Register Transducers Are Marble Transducers

Gaëtan Douéneau-Tabot
IRIF, Université de Paris, France
doueneau@irif.fr

Emmanuel Filiot
Université libre de Bruxelles and F.R.S.-FNRS, Belgium
efiliot@ulb.ac.be

Paul Gastin
LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France
paul.gastin@ens-paris-saclay.fr

Abstract

Deterministic two-way transducers define the class of regular functions from words to words. Alur and Cerný introduced an equivalent model of transducers with registers called copyless streaming string transducers. In this paper, we drop the "copyless" restriction on these machines and show that they are equivalent to two-way transducers enhanced with the ability to drop marks, named "marbles", on the input. We relate the maximal number of marbles used with the amount of register copies performed by the streaming string transducer. Finally, we show that the class membership problems associated with these models are decidable. Our results can be interpreted in terms of program optimization for simple recursive and iterative programs.

2012 ACM Subject Classification Theory of computation → Transducers

Keywords and phrases streaming string transducer, two-way transducer, marbles, pebbles

Digital Object Identifier 10.4230/LIPIcs.MFCS.2020.29

Related Version A full version of the paper is available at `https://arxiv.org/abs/2005.01342`.

Funding *Emmanuel Filiot*: associate researcher at F.R.S.-FNRS, and supported by the MIS project F451019F (F.R.S.-FNRS).

1 Introduction

Regular languages have been a cornerstone of theoretical computer science since the 1950's. They can be described by several equivalent models such as deterministic, non-deterministic, or two-way (the reading head can move in two directions) finite automata [16].

A natural extension consists in adding an output mechanism to finite automata. Such machines, called *transducers*, describe functions from words to words (or relations when non-deterministic). In this case, the landscape generally becomes more complex, as noted in 1967 by D. Scott: « the functions computed by the various machines are more important - or at least more basic - than the sets accepted by these device » [15]. Furthermore, transducers provide a natural way to model simple programs that produce outputs.

Regular functions and copyless register transducers. The particular model of *two-way transducer* consists in a two-way automaton enhanced with an output function. It describes the class of *regular functions* which has been intensively studied for its natural properties: closure under composition [4], logical characterization by monadic second-order transductions [8], decidable equivalence problem [11], etc.

In [1], the equivalent model of *copyless streaming string transducer* (SST) is described. This machine processes its input in a one-way fashion, while storing pieces of their output in a finite set of registers: it is at the same time simpler (since it reads the input only once) and more complex (since it needs registers) than a two-way transducer. Registers are updated by simple concatenation operations. However, the content of a register can never be duplicated ("copyless"), which allows one to implement the model efficiently for a streaming use.

Copyful register transducers. Regular functions remain quite limited in terms of expressiveness, since the size of the output can be at most linear in the input's. In this paper, we study the class of functions computed by copyful SSTs, i.e. register transducers that can duplicate their registers. With this model, it becomes possible to produce outputs that have a polynomial, or even exponential, size in the input's. Meanwhile, it preserves many "good properties" of regular functions, such as decidability of the equivalence problem [10].

Marble transducers. Our first objective is to extend the aforementioned correspondence between copyless SSTs and two-way transducers, by providing a model of transducer without registers that is equivalent to copyful SSTs. For this, we define *marble transducers* (introduced for trees in [9]) and show equivalence. This model consists in a two-way transducer that can drop several marks ("marbles") on its input, following a stack discipline. Indeed, new marbles can only be dropped on the left of the positions already marked. Informally, our result shows that copyful SSTs correspond to some recursive algorithms (hence the stack).

A very natural way to restrict the power of marble transducers is to bound the number of marks that can be used. We define *k-marble transducer* that can use at most k marks. Intuitively, it corresponds to iterative algorithms with "for" loops, such that the maximal depth of nested loops is $k+1$. In particular, 0-marble transducers are exactly two-way transducers (since they use no marbles). Whereas marble transducers in general can have an exponential execution time, a k-marble transducer runs in polynomial time, more precisely $\mathcal{O}(n^{k+1})$ when n is the input's length. Hence, it produces outputs of size $\mathcal{O}(n^{k+1})$.

As a second main result, we show that *k-marble transducers* are equivalent to a model of *k-layered SSTs*, i.e. SSTs with hierarchical restrictions on their copies. In particular for $k = 0$, we recover the correspondence between two-way transducers and copyless SSTs.

Optimization and class membership problems. As evoked above, our models of marble transducers have at most an exponential complexity (or "execution time"), but it becomes polynomial if we restrict the number of marks used. In practice, a natural question is that of optimization: can we transform an exponential algorithm in a polynomial equivalent one? Can we reach the smallest possible complexity? Having a tool to optimize programs is of foremost interest since it allows to write naive algorithms without worrying about the complexity. Due to well-known undecidability statements, optimizing any algorithm is hopeless in theory, thus having results for a "regular" kernel is already interesting.

From a theoretical point of view, the optimization problem is known as (effective) *class membership problem*. It instantiates as follows: given a function computed by a marble transducer, can it be computed by a k-marble transducer? An easy lower bound is given by the size of the output, since for instance we cannot produce a string of size $\Omega(n^2)$ with a two-way transducer. As shown in our third main result, it is in fact a sufficient criterion to decide membership: a function from our class is computable with k marbles if and only if it grows in $\mathcal{O}(n^{k+1})$ (and this property is decidable). This result shows the robustness of our k-marble model, since a simple syntactical restriction is sufficient to describe a semantical property. Its proof is the most involved of this paper; it uses the correspondence with SSTs.

Similar optimization results have recently been obtained in [13] for the class of *polyregular functions*, defined using k-pebble transducers [3] (an extension of k-marble). Interestingly, their conclusion is very similar to ours, that is: an output of size $\mathcal{O}(n^k)$ can always be produced using k nested loops. We shall discuss in conclusion how our results both refine and extend theirs. Contrary to us, the equivalence problem is an open problem for their model. Furthermore, pebble transducers have never been related to a class of streaming algorithms, contrary to what we show for marbles.

Outline. After recalling in Section 2 the basic definitions of two-way transducers and SSTs, we present in Section 3 the model of marble transducer and show equivalence. We then study in Section 4 the case of k-marble transducers and relate them to specific SSTs. Finally, we solve in Section 5 the class membership problems associated with these models. Due to space constraints, several proofs are only sketched; we chose to focus on the proofs of the last section since they describe an algorithm for program optimization.

