

Finally we suggest a series which we conjecture to belong to $\mathrm{MRat} \setminus \mathrm{MHad}$:

$$\mathrm{Id} \cdot \mathrm{Rev} = \sum_{w \in \Sigma^*} \sum_{u \cdot v = w} \{u \cdot \overline{v}\}w$$

We even more conjectures that this series could not be accepted by a $\mathbb{K}$-fa. Moreover, we may prove that $\mathrm{Id} \cdot \mathrm{Rev}$ belongs neither to Rat nor to $\overline{\mathrm{Rat}}$.

Proposition 22. $\mathrm{Rat} \cup \overline{\mathrm{Rat}} \subset \mathrm{MRat}$

Proof. Similarly to the proof of Proposition 21, we consider a slight modification of the series $\mathrm{Id} \cdot \mathrm{Rev}$:

$$\sigma := \mathrm{Id} \cdot (\{\#\}\epsilon) \cdot \mathrm{Rev}$$

It clearly belongs to MRat. We may restrict its support to the recognizable language $\#\Sigma^*$ thanks to Corollary 9, still belonging to MRat. Hence, we conclude as in the proof of Proposition 11. Similarly, we prove $\sigma \notin \overline{\mathrm{Rat}}$. □

Conjecture 1. *We conjecture that the series* $\mathrm{Id} \cdot \mathrm{Rev}$ *does not belong to* MHad *and that no transducer could accept it.*

Though the expressive power of the mirror and the Hadamard operations seems to be of completely different nature, it is possible to find nonrational examples in which the former operation can be traded for the latters.

Proposition 23. *The family* $(\mathrm{MRat} \cap \mathrm{Had}) \setminus \left(\mathrm{Rat} \cup \overline{\mathrm{Rat}}\right)$ *is nonempty.*

Proof. Let us start with the definition of three simple (thus rational) relations. Recall that uId denote the identity relation over a unary alphabet, say $\{a\}$, and suppose $\#$ is a symbol distinct from a. We define:

$$\mathrm{Sep} := \{(\#, \#)\} \qquad \mathrm{Copy} := \mathrm{uId} \cdot \mathrm{Sep} \qquad \mathrm{Ign} := \{(a^n\#, \epsilon) \mid n \in \mathbb{N}\}$$

Now, consider the following more complex relation:

$$\mathrm{B}_{132} := \mathrm{uId} \cdot \mathrm{Sep} \cdot \overline{\mathrm{uId} \cdot \mathrm{Sep} \cdot \mathrm{uId}} \cdot \mathrm{Sep}$$

It is in fact a function which maps every word $a^n\#a^m\#a^\ell\#$ for some integers n, m and ℓ, into the singleton $\{a^n\#a^\ell\#a^m\#\}$. By definition it belongs to MRat. Using the same argument as in the proof of Proposition 11, we can easily prove that it is neither rational nor the mirror of a rational relation. Finally, observe that the following equality holds

$$\mathrm{B}_{132} = (\mathrm{Copy} \cdot \mathrm{Ign} \cdot \mathrm{Ign}) \mathbin{Ⓗ} (\mathrm{Ign} \cdot \mathrm{Ign} \cdot \mathrm{Copy}) \mathbin{Ⓗ} (\mathrm{Ign} \cdot \mathrm{Copy} \cdot \mathrm{Ign})$$

Each factor is rational and thus B_{132} belongs to Had. □

The unary and the commutative cases

When Σ is unary, the mirror operation is the identity *i.e.*, $\overline{\sigma} = \sigma$. Thus:

$$\mathrm{MHad} = \mathrm{Had} \qquad \text{and} \qquad \mathrm{MRat} = \mathrm{Rat}$$

We have proved in Corollaries 7 and 8 that the same equalities hold when $\mathbb{K}$ is a commutative semiring.

By considering the series uId (the identity series over a unary alphabet) and $\mathrm{uMult} := \mathrm{uId}^{\circledast}$ in $\mathrm{Rat}(\Sigma^*)\langle\langle\Sigma^*\rangle\rangle$, we trivially obtain that in these cases too, the following hierarchy is strict:

$$\mathrm{Pol} \subset \mathrm{Rat} \subset \mathrm{Had}$$

Both ways rational series

Here we are interested in the study of the family $\mathrm{bwRat} := \mathrm{Rat} \cap \overline{\mathrm{Rat}}$, called *both ways rational*. By previous observations, it is a proper subfamily of Rat. In terms of $\mathbb{K}$-fa, a series belongs to bwRat, if there exists two $\mathbb{K}$-fas recognizing it, one reading the input from left to right and the other from right to left. This seems to strongly restrict the family Rat. The family bwRat enjoys some general closure properties, that are detailed in below.

A direct consequence of Proposition 13 is that it is closed under sum:

Corollary 11. *The family* $\mathrm{RAT} \cap \overline{\mathrm{RAT}}$ *is closed under sum.*

The family of recognizable series (Definition 15) is a very weak family of series. The Cauchy product with recognizable series does not really increase the "power" of the series in BWRAT. This is the purpose of the following property:

Proposition 24. *Let* τ *and* τ' *be two recognizable series, and let* σ *be a series in* BWRAT. *Then the series* $\tau \cdot \sigma \cdot \tau'$ *belongs to* BWRAT.

Proof. Thanks to Corollary 11 and by distributivity, it is sufficient to prove that for any recognizable language $L \subseteq \Sigma^*$, any $k \in \mathbb{K}$ and any σ in BWRAT, the series $\tau \cdot \sigma$ (*resp.* $\sigma \cdot \tau$) with $\tau = k_{\textcircled{H}/L}$ (see Section 4.1.3 and Section 4.1.4) belongs to BWRAT. By the inclusion $\mathrm{REC} \subset \mathrm{RAT}$, it trivially belongs to RAT. We prove now that $\overline{\tau \cdot \sigma}$ belongs to RAT. For $w \in \Sigma^*$ we have:

$$\langle \overline{\tau \cdot \sigma}, w \rangle = \langle \tau \cdot \sigma, \overline{w} \rangle = \sum_{uv=\overline{w}} \langle \tau, u \rangle \cdot \langle \sigma, v \rangle = \sum_{\substack{uv=\overline{w} \\ u \in L}} k \cdot \langle \sigma, v \rangle = k \sum_{\substack{uv=\overline{w} \\ u \in L}} \langle \sigma, v \rangle \cdot 1$$

$$= k \sum_{uv=\overline{w}} \langle \sigma, v \rangle \left\langle 1_{\textcircled{H}/L}, u \right\rangle = k \sum_{\overline{v} \cdot \overline{u} = w} \langle \overline{\sigma}, \overline{v} \rangle \left\langle \overline{1_{\textcircled{H}/L}}, \overline{u} \right\rangle = k \cdot \left\langle \overline{\sigma} \cdot 1_{\textcircled{H}/\overline{L}}, w \right\rangle$$

Hence, observing that $k \cdot \overline{\sigma} \cdot 1_{\textcircled{H}/\overline{L}}$ is rational, we have $\overline{\tau \cdot \sigma} \in \mathrm{RAT}$. Similarly, we prove $\overline{\sigma \cdot \tau}$ belongs to RAT. □

The family BWRAT is closed under restriction to a recognizable support. This is a corollary of Proposition 9

Corollary 12. *The family* BWRAT *is closed under restriction to a recognizable language.*

More results on the family BWRAT are obtained in the particular case $\mathbb{K} = 2^{\Sigma^*}$, *i.e.*, relations on words (see Section 4.3.4).

4.3 Relations on Σ^*

In this section, we are interested in the particular case: of $\mathbb{K} = \left\langle 2^{\Sigma^*}, \cup, \cdot, \{\epsilon\}, \varnothing \right\rangle$. A *relation on* Σ^* is a subset of $\Sigma^* \times \Sigma^*$. Given a relation $R \subseteq \Sigma^* \times \Sigma^*$, the *canonical formal series associated to* R is the series σ_R in $2^{\Sigma^*} \langle\langle \Sigma^* \rangle\rangle$ defined by:

$$\text{for each } u \in \Sigma^*,\ \langle \sigma_R, u \rangle = \{v \mid (u, v) \in R\}$$

This correspondence allows us to identify relations and associated series. In particular, observe that the sum on $2^{\Sigma^*} \langle\langle \Sigma^* \rangle\rangle$ corresponds to the union on $2^{\Sigma^* \times \Sigma^*}$. We may also speak of the mirror of the relation R: $\overline{R} := \{(\overline{u}, v) \mid (u, v) \in R\}$ with, trivially, $\overline{\sigma_R} = \sigma_{\overline{R}}$.

Caveat. In general, a rational series in $2^{\Sigma^*}\langle\langle\Sigma^*\rangle\rangle$ (Definition 5) is not associated to a rational relation in $\Sigma^* \times \Sigma^*$ (Definition 1): this is due to the fact that the images of polynomials are unrestricted. Also, a series in $\mathrm{RAT}(\Sigma^*)\langle\langle\Sigma^*\rangle\rangle$ is not necessarily rational: consider for instance the characteristic series of any non-recursive set. What is true is that, by Theorem 6, the rational relations are exactly the relations associated to rational series in $\mathrm{RAT}(\Sigma^*)\langle\langle\Sigma^*\rangle\rangle$.

In the sequel, we are cautious to make the difference: RAT denotes the family of relations associated to a rational series in $2^{\Sigma^*}\langle\langle\Sigma^*\rangle\rangle$. When speaking of rational relations, we explicitly mention that the semiring $\mathrm{RAT}(\Sigma^*)$ is intended.

4.3.1 Symmetry of relations

On relations in $\Sigma^* \times \Sigma^*$ the following operation is natural.

Definition 16. *The* symmetric of R, *denoted* $\mathrm{SYM}(R)$, *is the relation*

$$\mathrm{SYM}(R) := \{(v,u) \mid (u,v) \in R\}$$

Trivially, symmetry is an involution, *i.e.*, $\mathrm{SYM}(\mathrm{SYM}(R)) = R$. The usual way to prove that the rational relations are closed under *symmetry* is to work with the equivalent representation as two-tape finite automata (see Section 6.1.4). Here we follow the more algebraic approach (see [8, 66]).

The symmetry operator enjoys nice commutativity properties with rational operations.

Proposition 25. *The symmetry operator commutes with the sum (*i.e.*, union), the Cauchy product and the Kleene star.*

Proof. Let R and R' be two relations.

- sum:
$$\mathrm{SYM}(R + R') = \{(v,u) \mid (u,v) \in R + R'\} = \mathrm{SYM}(R) + \mathrm{SYM}(R')$$
- Cauchy product:
$$\begin{aligned}\mathrm{SYM}(R \cdot R') &= \{(v,u) \mid (u,v) \in R \cdot R'\} \\ &= \{(v_1v_2, u_1u_2) \mid (v_1,u_1) \in \mathrm{SYM}(R) \text{ and } (v_2,u_2) \in \mathrm{SYM}(R')\} \\ &= \mathrm{SYM}(R) \cdot \mathrm{SYM}(R')\end{aligned}$$

- Kleene star:

$$\begin{aligned}\textsc{Sym}(R^*) &= \{(v,u) \mid (u,v) \in R^*\}\\ &= \{(v_1\cdots v_k, u_1\cdots u_k) \mid (u_i,v_i) \in R \text{ for each } i\}\\ &= \{(v_1\cdots v_k, u_1\cdots u_k) \mid (v_i,u_i) \in \textsc{Sym}(R) \text{ for each } i\}\\ &= \textsc{Sym}(R)^*\end{aligned}$$

□

Contrary to the mirror operations, the symmetry operation does not preserves polynomial neither proper relations. However, under some assumptions, we can prove that the symmetric of a polynomial relation is a rational relation.

Lemma 8. *In the semiring* $\mathbb{K}' = \textsc{Rat}(\Sigma^*)$*, the symmetric of any polynomial is recognizable.*

Proof. Because the symmetry operator commutes with the sum (Proposition 25), it suffices to observe that the symmetric of any relation $\{(w, L)\}$ is the recognizable relation $L \times \{w\}$. □

Using Proposition 25, it immediately follows that the family of rational relations is closed under symmetry.

Corollary 13. *For* $\mathbb{K}' = \textsc{Rat}(\Sigma^*)$*, the family* $\textsc{Rat}(\mathbb{K}'\langle\langle\Sigma^*\rangle\rangle)$ *is closed under symmetry. In other words the family of rational relations is closed under symmetry.*

4.3.2 Bi-mirror of relations

We introduce a new operator on relation on words.

Definition 17. *Given a relation* R *in* $\Sigma^* \times \Sigma^*$*: The* bi-mirror of R*, denoted* $\widehat{R}$*, is the relation:*

$$\widehat{R} := \{(u,v) \mid (\overline{u},\overline{v}) \in R\}$$

As for the mirror, we have: $\widehat{\widehat{R}} = R$. Observe that in general $\overline{\textsc{Sym}(R)} \neq \textsc{Sym}\big(\overline{R}\big)$. However, for each $R \subseteq \Sigma^* \times \Sigma^*$ we have:

$$\widehat{R} = \overline{\textsc{Sym}\Big(\overline{\textsc{Sym}(R)}\Big)} = \textsc{Sym}\Big(\overline{\textsc{Sym}\big(\overline{R}\big)}\Big) \tag{4.2}$$

From this expression, it can be deduced that the bi-mirror commutes with the mirror. In fact, the bi-mirror operation admits commutative properties with all the operations considered until now:

Proposition 26. *The bi-mirror commutes with the mirror, the sum, the Kleene star, the Hadamard star and the symmetry operations. It anti-commutes with the Cauchy product and the Hadamard product.*

Proof. Let R and S be two relations

- mirror, sum, symmetry: a direct consequence of Equation 4.2

Observe that each single operator, mirror and symmetry, does not commute with the Kleene and Hadamard stars.

- Kleene star:

$$\begin{aligned}\widehat{R^*} &= \{(u,v) \mid (\overline{u},\overline{v}) \in R^*\} \\ &= \{(u_1\cdots u_n, v_1\cdots v_n) \mid n \in \mathbb{N} \text{ and } (\overline{u_i},\overline{v_i}) \in R \text{ for each } i\} \\ &= \left\{(u_1\cdots u_n, v_1\cdots v_n) \mid n \in \mathbb{N} \text{ and } (u_i,v_i) \in \widehat{R} \text{ for each } i\right\} \qquad = \widehat{R}^*\end{aligned}$$

- Hadamard star:

$$\begin{aligned}\widehat{R^{\circledast}} &= \left\{(u,v) \mid (\overline{u},\overline{v}) \in R^{\circledast}\right\} \\ &= \{(u, v_1\cdots v_n) \mid n \in \mathbb{N} \text{ and } (\overline{u},\overline{v_i}) \in R \text{ for each } i\} \\ &= \left\{(u, v_1\cdots v_n) \mid n \in \mathbb{N} \text{ and } (u,v_i) \in \widehat{R} \text{ for each } i\right\} \qquad = \widehat{R}^{\circledast}\end{aligned}$$

The Cauchy and Hadamard products anti-commutes with the bi-mirror.

- Cauchy product:

$$\begin{aligned}\widehat{R\cdot S} &= \{(u,v) \mid (\overline{u},\overline{v}) \in R\cdot S\} \\ &= \left\{(u'\cdot u'', v'\cdot v'') \mid (\overline{u''},\overline{v''}) \in R \text{ and } (\overline{u'},\overline{v'}) \in S\right\} \\ &= \left\{(u'\cdot u'', v'\cdot v'') \mid (u'',v'') \in \widehat{R} \text{ and } (u',v') \in \widehat{S}\right\} \qquad = \widehat{S}\cdot\widehat{R}\end{aligned}$$

- Hadamard product:

$$\begin{aligned}\widehat{R \text{Ⓗ} S} &= \{(u,v) \mid (\overline{u},\overline{v}) \in R \text{Ⓗ} S\} \\ &= \{(u, v'\cdot v'') \mid (\overline{u},\overline{v''}) \in R \text{ and } (\overline{u},\overline{v'}) \in S\} \\ &= \{(u, v'\cdot v'') \mid (u,v'') \in R \text{ and } (u,v') \in S\} \qquad = \widehat{S} \text{Ⓗ} \widehat{R}\end{aligned}$$

□

Since $\textsc{Supp}\big(\widehat{R}\big) = \overline{\textsc{Supp}(R)}$, both the families $\textsc{Pol}\big(2^{\Sigma^*}\langle\langle\Sigma^*\rangle\rangle\big)$ and $\textsc{Prop}\big(2^{\Sigma^*}\langle\langle\Sigma^*\rangle\rangle\big)$ are closed under bi-mirror. As a corollary, all families defined in Section 2.2 are closed under bi-mirror operation.

Corollary 14. *Each family among* Pol, Prop, Rat, $\overline{\text{Rat}}$ *(and hence,* bwRat*),* MRat, Had, $\overline{\text{Had}}$ *and* MHad *is closed under bi-mirror.*

The family of recognizable relations, characterized in Theorem 1, is a very restricted family of relations. Therefore, it is not surprising that it is stable by symmetry and bi-mirror.

Proposition 27. *The family of recognizable relations on Σ^* is closed under bi-mirror and symmetry.*

Proof. Because union commutes with bi-mirror and symmetry, in virtue of Theorem 1 it suffices to consider the case $R = A \times B$ for some recognizable languages A and B. We have: $\widehat{R} = \overline{A} \times \overline{B}$ and $\textsc{Sym}(R) = B \times A$ which are both recognizable. □

4.3.3 Morphism

A morphism μ of Σ^* into Σ^* is a mapping defined by a substitution rule that associates a fixed word to each letter. This mapping is thus uniquely defined by the image of each letter. The *mirror of* μ is the morphism $\overline{\mu}$ defined by:

$$\text{for each } c \text{ in } \Sigma, \qquad \overline{\mu}(c) = \overline{\mu(c)}$$

Clearly, it follows that morphisms commute with the mirror:

Proposition 28. *For every word $w \in \Sigma^*$, we have $\overline{\mu(w)} = \overline{\mu}(\overline{w})$.*

Proof. $\overline{\mu(w)} = \overline{\mu(w_1)\cdots\mu(w_n)} = \overline{\mu(w_n)}\cdots\overline{\mu(w_1)} = \overline{\mu}(w_n)\cdots\overline{\mu}(w_1) = \overline{\mu}(w_n\cdots w_1) = \overline{\mu}(\overline{w})$ □

We define the *left-* and the *right-application of a morphism* μ to a relation R, denoted $\mu_l(R)$ and $\mu_r(R)$ respectively, as follows:

$$\mu_l(R) := (\mu(u), v) \mid (u, v) \in R \qquad\qquad \mu_r(R) := (u, \mu(v)) \mid (u, v) \in R$$

Trivially:

$$\mu_r(\mu_l(R)) = \{(\mu(u), \mu(v)) \mid (u, v) \in R\} = \mu_l(\mu_r(R)) \tag{4.3}$$

$$\mu_r(R) = \textsc{Sym}(\mu_l(\textsc{Sym}(R))) \qquad \text{and} \qquad \mu_l(R) = \textsc{Sym}(\mu_r(\textsc{Sym}(R))) \tag{4.4}$$

As a consequence of the previous proposition, the left- and right-applications of a morphism admit a kind of commutativity property with the mirror, the bi-mirror and the symmetry.

Proposition 29. *The following holds:*

$$\overline{\mu_l(R)} = \overline{\mu}_l\big(\overline{R}\big) \qquad \widehat{\mu_l(R)} = \overline{\mu}_l\big(\widehat{R}\big) \qquad \widehat{\mu_r(R)} = \overline{\mu}_r\big(\widehat{R}\big) \qquad \overline{\mu_r(R)} = \mu_r\big(\overline{R}\big)$$

Proof. The proof is simple routine.