2 Preliminaries

We denote by $\mathbb{N}$ the set of nonnegative integers. Capital letters A and B are used to denote alphabets, i.e. finite sets of letters. If $w \in A^*$ is a word, let $|w| \in \mathbb{N}$ be its length, and for $1 \leq m \leq |w|$ let $w[m]$ be its m-th letter. The empty word is denoted ε. If $1 \leq m \leq n \leq |w|$, let $w[m{:}n] = w[m]w[m+1] \cdots w[n]$. We assume that the reader is familiar with the basics of automata theory, and in particular the notions of one-way and two-way deterministic automata (see e.g. [16]).

Two-way transducers. A deterministic two-way transducer is a deterministic two-way automaton enhanced with the ability to produce outputs along its run. The class of functions described by these machines is known as "regular functions" [4, 8].

▶ **Definition 1.** *A (deterministic) two-way transducer $\mathcal{T} = (A, B, Q, q_0, \delta, \lambda, F)$ consists of:*
- *an input alphabet A, an output alphabet B;*
- *a finite set of states Q, with an initial state $q_0 \in Q$ and a set of final states $F \subseteq Q$;*
- *a (partial) transition function $\delta \colon Q \times (A \uplus \{\vdash, \dashv) \to Q \times \{\triangleleft, \triangleright\}$;*
- *a (partial) output function $\lambda \colon Q \times (A \uplus \{\vdash, \dashv\}) \to B^*$ with same domain as δ.*

When given as input a word $w \in A^*$, the two-way transducer disposes of a read-only input tape containing $\vdash w \dashv$. The marks $\vdash$ and $\dashv$ are used to detect the borders of the tape, by convention we denote them as positions 0 and $|w|+1$ of w.

Formally, a *configuration* over $\vdash w \dashv$ is a tuple (q, m) where $q \in Q$ is the current state and $0 \leq m \leq |w|+1$ is the position of the reading head. The *transition relation* $\to$ is defined as follows. Given a configuration (q, m), let $(q', \star) := \delta(q, w[m])$. Then $(q, m) \to (q', m')$ whenever either $\star = \triangleleft$ and $m' = m - 1$ (move left), or $\star = \triangleright$ and $m' = m + 1$ (move right), with $0 \leq m' \leq |w| + 1$. A *run* is a sequence of configurations following $\to$. Accepting runs are those that begin in $(q_0, 0)$ and end in a configuration of the form $(q, |w| + 1)$ with $q \in F$.

The (partial) function $f \colon A^* \to B^*$ computed by the machine is defined as follows. If there exists an accepting run on $\vdash w \dashv$, then it is unique and $f(w)$ is the concatenation of all the $\lambda(q, w[m])$ along the transitions of this run. Otherwise $f(w)$ is undefined.

▶ **Example 2.** Let reverse: $A^* \to A^*$ be the function that maps a word *abac* to its mirror image *caba*. It can be performed by a two-way transducer that first goes to the right symbol $\dashv$, and then reads w from right to left while outputting the letters.

Streaming string transducers. Informally, a *streaming string transducer* [1] is a one-way deterministic automaton with a finite set $\mathfrak{X}$ of registers that store strings over the output alphabet B. These registers are modified using *substitutions*, i.e. mappings $\mathfrak{X} \to (B \uplus \mathfrak{X})^*$. We denote by $\mathfrak{S}_{\mathfrak{X}}^B$ the set of these substitutions. They can be extended morphically from $(B \uplus \mathfrak{X})^*$ to $(B \uplus \mathfrak{X})^*$ by preserving the elements of B. As explained in Example 3, they can be composed by setting $(s_1 \circ s_2)(x) := s_1(s_2(x))$ for $x \in \mathfrak{X}$.

▶ **Example 3.** Let $\mathfrak{X} = \{x, y\}$ and $B = \{b\}$. Consider the substitutions $s_1 := x \mapsto b, y \mapsto bxyb$ and $s_2 := x \mapsto xb, y \mapsto xy$, then $s_1 \circ s_2(x) = s_1(xb) = bb$ and $s_1 \circ s_2(y) = s_1(xy) = bbxyb$.

▶ **Definition 4.** *A* streaming string transducer *(SST) $\mathcal{T} = (A, B, Q, \mathfrak{X}, q_0, \iota, \delta, \lambda, F)$ is:*

- *an input alphabet A and an output alphabet B;*
- *a finite set of states Q with an initial state $q_0 \in Q$;*
- *a finite set $\mathfrak{X}$ of registers;*
- *an initial function $\iota \colon \mathfrak{X} \to B^*$;*
- *a (partial) transition function $\delta \colon Q \times A \to Q$;*
- *a (partial) register update function $\lambda \colon Q \times A \to \mathfrak{S}_{\mathfrak{X}}^B$ with same domain as δ;*
- *a (partial) output function $F \colon Q \to (\mathfrak{X} \cup B)^*$.*

This machine defines a (partial) function $f \colon A^* \to B^*$ as follows. Let us fix $w \in A^*$. If there is no accepting run of the *one-way* automaton $(A, Q, q_0, \delta, \mathrm{dom}(F))$ over w, then $f(w)$ is undefined. Otherwise, let $q_m := \delta(q_0, w[1{:}m])$ be the m-th state of this run. We define for $0 \leq m \leq |w|$, $\mathcal{T}^{w[1:m]} \in \mathfrak{S}_{\mathfrak{X}}^B$ ("the values of the registers after reading $w[1{:}m]$") as follows:

- $\mathcal{T}^{w[1:0]}(x) = \iota(x)$ for all $x \in \mathfrak{X}$;
- for $1 \leq m \leq |w|$, $\mathcal{T}^{w[1:m]} := \mathcal{T}^{w[1:(m-1)]} \circ \lambda(q_m, w[m])$. This formula e.g. means that if $\mathcal{T}^{w[1:(m-1)]}(x) = ab$ and $\lambda(q_m, w[m])(x) = xx$, then $\mathcal{T}^{w[1:m]}(x) = abab$.

In this case, we set $f(w) := \mathcal{T}^w(F(q_{|w|})) \in B^*$. In other words, we combine the final values of the registers following the output function.

▶ **Example 5.** The reverse of Example 2 can be computed by an SST with one state and one register x. When seeing a letter a, the SST updates $x \mapsto ax$ (a is added in front of x).