- $\overline{\mu_l(R)} = \Big\{\Big(\overline{\mu(u)}, v\Big) \mid (u,v) \in R\Big\} = \{(\overline{\mu}(\overline{u}), v) \mid (u,v) \in R\} = \big\{(\overline{\mu}(u), v) \mid (u,v) \in \overline{R}\big\} = \overline{\mu}_l\big(\overline{R}\big)$
- $\widehat{\mu_l(R)} = \Big\{\Big(\overline{\mu(u)}, \overline{v}\Big) \mid (u,v) \in R\Big\} = \{(\overline{\mu}(\overline{u}), \overline{v}) \mid (u,v) \in R\} = \big\{(\overline{\mu}(u), v) \mid (u,v) \in \widehat{R}\big\} = \overline{\mu}_l\big(\widehat{R}\big)$
- $\widehat{\mu_r(R)} = \Big\{\Big(\overline{u}, \overline{\mu(v)}\Big) \mid (u,c) \in R\Big\} = \{(\overline{u}, \overline{\mu}(\overline{v})) \mid (u,v) \in R\} = \big\{(u, \overline{\mu}(v)) \mid (u,v) \in \widehat{R}\big\} = \overline{\mu}_r\big(\widehat{R}\big)$
- $\overline{\mu_r(R)} = \{(\overline{u}, \mu(v)) \mid (u,v) \in R\} = \big\{(u, \mu(v)) \mid (u,v) \in \overline{R}\big\} = \mu_r\big(\overline{R}\big)$

□

In fact, the applications of a morphism commute with all operations defined until now *i.e.*, symmetry, rational operations, Hadamard operations.

Proposition 30. *The (left- and right-) applications of morphisms to relations commute with rational operations and Hadamard operations and anti-commute with symmetry.*

Proof. The commutation with rational and Hadamard operations directly follows from the definition of morphism. Considering the symmetry, we obtain $\mu_l(\textsc{Sym}(R)) = \textsc{Sym}(\mu_r(R))$ and $\mu_r(\textsc{Sym}(R)) = \textsc{Sym}(\mu_l(R))$ from Equation (4.4). □

Since the application of a morphism to a polynomial is still a polynomial, the following is a direct consequence of the preceding proposition.

Corollary 15. *The families* Pol, Prop, Rat, $\overline{\textsc{Rat}}$, MRat, Had, $\overline{\textsc{Had}}$ *and* MHad *are closed under left- and right-application of morphisms.*

Proof. Observe that if R is proper (*resp.* a polynomial), then so are $\mu_l(R)$ and $\mu_r(R)$. The other closure properties follow from Proposition 30. □

4.3.4 Both ways rational relations

We are interested in the family of both ways rational series (bwRat) in $\mathbb{K}$, *i.e.*, the relations belonging to $\textsc{Rat} \cap \overline{\textsc{Rat}}$ (see Section 4.2.2). Very simple rational relations do not belong to this family, for example Id. From the definition we obtain:

Corollary 16. *The family* BWRAT *is closed under mirror, bi-mirror, left- and right-application of a morphism. On* $\mathbb{K}' = \mathrm{RAT}(\Sigma^*)$ *it is closed under symmetry.*

Proof. The claims are direct consequences of: mirror (definition), bi-mirror (Corollary 14), left- and right-application of morphism (Corollary 15) and symmetry (Corollary 13). □

Here is a very simple subfamily of BWRAT.

Corollary 17. *Let* R *be in* RAT. *If for some* $c \in \Sigma$, *we have* $R \subseteq c^* \times \Sigma^*$ *then* $\overline{R}$ *is rational and thus* R *belongs to* BWRAT. *When* $\mathbb{K} = \mathrm{RAT}(\Sigma^*)$, *the same holds if* $R \subseteq \Sigma^* \times c^*$.

Proof. If $R \subseteq c^* \times \Sigma^*$, then $\overline{R} = R$. Thus, $R \in \mathrm{RAT}$ implies $R \in \mathrm{BWRAT}$. By the closure of BWRAT under symmetry when $\mathbb{K} = \mathrm{RAT}(\Sigma^*)$, the same holds for $R \subseteq \Sigma^* \times c^*$. □

Since the mirror of a recognizable relation is recognizable, we trivially have that the family REC is included in BWRAT. It is easy to find a relation in $\mathrm{BWRAT} \setminus \mathrm{REC}$. For example, the relation $\mathrm{OCCUR}_a := \{(w, a^n) \mid w \in \Sigma^* \text{ and } n = |w|_a\}$ is rational, and it is equal to its own mirror, so it belongs to BWRAT. But it does not belong to REC. In fact, as a consequence of Proposition 24, a relation in BWRAT can be concatenated in different ways with recognizable relations, leading to relations still in BWRAT.

Proposition 31. *Let* R *be in* BWRAT *and let* S *be recognizable. Then, both* $R \cdot S$ *and* $S \cdot R$ *belong to* BWRAT.

Proof. It suffices to observe that a recognizable relation has a recognizable associated series. Then, the result follows from Proposition 24. □

Chapter 5

Two-way transducers

5.1 Introduction

In the theory of words, two different terms are more or less indifferently used to describe the same objects: transductions and binary relations. The former term distinguishes an input and an output, even when the input does not uniquely determine the output. In certain contexts it is a synonym for translation where one source and one target are understood. The latter term is meant to suggest pairs of words playing a symmetric role.

Transducers and two-tape automata are the devices that implement the transductions and relations respectively. The concept of multitape- and thus in particular two-tape automata was introduced by Rabin and Scott [63] and also by Elgot and Mezei [23] almost fifty years ago. Most closure and structural properties were published in the next couple of years. As an alternative to a definition via automata it was shown that these relations were exactly the rational subsets of the direct product of free monoids. On the other hand, transductions, which are a generalization of (possibly partial) functions, is a more suitable term when the intention is that the input preexist the output. The present work deals with 2-way transducers which are such a model of machine using two tapes. An input tape is read-only and is scanned in both directions. An output tape is write-only, initially empty and is explored in one direction only. The first mention of 2-way transducers is traditionally credited to Shepherdson.

Our purpose is to define a structural characterization of these relations in the same way that the relations defined by multi-tape automata are precisely the rational relations. However we limit our investigation to the case where the input and output are words over a one letter alphabet, *i.e.*, to the case where they both belong to the free monoid a^* generated by the unique letter a. Our technique does not apply to non-unary alphabets. An output is written on a second write-only tape.

We are able to prove the following that every 2-way transductions in $a^* \times a^*$ (*i.e.*, a relation accepted by a 2-way transducer as defined in Section 2.3.4) is an Hadamard relation (see Definition 13). By characterizing the Hadamard relations of $a^* \times a^*$, our result can be reformulated as follows:

Theorem 15. *A relation of the monoid $a^* \times a^*$ is defined by a 2-way transducer if and only if it is Hadamard or, equivalently, it is a finite union of relations R satisfying the following condition: there exist two rational relations $S, T \subseteq a^* \times a^*$ such that for all $x \in a^*$ we have*

$$R(x) = S(x)T(x)^*$$

The relation $\{(a^n, a^{kn}) \mid n, k \geq 0\}$ is a simple example. It is of the previous form, however it is not rational. Indeed, identifying a^* with the additive monoid of integers $\mathbb{N}$ this relation defines the relation "being a multiple of". However rational subsets of $\mathbb{N}$ are first-order definable in Presburger arithmetics, *i.e.*, arithmetics with addition only.

We quickly review the few results which to the best of our knowledge are published on 2-way transducers when considering them for their own sake. Engelfriet and Hoogeboom showed that a function on the free monoid is defined by a deterministic 2-way transducer if and only if it is the set of models of an MSO formula, [24]. In [26] the authors show that given a transducer accepting a function, it is decidable whether or not it is equivalent to a 1-way transducer, and when it is, an equivalent 1-way transducer is computable.

We now turn to a short presentation of the content of the chapter. In the next section we show our main result: the characterization of 2-way transductions defined over unary input and output alphabet as Hadamard relations. Then, in the second section, we prove that the assumptions on the two alphabets of the above mentioned result is strongly required. Indeed, we exhibit two natural relations, one with a unary input alphabet, the second with a unary output alphabet, which are accepted by a 2-way transducer but by no sweeping transducer.

5.2 Unary two-way transductions

5.2.1 Hadamard relation with unary output

We consider relations on $\Sigma^* \times \Delta^*$. Whenever Δ is unary, the family of Hadamard relations (see Definition 13) has a simpler characterization.

Proposition 32. *A relation R belongs to* $\textsc{Had}(\Sigma^* \times \Delta^*)$ *with Δ unary if and only if there exists two finite families of rational relations R_is and S_is, such that:*

$$R = \bigcup_i R_i \mathbin{Ⓗ} S_i^{⊛}$$

Proof. Denote by $\mathcal{F}$ the family of relations of the form given in the proposition. By definition, $\mathcal{F}$ is included in $\textsc{Had}(\Sigma^* \times \Delta^*)$. Since any rational relation R is equal to $R \mathbin{Ⓗ} \varnothing^{⊛}$, we have $\textsc{Rat}(\Sigma^* \times \Delta^*) \subseteq \mathcal{F}$. Thus it suffices to prove that $\mathcal{F}$ is closed under Hadamard operations and union.

The closure under union is trivially obtained from the definition of $\mathcal{F}$. Let $T = \bigcup_{i \in I} R_i \mathbin{Ⓗ} S_i^{⊛}$ and $T' = \bigcup_{j \in J} R'_j \mathbin{Ⓗ} S'^{⊛}_j$ be in $\mathcal{F}$, and let u be a word in Σ^*. We consider the image of u by the Hadamard product $T \mathbin{Ⓗ} T'$:

$$\begin{aligned}
(T \mathbin{Ⓗ} T')(u) &= \left(\bigcup_{i \in I} R_i \mathbin{Ⓗ} S_i^{⊛}\right)(u) \cdot \left(\bigcup_{j \in J} R'_j \mathbin{Ⓗ} S'^{⊛}_j\right)(u) \\
&= \left(\bigcup_i R_i(u) \cdot S_i(u)^*\right) \cdot \left(\bigcup_j R'_j(u) \cdot S'_j(u)^*\right) \\
&= \bigcup_{i,j} R_i(u) \cdot S_i(u)^* \cdot R'_j(u) \cdot S'_j(u)^* \\
&= \bigcup_{i,j} \big(R_i(u) \cdot R'_j(u)\big) \cdot \big(S_i(u)^* \cdot S'_j(u)^*\big) && \text{— by commutativity} \\
&= \bigcup_{i,j} \big(R_i(u) \cdot R'_j(u)\big) \cdot \big(S_i(u) \cup S'_j(u)\big)^* && \text{— by commutativity} \\
&= \bigcup_{i,j} \big(R_i \mathbin{Ⓗ} R'_j\big)(u) \cdot \big(S_i \cup S'_j\big)^{⊛}(u) \\
&= \left(\bigcup_{i,j} \big(R_i \mathbin{Ⓗ} R'_j\big) \mathbin{Ⓗ} \big(S_i \cup S'_j\big)^{⊛}\right)(u)
\end{aligned}$$

By Theorem 14, each $R_i \mathbin{Ⓗ} R'_j$ is rational and by definition, each $S_i \cup S'_j$ is also rational. Hence, $T \mathbin{Ⓗ} T'$ belongs to $\mathcal{F}$.

We consider now the Hadamard star of T. We claims:

$$T^{⊛} = \left(\bigcup_{i \in I} R_i \mathbin{Ⓗ} S_i^{⊛}\right)^{⊛} = \bigcup_{X \subseteq I} \left(\bigmathbin{Ⓗ}_{i \in X} R_i\right) \mathbin{Ⓗ} \left(\bigcup_{i \in X} R_i \cup S_i\right)^{⊛} \tag{5.1}$$

Observe that there are finitely many $X \subseteq I$, and that for each such X, both relations $\bigmathbin{Ⓗ}_{i \in X} R_i$ (remember Theorem 14) and $\bigcup_{i \in X} R_i \cup S_i$ are rational. This implies that $T^{⊛}$ belongs to $\mathcal{F}$.

We now prove the equality (5.1). Let (u,v) be in $T^{\circledast}$, *i.e.*, $v \in \left(\bigcup_{i\in I} R_i \mathbin{Ⓗ} S_i^{\circledast}\right)(u)^*$. Thus, $v = v_0 v_1 \cdots v_n$ for some $n \in \mathbb{N}$ such that each v_k belongs to $\bigcup_{i\in I} R_i(u)\cdot S_i(u)^*$. For each k we fix an index $i_k \in I$ such that $v_k \in R_{i_k}(u)\cdot S_{i_k}(u)^*$, and we denote by X the set $\{i_k \mid 0 \le k \le n\}$. Hence, v belongs to $R_{i_0}\cdot S_{i_0}^* \cdots R_{i_n}\cdot S_{i_n}^*$. By commutativity, v belongs to $\left(\prod_{i\in X} R_i(u)\right)\cdot\left(\bigcup_{i\in X} R_i(u)\cup S_i(u)\right)^*$ which is equal to $\left(\left(\mathop{Ⓗ}_{i\in X} R_i\right)\mathbin{Ⓗ}\left(\bigcup_{i\in X} R_i\cup S_i\right)^{\circledast}\right)(u)$.

Conversely, let (u,v) belong to $\left(\mathop{Ⓗ}_{i\in X} R_i\right)\mathbin{Ⓗ}\left(\bigcup_{i\in X} R_i\cup S_i\right)^{\circledast}$ for some $X \subseteq I$. For some x and y, we have $v = xy$ where x belongs to $\prod_{i\in X} R_i(u)$ and y belongs to $\left(\bigcup_{i\in X} R_i(u)\cup S_i(u)\right)^*$. Using commutativity, we may decompose y into $y_0\cdots y_q y_{q+1}\cdots y_{q+p}$, where for each $0 \le h \le q$ (*resp.* each $q < h \le q+p$), the word y_h belongs to R_{i_h} (*resp.* S_{i_h}) for some $i_h \in X$. For each $i \in X$ we define w_i as the concatenation of each y_h, with $q < h \le q+p$, such that $i_h = i$, *i.e.*,

$$w_i = \prod_{q<h\le q+p \;\mid\; i_h = i} y_h$$

In particular, if for no $q < h \le q+p$ the equality $i_h = i$ holds, then w_i is equal to ϵ. We now decompose the word x into $\prod_{i\in X} x_i$ with each $x_i \in R_i$. Finally, we use commutativity to obtain $v = \prod_{i\in X}(x_i\cdot w_i)\cdot\prod_{0\le h\le q} y_h$. Each $x_i\cdot w_i$ belongs to $\left(R_i \mathbin{Ⓗ} S_i^{\circledast}\right)(u)$ and each y_h belongs to R_{i_h} and therefore to $\left(R_{i_h} \mathbin{Ⓗ} S_{i_h}^{\circledast}\right)(u)$. Hence, v belongs to $T^{\circledast}(u) = \left(\bigcup_{i\in I} R_i \mathbin{Ⓗ} S_i^{\circledast}\right)^{\circledast}(u)$. □

5.2.2 Main result

From now on we concentrate on unary 2-way transducers, *i.e.*, on those with input and output alphabets reduced to the singleton $\{a\}$. We fix a transducer $(\mathcal{A},\phi)$, with $\mathcal{A} = (Q,\{a\},\rhd,\lhd,I,F,\delta)$.

The following is a reformulation of Theorem 15 in terms of series (remember that binary relations in $a^*\times a^*$ are identified with formal series in $2^{a^*}\langle\langle a^*\rangle\rangle$ as observed by the convention of Section 4.3).

Theorem 16. *Let $\mathbb{K}$ denote the semiring* $\textsc{Rat}(a^*)$. *A series $s \in 2^{a^*}\langle\langle a^*\rangle\rangle$ is accepted by some 2-way finite transducer if and only if $s \in \textsc{Had}(\mathbb{K}\langle\langle a^*\rangle\rangle)$,* i.e., *there exist a finite*

collection of rational series $\alpha_i, \beta_i \in \mathrm{RAT}(\mathbb{K}\langle\langle a^* \rangle\rangle)$ *such that:*

$$s = \sum_i \alpha_i \mathbin{Ⓗ} \beta_i^{⊛}$$

The fact that the condition is sufficient has already be proved in Corollary 4. The other direction is more involved. We proceed as follows. We first show that if the transducer performs a unique hit, *i.e.*, it never visits endmarkers except at the beginning and at the end of the computation, it defines a rational, and therefore Hadamard, relation. Then, putting each such relation in a $(Q \times \{\rhd, \lhd\}) \times (Q \times \{\rhd, \lhd\})$-matrix as done in Corollary 3, we conclude by Proposition 2.

5.2.3 One-way simulation of hits

We fix two border points b_0 and b_1 (see Section 2.3.2) and we consider the set of pairs (u, v) such that v is produced by some b_0 to b_1 hit on u. The idea of the proof can be explained as follows where some technicalities are ignored.

We adapt Lemma 1 by saying that a b_0 to b_1 hit over an input u is of the form $c_0\lambda(c_1)\cdots\lambda(c_{\ell-1})c_\ell$ where $c_0c_1\cdots c_\ell$ is a loop-free b_0 to b_1 hit and $\lambda(c_i)$ is a central c_i-loop for $i = 1, \ldots, \ell-1$. Then the set produced on all possible b_0 to b_1 hits on u is the union over all loop-free b_0 to b_1 hits $c_0c_1\cdots c_\ell$ of the following subset (recall the notation of Definition 12):

$$\Phi(c_0c_1) \cdot \Phi(\lambda(c_1)) \cdot \Phi(c_1c_2) \cdots \Phi(c_{\ell-2}c_{\ell-1}) \cdot \Phi(\lambda(c_{\ell-1})) \cdot \Phi(c_{\ell-1}c_\ell) \tag{5.2}$$

Since the output alphabet is unary, the above terms commute and may be rewritten in as many products as there are positions in u. For each position $0 \le p \le n+1$ we group (1) the $\Phi(\lambda(c_i))$ around all the configurations in position p and (2) all $\Phi(c_ic_{i+1})$ involving a transition occurring at position p . The former product leads us to investigate all outputs of central loops and the latter product leads us to adapt the notion of crossing sequences for loop free runs.

Loop-free hits

We adapt the traditional notion of crossing sequences, e.g., [38, pages 36-42], to our purpose.

Fix a position $0 \le p \le n$ on the input u. The *crossing sequence* of a run at position p is the record, in the chronological order, of all the destination states in the transition performed between positions p and $p+1$, *i.e.*, the states at position $p+1$ in a left to right move and the states at position p in a right to left move, see Figure 5.1. The following is general and does not assume any condition on the run:

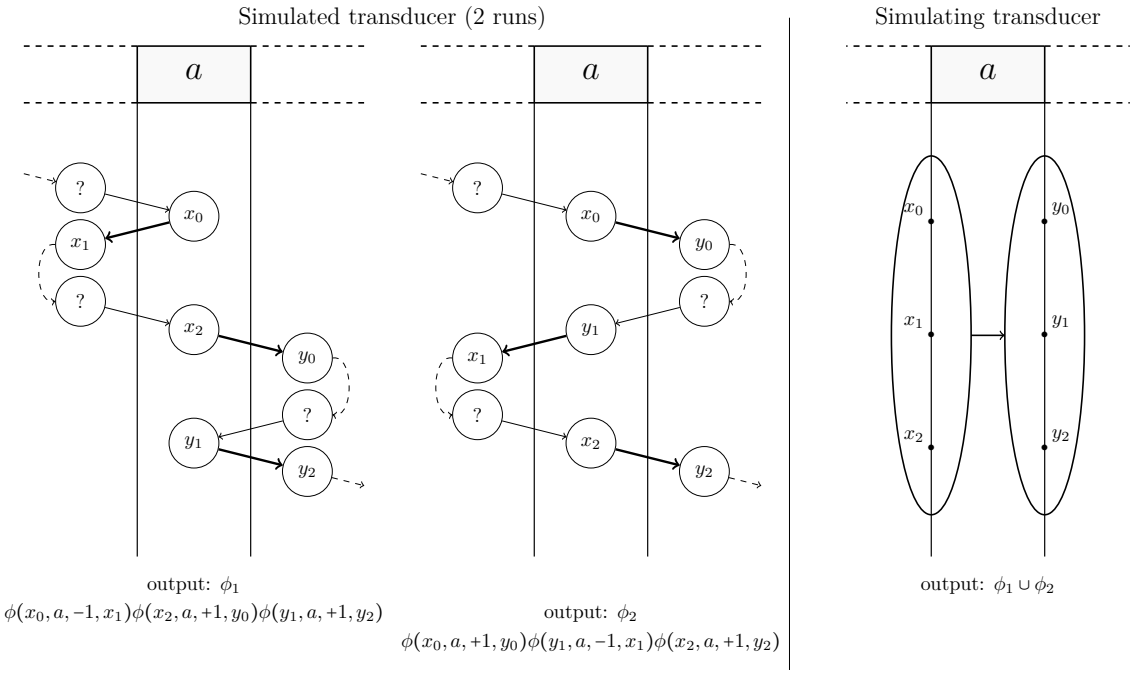


FIGURE 5.1: Two different runs producing the same crossing sequence

Definition 18. *Let* $\mathbf{r} = ((q_i, p_i))_{0 \le i \le \ell}$ *be a run over some input word* u *of length* n *and let* $0 \le p \le n$ *be a position on* u*. The* crossing sequence *of* $\mathbf{r}$ *at* p*, denoted* $\mathrm{X}_{\mathbf{r}}(p)$*, is the ordered state sequence extracted from the sequence* $q_0, \ldots, q_\ell$ *as follows: for each* $1 \le i \le \ell$*,* q_i *is selected if and only if:*

$$\left(\; p_{i-1} = p \quad \textit{and} \quad p_i = p + 1 \; \right) \quad \textit{or} \quad \left(\; p_{i-1} = p + 1 \quad \textit{and} \quad p_i = p \; \right)$$

In the run $\mathbf{r}$ there exist two types of states, those which are entered from the left and those which are entered from the right. These two types alternate. Which type occurs first depends on whether the initial border is left or right. In particular, if $\mathbf{r}$ is a successful run, then for each position p, the first and last state of $\mathrm{X}_{\mathbf{r}}(p)$ correspond to left to right moves. If $\mathbf{r}$ is loop-free, then for all crossing sequences $\mathbf{q} = q_0, q_1, \ldots$ and for all integers $0 \le i < j$ of the same parity we have $q_i \ne q_j$.