▶ **Example 6.** Consider the function $\mathsf{exp} \colon a^n \mapsto a^{2^n}$. It is computed by an SST with one register x initialized to a and updated $x \mapsto xx$ at each transition.

The function exp of Example 6 cannot be computed by a deterministic two-way transducer. Indeed, a two-way transducer computing a function f has only $|Q|(|w| + 2)$ possible configurations on input w, therefore we must have $|f(w)| = \mathcal{O}(|w|)$.

In order to make two-way transducers and SSTs coincide, the solution of [1] is to forbid duplications of registers. A substitution $\sigma \in \mathfrak{S}_{\mathfrak{X}}^B$ is said to be *copyless* if each register $x \in \mathfrak{X}$ appears at most once in the whole set of words $\{\sigma(x) \mid x \in \mathfrak{X}\}$. The substitution s_1 of Example 3 is copyless whereas s_2 is not. An SST is said to be copyless whenever it uses only copyless substitutions for the $\lambda(q, a)$; the SST of Example 5 is so.

▶ **Theorem 7** ([1, 5]). *Two-way transducers and copyless SSTs describe the same class of functions ("regular functions"). The right to left conversion is effective in* PTIME*, and the converse one in* EXPTIME*.*

▶ Remark 8. For the complexities, the "size" of the machines is that of a reasonable representation. For a two-way transducer, it is roughly the total size of the outputs that label its transitions. For an SST, it is the total size of its substitutions.

3 Marble transducers and streaming string transducers

As evoked in the introduction, our first goal is to extend Theorem 7 by describing a machine without registers that captures the expressiveness of SSTs with copies. For this purpose, we shall use a variant of two-way transducers that can drop/lift several marks on their input. However, the use of marks has to be strongly restricted so that the machine is not too expressive (see e.g. [12]). The model we propose here, named *marble transducer* after [9], can drop marks ("marbles") of different colors and the reading head has to stay on the left of marbles: a stack of marks is stored on the input and if the machine wants to move forward from a position where there is a marble, it has to remove it first.

▶ **Definition 9.** *A* (deterministic) marble transducer $\mathcal{T} = (A, B, Q, C, q_0, \delta, \lambda, F)$ *consists of:*
- *an input alphabet A;*
- *a finite set of states Q with an initial state $q_0 \in Q$ and a set $F \subseteq Q$ of final states;*
- *a finite set of marble colors C;*
- *a transition function $\delta \colon Q \times (A \uplus \{\vdash, \dashv\}) \times (C \uplus \{\varnothing\}) \to Q \times (\{\triangleleft, \triangleright, \mathsf{lift}\} \uplus \{\mathsf{drop}_c \mid c \in C\})$ such that $\forall q \in Q, a \in A, c \in C$ we have $\delta(q, a, c) \in Q \times \{\triangleleft, \mathsf{lift}\}$ (we cannot move right nor drop another marble when we see a marble).*
- *an output function $\lambda \colon Q \times (A \uplus \{\vdash, \dashv\}) \times (C \uplus \{\varnothing\}) \to B^*$ with same domain as δ.*

As for two-way transducers, the symbols $\vdash$ and $\dashv$ are used to denote the borders of the input. A *configuration* over $\vdash w \dashv$ is a tuple (q, m, π) where $q \in Q$ is the current state, m is the position of the reading head, and $\pi = (c_\ell, m_\ell) \cdots (c_1, m_1)$ is the stack of the positions and colors of the ℓ marbles dropped (hence $\ell \geq 0$ and $0 \leq m \leq m_\ell < \cdots < m_1 \leq |w| + 1$ and $c_i \in C$). An example of configuration is depicted in Figure 1.

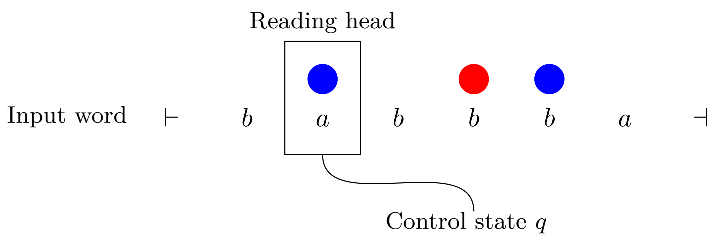

Figure 1 Configuration $(q, 2, ({\color{blue}\bullet}, 2)({\color{red}\bullet}, 5)({\color{blue}\bullet}, 6))$ of a marble transducer over *babbba*. Note that allowed transitions starting from this configuration are either lift or $\triangleleft$.

The *transition relation* $\to$ of $\mathcal{T}$ is defined as follows. Given a configuration (q, m, π), let $k := c$ if $\pi = (c, m) \cdots$ (marble c in position m) and $k := \varnothing$ otherwise (no marble in m). Let $(q', \star) := \delta(q, w[m], k)$. Then $(q, m, \pi) \to (q', m', \pi')$ whenever one of the following holds:
- move left: $\star = \triangleleft$, $m' = m - 1 \geq 0$ and $\pi = \pi'$;
- move right: $\star = \triangleright$, $m' = m + 1 \leq |w| + 1$ and $\pi = \pi'$ (only when $k = \varnothing$);
- lift a pebble: $\star = \mathsf{lift}$, $m = m'$ and $\pi = (c, m)\pi'$ (only when $k \neq \varnothing$);
- drop a pebble: $\star = \mathsf{drop}_c$, $m = m'$, $\pi' = (c, m)\pi$ (only when $k = \varnothing$).

The notion of *run* is defined as usual with $\to$. Accepting runs are finite runs that begin in $(q_0, 0, \varepsilon)$ and end in a configuration of the form $(q, |w| + 1, \varepsilon)$ with $q \in F$.

The partial function $f \colon A^* \to B^*$ computed by the machine is defined as follows. If there exists an accepting run on $\vdash w \dashv$, then it is unique and $f(w)$ is the concatenation of all the $\lambda(q, w[m], k)$ along the transitions of this run. Otherwise $f(w)$ is undefined.

▶ Remark 10. In case no marbles are used, the machine is simply a two-way transducer.

As they can have an exponential number of configurations, marble transducers can produce outputs of exponential size, like SSTs (see Example 11).