The following technical result implies that the set of all pairs $(u, v) \in \Sigma^* \times \Delta^*$ such that v is the output of a b_0 to b_1 loop-free hit on u is a rational relation. It works because the output alphabet is unary[1].

[1] In fact, the construction works for any $\mathbb{K}$-FA with $\mathbb{K}$ commutative.

Lemma 9. *Given a restless transducer $\mathcal{T} = (\mathcal{A}, \phi)$ and two border points b_0 and b_1 of $\mathcal{A}$, there exists a computable 1-way transducer $\mathcal{T}' = (\mathcal{A}', \phi')$ satisfying the following condition:*

Let $\mathbf{r} = c_0 \cdots c_\ell$ be a b_0 and b_1 loop free hit on u in $\mathcal{A}$, let $v \in \Phi_{\mathcal{T}}(\mathbf{r})$ and let $\mathrm{X}_{\mathbf{r}}(0), \mathrm{X}_{\mathbf{r}}(1) \ldots, \mathrm{X}_{\mathbf{r}}(|u|)$ be the associated crossing sequences. Then, the sequence $\mathbf{r}' = \mathrm{X}_{\mathbf{r}}(0) \cdot \mathrm{X}_{\mathbf{r}}(1) \cdots \mathrm{X}_{\mathbf{r}}(|u|)$ is a successful run[2] in $\mathcal{T}'$ with $v \in \Phi_{\mathcal{T}'}(\mathbf{r}')$.

Conversely, if $\mathbf{r}' = \mathbf{r}'_{\mathbf{0}} \cdots \mathbf{r}'_{|\mathbf{u}|} \cdot \mathbf{r}'_{|\mathbf{u}|+\mathbf{1}}$ is a successful run on u in $\mathcal{T}'$ with $v \in \Phi_{\mathcal{T}'}(\mathbf{r}')$, then there exists a b_0 and b_1 loop free hit $\mathbf{r}$ on u in $\mathcal{T}$ with crossing sequences $\mathbf{r}'_{\mathbf{0}} \cdots \mathbf{r}'_{|\mathbf{u}|}$ such that $v \in \Phi_{\mathcal{T}}(\mathbf{r})$.

Proof. Without loss of generality we assume by Proposition 7 that we are given a 2-way unary restless transducer $\mathcal{T}$. We fix two border points b_0 and b_1. For clarity, we consider left to right hits *i.e.*, we suppose that b_0 and b_1 are respectively a left and a right border point. The other three cases can be handled similarly.

In order to define the 1-way transducer simulating all loop-free runs of $\mathcal{T}$, we adapt the classical technique of crossing sequences used for proving the equivalence between 1-way and 2-way finite automata. We recall our convention: the states recorded between position p and $p+1$ are the state reached by the automaton in the destination position, *i.e.*, in position $p+1$ in a left to right move and in position p in a right to left move (in Figure 5.2, the crossing sequence at position labelled by b begins with y_1, y_2, y_3, y_4). Because the run is loop-free all crossing sequences $\mathbf{x} = x_0, x_1, \ldots, x_{2k}$ satisfy the condition

$$\text{for all integers of the same parity } 0 \le i \le j : \; x_i = x_j \;\text{ implies } i = j \tag{5.3}$$

Sequences satisfying the condition are called *valid*. There clearly is a finite number of such sequences. Thus they can be stored in the finite control of our simulating transducer.

Given two valid sequences $\mathbf{x}$ and $\mathbf{y}$ and a letter a, we investigate under which condition the tuple $(\mathbf{x}, a, +1, \mathbf{y})$ is a transition of the 1-way transducer $\mathcal{T}'$ simulating left to right loop-free hits of $\mathcal{T}$ and we determine its value in the production function of $\mathcal{T}'$. Because simulated runs are loop-free, the automaton may not enter a state twice at the same position. Since this state may be entered when coming from the right or from the left, for all i, j we require $x_{2i} \neq y_{2j+1}$.

We do as in the case of the 1-way simulation of a 2-way automaton: the tuple $(\mathbf{x}, a, +1, \mathbf{y})$ is a transition of $\mathcal{T}'$ if there is a left to right hit of $\mathcal{T}$ over some input, having crossing sequence $\mathbf{x}$ at the left border of cell labelled by the letter a and $\mathbf{y}$ at its

[2] Actually, this is not really a run as defined in Section 2.3.2, but since our simulating transducer is restless and 1-way, we omit the position component which is equal to the index of the configuration in the run. We also omit the accepting configuration at position $|u|+1$ which can easily be added. Indeed, we consider a kind of transducer $(\mathcal{A}', \phi')$ where $\mathcal{A}$ is a cFA (see Definition 6).

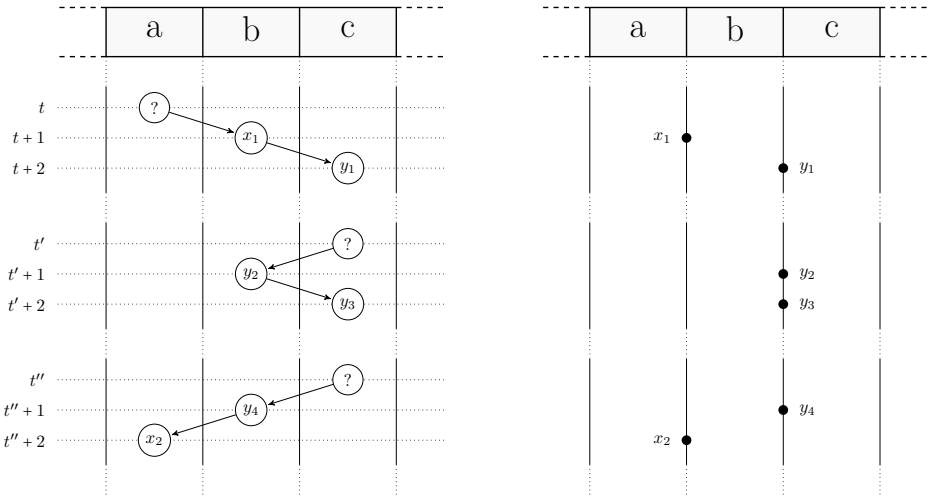


FIGURE 5.2: The crossing sequence at position labelled by b begins with y_1, y_2, y_3, y_4.

right border, see Figure 5.2 (this is defined formally exactly as in the case of automata and we omit the details).

In order to define the image of such a transition by the production function ϕ' of $\mathcal{T}'$, we proceed as follows. We must distinguish the runs having $\mathbf{x}$ and $\mathbf{y}$ as consecutive crossing sequences. We group the first two states in the run, then the next two states etc. As an example, Figure 5.1 represents two different runs defining the same consecutive crossing sequences $\mathbf{x} = x_0x_1x_2$ and $\mathbf{y} = y_0y_1y_2$. In the left run we group $(x_0, x_1), (x_2, y_0), (y_1, y_2)$ and in the right run we group $(x_0, y_0), (y_1, x_1), (x_2, y_2)$. We call *matching* such a sequence of pairs of states belonging to either crossing sequences. Observe that a pair of states in a matching satisfies one of the following predicates

- $A(x_{2i}, x_{2i+1})$ if x_{2i}, x_{2i+1} is a U-turn of the run changing direction from right to left (e.g., (x_0, x_1) in the left run of Figure 5.1).
- $B(y_{2j-1}, y_{2j})$ if y_{2j-1}, y_{2j} is a U-turn of the run changing direction from left to right (e.g., (y_1, y_2) in the right run of Figure 5.1).
- $C(x_{2i}, y_{2j})$ if x_{2i}, y_{2j} is a progression of the run to the right (e.g., (x_0, y_0) to the left of Figure 5.1).
- $D(y_{2j-1}, x_{2i-1})$ if y_{2j-1}, x_{2i-1} is a progression of the run to the left (e.g., (y_1, x_1) to the right of Figure 5.1).

We are now in a position to define the production function ϕ' of $\mathcal{T}'$. Consider first a fixed matching $(s_1, s'_1), \ldots, (s_h, s'_h)$ of $\mathbf{x}$ and $\mathbf{y}$ set

$$\Big(\prod_{A(s_i,s'_i)\vee D(s_i,s'_i)} \phi(s_i, a, -1, s'_i)\Big)\Big(\prod_{B(s_i,s'_i)\vee C(s_i,s'_i)} \phi(s_i, a, 1, s'_i)\Big)$$

Then we set $\phi'(\mathbf{x}, a, +1, \mathbf{y})$ as the union of the above value for all matchings of $\mathbf{x}$ and $\mathbf{y}$.

In order to justify this construction, we fix once and for all a sequence of crossing sequences $\mathbf{x_0}, \ldots, \mathbf{x_n}$ and compute

$$\{\Phi(\mathbf{r}) \mid \mathbf{r} \text{ is a } b_0 \text{ to } b_1 \text{ hit} \mid \mathrm{X}_\mathbf{r}(0) = \mathbf{x_0}, \ldots, \mathrm{X}_\mathbf{r}(n) = \mathbf{x_n}\}$$

We must take into account all possible transitions of all runs once and only once. To that purpose we assign each transition to the cell to which its source belongs. In Figure 5.1 these are in the two different cases the transitions with no question mark. Furthermore, instead of considering over all runs the product of all images by Φ of its transitions and taking the union, we proceed differently by inverting union and product: we consider for each position the union of the images of all transitions assigned to that position and then take their product. In the example of the figure, we assume that there exist two runs at the position in question. Then the contribution of this position to the image by Φ is the subset consisting of the two elements

$$\phi(q_0, a, -1, q_1)\phi(q_2, a, +1, r_0)\phi(r_1, a, +1, r_2)$$

and

$$\phi(q_0, a, +1, r_0)\phi(r_1, a, -1, q_1)\phi(q_2, a, +1, r_2)$$

□

Computing the outputs of central loops.

As said previously we now turn to the investigation of central loops. We show that there are only finitely many different sets of words produced by central loops. Intuitively for an initial configuration (q, p) the set of outputs does not depend on p provided it is sufficiently far away from each endmarker.

Given a central loop $\mathbf{r} = (q_0, p_0), \ldots, (q_k, p_k)$, and two elements ℓ and r in $\mathbb{N} \cup \{\infty\}$, we say that $\mathbf{r}$ is (ℓ, r)-*limited* if for all $0 \le i \le k$ we have $p_0 - \ell \le p_i \le p_0 + r$. Observe that if $\ell' \ge \ell$ and $r' \ge r$, every (ℓ, r)-limited loop is (ℓ', r')-limited. Any central loop is (∞, ∞)-limited.

We denote by $\mathcal{O}^{(q)}_{\ell,r}$ the union of all $\Phi(\mathbf{r})$ where $\mathbf{r}$ is a (ℓ, r)-limited central q-loop (observe that it does not depend on the initial position since no endmarker is visited). In

particular, for each $\ell' \geq \ell$ and each $r' \geq r$, $\mathcal{O}^{(q)}_{\ell,r}$ is included in $\mathcal{O}^{(q)}_{\ell',r'}$. The language $\mathcal{O}^{(q)}_{\infty,\infty}$ is the set of all outputs of central q-loops. For any ℓ, r and q, $\mathcal{O}^{(q)}_{\ell,r}$ contains in particular the empty word ϵ, since for any central configuration (q,p), the run reduced to (q,p) is a trivial central-loop. The following shows that each language $\mathcal{O}^{(q)}_{\ell,r}$ is rational via Parikh's Theorem [60] and that there exist finitely many different such languages:

Lemma 10. *Let $(\mathcal{A},\phi)$ be a transducer. For any ℓ and r in $\mathbb{N}\cup\{\infty\}$, and any state q of $\mathcal{A}$ there exists a computable 1-way automaton accepting the language $\mathcal{O}^{(q)}_{\ell,r}$. It follows that the language is rational.*

There exists a computable N, such that for each $r \geq N$, each $\ell \geq N$ and each state q, $\mathcal{O}^{(q)}_{\ell,r} = \mathcal{O}^{(q)}_{N,N} = \mathcal{O}^{(q)}_{\infty,\infty}$.

Proof. Without loss of generality, by Proposition 6 and Proposition 7, we can suppose that the simulated automaton is restless and single-valued. For ℓ and r finite it suffices to construct a finite automaton recognizing $\mathcal{O}^{(q)}_{\ell,r}$. The automaton has $Q\times[-\ell,r]$ as set of states, $(q,0)$ as initial and final states. The transitions are more general than the transitions for ordinary automata since we allow triples in $Q\times[-\ell,r]\times a^*\times Q\times[-\ell,r]$ but it should be clear that this can be transformed into an ordinary automaton reading a unique letter at the price of some technicalities. The transitions are of the form

$$((q,i),a^k,(p,i+d)) \text{ if } \begin{cases} (q,a,d,p)\in\delta \\ -\ell\leq i+d\leq r \\ \text{ and } \phi(q,a,d,p)=a^k \end{cases}$$

Now consider the case $\mathcal{O}^{(q)}_{\ell,\infty}$ (the remaining cases $\mathcal{O}^{(q)}_{\infty,r}$ and $\mathcal{O}^{(q)}_{\infty,\infty}$ can be treated similarly). We describe informally a nondeterministic one-counter pushdown automaton that accepts all the possible outputs a^n of a central q-loop when the loop is prohibited the go further than ℓ cells to the left of the initial position. The pushdown automaton accepts by empty stack and accepting state. Before starting reading a^n the counter is set to ℓ. Then the automaton interprets each transition $(q,a,d,p)\in\delta$ and its image $\phi(q,a,d,p)=a^k$ as follows: it reads the next k occurrences of a^n, changes state accordingly and increments the counter if $d=1$ or decrements it if $d=-1$. When it finishes reading the input, it checks that the counter equals ℓ.

Since $\mathcal{O}^{(q)}_{\ell,\infty}$ is a language L over a one-letter alphabet, by Parikh Theorem [60] it is recognized by some finite automaton. Furthermore L is a submonoid of a^* since loops are composable thus it is finitely generated. The minimum set generators is defined as $(L\setminus\{\epsilon\})\setminus(L\setminus\{\epsilon\})^2$ and is computable. Each generator is produced by a central q-loop on some large enough input word. Hence, the least integer N such that each generator belongs to $\mathcal{O}^{(q)}_{\ell,N}$ exists and we can compute it by brute-force thanks to the automata

accepting each $\mathcal{O}_{\ell,k}^{(q)}$, defined at the beginning of the proof. Then for all integers $r \geq N$ and all states q' we have $\mathcal{O}_{\ell,N}^{(q')} = \mathcal{O}_{\ell,\infty}^{(q')}$ □

Putting things together.

We are now able to simulate all hits:

Proposition 33. *Given two border points b_0 and b_1, we can compute a 1-way transducer accepting the set of pairs (u, v) such that v is produced by a b_0 to b_1 hit on u. The relation accepted is thus rational.*

Proof. Let $\mathcal{T} = (\mathcal{A}, \phi)$ be a transducer that we suppose thanks to Proposition 7 restless. We denote by $N+1$ the integer computed from Lemma 10.

We define the following two 1-way restless automata which share the same state set $Q_B = \{0, 1, \ldots, N, \infty\}$. The first one $\mathcal{B}_L = (Q_B, 0, Q_B, \delta_L)$ counts the distance to the left endmarker up to N. The second one $\mathcal{B}_R = (Q_B, \infty, \{0\}, \delta_R)$ guesses whether the distance to the right endmarker is greater than N or is equal to some integer less than or equal to N and checks this guess by counting down until reaching the right endmarker. The transition sets δ_L and δ_R are defined as follows with the convention $N+1 = \infty+1 = \infty$ and $\infty - 1 = \infty$

- $\delta_L = \{(q, s, +1, q+1)\}$ where $s \in \Sigma_{\triangleright\triangleleft}$
- $\delta_R = \{(\infty, \triangleright, +1, i) \mid i \in Q_B\} \cup \{(\infty, a, +1, N)\} \cup \{(q, a, +1, q-1) \mid q \neq 0\}$

We build a 1-way restless transducer $(\mathcal{A}', \phi')$ from $\mathcal{T}$ as in Lemma 9. Because $\mathcal{A}'$, $\mathcal{B}_L$ and $\mathcal{B}_R$ are 1-way restless, we can take the product automaton $\mathcal{A}'' = \mathcal{A}' \times \mathcal{B}_L \times \mathcal{B}_R$. Let us now consider a transition $t = ((\mathbf{q}, \ell, r), a, +1, (\mathbf{q}', \ell', r'))$ of $\mathcal{A}''$ performed at position p. The states appearing in $\mathbf{q}$ (*resp.* $\mathbf{q}'$) correspond to states entered by $\mathcal{T}$ in position $p-1$ or p (*resp.* p or $p+1$). In any case, this difference can be determined from the initial border point and the parity of the index of the state in the crossing sequence. In particular, we may extract from both crossing sequences $\mathbf{q}$ and $\mathbf{q}'$ the set S_t of states that are entered by $\mathcal{T}$ in position p. Hence, we can add for each such state q, the possible output of (ℓ, r)-limited central q-loops. Formally, the image of t by ϕ'' is defined by:

$$\phi''(t) = \phi'(t)\left(\bigcup_{q \in S_t} \mathcal{O}_{\ell,r}^{(q)}\right)$$

By construction, the 1-way restless transducer $(\mathcal{A}'', \phi'')$ simulates any b_0 to b_1 hit of $\mathcal{T}$. Hence, the relation of pairs (u, v) such that v is produced by some b_0 to b_1 hit of $\mathcal{T}$ on u is rational. □

5.2.4 Unlimited number hits

We prove Theorem 16 by considering an unlimited number of hits. We show that the series associated with the relation defined by a 2-way transducer can be expressed via a transitive closure of a square matrix with entries in $\textsc{Had}(\mathbb{K}\langle\langle\Sigma^*\rangle\rangle)$. More precisely, by Lemma 2, if a run $\mathbf{r}$ is successful, then there exists a sequence of composable hits $\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_k$ such that $\mathbf{r} = \mathbf{r}_0 @ \mathbf{r}_1 @ \ldots @ \mathbf{r}_k$.