▶ **Example 11.** The function $\mathsf{exp}\colon a^n \mapsto a^{2^n}$ of Example 6 is computed by a marble transducer with $C = \{0, 1\}$. The idea is to use marbles to count in binary on the input a^n. We first write 0^n on the input, then we increment it to 10^{n-1}, then 010^{n-2}, 110^{n-2}, …, 1^n (there are 2^n numbers). These increments can be done while preserving the stack discipline of the marbles: we move right and lift while we see 1's, when a 0 is met we replace it by 1, then we move left dropping 0's. Initially and after each increment, we output an a to produce a^{2^n}.

We are now ready to state our first generalization of Theorem 7. For the complexity, the size of a marble transducer is that of its output labels plus its number of marbles.

▶ **Theorem 12.** *Marble transducers and SSTs describe the same class of functions. The right to left conversion is effective in* PTIME*, and the converse one in* EXPTIME*.*

Proof sketch. Let $\mathcal{T} = (A, B, Q, C, q_0, \delta, \lambda, F)$ be a marble transducer. We simulate it with an SST by adapting the classical reduction from two-way automata to one-way automata via *crossing sequences* [16]. In position m of input $w \in A^*$, the SST keeps track of the right-to-right runs of the marble transducer on the prefix $w[1{:}m]$, see Figure 2. This is stored as a function $f\colon Q \to Q \uplus \{\perp\}$ such that when $\mathcal{T}$ starts in configuration (m, q, ε), the first time it reaches position $m+1$, $\mathcal{T}$ is in state $f(q)$ (with $\perp$ if $\mathcal{T}$ never reaches $m+1$, blocking or looping on the prefix $w[1{:}m]$). Hence, we have $(m, q, \varepsilon) \to^* (m+1, f(q), \varepsilon)$. We also store the state reached after the left-to-right run on the prefix $w[1{:}m]$: $(0, q_0, \varepsilon) \to^* (m+1, \mathrm{first}, \varepsilon)$. This abstraction is updated at each new letter by considering the transitions it induces.

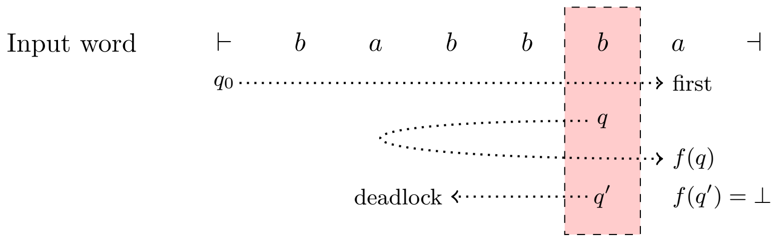

Figure 2 Crossing sequences in a marble transducer.

For instance, suppose that after the prefix $w[1{:}m]$ the abstraction f is such that $f(q_2) \neq \perp$. Assume also that $\delta(q, a, \varnothing) = (q_1, \mathsf{drop}_c)$, $\delta(q_1, a, c) = (q_2, \triangleleft)$, $\delta(f(q_2), a, c) = (q_3, \mathsf{lift})$, $\delta(q_3, a, \varnothing) = (q_2, \triangleleft)$ and $\delta(f(q_2), a, \varnothing) = (q', \triangleright)$. Then, after the prefix $w[1{:}m]a$ the abstraction f' is such that $f'(q) = q'$, see Figure 3. Notice that the right-to-right run from q_2 to $f(q_2)$ on the prefix $w[1{:}m]$ is used twice, first with marble c dropped on position $m+1$ and then without any marble on position $m+1$.

For each state $q \in Q$, we also use a register x_q to store the output produced by the right-to-right run $(m, q, \varepsilon) \to^* (m+1, f(q), \varepsilon)$ of $\mathcal{T}$ on the prefix $w[1{:}m]$ (provided $f(q) \neq \perp$). Due to the presence of marbles, the same right-to-right run can be executed multiple times, but with different stack of marbles (thus avoiding looping behaviors). These multiple similar executions are handled using copies in the SST model when updating the registers $(x_q)_{q \in Q}$. Continuing the example above, the update of register x_q is given by

$$x_q \mapsto \lambda(q, a, \varnothing)\lambda(q_1, a, c)x_{q_2}\lambda(f(q_2), a, c)\lambda(q_3, a, \varnothing)x_{q_2}\lambda(f(q_2), a, \varnothing)\,.$$

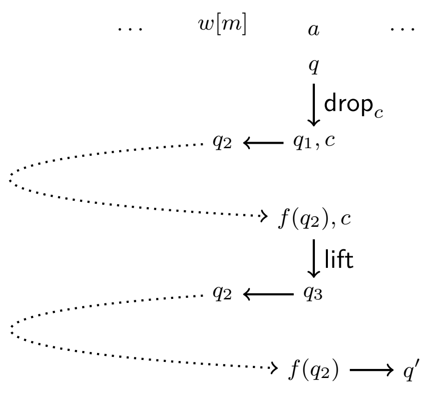

Figure 3 Example of run starting from configuration $(q, m+1, \varepsilon)$.

There is also a register x storing the output of the left-to-right run $(0, q_0, \varepsilon) \to^* (m+1, \text{first}, \varepsilon)$ and x is updated similarly. When the whole word is read, we can recombine all pieces of information in order to obtain the output of the marble transducer (when it accepts). The construction is similar to that of the update, by stitching the different pieces of the run.

From SSTs to marble transducers, we execute a recursive algorithm to compute the contents of the registers, and implement it with a marble transducer using the marbles to code the stack of calls (recursive calls are done from right to left, which corresponds to the orientation of the marble stack). ◀

▶ Remark 13. Considering the domains, we note that marble automata (transducers without the output) recognize exactly regular languages. Indeed an SST is easily seen to have a regular domain, since it is an extended one-way automaton. See [9] for another proof of this fact.

4 Bounded number of marbles

A natural restriction of our marble transducers is to bound the number of marbles that can be simultaneously present in the stack. Indeed, if a machine uses at most k marbles, it has $\mathcal{O}(|w|^{k+1})$ possible configurations on input w. As a consequence, *it performs its computation in polynomial time*, and the function f it computes is such that $|f(w)| = \mathcal{O}(|w|^{k+1})$. In particular, the exponential behaviors of Example 11 are no longer possible.

▶ **Definition 14.** *A* k-marble transducer *is a marble transducer such that every accessible configuration (i.e. reachable from the initial configuration) has a stack of at most k marbles.*

This definition is semantical, but it can easily be described in a syntactical way by storing the (bounded) number of marbles that are currently dropped on the input.

▶ Remark 15. 0-marble transducers are exactly two-way transducers.