We first adapt the matrix multiplication to the Hadamard product. Given an integer N and two matrices $X, Y \in (\mathbb{K}\langle\langle\Sigma^*\rangle\rangle)^{N\times N}$ we define their H-product as the matrix $Z = X \,Ⓗ\, Y \in (\mathbb{K}\langle\langle\Sigma^*\rangle\rangle)^{N\times N}$ where for each $1 \le i, j \le N$

$$Z_{i,j} = \sum_{k=1}^{N} X_{i,k} \,Ⓗ\, Y_{k,j}$$

Also, the H-star of the matrix X is defined as the infinite sum

$$(X)^{⊛} = \sum_{k=0}^{\infty} \overbrace{X \,Ⓗ \cdots Ⓗ\, X}^{k \text{ times}}$$

We proceed as in the proof of Corollary 3. Indeed, by applying Theorem 2 to the semiring $\textsc{Had}(\Sigma^* \langle\langle\Sigma^*\rangle\rangle)$ we obtain that the entries of the star of a matrix in $\textsc{Had}(\Sigma^* \langle\langle\Sigma^*\rangle\rangle)^{N\times N}$ are well-defined and belong $\textsc{Had}(\Sigma^* \langle\langle\Sigma^*\rangle\rangle)$.

Corollary 18. *If the matrix X is in $(\textsc{Had}(\mathbb{K}\langle\langle\Sigma^*\rangle\rangle))^{N\times N}$ then so is $(X)^{⊛}$.*

We are now able to complete the proof of Theorem 16.

Proof of Theorem 16. In one direction this is an immediate consequence Corollary 4, since the input alphabet is unary.

It remains to prove the converse. Let $\mathcal{T}$ be transducer. Consider a matrix X whose rows and columns are indexed by the pairs $Q \times \{\triangleright, \triangleleft\}$ of border points. For all pairs of border points b_0 and b_1, its (b_0, b_1) entry is, by Proposition 33, the rational series associated to b_0 to b_1 hits. The series accepted by $\mathcal{T}$ is the sum of the entries of $X^{⊛}$ in positions $((q_-, \triangleright), (q, \triangleleft))$ for $q \in Q_+$. Since all rational series are also Hadamard series, we conclude by Proposition 18. □

5.2.5 Conclusion

Our main result of Theorem 15 gives a characterization of relations (series) accepted by 2-way unary transducers, *i.e.*, general transductions (see Section 2.3.4), when both

alphabets are unary. A key point is that crossing sequences of loop-free runs have bounded size. Consequently, any loop-free run can be simulated by a 1-way transducer as done in Lemma 9. We point out that this simulation does not require any hypothesis on the size of the input alphabet.

We fix a transducer $\mathcal{T} = (\mathcal{A}, \phi)$ accepting a relation $R \subseteq \Sigma^* \times \Delta^*$, with $|\Delta| = 1$. If $\mathcal{A}$ is deterministic or *unambiguous* (*i.e.*, for each input word u, there exists at most one successful run of $\mathcal{A}$ on u), then every successful run is loop-free. Therefore, by Lemma 9, $\mathcal{T}$ is equivalent to some constructible 1-way transducer. Another interesting case is when R is a function. Then for each u, all the successful runs on u produce the same output word. Hence, considering only loop-free runs preserves the acceptance of $\mathcal{T}$. We thus recover the results of Anselmo [2].

Corollary 19. *Let $R \subseteq \Sigma \times \Delta$ with $|\Delta| = 1$ be accepted by some 2-way transducer $\mathcal{T} = (\mathcal{A}, \phi)$. If $\mathcal{A}$ is unambiguous or if R is a function then R is rational. Moreover, a 1-way transducer $\mathcal{T}'$ equivalent to $\mathcal{T}$ can be effectively build.*

A *rational uniformization* of a relation $R \subseteq \Sigma^* \times \Delta^*$, is a rational function $F \subseteq R$, such that the domain of F is equal to the one of R. Under the hypothesis $|\Delta| = 1$, it is possible to build, from Lemma 9, a 1-way transducer accepting such a F. Since the transducer obtained from Lemma 9 is not necessarily functional, the construction involves a result of Eilenberg [21, Prop. IX 8. 2] solving the rational uniformization problem for rational relation.

Corollary 20. *There exists a computable 1-way transducer accepting a rational uniformization of R.*

Finally, observe that since outer-nondeterministic transducers (see Section 2.3.4) cannot have a successful run admitting a central loop as factor, Lemma 9 can be applied on every useful hits, and thus the same conclusion as that of Theorem 15 holds for outer-nondeterministic transducers, provided the output alphabet is unary.

Corollary 21. *Let $R \subseteq \Sigma \times \Delta$ with $|\Delta| = 1$ be accepted by some 2-way outer-nondeterministic transducer. Then R belongs to* Had.

5.3 Sweeping weakens two-way transducers

5.3.1 Revisiting the family Rat(a^*)

Taking advantage of the observation that $(a^*, \cdot, \epsilon)$ is isomorphic with the additive monoid $(\mathbb{N}, +, 0)$ in the mapping $n \mapsto a^n$, we prefer for notational reasons to work in $\mathbb{N}$. With this

identification we may speak of the subset of $\mathbb{N}$ accepted by an automaton over a unary alphabet. From now on instead of working in $\Sigma^* \times a^*$ we will work in the equivalent structure $\Sigma^* \times \mathbb{N}$. All the terminology on the former structure carries over to the latter. Beware that the concatenation in a^* converts to the addition in $\mathbb{N}$. In particular, the set product on $\mathbb{N}$ is denoted $X + Y$. Its neutral element is the singleton $\{0\}$ (or simply denoted 0) and $\varnothing$ is an absorbing element.

First, we introduce some notations. Speaking of the singleton $\{n\}$, we use the abusive but convenient notation n when the context is clear. Hence, for a subset $X \subseteq \mathbb{N}$, we write $n + X$ for $\{n\} + X$. The multiplication of subset X by a scalar $p \in \mathbb{N}$ *i.e.*, the set $\{px \mid x \in X\}$, is denoted pX. In particular, $p\mathbb{N}$ is the set of multiples of p. We say that a subset X of $\mathbb{N}$ is *bounded by* k, if $x \in X$ implies $x < k$ *i.e.*, if $X \subseteq \{0, \ldots, k-1\}$.

Rational subsets of $\mathbb{N}$

The proof of the following simple result that characterizes rational sets, is left to the reader. This is basically due to the famous characterization of rational subsets of $\mathbb{N}$ as semilinear sets *i.e.*, finite unions of linear sets.

Proposition 34. *A subset X of $\mathbb{N}$ is rational if and only if there exist two integers t and p and two finite sets A and M respectively bounded by t and p such that $X = A \cup (t + M + p\mathbb{N})$.*

If $X = A \cup (t + M + p\mathbb{N})$, for two integers t and p and two subsets A and M respectively bounded by t and p, we say that $A \cup (t + M + p\mathbb{N})$ is a *rat-expression* for X. The integers t and p are respectively *the threshold* and *the period of the rat-expression* or simply *a threshold* and *a period* for X, when the rat-expression is not made precise. It is possible to choose t and p minimal. In this case t and p are called *the threshold* and *the period* of the rational set X. Observe that if $A \cup (t + M + p\mathbb{N})$ is a rat-expression of a finite set X, then $p = 0$ and so $M = \varnothing$; thus $X = A$. Conversely, if X is infinite, then $p > 0$ and $M \neq \varnothing$.

Equivalent rat-expressions of rational sets

The same rational subset is definable by different rat-expressions with different thresholds and periods. We show how these parameters can be modified.

Lemma 11. *Let X be a rational set and let t and p be the threshold and period of some rat-expression of X. Then, for any $u \geq t$, there exists a computable rat-expression of X with threshold u and period p.*

Proof. Let $A \cup (t + M + p\mathbb{N})$ be a rat-expression of a rational set X and let u be greater than or equal to t. Since $u \geq t$, for some $k \geq 0$ and $0 \leq s < p$ we have $u = t + s + kp$. Define

a subset M' of $\{0,\dots,p-1\}$ as follow: $j \in \{0,\dots,p-1\}$ belongs to M' if and only if $j+s$ belongs to M or to $p+M$ *i.e.*,

$$M' = \left\{ \begin{array}{lll} i-s & | \, i \in M & \text{and} \quad i \geq s \\ p+i-s & | \, i \in M & \text{and} \quad i < s \end{array} \right\}$$

and define A' as follows:

$$A' = (t + M + p\{0,\dots,k-1\}) \cup (t + \{i \in M \mid i < s\} + kp)$$

Observe that: $X \cap \{0,\dots,t-1\} = A$; $X \cap \{t,\dots,u-1\} = A'$ and $X \cap (u+\mathbb{N}) = u+M'+p\mathbb{N}$. Thus, $X = (A \cup A') \cup (u + M' + p\mathbb{N})$. □

Lemma 12. *Let X be a rational set, and let t and p be the threshold and period of some rat-expression of X. Then, for any $r > 0$, there exists a computable rat-expression of X with threshold t and period rp.*

Proof. Let $A \cup (t + M + p\mathbb{N})$ be a rat-expression of a rational set X and let r be a positive integer. Simply set M' to be equal to the set $M + p\{0,\dots,r-1\}$. We prove $X = A \cup (t + M' + rp\mathbb{N})$. It suffices to prove $t + M + p\mathbb{N} = t + M' + rp\mathbb{N}$.

Let $x = t + m + kp$ for some $m \in M$ and some $k \geq 0$. Then, the euclidian division of k by r gives $k = qr + \ell$ with $0 \leq \ell < r$. Hence $x = (t + m + \ell p) + qrp$. By definition of M', $m + \ell p$ belongs to M', and thus $x \in t + M' + p\mathbb{N}$.

Reciprocally, if $x = t + m' + qrp$ for some $m' \in M'$ and $q \geq 0$, then there exist $m \in M$ and $0 \leq \ell < r$ such that $m' = m + \ell p$. Thus, $x = t + m + p(\ell + qr)$ belongs to $t + M + p\mathbb{N}$. This concludes the proof. □

By combining both Lemmas 11 and 12, a unique threshold and period can be chosen to work with every rational sets of a finite family:

Corollary 22. *Let $\mathcal{F}$ be a finite family of rational sets. For each $X \in \mathcal{F}$, let t_X and p_X denote the threshold and the period of some rat-expression of X. Then, there exists for each $X \in \mathcal{F}$ a rat-expression of X with threshold $\max_{X \in \mathcal{F}}(t_X)$ and period $\operatorname{lcm}_{X \in \mathcal{F}}(p_X)$.*

It is then easy to compute the union or the intersection of rational sets. Indeed, if $A_X \cup (t + M_X + p\mathbb{N})$ and $A_Y \cup (t + M_Y + p\mathbb{N})$ are two rat-expressions, then their union is equal to $(A_X \cup A_Y) \cup (t + (M_X \cup M_Y) + p\mathbb{N})$, which is a rat-expression with the same threshold and the same period. This particular case is instrumental to the last part of our proof.

Proposition 35. *Given a finite family $p_1, p_2, \dots, p_n$ of distinct prime integers, the period of the set $\bigcup_{0<i\leq n} p_i\mathbb{N}$ is equal to $\prod_{0<i\leq n} p_i$.*

Proof. Let X denote the set $\bigcup_{0<i\leq n} p_i\mathbb{N}$. By Corollary 22 and previous observation, it is clear that $p = p_1 \times \cdots \times p_n$ is a period for X.

Indeed, the minimal period divides p. If it is not equal to p then for some p_i the integer $\widehat{p_i} = \frac{p_1\times\cdots\times p_n}{p_i}$ is a period. Then $p_i \in X$ implies $p_i + \widehat{p_i} \in X$ i.e., $p_i + \widehat{p_i} = rp_j$ for some $1 \leq j \leq n$ and some $r \in \mathbb{N}$. If $i = j$ then the left handside is divisible by p_i thus $\widehat{p_i}$ is divisible by p_i, a contradiction. Otherwise, p_j divides $\widehat{p_i}$ thus p_i, a contradiction. □

The sum of rational subsets of $\mathbb{N}$

By Kleene Theorem we know that the sum of two rational sets is rational. Here we discuss the value of the threshold and the period of the rat-expression of the sum of two subsets of $\mathbb{N}$. We start by proving intermediate result:

Proposition 36. *Let t and p be a threshold and a period for a rational set X. Let Y be bounded by some $s \in \mathbb{N}$. Then $X + Y$ admits a rat-expression of threshold $t + s$ and period p.*

Proof. For some A bounded by t and some M bounded by p, we have $X = A \cup (t + M + p\mathbb{N})$. Since $Y = \bigcup_{y\in Y} \{y\}$, we have $X + Y = (A + Y) \cup \bigcup_{y\in Y} (y + t + M + p\mathbb{N})$. We fix $y \in Y$. By assumption $t + y < t + s$. By Lemma 11, there exist A_y and M_y respectively bounded by $t + s$ and p such that $y + X = A_y \cup ((t + s) + M_y + p\mathbb{N})$. Finally

$$X + Y = \left(\bigcup_{y\in Y} A_y\right) \cup \left((t + s) + \left(\bigcup_{y\in Y} M_y\right) + p\mathbb{N}\right)$$

□

Lemma 13. *Let t, s and p be three integers, and let J and K be two subsets bounded by p. Then there exist A and M, respectively bounded by $t + s + p$ and p, such that*

$$(t + J + p\mathbb{N}) + (s + K + p\mathbb{N}) = A \cup ((t + s + p) + M + p\mathbb{N})$$

Proof. Observe that $(t + J + p\mathbb{N}) + (s + K + p\mathbb{N})$ is equal to $K + (t + s + J + p\mathbb{N})$. Since K is bounded by p, the result follows directly from Proposition 36. □

We are now able to consider the sum of two general rational sets.

Proposition 37. *Let $A_X \cup (t_X + M_X + p_X\mathbb{N})$ and $A_Y \cup (t_Y + M_Y + p_Y\mathbb{N})$ be the respective rat-expressions of two rational sets X and Y. Fix $t = \max(t_X, t_Y)$ and $p = \operatorname{lcm}(p_X, p_Y)$. Then the rational set $X + Y$ admits a rat-expression of threshold $(2t + p)$ and period p.*

Proof. By Corollary 22, we may find A'_X and A'_Y bounded by t and M'_X and M'_Y bounded by p such that: $X = A'_X \cup (t + M'_X + p\mathbb{N})$ and $Y = A'_Y \cup (t + M'_Y + p\mathbb{N})$. By distributivity:

$$X + Y = \bigcup \begin{cases} (A'_X + A'_Y) \\ (A'_X + t + M'_Y + p\mathbb{N}) \\ (A'_Y + t + M'_X + p\mathbb{N}) \\ (2t + M'_X + M'_Y + p\mathbb{N}) \end{cases}$$

We consider each of the four subsets separately:

1. the set $A_0 = A'_X + A'_Y$ is a finite set bounded by $2t$ and so by $2t + p$;
2. the set $A'_X + t + M'_Y + p\mathbb{N}$ can be rewritten, thanks to Proposition 36 and Lemma 11, as $A_1 \cup ((2t + p) + M_1 + p\mathbb{N})$ with $A_1 \subseteq \{0, \ldots, 2t + p - 1\}$ and $M_1 \subseteq \{0, \ldots, p - 1\}$;
3. similarly the set $A'_Y + t + M'_X + p\mathbb{N}$ is rewritten as $A_2 \cup ((2t + p) + M_2 + p\mathbb{N})$;
4. from Lemma 13 follows the existence of the sets A_3 and M_3, respectively bounded by $2t + p$ and p, such that $2t + M'_X + M'_Y + p\mathbb{N} = A_3 \cup (2t + p + M_3 + p\mathbb{N})$.

We pose $A = A_0 \cup A_1 \cup A_2 \cup A_3$ and $M = M_1 \cup M_2 \cup M_3$. We have

$$X + Y = A \cup ((2t + p) + M + p\mathbb{N})$$

□

Star of rational subsets of $\mathbb{N}$

The following lemma gives a characterization of star-generated subsets of integers.

Lemma 14. *Let X be a subset of $\mathbb{N}$. Then, denoting by r the greatest common divisor of the elements of X,* i.e., *$r = \gcd(X)$, there exist an integer $t \in \mathbb{N}$ and a finite set $A \subseteq \{0, \ldots, t - 1\}$ such that:*

$$X^* = r\,(A \cup (t + \mathbb{N}))$$

In particular, X^* is rational and is included in $r\mathbb{N}$.

Proof. Whenever $r = \gcd(X) = 1$, it is known that $X^* = A \cup (t + \mathbb{N})$ for some integer t and some subset A bounded by t, *e.g.*, [64, Theorem 1.0.1]. Observe that $t - 1$ when t is minimal is known as the Frobenius number.

We now extend this result to the general case, where r is arbitrary. We define $X_{/r} = \{x \mid rx \in X\}$. Observe that $X^* = rX_{/r}{}^*$. Hence: $X^* = r\,(A \cup t + \mathbb{N})$. □

Considering the Kleene star of a rational set, it happens that both the threshold and the period have no simple expressions. However, we are able to bound the value of the period which is enough for our purpose.

Lemma 15. *Let $A \cup (t + M + p\mathbb{N})$ be a rat-expression of some non-empty rational set X. Then X^* admits a period less than or equal to* $\max(t, p)$.

Proof. Let r denote the greater common divisor of the elements of X. By Lemma 14, $X^* = r(K \cup (\ell + \mathbb{N}))$ and thus r is a period of X^*.

Now we prove an upper bound on r in the two disjoint cases: finite or infinite. If X is finite, then X is equal to A and is thus bounded by t. Since r divides all the elements of X (supposed non-empty), we have $r < t$. Else, if X is infinite, then $p > 0$ and $M \neq \varnothing$. For any $x \in t + M$, r divides both x and $x + p$, and thus r divides p. So $r \leq p$. This concludes the proof. □

5.3.2 The unary output case

In Section 5.2, we proved that when Σ and Δ are unary, the family of relations in $\Sigma^* \times \Delta^*$ accepted by 2-way transducers is exactly the family of Hadamard relations. The crux of the proof seems to rely on the hypothesis that Δ is unary, since it is strongly required by the characterization of the family $\textsc{Had}(\Sigma^* \times \Delta^*)$ in terms of semi-linear forms (see Proposition 32). We left open the general case where Σ has at least two elements and Δ is unary. Here we show that in this case the family of relations realized by 2-way transducers strictly contains the family of Hadamard relations.

Massaging the productions

In this section we give a kind of normal form for transducers with unary output. Thanks to the identification between unary languages and subsets of $\mathbb{N}$, we may associate to each production function of such transducers, a production function that maps transitions into rational subsets of $\mathbb{N}$.

We show that transducers over Σ and $\mathbb{N}$ admit a simple form:

Lemma 16. *Let $\mathcal{T}$ be a transducer with transition set δ and production function $\phi : \delta \to \textsc{Rat}(\mathbb{N})$. Then there exists an equivalent transducer $\mathcal{T}'$ such that the image of each transition by the production function is of the form $t + p\mathbb{N}$ for some non-negative integers t and p. Moreover, if $\mathcal{T}$ is 1-way or restless or both, so is the resulting transducer.*

Proof. We fix the transducer $\mathcal{T} = (\mathcal{A}, \phi)$. By Proposition 34, for each transition e of $\mathcal{A}$ the language $\phi(e)$ admits a rat-expression $A_e \cup (t_e + M_e + p_e\mathbb{N})$. By decomposing A_e and

M_e as finite union of singletons, we obtain:

$$\phi(e) = \left(\bigcup_{a\in A_e} a + 0\mathbb{N}\right) \cup \left(\bigcup_{m\in M_e} (t_e + m) + p_e\mathbb{N}\right)$$

Hence, by indexing the disjoint union $A_e \uplus M_e$ by $I_e = \{0, \ldots, |A_e| + |M_e| - 1\}$, the set $\phi(e)$ may be written as $\bigcup_{i\in I_e} t_{i,e} + p_{i,e}\mathbb{N}$.