As special instances of 1-marble transducers we get programs with 2 nested for loops of shape `for i in {1,...,|w|} ( for j in {1,...,i} (...)  )`. Indeed, the outer index i corresponds to the marble, and the inner j to the reading head that cannot move on the right of i. However, this interpretation does not take the two-way moves into account.

▶ **Example 16.** Consider the function mul: $w\#0^n \mapsto (w\#)^n$ that produces n copies of $w\#$. It can be realized by a 1-marble transducer that successively drops the marble from first to last 0, and each time scans and outputs $w\#$.

▶ Remark 17. For a k-marble transducer, it is enough to have one marble color. Indeed, the colors of the marbles dropped form a finite information that can be encoded in the states.

The correspondence given by Theorem 12 does not take the number of marbles into account. In particular, it does not produce a copyless SST if we begin with a 0-marble transducer. We shall now give a more precise statement that relates the maximal number of marbles to the number of "copy layers" in the SST, as defined below.

▶ **Definition 18.** *An SST $(A, B, Q, \mathfrak{X}, q_0, \iota, \delta, \lambda, F)$ is said to be k-layered if $\mathfrak{X}$ has a partition of the form $\mathfrak{X}_0, \dots, \mathfrak{X}_k$, such that $\forall q \in Q$, $\forall a \in A$, the following are true:*

- *$\forall 0 \leq i \leq k$, only registers from $\mathfrak{X}_0, \dots, \mathfrak{X}_i$ appear in $\{\lambda(q,a)(x) \mid x \in \mathfrak{X}_i\}$;*
- *$\forall 0 \leq i \leq k$, each register $y \in \mathfrak{X}_i$ appears at most once in $\{\lambda(q,a)(x) \mid x \in \mathfrak{X}_i\}$.*

Note that 0-layered SSTs are exactly copyless SSTs (only the second condition is useful). For $k \geq 1$, Definition 18 forces each layer $\mathfrak{X}_i$ to be "copyless in itself", but it can do many copies of deeper layers ($\mathfrak{X}_j$ for $j < i$). This update mechanism is depicted in Figure 4; it mainly avoids copying twice a register in itself.

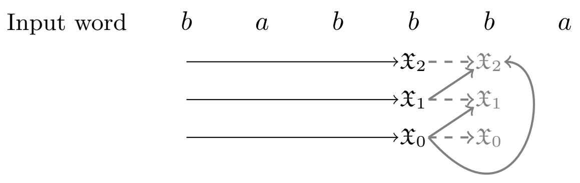

Figure 4 Update of the registers in a 2-layered SST.

▶ **Example 19.** The function mul: $w\#0^n \mapsto (w\#)^n$ (Example 16) can be computed by a 1-layered SST with $\mathfrak{X}_0 = \{x\}$ and $\mathfrak{X}_1 = \{y\}$ as follows. First, when reading $w\#$, it stores $w\#$ in x, while keeping ε in y. Then, each time it sees a 0, it applies $x \mapsto x, y \mapsto xy$.

We now provide a fine-grained correspondence between marbles and registers. Our result indeed extends Theorem 7, which corresponds to the case $k = 0$.

▶ **Theorem 20.** *For all $k \geq 0$, k-marble transducers and k-layered SSTs describe the same class of functions. The right to left conversion is effective in* PTIME.

Proof sketch. To convert a k-layered SST in a k-marble transducer, we adapt the transformation of Theorem 12 in order to use no more than k marbles. The idea is to write only the recursive calls that correspond to the copy of a register, the others being kept implicitly. The PTIME complexity is obtained by adapting the construction of [5]. For the converse implication, we first transform the k-marble transducer in an SST using Theorem 12. Since the function f computed by this SST is such that $|f(w)| = \mathcal{O}(|w|^{k+1})$, we use Lemmas 34 and 36 in order to build a k-layered SST for f. A large amount of additional work is required to obtain these results, and it is the purpose of Section 5. ◀

5 Membership problems

It is clear that a k-marble transducer is a particular case of $(k+1)$-marble transducer, which is a particular case of marble transducer (without restrictions). In other words, the classes of functions they define are included in each other. More precisely, these classes describe a strict hierarchy of increasing expressiveness, since k-marble transducers can only describe functions such that $|f(w)| = \mathcal{O}(|w|^{k+1})$ (see Examples 11 and 21 for separation).

▶ **Example 21.** The function $\mathsf{pow}^k \colon a^n \mapsto a^{n^k}$ can be computed with k marbles, but not less (since $|\mathsf{pow}^k(w)|$ is $\Omega(|w|^k)$). Let us explain the computation of pow^2 with 1 marble on input a^n. We first drop the marble on position $n-1$, go to 1, and move forward from 1 to $n-1$ while outputting aa at each transition. Then we lift the marble and drop it on $n-2$, and perform the same outputs from 1 to $n-2$, etc. At the end of this procedure, we have output $a^{2((n-1)+\cdots+1)} = a^{n^2-n}$. It remains to output a^n by reading the input once.

▶ Remark 22. Generalizing the construction of Example 21, we can show that if $P \in \mathbb{N}[X]$ is a polynomial of degree $k \geq 0$, then $a^n \mapsto a^{P(n)}$ is computable with k marbles (but not less).

A natural problem when considering a hierarchy is that of *membership*: given a function in some class, does it belong to a smaller one? The objective of this section is to provide a positive answer by showing Theorem 23 below.

▶ **Theorem 23.** *Given a function f described by a marble transducer, it is decidable in* EXPTIME *whether f can be computed by a k-marble transducer for some $k \geq 0$. In that case, we can compute the least possible $k \geq 0$ and build a k-marble transducer for f.*

Proposition 25 below is the key element for the proof, and is also interesting in itself. Indeed, it states that *a polynomial-growth function can always be computed in polynomial time!* In other words, f described by a marble transducer is computable with k marbles if and only if $|f(w)| = \mathcal{O}(|w|^{k+1})$. Given $f \colon A^* \to B^*$, let $|f| \colon A^* \to \mathbb{N}, w \mapsto |f(w)|$.

▶ **Definition 24.** *Let $g \colon A^* \to \mathbb{N}$, we say that g has:*

- exponential growth, *if $g(w) = \mathcal{O}(2^{\mathcal{O}(|w|)})$ and there exists an infinite set $L \subseteq A^*$ such that $g(w) = 2^{\Omega(|w|)}$ when $w \in L$;*
- k-polynomial growth *for $k \geq 0$, if $g(w) = \mathcal{O}(|w|^k)$ and there exists an infinite set $L \subseteq A^*$ such that $g(w) = \Omega(|w|^k)$ when $w \in L$;*

▶ **Proposition 25.** *Let $f \colon A^* \to B^*$ be a total function computed by a marble transducer. Then exactly one of the following is true:*

- *$|f|$ has exponential growth, and f is not computable with k marbles for any $k \geq 0$;*
- *$|f|$ has $(k+1)$-polynomial growth for some $k \geq 0$, and f is computable with k marbles and k is the least possible number of marbles;*
- *$|f|$ has 0-polynomial growth (i.e. a finite image), and f is computable with 0 marbles.*

Moreover, these three properties are decidable in EXPTIME.

▶ Remark 26. If f has a finite image, it is a trivial “step function”: $\mathrm{dom}(f)$ is a finite union $\bigcup_i L_i$ of regular languages such that f is constant on each L_i.

The rest of this section is devoted to the proof of these results. By Theorem 12, we first convert our marble transducer in an SST (in EXPTIME), and only reason about SSTs in the sequel. In fact, considering SSTs is crucial: contrary to a marble transducer, an SST performs only one pass on its input, which makes it possible to apply pumping-like arguments for understanding the asymptotic growth of the outputs. Some proof techniques used below are inspired from [10] which only considers deciding 1-polynomial growth of SSTs.

5.1 Simplification of SST

An SST is said to be *total* whenever its transition, update and output functions are total. We shall assume that our machine is so. Indeed, it can be completed like a finite automaton, by outputting ε when out of the domain. Furthermore, this operation does not modify the asymptotic growth of the computed function.

We say that an SST is *simple* if it is total, it has a single state (i.e. $Q = \{q_0\}$), and its substitutions and output do not use letters (i.e. $\lambda\colon Q \times A \to \mathfrak{S}_{\mathfrak{X}}^{\varnothing}$ and $F\colon Q \to \mathfrak{X}^*$). To simplify the notations, we write $(A, B, \mathfrak{X}, \iota, \lambda, F)$ for a simple SST, where $\lambda\colon A \to \mathfrak{S}_{\mathfrak{X}}^{\varnothing}$ and $F \in \mathfrak{X}^*$. Indeed, states and the transition function are useless.

▶ **Lemma 27.** *Given a total SST, we can build an equivalent simple SST in* PTIME.

Proof. Let $\mathcal{T} = (A, B, Q, \mathfrak{X}, q_0, \iota, \delta, \lambda, F)$ be the original SST. We can assume that it uses no letters in the substitutions by storing them in constant registers (using the initial function). To remove the states, we let $\mathfrak{X}' := Q \times \mathfrak{X}$ be our new register set. In our new machine, register (q, x) will contain the value of x if q is the current state of $\mathcal{T}$, and ε otherwise. The update function $\lambda'\colon A \to \mathfrak{S}_{\mathfrak{X}'}^{B}$ and output $F' \in \mathfrak{X}'^*$ are defined as follows:

$$\lambda'(a)(q, x) = \prod_{p \mid \delta(p,a)=q} \mu_p(\lambda(p, a)(x)) \text{ and } F' = \prod_{q \in \mathrm{dom}(F)} \mu_p(F(q))$$

where μ_p replaces $y \in \mathfrak{X}$ by $(p, y) \in \mathfrak{X}'$. During a run, at most one term of the concatenation defining $\lambda'(a)(q, x)$ is nonempty, the one which corresponds to the true state p of $\mathcal{T}$. ◀

▶ Remark 28. However, this construction does not preserve copylessness nor k-layeredness.

5.2 Asymptotic behavior of ℕ-automata and SSTs

Given a simple SST, we first build an ℕ-automaton that "computes" the size of the words stored in the registers along a run of the SST. As we shall see, the growth of functions produced by ℕ-automata exactly matches the case disjunction of Proposition 25.

▶ **Definition 29.** *An* ℕ*-automaton* $\mathcal{A} = (A, \mathcal{Q}, \alpha, \mu, \beta)$ *consists in:*

- *an input alphabet A;*
- *a finite set $\mathcal{Q}$ of states;*
- *an initial row vector $\alpha \in \mathbb{N}^{\mathcal{Q}}$ and a final column vector $\beta \in \mathbb{N}^{\mathcal{Q}}$;*
- *a monoid morphism $\mu\colon A^* \to \mathbb{N}^{\mathcal{Q} \times \mathcal{Q}}$ (weight function).*

The automaton $\mathcal{A}$ computes the total function $A^* \to \mathbb{N}, w \mapsto \alpha\mu(w)\beta$. We say that it is *trim* if $\forall q \in \mathcal{Q}$, $\exists u, v \in A^*$ such that $(\alpha\mu(u))(q) \geq 1$ and $(\mu(v)\beta)(q) \geq 1$.

Let $\mathcal{T} = (A, B, \mathfrak{X}, \iota, \lambda, F)$ be a simple SST, we define its *flow automaton* $\widetilde{\mathcal{T}} := (A, \mathfrak{X}, \alpha, \mu, \beta)$ as the ℕ-automaton with input A, states $\mathfrak{X}$, and:

- for all $x \in \mathfrak{X}$, $\alpha(x) = |\iota(x)|$ (number of letters initialized in x);
- for all $x \in \mathfrak{X}$, $\beta(x)$ is the number of occurrences of x in F;
- for all $a \in A, x, x' \in \mathfrak{X}$, $\mu(a)(x, x')$ is the number of occurrences of x in $\lambda(a)(x')$.

Recall that $\mathcal{T}^w(x)$ is "the value of x after reading w in $\mathcal{T}$"; the flow automaton indeed computes the size of these values. We get the following by induction.

▷ Claim 30. For all $w \in A^*$ and $x \in \mathfrak{X}$, we have $(\alpha\mu(w))(x) = |\mathcal{T}^w(x)|$. In particular, if f is the function computed by $\mathcal{T}$, then $\widetilde{\mathcal{T}}$ computes $|f|$.

Without loss of generality, we can assume that $\widetilde{\mathcal{T}}$ is trim. Indeed, if $x \in \mathfrak{X}$ is such that $(\alpha\mu(u))(x) = 0$ for all $u \in A^*$, then x always has value ε and can be erased everywhere in $\mathcal{T}$. Similarly, if $(\mu(v)\beta)(x) = 0$ for all $v \in A^*$, x is never used in the output.