Now we modify the transducer $(\mathcal{A}, \phi)$ into $(\mathcal{A}', \phi')$ in such a way that the transitions distinguish the indices i chosen in I_e. This can easily be done by recording in the finite control of $\mathcal{A}'$ which choice has been done at the last transition. Formally, a state of $\mathcal{A}'$ is a pair (q, i) where q is a state of $\mathcal{A}$ and i is an index in $\bigcup_e I_e$. For each transition $f = (q, a, d, q')$ of $\mathcal{A}$ and each index $i \in \bigcup_e I_e$ there are $|I_f|$ transitions: $((q, i), a, d, (q', j))$ for $j \in I_f$. Finally, the image of a transition $((q, i), a, d, (q', j))$ by ϕ' is defined as $t_{j,f} + p_{j,f}\mathbb{N}$. By construction the resulting transducer is equivalent to $\mathcal{T}$. Observe that the directions are kept. □

Images of 1-way transducers

Let $R \subseteq \Sigma^* \times \Delta^*$ be a rational relation, i.e., a relation realized by a 1-way transducer. For all words $u \in \Sigma^*$ the set $R(u) = \{v \in \Delta^* \mid (u, v) \in R\}$ is a rational subset of Δ^*, [66, Theorem IV.1.3]. Here we show that when Δ is unary the collection of all possible images satisfies a uniform property. We keep identifying Δ^* and $\mathbb{N}$.

Theorem 17. *Let Σ be an arbitrary alphabet. Let R be a rational relation in $\Sigma^* \times \mathbb{N}$. Then, there exist two integers t and p such that, for all $w \in \Sigma^*$, the rational language $R(w)$ admits a rat-expression of threshold $t\,(|w| + 1)$ and period p.*

Proof. By Theorem 6, R is accepted by a 1-way transducer $\mathcal{T} = (\mathcal{A}, \phi)$ which we can suppose restless by Proposition 7. Let w be an input word in Σ^* and let n denote its length. Let $\mathbf{R}$ be the set of all successful runs of $\mathcal{T}$ on w. Observe that since $\mathcal{T}$ is 1-way restless, every run $\mathbf{r}$ in $\mathbf{R}$ has length $n+2$ (there is exactly one configuration per position, including endmarkers). Thus $\mathbf{R}$ is finite. The image of w is:

$$R(w) = \bigcup_{\mathbf{r}\in\mathbf{R}} \Phi(\mathbf{r})$$

Via Lemma 16 we suppose without loss of generality that for each $e \in \delta$, $\phi(e) = t_e + p_e\mathbb{N}$ for some integers t_e and p_e.

We fix one run $\mathbf{r} \in \mathbf{R}$ of trace $\mathbf{t}$. For each $e \in \delta$, we denote by r_e the number of occurences of e in $\mathbf{t}$. By commutativity,

$$\Phi(\mathbf{r}) = \sum_{e\in\delta} (t_e r_e + (p_e\mathbb{N})^{r_e}) = \left(\sum_{e\in\delta} t_e r_e\right) + \left(\sum_{e\in\delta} (p_e\mathbb{N})^{r_e}\right)$$

Note that $|\mathbf{t}| = n+1$. Thus, $s_{\mathbf{r}} = \sum_{e\in\delta} t_e r_e$ is an integer less than or equal to $m(n+1)$ where $m = \max_{e\in\delta}(t_e)$. Then, define $C_{\mathbf{r}} = \sum_{e\in\delta} (p_e\mathbb{N})^{r_e}$. Denote by $I_{\mathbf{r}}$ the set of transitions e such that $r_e > 0$. Since for any ℓ we have $\ell\mathbb{N} + \ell\mathbb{N} = \ell\mathbb{N}$, the set $C_{\mathbf{r}}$ is equal to $\sum_{e\in I_{\mathbf{r}}} p_e\mathbb{N}$. Observe that there are finitely many possible $I_{\mathbf{r}}$. By Corollary 22, there exist two integers k and p such that for each subset I of transitions, there are two sets A_I and M_I, respectively bounded by k and p, such that $\sum_{e\in I} p_e\mathbb{N} = A_I \cup (k + M_I + p\mathbb{N})$. In particular, $C_{\mathbf{r}} = A_{I_{\mathbf{r}}} \cup (k + M_{I_{\mathbf{r}}} + p\mathbb{N})$. Finally:

$$\begin{aligned}\Phi(\mathbf{r}) &= s_{\mathbf{r}} + (A_{I_{\mathbf{r}}} \cup (k + M_{I_{\mathbf{r}}} + p\mathbb{N})) \\ &= (s_{\mathbf{r}} + A_{I_{\mathbf{r}}}) \cup (s_{\mathbf{r}} + k + M_{I_{\mathbf{r}}} + p\mathbb{N})\end{aligned}$$

As previously claimed, $s_{\mathbf{r}} < m(n+1)$. We can thus find an integer t, independent on n, such that $k + m(n+1) < t(n+1)$. Then, using Lemma 11, we can find $B_{\mathbf{r}}$ bounded by $t(n+1)$ and $M'_{\mathbf{r}}$ bounded by p such that $\Phi(\mathbf{r}) = B_{\mathbf{r}} \cup (t(n+1) + M'_{\mathbf{r}} + p\mathbb{N})$.

Now we consider all successful runs of $\mathcal{T}$ on w, *i.e.*, all runs in $\mathbf{R}$. It follows from previous study:

$$R(w) = \bigcup_{\mathbf{r}\in\mathbf{R}} B_{\mathbf{r}} \cup (t(n+1) + M'_{\mathbf{r}} + p\mathbb{N})$$

and hence, by commutativity and associativity of the set union operation:

$$R(w) = \left(\bigcup_{\mathbf{r}\in\mathbf{R}} B_{\mathbf{r}}\right) \cup \left(t(n+1) + \left(\bigcup_{\mathbf{r}\in\mathbf{R}} M'_{\mathbf{r}}\right) + p\mathbb{N}\right)$$

Because each $B_{\mathbf{r}}$ and $M'_{\mathbf{r}}$ are respectively bounded by $t(n+1)$ and p, so are their respective unions over $\mathbf{R}$. □

Back to 2-way transducers

From the study of Section 5.3.1, we are now able to extend Theorem 17 to the relations of the special form $R(\!|H|\!)S^{\circledast}$ for some rational relations R and S.

Lemma 17. *Let Σ be an arbitrary alphabet. Let R and S be two rational relations in $\Sigma^* \times \mathbb{N}$. The rational set $(R(\!|H|\!)S^{\circledast})(w)$ admits a period in $\mathcal{O}(|w|)$.*

Proof. By Theorem 17, for $Z = R, S$, there exist two integers t_Z and p_Z, such that for every $w \in \Sigma^*$, there are two finite subsets $A_Z(w), M_Z(w) \subseteq \mathbb{N}$ respectively bounded by $t_Z(|w|+1)$ and p_Z that satisfy:

$$Z(w) = A_Z(w) \cup (t_Z(|w|+1) + M_Z(w) + p_Z\mathbb{N})$$

Consider $S^{\circledast}$. By Lemma 15, the set $S^{\circledast}(w) = (S(w))^*$ admits a period $q_{S,w}$ less than or equal to $\max(t_S(|w|+1), p_S)$. Applying Lemma 12, the integer $p_w = p_R \times q_{S,w}$ is a period for both $R(w)$ and $S(w)^*$ and thus for $R(w) + S(w)^* = \left(R \text{Ⓗ} S^{\circledast}\right)(w)$ by Proposition 37. Observe that $p_w = p_R \times q_{S,w} \leq p_R \times \max(t_S \times (|w|+1), p_S)$. This concludes the proof. □

Finally, we prove that the period of the image of an input of length n in a 2-way transduction is bounded by a polynomial in n:

Theorem 18. *Let Σ be an arbitrary alphabet. Let R be an Hadamard relation in $\Sigma^* \times \mathbb{N}$. Then there exists an integer k such that for each input word $w \in \Sigma^*$, the rational set $R(w)$ admits a period in $\mathcal{O}\left(|w|^k\right)$.*

Proof. Let R be an Hadamard relation in $\Sigma^* \times \mathbb{N}$. Then, by Proposition 32, for some finite families of rational relations $(X_i)_{0 \leq i < k}$ and $(Y_i)_{0 \leq i < k}$:

$$R = \bigcup_{0 \leq i < k} X_i \text{Ⓗ} Y_i^{\circledast}$$

By Lemma 17, for every $0 \leq i < k$ there exists an integer c_i such that for every $w \in \Sigma^*$, the set $\left(X_i \text{Ⓗ} Y_i^{\circledast}\right)(w)$ admits a rat-expression of period p_i less than or equal to $c_i(|w|+1)$. We define $p = \prod_{0 \leq i < k} p_i$. Thus, $p \leq \prod_{0 \leq i < k} c_i(|w|+1) = \mathcal{O}\left(|w|^k\right)$. By Lemma 12, $p(w)$ is a period for every $\left(X_i \text{Ⓗ} Y_i^{\circledast}\right)(w)$. Thus, using Lemma 11 that preserves the period, we can show that p is a period of $R(w)$. □

Separating general 2-way and sweeping transducers

Theorem 18 allows us to prove that the family $\textsc{Had}(\Sigma^* \times a^*)$ is strictly included in the family of 2-way transductions, *i.e.*, the family of relations in $\Sigma^* \times a^*$ accepted by 2-way transducers. We define the relation MULTBLOCK on $\Sigma = \{a, \#\}$ by setting for each input word $w \in \Sigma^*$:

$$\textsc{MultBlock}(w) = \{kn \mid k, n \in \mathbb{N} \text{ and } w \in \Sigma^* \# a^n \# \Sigma^*\}$$

The relation is accepted by a 2-way transducer:

Proposition 38. *The relation* MULT-BLOCK *is accepted by a 2-way transducer.*

Proof. The automaton underlying our 2-way transducer accepting MULTBLOCK is depicted in Figure 5.3. For clarity, the production function is given in caption. It should be clear that the relation accepted is MULTBLOCK. □

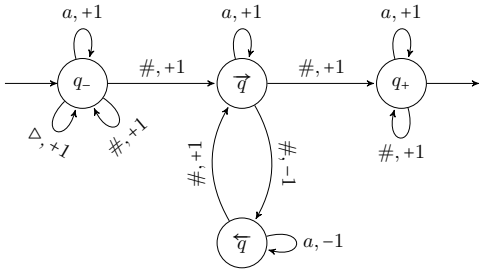


FIGURE 5.3: The underlying automaton of a 2-way transducer accepting MULTBLOCK. The production function maps the transition $(\overleftarrow{q}, a, -1, \overleftarrow{q})$ to 1 and all other transitions to 0.

We prove now that MULTBLOCK is not Hadamard:

Lemma 18. *The relation* MULTBLOCK *is not Hadamard.*

Proof. By Theorem 18, it suffices to prove by contraposition that there exists an infinite sequence of input words $w_n \in \Sigma^*$ of strictly increasing length such that the minimal period p_n of MULTBLOCK(w_n) is superpolynomial in the length $|w_n|$.

For $n > 0$, we define $w_n = \#a^{r_1}\#a^{r_2}\#\cdots\#a^{r_n}\#$ where r_i denotes the i-th prime number. The image of w_n by MULTBLOCK is given by:

$$\text{MULTBLOCK}(w_n) = \bigcup_{0<i\leq n} (r_i\mathbb{N})$$

By Proposition 35, the minimal period p_n is equal to $\prod_{0<i\leq n} r_i$. Asymptotically p_n is in $\Omega\left(e^{n\log(n)}\right)$ [29]. On the other hand, $|w_n| = n + 1 + \sum_{0<i\leq n} r_i$ is in $\theta\left(\frac{1}{2}n^2\log(n)\right)$ [4]. A simple computation shows that p_n is in $\Omega\left(e^{\sqrt[3]{|w_n|}\log(|w_n|)}\right)$ which is superpolynomial in $|w_n|$. □

Observe that for every integer k, the period of the image of $w \in \Sigma^*$ in the restriction MULTBLOCK $\cap\, ((\#a^*)^k\# \times \mathbb{N})$ is in $\mathcal{O}\left(|w|^k\right)$.

Recall Corollary 4 and Corollary 8 asserting the equivalence between Hadamard relations and relations accepted by sweeping transducers.

Theorem 19. *Let Σ and Δ be two alphabets. If Σ has cardinality at least 2 then the family of Hadamard (*resp. *Mirror-Hadamard) relations in* HAD$(\Sigma^* \times \Delta^*)$ *(*resp. MHAD$(\Sigma^* \times \Delta^*)$*), or equivalently the family of rotating (*resp. *sweeping) transductions over Σ and Δ, is strictly included in the family of 2-way transductions.*

A corollary

Recall that the componentwise concatenation of two relations $A_1, A_2 \subseteq \Sigma^* \times \mathbb{N}$ is the relation given by $A_1 \cdot A_2 = \{(u_1u_2, n_1 + n_2) \mid (u_1, n_1) \in A_1,\ (u_2, n_2) \in A_2\}$. Define the two relations:

$$\begin{aligned} \textsc{Erase} &= \{(w, 0) \mid w \in \Sigma^*\}; \\ \textsc{MultOneBlock} &= \{(\#a^n\#, kn) \mid n, k \in \mathbb{N} \text{ and } w \in \mathbb{N}\}. \end{aligned}$$

Observe that Erase is rational therefore Hadamard and MultOneBlock is Hadamard[3]. Then we have:

$$\textsc{MultBlock} = \textsc{Erase} \cdot \textsc{MultOneBlock} \cdot \textsc{Erase}$$

The following is hence a consequence of Lemma 18.

Corollary 23. *The family of Hadamard relations is not closed under componentwise concatenation, even when the output alphabet is unary.*

5.3.3 The unary input case

We know that 2-way transducers are equivalent to sweeping transducer when both the input and the output alphabets are unary. Theorem 19 shows that this is not the case anymore, when the output alphabet only is unary. In this section we give an example of a relation in $\{a\}^* \times \{a, b\}^*$ which is accepted by a 2-way transducer but not by a sweeping transducer.

Example 13.

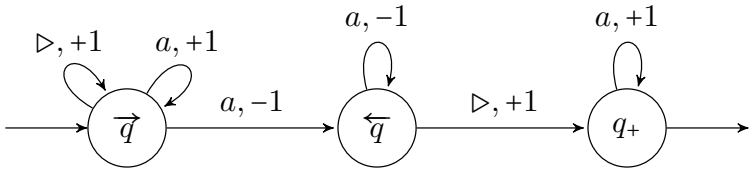
$$\text{LR-}\textsc{Prefix} = \{(a^n, a^pb^p) \mid 0 < p \leq n,\ n \in \mathbb{N}\}$$

It is an easy exercise to build a 2-way transducer accepting LR-Prefix. *We give one in Figure 5.4. Observe that it has a nondeterministic choice in state $\overrightarrow{q}$ scanning a. This nondeterminism is strongly required for our purpose, since every deterministic output-unary transducer admits an equivalent 1-way transducer [2].*

The relation LR-Prefix cannot be accepted by a sweeping transducer.

Lemma 19. *No sweeping transducer may accept the relation* LR-Prefix.

[3]But it is not rational: compare it with the relation uMult defined in Section 2.3.4.



$\phi((q, c, d, q'))$ is equal to a if $q' = \overrightarrow{q}$, to b if $q' = \overleftarrow{q}$ and to ϵ else.

FIGURE 5.4: A 2-way transducer accepting LR-PREFIX.

Proof. Suppose there exists a sweeping transducer $\mathcal{T}$ accepting LR-PREFIX. Without loss of generality, we assume $\mathcal{T}$ is restless.

We first show that this is no loss of generality to suppose that ϕ is single-valued. Indeed, since the image of any input word is finite, for every transition t, $\phi(t)$ is finite. It is thus possible to store the last chosen output in the finite control of the automaton, in order to make $\mathcal{T}$ single-valued. Thus, we assume from now on that $\mathcal{T}$ is a sweeping restless single-valued transducer accepting LR-PREFIX.

Observe that for each $n \in \mathbb{N}$, the language LR-PREFIX(a^n) has cardinality n, by definition. We will prove that for some large enough n, the set of outputs produced by the successful runs of $\mathcal{T}$ on a^n has cardinality less than n, leading to a contradiction.

Let $\mathbf{r}$ be a successful run on some input u. We can decompose r into three runs as follows:

$$\mathbf{r} = \mathbf{r}_1@(q,p)@\mathbf{r}_2@(q',p')@\mathbf{r}_3$$

such that $\mathbf{r}_2$ is a (q,p) to (q',p') hit, and:

$$\phi(\mathbf{r}_1) \in a^* \qquad \phi(\mathbf{r}_2) \in a^*b^+ \qquad \phi(\mathbf{r}_3) \in b^*$$

Observe that, because $\mathcal{T}$ is sweeping, p' is the border opposite to p. We may thus identify both using the one-bit information $b \in \{\triangleright, \triangleleft\}$, meaning that $u_p = b$. The border opposite to b is denoted $\eth$ and $u_{p'} = \eth$. The tuple $(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3)$ is called (q, b, q')-*phase decomposition of* $\mathbf{r}$.

For each state q and q' and each $b \in \{\triangleright, \triangleleft\}$, we define the following relations:

$$\textsc{Prefix}_{q,b,q'} = \left\{ (u,v) \mid \begin{array}{c}\text{there exists a successful run on } u \text{ of } (q,b,q')\text{-phase} \\ \text{decomposition } (\mathbf{r}_1,\mathbf{r}_2,\mathbf{r}_3) \text{ and } v = \phi(\mathbf{r}_1)\end{array} \right\}$$

$$\textsc{Transit}_{q,b,q'} = \left\{ (u,v) \mid \begin{array}{c}\text{there exists a successful run on } u \text{ of } (q,b,q')\text{-phase} \\ \text{decomposition } (\mathbf{r}_1,\mathbf{r}_2,\mathbf{r}_3) \text{ and } v = \phi(\mathbf{r}_2)\end{array} \right\}$$

$$\textsc{Suffix}_{q,b,q'} = \left\{ (u,v) \mid \begin{array}{c}\text{there exists a successful run on } u \text{ of } (q,b,q')\text{-phase} \\ \text{decomposition } (\mathbf{r}_1,\mathbf{r}_2,\mathbf{r}_3) \text{ and } v = \phi(\mathbf{r}_3)\end{array} \right\}$$

There are finitely many triples (q,b,q') and by definition:

$$\text{LR-}\textsc{Prefix} = \bigcup_{q,b,q'} \textsc{Prefix}_{q,b,q'} \circledcirc \textsc{Transit}_{q,b,q'} \circledcirc \textsc{Suffix}_{q,b,q'} \tag{5.4}$$

Observe that, for each q, q' and b, the three relations defined above have the same domain which is equal to:

$$D_{q,b,q'} = \left\{ u \in a^* \mid \begin{array}{c}\text{there exists a successful run on } u \\ \text{which admits a } (q,b,q')\text{-phase decomposition}\end{array} \right\}$$

It is easy to prove that it is recognizable. Thus, each relation $\textsc{Transit}_{q,b,q'}$ is accepted by a 1-way transducer $\mathcal{T}_{q,b,q'}$ obtained from $\mathcal{T}$. The transducer $\mathcal{T}_{q,b,q'}$ does not simply simulate the (q,b) to (q',d) hits of $\mathcal{T}$, but should also check that the input belongs to $D_{q,b,q'}$. We denote by N the maximal number of states of these 1-way transducers *i.e.*, $N = \max_{q,b,q'} |T_{q,b,q'}|$.

We fix (q,b,q'). It is easy to see that both $\textsc{Prefix}_{q,b,q'}$ and $\textsc{Suffix}_{q,b,q'}$ should be functional. It follows that for any $n \in \mathbb{N}$ we have:

$$|(\textsc{Prefix}_{q,b,q'} \circledcirc \textsc{Transit}_{q,b,q'} \circledcirc \textsc{Suffix}_{q,b,q'})(a^n)| = |\textsc{Transit}_{q,b,q'}(a^n)| \tag{5.5}$$

Moreover, if v and v' in a^*b^+ belong to $\textsc{Transit}_{q,b,q'}(a^n)$ then we have:

$$|v|_b - |v|_a = |v'|_b - |v'|_a \tag{5.6}$$

We will use this property, in order to bound in function of N, the number of words in each $\textsc{Transit}_{q,b,q'}(a^n)$.