Let us now study in detail the asymptotic behavior of ℕ-automata.

▶ **Lemma 31.** *Let $\mathcal{A} = (A, \mathcal{Q}, \alpha, \mu, \beta)$ be a trim $\mathbb{N}$-automaton that computes a function $g\colon A^* \to \mathbb{N}$. Then one of the following holds, and it can be decided in* PTIME*:*

- *g has an exponential growth;*
- *g has k-polynomial growth for some $k \geq 0$ and $\mathcal{Q} = \biguplus_{0 \leq i \leq k} S_i$ is such that:*
  - *$\forall q, q' \in \mathcal{Q}$, if $\exists w \in A^*$ such that $\mu(w)(q, q') \geq 1$ then $q \in S_i, q' \in S_j$ for some $i \leq j$;*
  - *$\exists B \geq 0$ such that $\forall 0 \leq i \leq k$, $\forall q, q' \in S_i, \forall w \in A^*$, $\mu(w)(q, q') \leq B$;*
  - *$\forall q \in S_i, (\alpha\mu(w))(q) = \mathcal{O}(|w|^i)$.*

Furthermore, k and $S_0, \dots, S_k$ are computable in PTIME*.*

▶ Remark 32. An upper bound B can be described explicitly, see e.g. [14].

Proof sktech. Very similar results are obtained in [17] for computing ambiguity of finite automata (which corresponds to $\mathbb{N}$-automata with weights in $\{0, 1\}$ only).

Mainly, we look for the presence of the two patterns from [14] in the weights of $\mathcal{A}$:

- *heavy cycles* ($\exists q \in \mathcal{Q}, v \in A^*$ such that $\mu(v)(q, q) \geq 2$), that creates exponential growth ;
- *barbells* ($\exists q \neq q', v \in A^+$, such that $\mu(v)(q, q) \geq 1$, $\mu(v)(q, q') \geq 1$ and $\mu(v)(q', q') \geq 1$) such that a chain of k barbells induces k-polynomial growth. ◀

As a consequence, if f is computed by an SST, then $|f|$ has either exponential growth or k-polynomial growth for some $k \geq 0$. Furthermore, we can decide it in PTIME.

It remains to show that if $|f|$ has a $(k+1)$-polynomial growth, then f is computable by a k-layered SST. For this, we shall use the partition of Lemma 31 that splits the simple SST (via the states of its flow automaton) in a somehow k-layered way. However, the layers obtained are not directly copyless, and another transformation is necessary.

5.3 Construction of k-layered SST in the polynomial case

If $|f|$ has $(k+1)$-polynomial growth, then Lemma 31 partitions the simple SST in $k+2$ layers. Our first concern is to get $k+1$ layers only, since we want a k-layered SST. In the next definition, $\lambda(p, w)$ denotes the substitution applied when reading $w \in A^*$ from $p \in Q$, that is $\lambda(p, w[1]) \circ \cdots \lambda(\delta(p, w[1{:}(i-1)]), w[i]) \circ \cdots \circ \lambda(\delta(p, w[1{:}(|w|-1)]), w[|w|])$.

▶ **Definition 33.** *We say that an SST $(A, B, Q, \mathfrak{X}, q_0, \iota, \delta, \lambda, F)$ is* (k, B)-bounded *if there exists a partition $\mathfrak{X}_0, \mathfrak{X}_1, \dots, \mathfrak{X}_k$ of $\mathfrak{X}$ such that $\forall q \in Q, a \in A, w \in A^*$:*

- *$\forall 0 \leq i \leq k$, only registers from $\mathfrak{X}_0, \dots, \mathfrak{X}_i$ appear in $\{\lambda(q, a)(x) \mid x \in \mathfrak{X}_i\}$;*
- *$\forall 0 \leq i \leq k$, each $y \in \mathfrak{X}_i$ appears at most B times in $\{\lambda(q, w)(x) \mid x \in \mathfrak{X}_i\}$.*

For $k = 0$, Definition 33 means that at most B copies of y are "useful" during a run. The $(0, B)$-bounded SSTs are known as *B-bounded (copy) SSTs* in [6] (however, contrary to what is said in [6, 10], it is not the same definition as the "bounded copy" of [2]). For some $k \geq 1$, a (k, B)-bounded SST is similar to a k-layered SST, except that each layer is no longer "copyless in itself" but "B-bounded in itself". In particular, we note that $(k, 1)$-bounded SSTs exactly correspond to k-layered SSTs.

▶ **Lemma 34.** *Given a simple SST that computes a function f such that $|f|$ has $(k+1)$-polynomial growth, we can build an equivalent (k, B)-bounded SST for some $B \geq 0$.*

Proof sketch. Let $S_0, \dots, S_{k+1}$ be the partition of the registers given by Lemma 31. During a run, note that the registers in S_0 can only store strings of a bounded size. The idea is to remove S_0 and hardcode the content of each $x \in S_0$ in a finite set of states. The transition function is defined following their former updates. The new update function is defined by replacing the mention of $x \in S_0$ by its explicit content (given by the current state). ◀

▶ Remark 35. As for weighted automata above, an upper bound B can effectively be computed.

It is known that a $(0, B)$-bounded SST can be transformed in a copyless SST. The proof requires rather complex constructions, that we generalize for a (k, B)-bounded SST.

▶ **Lemma 36.** *Given a (k, B)-bounded SST, we can build an equivalent k-layered SST.*

Proof sketch. The proof is done by induction on $k \geq 0$. Indeed a (k, B)-bounded (resp. k-layered) SST is somehow a B-bounded (resp. copyless) SST, that can also "call" registers from the lower layers. Hence we only need to focus on transforming *one* layer. The difficulty is to take into account copies coming from the lower layers. This is done by introducing an intermediate model of *SST with external functions* (SST-F), which corresponds to an SST with a set of functions $\mathfrak{F}$ that can be called in an oracle-like style.