Let n be fixed and let $\mathbf{r}$ be a successful run of $\mathcal{T}_{q,b,q'}$ (*i.e.*, the 1-way transducer accepting $\textsc{Transit}_{q,b,q'}$) on a^n. Suppose:

$$\mathbf{r} = \mathbf{r}_1 @ \ell_1 @ \mathbf{r}_2 @ \ell_2 @ \mathbf{r}_3$$

where both ℓ_1 and ℓ_2 are cycles (*i.e.*, sub-runs starting and ending in the same state), with $\phi(\ell_1) \in a^+$ and $\phi(\ell_2) \in b^*$ (or similarly, a^* and b^+ respectively). Denote by h_1 and h_2 the length, in terms of head moves, of ℓ_1 and ℓ_2 respectively. Then, on the input word $u = a^{n+h_1h_2-h_1-h_2}$, we may find two different successful runs: one repeating h_2 times the cycle ℓ_1 and 0 times the cycle ℓ_2, and the second repeating 0 times the cycle ℓ_1 and h_1 times the cycle ℓ_2. The associated outputs trivially violates (5.6). Hence, every successful run $\mathbf{r}$ of $\mathcal{T}_{q,b,q'}$ satisfies one of the three following properties:

- all the cycles of $\mathbf{r}$ have their output in a^+;
- all the cycles of $\mathbf{r}$ have their output in b^+;
- all the cycles of $\mathbf{r}$ have their output in $\{\epsilon\}$.

For each case, the number of a or the number of b produced during $\mathbf{r}$ is linearly bounded in N, say, less than kN. Since, for any number of a (*resp.* of b) in the output, there is only one possible number of b (*resp.* of a) by (5.6), the total number of $v \in \textsc{Transit}_{q,b,q'}(a^n)$ is less than kN.

We conclude by observing that, by the equations (5.4) and (5.5), the number of outputs associated to any input word a^n is bounded by $2kN|Q|^2$. This is a contradiction, since any input a^n with $n > 2kN|Q|^2$ has more associated outputs. □

It follows that the family of Hadamard relations does not capture the family of relations accepted by 2-way transducers, even when the input alphabet is unary. See the analogy with Theorem 19.

Theorem 20. *Let Σ and Δ be two alphabets. If Σ has cardinality at least 2 then the family of Hadamard (*resp. *Mirror-Hadamard) relations in* $\textsc{Had}(\Sigma^* \times \Delta^*)$ *(*resp. $\textsc{MHad}(\Sigma^* \times \Delta^*)$*), or equivalently the family of rotating (*resp. *sweeping) transductions over Σ and Δ, is strictly included in the family of general transductions,* i.e., *relations accepted by 2-way transducers.*

As for Theorem 19, we can deduce a non-closure property of the family of Hadamard relations.

Corollary 24. *The family of Hadamard relations is not closed under componentwise concatenation, even when the input alphabet is unary.*

Proof. We define the following relation:

$$\textsc{Rename}_{a,b} = \{(a^n, b^n) \mid n \in \mathbb{N}\}$$

Obviously, it is rational. Hence, the Hadamard product: $\textsc{Id}\,Ⓗ\,\textsc{Rename}_{a,b}$ belongs to $\textsc{Had}\big(a^* \times \{a,b\}^*\big)$. We conclude the proof by observing:

$$\text{LR-}\textsc{Prefix} = (\textsc{Id}\,Ⓗ\,\textsc{Rename}_{a,b}) \cdot \textsc{Erase}$$

where the relation Erase is defined as in Section 5.3.2. □

5.3.4 Conclusion

Corollary 4 together with Corollary 8, claims that on unary input and output alphabets, sweeping transducers have the same recognition power as general 2-way transducers. In this section, we have shown how crucial is the hypothesis asking for both input and output alphabets. Indeed, we have exhibited two relations, one with a unary input alphabet, the other with a unary output alphabet, that separate the two families of relations (Theorems 19 and 20). Despite the intuition and the simplicity of the two examples, the proofs involve many intermediate results and some technical material, showing the complexity of the dynamics of 2-way devices. Some of these intermediate results are interesting for their own sake, in particular, the bound on the period of the image of unary 1-way and unary sweeping transducers (Theorems 17 and 18).

Chapter 6

Iteration of arity 2 relations on words

6.1 Preliminaries

In this chapter we consider relations in $\Sigma^* \times \Sigma^*$, for some fixed alphabet Σ. When Σ is supposed to be unary, it is more convenient to work in $\mathbb{N} \times \mathbb{N}$ which can be identified with $a^* \times a^*$ by the mapping $a^n \mapsto n$ extended to $a^* \times a^*$.

Notations: We will consider sequences of words, that it is convenient to write $w_1, w_2, \ldots, w_n$. Hence, w_i does no longer refer to the i-th symbol of w, as denoted in Section 2.1.2, but to the i-th word in the sequence. We are thus led to change the former notation: given a word w of length n, and given $1 \leq i \leq n$, the i-th symbol of w is denoted by $w[i]$.

6.1.1 Composition and iteration of relations

Given R and S in $\Sigma^* \times \Sigma^*$, we define the *composition* of R and S in a natural way:

$$R \circ S := \{(u, w) \mid \exists v \in \Sigma^*,\ (u, v) \in S \text{ and } (v, w) \in R\}$$

The identity relation $\textsc{Id} = \{(u, u) \mid u \in \Sigma^*\}$ is a neutral element for the composition. Moreover, the composition is distributive over the union. Thus, the structure $\left\langle 2^{\Sigma^* \times \Sigma^*}, \cup, \circ, \textsc{Id}, \varnothing \right\rangle$ is a semiring.

We can inductively define the *composition powers* of a relation as follows:

$$R^{(0)} := \textsc{Id} \qquad \text{and for each } k \geq 0,\ \ R^{(k+1)} := R \circ R^{(k)}$$

The *iterative star* of R, which is the reflexive and transitive closure of R, denoted $R^{(*)}$, is defined by:

$$R^{(*)} := \bigcup_{k \in \mathbb{N}} R^{(k)}$$

Concerning rational relations, we have the important result:

Theorem 21 (Elgot and Mezei [23]). *If R and S are rational relations, so is $R \circ S$.*

6.1.2 Decidability problems

This chapter is motivated by the case of rational relations. Indeed, if the different composition powers of a rational relation are still rational, their union is not, even in the case of unary alphabet (see Theorem 29). We ask the following questions, given a relation R and a class of relation X:

- MEMBERSHIP TO A RELATION R: can we decide whether or not a given a pair (u, v) belongs to $R^{(*)}$?
- CLASSIFICATION FOR A FAMILY OF RELATIONS X: given a relation R, does $R^{(*)}$ belongs to X?

We will prove that for some very restricted R, the MEMBERSHIP problem is decidable (see Example 14, 26 and 28), and we determinate its complexity (which is a kind of answer to the CLASSIFYING problem). However, in most cases, the MEMBERSHIP problem has a negative (see [75]) and we propose some new approaches (see Theorems 25, and 29).

We did not have the time to study the following questions but there are worthwhile being considered. Given a relation R,

- LOCAL FINITE POWER PROPERTY: does there exist for each word u, an integer N such that $R(u)^{(*)} = \bigcup_{0 \leq k < N} R(u)^{(k)}$?
- FINITE POWER PROPERTY: does there exist an integer N such that $R^{(*)} = \bigcup_{0 \leq k < N} R^{(k)}$?
- UNIVERSALITY: is $R^{(*)}$ equal to $\Sigma^* \times \Sigma^*$?
- LOCAL NILPOTENCY: does there exist for each u, an integer N such that $R(u)^{(N)} = \varnothing$?
- NILPOTENCY: does there exists an integer N such that $R^{(N)} = \varnothing$?

Example 14 (Example: iteration of recognizable relation). *In [66], the following is given as exercise: if R is recognizable, then $R \circ R^{(*)}$ is recognizable.*

Indeed, let R be a recognizable binary relation. By Theorem 1, we may suppose $R = \bigcup_{i \in I} A_i \times B_i$ for some finite set of indexes I, and some regular languages A_i's and B_i's. We associate to R the bipartite graph G built as follows:

1. *The vertices of G are the A_i's and B_i's.*
2. *For each i, we define an edge from A_i to B_i.*
3. *For each i, j such that $A_i \cap B_j \neq \varnothing$, we define an edge from node B_j to node A_i.*

Suppose $(w, w') \in R \circ R^{(*)}$. Thus there exists $w_0, w_1, \ldots, w_n$ such that $w_0 = w$, $w_n = w'$, and for each j, $(w_j, w_{j+1}) \in R$. Because $R = \bigcup_{i \in I} A_i \times B_i$, there exist a sequence $i_1, \ldots, i_n$ of indexes from I, such that, $w_0 \in A_{i_1}$, $w_n \in B_{i_n}$ and for each $1 \leq j < n$, $w_j \in B_{i_j} \cap A_{i_{j+1}}$, which, in particular implies that $B_{i_j} \cap A_{i_{j+1}} \neq \varnothing$, or equivalently, that $(B_{i_j}, A_{i_{j+1}})$ is an edge of G. Moreover, by construction of G there are edges from A_{i_j} to B_{i_j} for each j. Thus, $(w, w') \in R \circ R^{(*)}$ implies that there exists a path in G from a node A_{i_1} to a node B_{i_n} such that, $w \in A_{i_1}$ and $w' \in B_{i_n}$.

Conversely, suppose we have a path in G from a node A_{i_1} to a node B_{i_n}, and let be $w \in A_{i_1}$ and $w' \in B_{i_n}$. Because the graph is bipartite, the path is necessary of the following form: $A_{i_1}, B_{i_1}, \ldots, A_{i_n}, B_{i_n}$. Moreover, for $j < n$, $B_{i_j} \cap A_{i_{j+1}} \neq \varnothing$, hence we can find and fix a $w_j \in B_{i_j} \cap A_{i_{j+1}}$. We obtain a sequence $w_1, w_2, \ldots, w_{n-1}$, that we extend with $w_0 = w$ and $w_n = w'$. By definition, each (w_j, w_{j+1}) belongs to R. This implies $(w, w') \in R \circ R^{(*)}$.

The graph G is finite (it does not depend on w), thus we can pre-compute the set of accessible indexes:

$$\textsc{Acc}_G := \{(i, j) \mid \textit{there exists a path from node } A_i \textit{ to node } B_j\}$$

Hence, $R \circ R^{(*)} = \bigcup_{(i,j) \in \textsc{Acc}_G} A_i \times B_j$, which is recognizable.

In particular, the Finite Power Property*, and thus its local version, are always true for recognizable relations. Indeed, the minimal integer N such that $R^{(*)} = R^{(N)}$ is the maximal distance in the graph G divided by 2. In particular, it is bounded by $|I|$[1]. Concerning the* Nilpotency*, the problem reduces to the existence of a cycle in G. It is thus decidable.*

[1] This answer to the Finite Power Property can be generalize to any relation of the form $\bigcup_{i \leq M} A_i \times B_i$.

6.1.3 Length-preserving relations, padding and completion

We introduce here a very basic property of relations. A relation $R \subseteq \Sigma^* \times \Sigma^*$ is *length-preserving* if each $(u,v) \in R$ satisfies $|u| = |v|$. Clearly, if R and S are length-preserving, so are $R \circ S$, $R^{(k)}$ for $k \in \mathbb{N}$ and $R^{(*)}$.

We can map a relation into a length-preserving one in an injective way. To this end, we define some one-to-one mappings of $2^{\Sigma^* \times \Sigma^*}$ into $2^{(\Sigma \cup \{\#\})^* \times (\Sigma \cup \{\#\})^*}$ for some new symbol $\#$. For $(u,v) \in \Sigma^* \times \Sigma^*$ and $k \in \mathbb{N}$, the *k-padding* of (u,v), denoted $(u,v)^{\#_k}$, is the pair of words obtained from (u,v) by adding exactly k symbol $\#$ at the right of the longest component, and adding enough $\#$ at the right of the shortest one in order to obtained a pair of words of same length. The *completion* of (u,v), denoted $(u,v)^{\#}$, is its 0-padding. Formally:

$$(u,v)^{\#_k} := \left(u\#^i, v\#^j\right) \text{ with } |u| + i = |v| + j \text{ and } \min(i,j) = k \qquad (u,v)^{\#} := (u,v)^{\#_0}$$

The definition of *k-padding* (*resp. completion*) of a relation R, denoted $R^{\#_k}$ (*resp.* $R^{\#}$), is the natural extension from pair of words to subsets of pair of words:

$$R^{\#_k} := \left\{(u,v)^{\#_k} \mid (u,v) \in R\right\} \qquad R^{\#} := \left\{(u,v)^{\#} \mid (u,v) \in R\right\}$$

Observe that the k-paddings (and so, the completion) of a relation $R \subset \Sigma^* \times \Sigma^*$ is a relation on the alphabet $\Sigma \cup \{\#\}$. In particular, if Σ is unary, the completion of $R \subset \Sigma^* \times \Sigma^*$ is generally no longer a relation on a unary alphabet. The completion of a length-preserving relation is equal to the relation itself.

6.1.4 One-way two-tape finite automata

Here we use the model of 2-tape automaton introduced by Rabin and Scott [63]. and we study some restricted variants. This device may be seen as an extension of the 1-way finite automata (see Definition 7) in which an additional read-only tape is scanned by a second input head. It can easily be extended to the model of n-tape finite automata[2] for arbitrary $n > 0$, but, since we aim to study the composition and iteration of relations of arity 2, we are essentially interested in the 2-tape case. All the definitions below (configurations, runs...) are straightforward adaptations of those given in Section 2.3.

[2]The 2-way variants of 2-tape finite automata has also been studied (see for instance [28]), but their dynamics is really difficult to catch. They recognize a much broader family of relations than the rational one. It is for example easy to build a 2-way 2-tape finite automaton accepting the relation $\{(a^n b^n c^n, a^n b^n c^n) \mid n \in \mathbb{N}\}$.

Device

A *2-tape finite automaton* (FA_2) scans two read-only input tapes, each with an independent head. At each step, the *transition function* determines the possible next *states* and head movements, based on the current state and the symbols currently scanned by the heads. Two special symbols $\triangleright$ and $\triangleleft$ mark the left and the right ends of each input tape, preventing the input heads to fall out the tape. Formally:

Definition 19. *A* 2-tape finite automaton *is a tuple* $(Q, \Sigma, \triangleright, \triangleleft, \delta, I, F)$*, where:*

- Q *is the finite set of* states*;*
- I *(*resp. F*) is a subset of* Q*, whose elements are the* initial *(*resp. accepting*) states;*
- Σ *is an alphabet, called* input alphabet*, not including the symbols* $\triangleright$ *and* $\triangleleft$*, which are respectively the* left *and* right endmarkers*;*
- δ *is the* transition set*,* i.e.*, a subset of* $Q \times {\Sigma_{\triangleright\triangleleft}}^2 \times \{0,1\}^2 \times Q$ *where* $\Sigma_{\triangleright\triangleleft}$ *denotes the extended alphabet* $\Sigma \cup \{\triangleright, \triangleleft\}$*. It is supposed to be disjoint from the set* $F \times \{\triangleleft\}^2 \times \{0,1\}^2 \times Q$*.*

Configurations, runs...

Given an automaton $\mathcal{A} = (Q, \Sigma, \triangleright, \triangleleft, I, F)$ and given two words w_1 and w_2, a *configuration of* $\mathcal{A}$ *on* (w_1, w_2) is a tuple $(q, p_1, p_2) \in Q \times \{0, \ldots, |w_1| + 1\} \times \{0, \ldots, |w_2| + 1\}$ where q is the current state and p_1 (*resp.* p_2) is the current position of the first (*resp.* second) head on $\widetilde{w_1}$ (*resp.* $\widetilde{w_2}$) (as denoted in Section 2.3.2, $\widetilde{w}$ denote the word $\triangleright w \triangleleft$ for any w). In particular when $p_i = 0$ (*resp.* $|w_i| + 1$) the i-th head is scanning the left (*resp.* right) endmarker, that we denote $w_i[0]$ (*resp.* $w_i[|w_i| + 1]$). An *initial configuration* is a configuration $(q, 0, 0)$ with $q \in I$. When q belongs to F, $p_1 = |w_1| + 1$ and $p_2 = |w_2| + 1$, the configuration is *accepting*.

The *successor relation* between configurations, denoted $\to$, describes one computational step of the machine. It naturally derives from δ:

$$(q, p_1, p_2) \to (q', p_1', p_2') \quad \text{if and only if} \quad (q, (w_1[p_1], w_2[p_2]), (p_1' - p_1, p_2' - p_2), q') \in \delta$$

A configuration is said *halting* if it does not admit a successor. Due to the last restriction on δ in Definition 19, every accepting configuration is halting.

A *run* ρ of $\mathcal{A}$ on input words (w_1, w_2) is a possibly infinite sequence of configuration, $c_0, c_1, \ldots$ such that for all $k \geq 0$, $c_k \to c_{k+1}$. It is *initial* if c_0 is initial. It is *halting* if it is finite and its last configuration is halting. A *successful run* is a run which is both

initial and halting, and whose last configuration is accepting. The automaton $\mathcal{A}$ *accepts* an input $\mathbf{w} = (w_1, w_2)$ if and only if there exists a successful run of $\mathcal{A}$ on $\mathbf{w}$. The *relation accepted* by $\mathcal{A}$, denoted $\|\mathcal{A}\|$, is the set of pair of words accepted by $\mathcal{A}$.

Recognition power of FA_2

The main result concerning FA_2s is:

Theorem 22 (Elgot and Mezei [23])**.** *A relation in $\Sigma^* \times \Sigma^*$ is accepted by a FA_2 if and only if it is a rational subset of $\Sigma^* \times \Sigma^*$.*

Restrictions

As for FAs, $\mathbb{K}$-FAs and transducers (see Sections 2.3.2, 2.3.3, 2.3.4), we define some restricted variants of the device. A FA_2 is:

- *deterministic* (DFA_2), if it is *initial*, *i.e.*, I is a singleton, and for each state q and each symbols c_1 and c_2, there exist at most one pair (d_1, d_2) and one state q' such that $(q, (c_1, c_2), (d_1, d_2), q')$ is a transition;
- *unambiguous* (UFA_2), if for each input there exists at most one successful run;
- *synchronous* (SFA_2), if it enforces the two heads to have always the same position, *i.e.*, if
$$\delta \subseteq \left(Q \times {\Sigma_{\triangleright\triangleleft}}^2 \times \{(0,0),(1,1)\} \times Q\right)$$
In particular, it may accept length-preserving relations only.

As expected, deterministic implies unambiguous. The synchronous variant is relevant when considering the family of series whose completion is accepted by the device.

Compacting

Since the two heads of a SFA_2 are always at the same position, they may be "glued" together into one input head scanning a 2-track tape. In other words, a relation accepted by a synchronous 2-tape automaton may be seen as a rational language on the alphabet $(\Sigma \times \Sigma)$.

Definition 20. *The* compacting *of a length-preserving relation R is the language $\breve{R}$ on the alphabet $(\Sigma \times \Sigma)$, such that a word w belongs to $\breve{R}$ if and only if, denoting (u_i, v_i) the symbol $w[i] \in \Sigma \times \Sigma$, we have $\left(u_1 \cdots u_{|w|}, v_1 \cdots v_{|w|}\right) \in R$.*

Compacting is aimed to better describe the relations accepted by SFA_2. The following result gives a trivial characterization of such relations.

Proposition 39. *A length-preserving relation in $\Sigma^* \times \Sigma^*$ is accepted by a SFA_2 if and only if, its compacting is accepted by a FA working on the alphabet $\Sigma \times \Sigma$.*

Proof. The proof is trivial. It is easily done by bijectively mapping every transition $(q, (c_1, c_2), (d, d), q')$ of a synchronous FA_2 into a transition $(q, (c_1, c_2), d, q')$ of a FA, and conversely. □

As said before, synchronous FA_2 will be applied to the completion of relations. Thus, we speak of the compacting of an arbitrary (that is, not length-preserving) relation R to refer to the compacting of its completion, and we denote $\breve{R} := \widetilde{R^{\#}}$.

Links with transducers

Input, output: In the case of transducers, the roles of the two tapes are clearly identified: one holds an input and the other one the output produced by the machine. For 2-tape automata, this distinction is less relevant, since they play a symmetric role. However, for convenience and since our model is meant to study the composition of relations, by convention we name the first (*resp.* the second) tape the *input* (*resp. output*) *tape*. Similarly, we will speak of the *input*/*output* heads or words, and of the *domain*/*image* of a FA_2.

From a FA_2 accepting a relation R, we can easily construct a second FA_2 accepting $\text{SYM}(R)$, by exchanging the roles of the two tapes. The resulting FA_2 is the *symmetric* of the first one.

Equivalence with transducers: It is well known that 2-tape finite automata are equivalent to 1-way transducers, and characterize the class of rational relations.

Proposition 40. *Every FA_2 is effectively equivalent to a 1-way transducer.*

Proof. A FA_2 can easily simulate an elementary transducer (see Section 2.3.4 and Proposition 6). Indeed, every time the simulated device writes a symbol on its output tape, the simulating FA_2 can check that the same symbol is scanned by its output head, and move it one cell to the right.

Conversely, a transducer can simulate a FA_2, by nondeterministically guessing which symbol is scanned by the simulated FA_2 after each move, and writing it on the output tape. In order to simulate transitions that does not move the output head, it has to store the previous guessed symbol in its finite control. □

Despite this correspondence, the two devices have significant differences, whenever deterministic aspects are considered. Indeed, deterministic 2-tape FAs are more powerful than deterministic transducers. Observe for example that, since the output preexists the computation, a relation accepted by a DFA_2 is not necessarily *functional*.

6.1.5 Classes of relations and hierarchy

Classes of relations

We distinguish now different families of relations. The distinction may arise either by considering their natural acceptor, or by their algebraic properties.

The family of recognizable relations, denoted REC, has been defined in Definition 2 and is characterized in Theorem 1. The rational relations have been introduced in Definition 1 and are accepted by FA_2s, thanks to Theorem 22. However, the formalism of FA_2 allows us to define further families. We denote by DRAT, URAT and NRAT the family of relations respectively recognized by deterministic, unambiguous and nondeterministic FA_2. The family of *synchronous relations* is the family of relations whose completion is recognized by a synchronous FA_2 and is denoted SYNC.

Another less traditional class is the class of *special* relations, denoted SPEC. Because no natural acceptor exists for this class, we define it as it was introduced in [1, 49]. A relation R on $\Sigma^* \times \Sigma^*$ is *special* if and only if there exists a rational language L over Σ and a recognizable relation T over Σ such that: $R = \{(wu, wv) \mid w \in L,\ (u, v) \in T\}$. By taking $L = \{\epsilon\}$, we directly obtain that every recognizable relation is special. It is also trivial to see that $\text{SPEC} \subsetneq \text{SYNC}$. Finally, the family FINITE of finite relations is included in every other class.

Notation: We will use the small-capital letter U to restrict the considered class to relations on unary a alphabet, and the prefix LP will denote the length-preserving restriction. For instance, LPDRAT refer to the family of length-preserving deterministic rational relations.

Based on the characterization of Elgot and Mezei (Theorem 1), it is not difficult to prove the following result.

Proposition 41. *A relation $R \subseteq \Sigma^* \times \Sigma^*$ is recognizable if and only if the language $\{u\#v \mid (u, v) \in R\}$ is rational, for $\#$ not belonging to Σ.*

A more powerful and well-known property holds for rational relations:

Proposition 42. *A relation $R \subseteq \Sigma^* \times \Sigma^*$ is rational, if and only if the language $\{u\#\overline{v} \mid (u, v) \in R\}$ is context-free, for $\#$ not belonging to Σ.*

This result has already been used in order to prove that the relation REV, which maps every word to its mirror, is not rational (see Proposition 11).

Concerning synchronous relations, a corollary of Proposition 39, is that synchronous relations are no more than rational languages on the alphabet $(\Sigma \cup \{\#\})^2 \setminus \{(\#,\#)\}$.

Corollary 25 (of Proposition 39)**.** *A relation R in $\Sigma^* \times \Sigma^*$ is synchronous, if and only if the relation $R^\#$ is a rational language over the alphabet $(\Sigma \cup \{\#\})^2 \setminus \{(\#,\#)\}$.*

Eilenberg proved that if a relation is both rational and length-preserving, then it is synchronous.

Theorem 23 (Eilenberg [21])**.** *If R is length-preserving and rational, then it is synchronous.*

Hierarchy

The following hierarchy is a direct consequence of the definitions:

$$\text{FINITE} \subseteq \text{REC} \subseteq \text{SPEC} \subseteq \text{SYNC} \subseteq \text{DRAT} \subseteq \text{URAT} \subseteq \text{NRAT} \tag{6.1}$$

Furthermore it is well known that it is strict. We give simple examples that witness the strictness of the above inclusions. When possible, they are taken unary or length-preserving. The hierarchy is depicted in Figure 6.1.

A natural example of non-recognizable relation in SPEC is the identity relation $\text{ID} = \{(w,w) \mid w \in \Sigma^*\}$. Even its unary version, denoted UID, belongs to $\text{SPEC} \setminus \text{REC}$. Moreover it is length-preserving. In fact, we may easily prove from Proposition 41, that if a relation is both length-preserving and recognizable then it is finite.

The relation $\text{USQUARE} := \{(\widetilde{a^n, a^{2n}}) \mid n \in \mathbb{N}\}$ belongs to $\text{UDRAT} \setminus \text{USYNC}$. Indeed, suppose it is synchronous. Then USQUARE is a rational language on $\Delta = \{(a,a),(a,\#),(\#,a)\}$ by Corollary 25. Considering the mapping μ which maps every symbol $(c,c') \in \Delta$ into its first component c, we obtain that the language $\mu\Big(\widetilde{\text{USQUARE}}\Big)$ is a rational language on $\Sigma \cup \{\#\}$. This is a contradiction since $\mu\Big(\widetilde{\text{USQUARE}}\Big) = \{a^n \#^n \mid n \in \mathbb{N}\}$ is clearly not rational.

Always on the unary alphabet $\Sigma = \{a\}$, it is easy to find a relation in URAT which does not belong to DRAT. In Section 6.2.2 we consider the following relation, which implements the well-known Collatz (also called Syracuse) function:

$$\text{SYRACUSE} = \Big\{\Big(a^{2n}, a^n\Big) \mid n \in \mathbb{N}\Big\} \cup \Big\{\Big(a^{2n+1}, a^{6n+4}\Big) \mid n \in \mathbb{N}\Big\}$$

A simple pumping argument may be used in order to prove that it does not belong to DRAT.

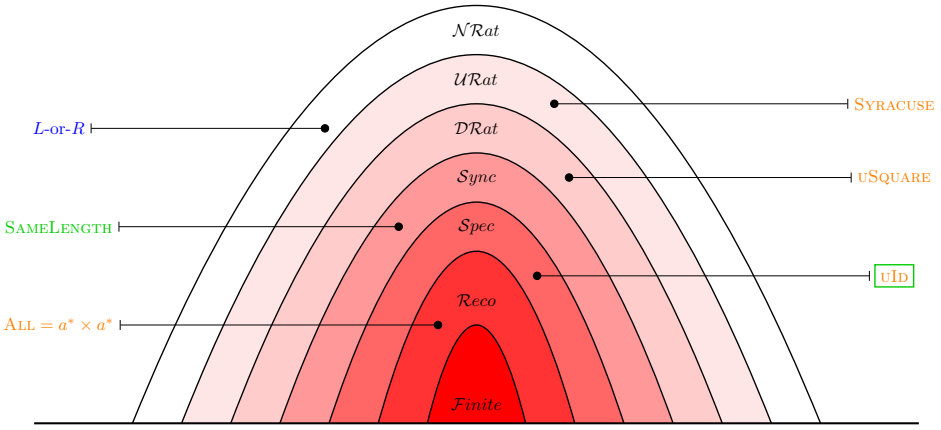


FIGURE 6.1: Hierarchy of the main relation classes with the examples of separating relation (orange: unary; green: length-preserving)

The probably simplest example of length-preserving synchronous relation is:

$$\text{SAMELENGTH} = \{(u, v) \mid u, v \in \Sigma^*,\ |u| = |v|\}$$

It is not special provided Σ has at least two symbols. Indeed, as shown with Corollary 26, the two families SYNC and SPEC coincide in the unary case.

The last example introduced here is a bit artificial. It is aimed to belong to NRAT $\setminus$ URAT, which can be proved by a simple pumping argument.

$$\text{L-OR-R} = \left\{\left(a^n b^m, c^k\right) \mid n, m \in \mathbb{N},\ k = n \text{ or } k = m\right\}$$

Unary synchronous relations

As already said at the beginning of this chapter, a unary word is characterized by its length, in the mapping $a^n \mapsto n$. Thus, unary languages are viewed as integer sets, and unary relations as subset of $\mathbb{N}^2$. Unary rational languages (view as integers set) are simply characterized by semi-linear sets (see [22]), that is finite union of linear sets, which may take the form given in Proposition 34. From this characterization, we are able to characterize unary synchronous relations.

Theorem 24. *A unary relation R is synchronous if and only if,*

$$R = \bigcup_{i \in I} \{(Mx + a_i, My + b_i),\ x, y \in \mathbb{N},\ C_i(x, y)\} \tag{6.2}$$

where I is a finite family of indices, M is a constant, and for each $i \in I$, a_i and b_i are two positive integers less than M, and C_i is one of the following predicates:

$$C_i \in \begin{Bmatrix} 0 = x = y & , & 0 = x \leq y & , & 0 = y \leq x \\ 0 \leq x = y & , & 0 \leq x \leq y & , & 0 \leq y \leq x \end{Bmatrix}$$

Proof. We start by proving that if a relation is of the form (6.2) then it is synchronous. Since synchronous relations are closed under union, it is sufficient to prove that, for any constants M, any c and d less than M, and any predicate C as in the Theorem, the relation $R = \{(Mx + c, My + d) \mid x, y \in \mathbb{N},\ C(x, y)\}$ is synchronous.

First of all, observe that, if C implies $x = 0$ or $y = 0$, then R is trivially recognizable, and thus it is synchronous. Secondly, the cases in which C implies $x = y$ obviously leads to a special relation which is thus synchronous too. The remaining cases are: $0 \leq x \leq y$ and $0 \leq y \leq x$. By symmetry, it is sufficient to consider one of these two cases only. From now on, we suppose that C is $0 \leq x \leq y$. Then:

- if $c \leq d$, then the language $R^{\#}$ over the alphabet $\Delta = \{(a, a), (a, \#), (\#, a)\}$ is:

$$R = \left((a, a)^M\right)^* (a, a)^c \cdot (\#, a)^{d-c} \cdot \left((\#, a)^M\right)^*$$

  It is thus rational on Δ, which directly implies that R is synchronous.

- else, we have $d < c < M$. In this case, the language $R^{\#}$ is:

$$R = \left((a, a)^M\right)^* \cdot \left[\left((a, a)^d \cdot (a, \#)^{c-d}\right) \cup \left((a, a)^c \cdot (\#, a)^{M-(c-d)} \cdot \left((\#, a)^M\right)^*\right) \right]$$

  Thus R is synchronous.

In any cases we have shown that R is synchronous, this proves the "if" direction of the theorem.

Conversely, let R be a unary synchronous relation. By Corollary 25, there exists a 1-way deterministic finite automaton $\mathcal{A} = (Q, \Delta, \delta, q_0, F)$ over the alphabet $\Delta = \{(a, a), (a, \#), (\#, a)\}$, accepting the language $R^{\#}$.

The automaton $\mathcal{A}$ induces three families of unary automata, indexed by Q, defined as follows:

$$A_0^{(q)} := \left(Q, \{(a, a)\}, q_0, \{q\}, \delta_{|\{(a,a)\}}\right)$$

$$A_1^{(q)} := \left(Q, \{(a, \#)\}, q, F, \delta_{|\{(a,\#)\}}\right) \qquad A_2^{(q)} := \left(Q, \{(\#, a)\}, q, F, \delta_{|\{(\#,a)\}}\right)$$

Each of these automaton recognizes a unary rational language, which can be viewed as a recognizable subset of $\mathbb{N}$. For each $k \in \{0,1,2\}$ and each $q \in Q$, we denote by $L_k^{(q)}$ this set recognized by the automaton $\mathcal{A}_k^{(q)}$. By Proposition 34 and Corollary 22, there exists two constants t and p such that, there exists for each pair k, q two finite sets $A_k^{(q)} \subseteq \{0, \dots, t-1\}$ and $M_k^{(q)} \subseteq \{0, \dots, p-1\}$ such that:

$$L_0^{(q)} = \left\{(a,a)^n \mid n \in A_0^{(q)} \cup \left(t + M_0^{(q)} + p\mathbb{N}\right)\right\}$$

$$L_1^{(q)} = \left\{(a,\#)^n \mid n \in A_1^{(q)} \cup \left(t + M_1^{(q)} + p\mathbb{N}\right)\right\}$$

$$L_2^{(q)} = \left\{(\#,a)^n \mid n \in A_2^{(q)} \cup \left(t + M_2^{(q)} + p\mathbb{N}\right)\right\}$$

By construction of the automata $\mathcal{A}_k^{(q)}$, the language $R^{\#}$ is:

$$R^{\#} = \left(\bigcup_{q\in Q} L_0^{(q)} \cdot L_1^{(q)}\right) \cup \left(\bigcup_{q\in Q} L_0^{(q)} \cdot L_2^{(q)}\right) \tag{6.3}$$

The first (*resp.* second) member of the union contains all the pairs (u,v) in R such that $|u| \geq |v|$ (*resp.* $|u| \leq |v|$). Observe that the language $\bigcup_{q\in F} L_0^q$ of pairs $(u,v) \in R$ with $|u| = |v|$ is included in the two members of the union, since when q belongs to F, both $\mathcal{A}_1^{(q)}$ and $\mathcal{A}_2^{(q)}$ accept the empty word.

By simple rewriting, we obtain Expression (6.2). □

A direct consequence is that unary synchronous relations are special.

Corollary 26. *When Σ is unary, the families* Sync *and* Spec *coincide.*

6.2 Iteration

In this section, we are interested in the Membership problem, when restricting the considered relation to belong to some class. Theorem 25 and Theorem 29 shows that in the majority of the cases, the answer is negative, *i.e.*, the membership problem is undecidable.

However, when considering low-level classes such as Rec or Sync, we are able to decide the membership problem (see Example 14, Theorem 28 and Theorem 26). Moreover, the algorithm is itself an answer to the Classifying problem.

6.2.1 Iteration of synchronous relations

Synchronous relations are simpler than rational. However we prove in Theorem 25 that iterated synchronous relations captures in some sense the complexity of Turing Machine computations. This result is based on an encoding of configurations of Turing Machine (*i.e.*, the instantaneous descriptions of a Turing Machine at a fixed time) into words.

We encode a configuration (q, w, h) of a Turing Machine $\mathcal{M}$ in a word $\dot{w}$ in the recognizable language $\Sigma^* \cdot (\Sigma \times Q) \cdot \Sigma^*$ as follows: for each $i \neq h$, $\dot{w}_i = w_i$, and $\dot{w}_h = (w_h, q)$. For example the word $\dot{w} = aabbaaa(b, q)aabbbabb$ encodes the configuration where the tape contains the word $aabbaaabaabbbabb$, the current state is q and the input head is in position 8 (reading a symbol b). Suppose now that $\delta(q, b)$ is equal to $(q', a, -1)$, then the next configuration is composed of tape word $aabbaaaaaabbbabb$, state q' and head position 7. Thus the encoding of this configuration is $aabbaa(a, q')aaabbbabb$.

Because a Turing Machine works locally, the relation consisting of pairs of words encoding successive configurations is synchronous.

Theorem 25. *There exists a synchronous binary relation X such that the iterated relation is non-recursive.*

Proof. The proof is based on the previous encoding technique, by reducing the MEMBERSHIP problem (*i.e.*, does an element (x, y) belong to the iterated relation?) to the Turing-Accessibility problem (*i.e.*, does a configuration c' be accessible from a configuration c?). □

Length-preserving synchronous relations

The general synchronous relations are able to mimic the transitions of any Turing Machine. Subclasses may only capture the transitions of less powerful models of computation. This is the case of length-preserving rational relations, which are synchronous thanks to Theorem 23). The following shows that the MEMBERSHIP problem is decidable for length-preserving rational relations.

Theorem 26. *Let R be a binary length-preserving rational relation, then the compacting of $R^{(*)}$ (see Definition 20) is accepted by a (nondeterministic) linear bounded finite automaton.*

Proof. Let R be a binary length-preserving rational relation over an alphabet Σ, and let $\mathcal{A}$ be a synchronous two-tape automaton accepting R. We build a nondeterministic linear bounded automaton which accepts an input $\mathbf{w} \in (\Sigma^2)^*$ if and only if it is the compacting of some $(w, w') \in R^{(*)}$. The alphabet is $\Sigma \times \Sigma$ thus we can see the tape as two tracks.

Initially the first track contains w, while the second contains w' and remains unchanged during the computation. It works in two alternating modes, TESTING and SIMULATING. The objective of the later mode is to overwrite the first track with one of its possible image by R. The former mode simply tests equality between the two tracks.

TESTING Starting from the rightmost position, the input head scans the entire tape (from right to left), testing whether or not the two tracks contain the same word. If this is the case, the machine accepts, otherwise, it enters the second mode.

SIMULATING Starting from the leftmost position, the machine simulates $\mathcal{A}$. At each position, the machine guesses which letter stands at the same position of the second tape of $\mathcal{A}$, and overwrites it on the first track. It performs the transition of $\mathcal{A}$ and proceeds to the right. After having reached the rightmost position of the input, the machine either enters the TESTING mode if the simulated state of $\mathcal{A}$ is accepting, or halts and rejects otherwise.

□

The following result is a kind of converse of the previous Theorem. It is in the same vein as [75, Corollary 5.5].

Theorem 27. *Let L be a context-sensitive language in Σ^*. Then there exists an alphabet Δ, a synchronous length-preserving relation $R \subseteq \Delta^* \times \Delta^*$ and a recognizable relation $S \subseteq \Delta^* \times \Delta^*$, such that $L = \pi(R^{(*)} \cap S)$, where π is the projection of $\Delta^* \times \Delta^*$ on the first component.*

Proof. We only give a sketch of the proof, since it is essentially an encoding of the configurations of linear bounded automata. Let $\mathcal{A}$ be a linear bounded automaton accepting L, let q_0 be its initial state, Q be its state set and assume the acceptance is done by entering a halting state $h \in Q$. In our model, the head never leaves the space initially occuped by the input word. We define $\Delta = \Sigma \cup (\Sigma \times Q)$. As done above we encode a configuration of $\mathcal{A}$ as a word in $\Sigma^* \cdot (\Sigma \times Q) \cdot \Sigma^*$. The relation R consists all the pairs that encode successive configurations of $\mathcal{A}$, along with the pairs $(aw, (a, q_0) \cdot w)$ which allows to initialize the computation. Then it suffices to define S as the recognizable relation $\Sigma^* \times (\Sigma^* \cdot (\Sigma \times \{h\}) \cdot \Sigma^*)$. □

6.2.2 The unary case

Unary synchronous relations

Theorem 28. *If R is a synchronous binary relation over a unary alphabet, then $R^{(*)}$ is also synchronous.*

Proof. Let R be a synchronous binary relation over the unary alphabet $\{a\}$. Interpreting the two components as non-negative integers, by Property 6.2 we may suppose R is of the form:

$$\bigcup_{i\in I}\{(Mx+a_i, My+b_i)\,,\ C_i(x,y)\} \tag{6.4}$$

for some positive integers M, a_i's and b_i's and some total ordering C_i of 0, x and y.

We build a 2-way finite automaton $\mathcal{A}$, working on the alphabet $\{a,\#\}\times\{a,\#\}$, such that a word $\mathbf{w} = (w,w')$ belongs to $R^{(*)}$ if and only if there exists k such that the k-padding of the compacting of $\mathbf{w}$ (see Section 6.1.3 and Definition 20) is accepted by $\mathcal{A}$. The input of $\mathcal{A}$ is the word which is the compacting of the pair $(w\#^i, w'\#^j)$ with $k = \min(i,j)$ and $|w|+i = |w'|+j$.

By definition, a pair $\mathbf{w} = (w,w')$ belongs to $R^{(*)}$ if and only if there is a sequence $w_0, w_1, \ldots, w_n$ such that $w = w_0$, $w' = w_n$ and for each $0 \le \ell < n$, $(w_\ell, w_{\ell+1}) \in R$. Since $\mathcal{A}$ is an FA, it cannot write, but it will use its head position in order to encode each successive w_i. The simulation proceeds in n steps. At the end of step ℓ, the head is in position $p = |w_\ell|$ and the value $p \bmod M$ is stored in the finite control (this can be done by updating this state component at each move). The step $\ell+1$ consists of choosing one $i \in I$ such that $p = a_i \bmod M$, in Expression (6.4), and to perform the relation $R_i = \{(Mx+a_i, My+b_i) \mid C_i(x,y)\}$. The action to be carried out depends on the predicate C_i. The case study is detailed below, with the implicit assumption that if an action fails then the automaton rejects.

case $0 = x = y$: the head moves a_i cells backwards, verifies that it is at position 1 and then moves b_i cells to the right;

case $0 = x \le y$: the head moves a_i cells backwards, verifies that it is at position 1 and then it moves $b_i + kM$ cells to the right where k is arbitrary;

case $0 = y \le x$: the head moves backward to position 1 and then moves b_i cells to the right;

case $0 \le x = y$: it moves to position $p - a_i + b_i$;

case $0 \le x \le y$: the head moves to position $p - a_i + b_i$ and then moves an arbitrary number of times M cells to the right;

case $0 \le y \le x$: the head moves to position $p - a_i + b_i$ and then moves an arbitrary number of times M cells to the left.

The computation begins with positioning the head on the rightmost occurrence of a on track 1. At the end of each step (including this initial step), it tests whether or not

the head is positioned on the rightmost occurrence of a on track 2. If it the case, it halts and accepts.

By construction, a pair $\mathbf{w}$ belongs to $R^{(*)}$ if and only if there exists k such that the k-padding of $\mathbf{w}$ is accepted by $\mathcal{A}$. We define the homomorphism h which maps $(\#,\#)$ to ϵ and leaves the other symbols unchanged. It suffices to observe that the compacting of the completion of $R^{(*)}$ is the image by h of the language recognized by $\mathcal{A}$. Thus, it is rational, and so $R^{(*)}$ is synchronous. □

Unary rational relations

Here we still study the case of unary rational relations, but we relax the condition that the relation is synchronous. The situation is completely different, even under the stronger assumption that the relation is functional. More surprisingly, the following example known as the Collatz function, is the union of two unary sequential relations (that is in our terminology, accepted by 1-way deterministic transducer).

$$f(n) = \begin{cases} n/2 & \text{if } n \text{ is even} \\ 3n+1 & \text{if } n \text{ is odd} \end{cases} \qquad \text{and} \qquad R = \{(n, f(n)) \mid n \in \mathbb{N}\}$$

It is straightforward that R is rational as the union of the linear sets $R_1 = \{(2n, n) \mid n \in \mathbb{N}\}$ and $R_2 = \{(2n+1, 6n+4) \mid n \in \mathbb{N}\}$.

As of today, it is unknown whether the MEMBERSHIP problem for $R^{(*)}$ is decidable. It is conjectured that for every $n > 0$ there exists k such that $f^k(n) = 1$. In other words:

Conjecture 2. *$\{(n, 1) \mid n \geq 1\}$ is included in $R^{(*)}$.*

We recall the following generalization due to Conway [17].

Definition 21 (Generalized Collatz Functions)**.** *A function g of $\mathbb{N}$ into $\mathbb{N}$ is called a* Collatz function *if there exists an integer m together with non-negative rational numbers $\{a_i, b_i \mid i < m\}$, such that whenever $x \bmod m$, then $g(x) = a_i x + b_i$ is integral.*

Proposition 43. *A Collatz function of $\mathbb{N}$ into $\mathbb{N}$ is rational and total.*

Proof. We use the notation of Definition 21, and we prove that a Collatz function is rational. It suffices to prove that the restriction of the function to the integers equal to $i \bmod m$ is rational. We set $a_i = \frac{p}{q}$ and $b_i = \frac{r}{s}$ and because $g(i)$ and $g(i+m)$ are integral, it follows that $a_i m$ is integral and therefore q divides m.

Let t be the least common multiple of q and s and set $t = \alpha q = \beta s$. We compute

$$g(km+i) = \frac{p}{q}(km+i) + \frac{r}{s} = \frac{\alpha p(km+i) + \beta r}{t} = \frac{\alpha p i + \beta r}{t} + \frac{\alpha p m}{t} k$$

Observe that the first term of the last expression is equal to $g(i)$ and is therefore an integer. Concerning the second term, we have $\frac{\alpha pm}{t} = \frac{pm}{q}$ which is an integer since, as observed above, q divides m.

Finally the graph of the restriction of the function g to the integers equal to $i \bmod m$, *i.e.*, the relation $\#g_i \coloneqq \{(km+i, g(km+i)) \mid k \in \mathbb{N}\}$ is the linear relation:

$$\left(i, \frac{\alpha pi + \beta r}{t}\right) + \left(m, \frac{pm}{q}\right)\mathbb{N}$$

□

The following two results are direct consequences of Theorem 1 and Theorem 2 of [51] and give a negative answer to the MEMBERSHIP problem for unary rational relations.

Theorem 29. *There exists a unary functional rational relation R such that $R^{(*)}$ is non-recursive.*

Theorem 30. *It is undecidable given a unary rational relation R whether or not the relation $a^* \times a$ is included in $R^{(*)}$.*

FracTran: The previous two results are obtained by two reduction steps: the first one from Minsky machines to FRACTRAN *programs*, and the second one, from FRACTRAN programs to Collatz generalized functions. Because FRACTRAN programs are a beautiful illustration of unary rational functions, we recall that they are specified by an ordered finite family of rational numbers such as

$$\mathcal{P} = \left\{\frac{4}{7}, \frac{5}{4}, \frac{12}{11}, \frac{1}{3}, \frac{7}{10}, \frac{13}{5}\right\}$$

The computation on an input n consists of repeatedly performing the following instruction: replace n by nr where r is the first rational number in the list such that nr is integral. For example, if we start with $n = 24$, the next value is $24 \times \frac{5}{4} = 30$. And then, the following value is $30 \times \frac{1}{3} = 10$. The entire sequence is thus: $\{24, 30, 10, 7, 4, 5, 13\}$. From 13 no transition is possible, so the process halts, with *output* value 13.

Given a FRACTRAN program $\mathcal{P} = \left\{\frac{p_1}{q_1}, \ldots, \frac{p_k}{q_k}\right\}$ with each $\frac{p_i}{q_i}$ irreducible, we define the successor relation

$$R_{\mathcal{P}} \coloneqq \{(n, m) \mid m \text{ is obtained from } n \text{ in one step of } \mathcal{P}\}$$

We show that this relation is rational. We build a nondeterministic two-tape finite automaton $\mathcal{A}$ that accepts $R_{\mathcal{P}}$. We fix $0 < i \leq k$. It is easy to build a FA_2 accepting the relation $\{(a^{q_i n}, a^{p_i n}) \mid n \in \mathbb{N}\}$. Moreover, we can restrict the domain of its accepted relation to the recognizable language $\bigcup_{1 \leq j < i} \{a^n \mid n \notin q_j\mathbb{N}\}$, using a state component

of size $\operatorname{lcm}(q_1, \ldots, q_{i-1})$. Hence, we may obtain a 2-tape 1-way automaton $\mathcal{A}_i$ which accepts the relation:

$$\{(a^{q_i n}, a^{p_i n}) \mid n \in \mathbb{N} \text{ and, for each } 1 \leq j < i,\ q_i n \notin q_j \mathbb{N}\}$$

Finally, we simulate any transition of the FRACTRAN program by the 2NFA_2 $\mathcal{A}$, which nondeterministically chooses an index $1 \leq i \leq k$ at the beginning of the computation, and then simulates the automaton $\mathcal{A}_i$. Observe that $\mathcal{A}$ has $\sum_{1 \leq i \leq k} \operatorname{lcm}\{q_j \mid 1 \leq j < i\} \times \max(q_i, p_i)$ states.

Bibliography

[1] Dana Angluin and Douglas N. Hoover. Regular Prefix Relations. *Mathematical Systems Theory*, 17(3):167–191, 1984.

[2] Marcella Anselmo. Two-Way Automata with Multiplicity. In *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, July 16-20, 1990, Proceedings*, pages 88–102, 1990.

[3] Marcella Anselmo and Alberto Bertoni. On 2PFA's and the Hadamard quotient of formal power series. *Bulletin of the Belgian Mathematical Society - Simon Stevin*, 1(2):165–173, 1994.

[4] Eric Bach and Jeffrey Outlaw Shallit. *Algorithmic Number Theory: Efficient algorithms*. Number vol. 1 in Algorithmic Number Theory. MIT Press, 1996.

[5] Félix Baschénis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. One-way definability of sweeping transducers. 2015.

[6] Piotr Berman. A note on sweeping automata. In Jaco de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming*, volume 85 of *Lecture Notes in Computer Science*, pages 91–97. Springer Berlin Heidelberg, 1980.

[7] Piotr Berman and Andrzej Lingas. *On Complexity of Regular Languages in Terms of Finite Automata*. Instytut Podstaw Informatyki Warszawa: Prace IPI PAN. Institute of Computer Science, Polish Academy of Sciences, 1977.

[8] Jean Berstel. *Transductions and context-free languages*. Vieweg+Teubner Verlag, 1979.

[9] Jean Berstel. *Transductions and context-free languages*. Springer-Verlag, 2013.

[10] Jean Berstel and Dominique Perrin. *Theory of codes*. Number 117 in Pure and applied mathematics. Academic Press, Orlando, 1985.

[11] Jean Berstel and C. Reutenauer. *Rational series and their languages.* Springer, [Place of publication not identified], 2012.

[12] Vincent Carnino and Sylvain Lombardy. On Determinism and Unambiguity of Weighted Two-way Automata. In *Proceedings 14th International Conference on Automata and Formal Languages, AFL 2014, Szeged, Hungary, May 27-29, 2014.*, pages 188–200, 2014.

[13] Vincent Carnino and Sylvain Lombardy. Tropical Two-Way Automata. In *Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Proceedings*, pages 195–206, 2014.

[14] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, January 1981.

[15] Christian Choffrut and Bruno Guillon. An Algebraic Characterization of Unary Two-Way Transducers. *Lecture Notes in Computer Science*, pages 196–207, 2014.

[16] Marek Chrobak. Finite automata and unary languages. *Theoretical Computer Science*, 47(0):149 – 158, 1986.

[17] John H. Conway. FRACTRAN: A Simple Universal Programming Language for Arithmetic. In Thomas M. Cover and B. Gopinath, editors, *Open Problems in Communication and Computation*, pages 4–26. Springer New York, New York, NY, 1987.

[18] Karel II. Culik and Juhani Karhumäki. The Equivalence Problem for Single-Valued Two-Way Transducers (on NPDT0L Languages) is Decidable. *SIAM J. Comput.*, 16(2):221–230, 1987.

[19] Pavol Ďuriš, Juraj Hromkovič, José D.P. Rolim, and Georg Schnitger. Las Vegas versus determinism for one-way communication complexity, finite automata, and polynomial-time computations. In Rüdiger Reischuk and Michel Morvan, editors, *STACS 97*, volume 1200 of *Lecture Notes in Computer Science*, pages 117–128. Springer Berlin Heidelberg, 1997.

[20] Roger W. Ehrich and Stephen S. Yau. Two-Way Sequential Transductions and Stack Automata. *Information and Control*, 18(5):404–446, 1971.

[21] Samuel Eilenberg. *Automata, Languages and Machines*, volume A. Academic Press, 1974.

[22] Samuel Eilenberg and Marcel-Paul Schützenberger. Rational sets in commutative monoids. *J. Algebra*, 13:173–191, 1969.

[23] Calvin C. Elgot and Jorge E. Mezei. On Relations Defined by Finite Automata. *IBM Journal*, 10:47–68, 1965.

[24] Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Logic*, 2(2):216–254, April 2001.

[25] Zoltán Ésik and Werner Kuich. Locally closed semirings. *Monatshefte für Mathematik*, 137(1):21–29, 2002.

[26] Emmanuel Filiot, Olivier Gauwin, Pierre-Alain Reynier, and Frédéric Servais. From Two-Way to One-Way Finite State Transducers. abs/1301.5197, 2013.

[27] Rusins Freivalds. Projections of Languages Recognizable by Probabilistic and Alternating Finite Multitape Automata. *Inf. Process. Lett.*, 13(4/5):195–198, 1981.

[28] Carlo A. Furia. A Survey of Multi-Tape Automata. abs/1205.0178, 2012.

[29] Dan Fux. OEIS Foundation Inc. (2011), 2001.

[30] Viliam Geffert. An alternating hierarchy for finite automata. pages 15–36, 2011.

[31] Viliam Geffert, Bruno Guillon, and Giovanni Pighizzini. Two-way automata making choices only at the endmarkers. *Inf. Comput.*, 239:71–86, 2014.

[32] Viliam Geffert, Carlo Mereghetti, and Giovanni Pighizzini. Converting two-way nondeterministic unary automata into simpler automata. *Theoretical Computer Science*, 295(1–3):189 – 203, 2003.

[33] Viliam Geffert, Carlo Mereghetti, and Giovanni Pighizzini. Complementing two-way finite automata. *Information and Computation*, 205(8):1173 – 1187, 2007.

[34] Viliam Geffert and Giovanni Pighizzini. Two-way unary automata versus logarithmic space. *Information and Computation*, 209(7):1016 – 1025, 2011.

[35] Sheila A. Greibach. Hierarchy Theorems for Two-Way Finite State Transducers. *Acta Inf.*, 11:80–101, 1978.

[36] Eitan M. Gurari. The Equivalence Problem for Deterministic Two-Way Sequential Transducers Is Decidable. pages 83–85, 1980.

[37] Kosaburo Hashiguchi. Limitedness theorem on finite automata with distance functions. *Journal of Computer and System Sciences*, 24(2):233–244, April 1982.

[38] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[39] Juraj Hromkovič and Georg Schnitger. Nondeterminism versus Determinism for Two-Way Finite Automata: Generalizations of Sipser's Separation. In JosC.M. Baeten, JanKarel Lenstra, Joachim Parrow, and GerhardJ. Woeginger, editors, *Automata, Languages and Programming*, volume 2719 of *Lecture Notes in Computer Science*, pages 439–451. Springer Berlin Heidelberg, 2003.

[40] Neil Immerman. Number of quantifiers is better than number of tape cells. *Journal of Computer and System Sciences*, 22(3):384 – 406, 1981.

[41] Jānis Kaņeps and Rūsiņš Freivalds. Minimal nontrivial space complexity of probabilistic one- way turing machines. In Branislav Rovan, editor, *Mathematical Foundations of Computer Science 1990*, volume 452, pages 355–361. Springer-Verlag, Berlin/Heidelberg, 1990.

[42] Christos A. Kapoutsis. Small Sweeping 2NFAs Are Not Closed Under Complement. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming*, volume 4051 of *Lecture Notes in Computer Science*, pages 144–156. Springer Berlin Heidelberg, 2006.

[43] Christos A. Kapoutsis. Size Complexity of Two-Way Finite Automata. In Volker Diekert and Dirk Nowotka, editors, *Developments in Language Theory*, volume 5583 of *Lecture Notes in Computer Science*, pages 47–66. Springer Berlin Heidelberg, 2009.

[44] Christos A. Kapoutsis. Two-Way Automata versus Logarithmic Space. In Alexander Kulikov and Nikolay Vereshchagin, editors, *Computer Science – Theory and Applications*, volume 6651 of *Lecture Notes in Computer Science*, pages 359–372. Springer Berlin Heidelberg, 2011.

[45] Christos A. Kapoutsis. Nondeterminism is essential in small two-way finite automata with few reversals. *Information and Computation*, 222(0):208 – 227, 2013.

[46] Christos A. Kapoutsis and Giovanni Pighizzini. Two-Way Automata Characterizations of L/poly versus NL. In EdwardA. Hirsch, Juhani Karhumäki, Arto Lepistö, and Michail Prilutskii, editors, *Computer Science – Theory and Applications*, volume 7353 of *Lecture Notes in Computer Science*, pages 217–228. Springer Berlin Heidelberg, 2012.

[47] Richard M. Karp and Richard J. Lipton. Turing machine that take advice. 28(1-2):191–209, 1982.

[48] Daniel Kirsten. Distance desert automata and the star height problem. *RAIRO - Theoretical Informatics and Applications*, 39(3):455–509, July 2005.

[49] Felix Klaedtke and Harald Rueß. Monadic Second-Order Logics with Cardinalities. pages 681–696, 2003.

[50] Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *International Journal of Algebra and Computation*, 04(03):405–425, September 1994.

[51] Stuart A. Kurtz and Janos Simon. The Undecidability of the Generalized Collatz Problem. pages 542–553, 2007.

[52] Michel Latteux, David Simplot, and Alain Terlutte. Iterated Length-Preserving Rational Transductions. pages 286–295, 1998.

[53] Hing Leung. On the topological structure of a finitely generated semigroup of matrices. In *Semigroup Forum*, volume 37, pages 273–287. Springer, 1988.

[54] Hing Leung and Viktor Podolskiy. The limitedness problem on distance automata: Hashiguchi's method revisited. *Theoretical Computer Science*, 310(1-3):147–158, January 2004.

[55] Sylvain Lombardy. Two-Way Representations and Weighted Automata. August 2015.

[56] George H. Mealy. A method for synthesizing sequential circuits. *The Bell System Technical Journal*, 34(5):1045–1079, September 1955.

[57] Albert R. Meyer and Michael J. Fisher. Economy of description by automata, grammars, and formal systems. 1971.

[58] Silvio Micali. Two-way deterministic finite automata are exponentially more succinct than sweeping automata. *Information Processing Letters*, 12(2):103 – 105, 1981.

[59] Edward F. Moore. Gedanken-experiments on sequential machines. In *Automata studies*, Annals of mathematics studies, no. 34, pages 129–153. Princeton University Press, Princeton, N. J., 1956.

[60] R. Parikh. On Context-Free Languages. *J. ACM*, 13(4):570–581, 1966.

[61] Azaria Paz. Fuzzy star functions, probabilistic automata, and their approximation by nonprobabilistic automata. *Journal of Computer and System Sciences*, 1(4):371–390, 1967.

[62] Azaria Paz and Werner Rheinboldt. *Introduction to probabilistic automata*. Computer science and applied mathematics. Academic Press, New York, 1971.

[63] Michael Rabin and Dana Scott. Finite automata and their decision problems. *IBM J.Res.Develop.*, 3:114–125, 1959.

[64] Jorge L. Ramírez Alfonsín. *The Diophantine Frobenius Problem*. Oxford Lecture Series in Mathematics and Its Applications. OUP Oxford, 2005.

[65] K. Reinhardt and E. Allender. Making Nondeterminism Unambiguous. *SIAM Journal on Computing*, 29(4):1118–1131, 2000.

[66] Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

[67] 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 '78, pages 275–286, New York, NY, USA, 1978. ACM.

[68] Arto Salomaa and Matti Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Springer New York, New York, NY, 1978.

[69] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177 – 192, 1970.

[70] Marcel-Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2-3):245–270, September 1961.

[71] Marcel-Paul Schützenberger. Certain elementary families of automata. pages 139–153, New York, NY, USA, 1962. Polytechnic Press of Polytechnic Inst. of Brooklyn, Brooklyn.

[72] John C. Shepherdson,. The reduction of two-way automata to one-way automata. *IBM J.Res.Develop.*, 3:198–200, 1959.

[73] Michael Sipser. Halting space-bounded computations. *Theoretical Computer Science*, 10(3):335 – 338, 1980.

[74] Michael Sipser. Lower Bounds on the Size of Sweeping Automata. *J. Comput. Syst. Sci.*, 21(2):195–202, 1980.

[75] Alain Terlutte and David Simplot. Iteration of rational transductions. 34(2):99–130, 2000.

Index