Thus our proof roughly consists in showing that a B-bounded SST-F (in the sense of Definition 33) can be transformed in a copyless SST-F. This is done in two steps. First, we transform the B-bounded SST-F in a copyless *non-deterministic* SST-F, following the ideas of [6] for SST. Non-deterministic transducers usually compute relations between words, but we in fact obtain an *unambiguous* machine (i.e. that has at most one accepting run on each input), hence describing a function. Second, we show that a copyless unambiguous non-deterministic SST-F can be converted in a copyless SST-F. This transformation relies on the techniques of [2] (developped for SST over infinite words). ◀

Proof of Theorem 23. We first transform an SST into a simple SST (Lemma 27) and build its flow automaton. Using this machine, one can decide what is the growth of $|f|$ (Lemma 31). If $|f|$ has $(k+1)$-polynomial growth, we then build a (k, B)-bounded SST that computes it (Lemma 34) and finally a k-layered SST (Lemma 36). ◀

6 Conclusion and outlook

We presented in this paper a new correspondence between SSTs and marble transducers. Showing that two models are equivalent is always interesting in itself, but our result also provides a deeper understanding of their behaviors. Indeed, it relates recursive and iterative programs (marbles) to streaming algorithms (SSTs), which are suitable for program optimization problems. Since the equivalence problem is decidable for SSTs [10], we also obtain for free that it is the case for marble transducers (which was not previously known).

Note that our model is not closed under composition. It is the case for obvious asymptotic growth reasons, since marble transducers can compute one exponential ($\mathsf{exp}\colon a^n \mapsto a^{2^n}$) but not two of them ($\mathsf{exp} \circ \mathsf{exp}$). More surprisingly, there exist polynomial-size compositions that cannot be expressed by our transducers, as shown below.

▷ Claim 37. $\mathsf{mul}\colon w\#0^n \mapsto (w\#)^n$ is computable by an SST, but not $0^n\#w \mapsto (w\#)^n$.

This result mainly comes because marbles and SSTs give an *orientation* on the input: we stack marbles "on the right", and the SST is a streaming process "from left to right".

Marbles and pebbles. A non-oriented generalization of k-marbles, named *k-pebble transducers* [3], has recently been investigated in detail. In this case, the reading head is allowed to move on the right of a mark without lifting it, while keeping a stack discipline. The typical example of function computable with 1 pebble is $\mathsf{square}\colon A \to A \uplus \{\overline{a} \mid a \in A\}, abc \mapsto \overline{a}bca\overline{b}cab\overline{c}$, which associates to w the concatenation with $|w|$ copies of itself, the i-th copy having its i-th letter overlined. This function cannot be computed by a marble transducer.

In [13], the membership problem is solved for the classes of k-pebble transducers. Despite their similarity, neither their result (Theorem 38 below) nor our Proposition 25 imply each other, and the proof techniques are significantly different. Indeed we consider different classes of functions, ours being designed for streaming implementations, but not theirs. The relationship between marbles and pebbles is depicted in Figure 5.

▶ **Theorem 38** ([13]). *A function f described by a k-pebble transducer is computable by an ℓ-pebble transducer if and only if $|f(w)| = \mathcal{O}(|w|^{\ell+1})$ (and this property is decidable).*

Contrary to us, they do not obtain tight asymptotic bounds. Furthermore, they only consider machines with a bounded number of marks, i.e. no exponential growths.

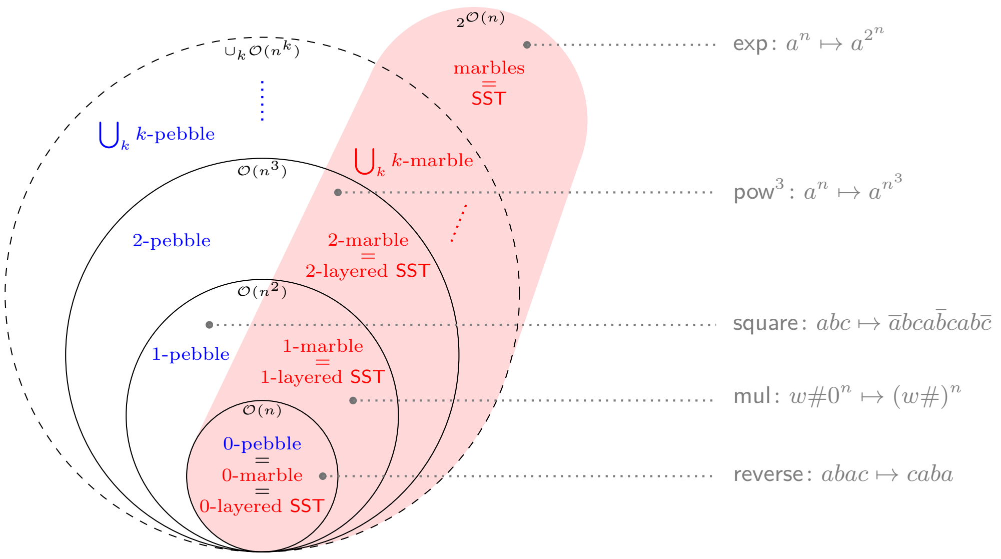
$2^{\mathcal{O}(n)}$
marbles = SST
$\cup_k \mathcal{O}(n^k)$
$\bigcup_k$ k-pebble
$\bigcup_k$ k-marble
$\mathcal{O}(n^3)$
2-pebble
2-marble = 2-layered SST
$\mathcal{O}(n^2)$
1-pebble
1-marble = 1-layered SST
$\mathcal{O}(n)$
0-pebble = 0-marble = 0-layered SST
exp: $a^n \mapsto a^{2^n}$
pow^3: $a^n \mapsto a^{n^3}$
square: $abc \mapsto \overline{a}bca\overline{b}cab\overline{c}$
mul: $w\#0^n \mapsto (w\#)^n$
reverse: $abac \mapsto caba$

Figure 5 Classes of functions studied in this paper (red) and in [13] (blue).

Future work. Our work opens the way to a finer study of the classes of functions described by marble and pebble transducers. The *membership problem from k-pebble to k-marble* is worth being studied to complete the decidability picture. It reformulates as follows: given a function computed by a pebble transducer, can we implement it in streaming way? The answer seems to rely on combinatorial properties of the output. Another perspective is to define a *logical description* of our transducers, which allows to specify their behavior in a non-operational fashion. No formalism is known for marble transducers, but it is known since [8] that two-way transducers correspond to *monadic-second-order transductions*. On the other hand, [7] studies in detail a *weighted logics* which describes the functions computed by weighted automata. Using proof techniques which are similar to ours, they relate the asymptotic growth of the function to the logical connectors required to describe it.

References

1 Rajeev Alur and Pavol Cerný. Expressiveness of streaming string transducers. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010*, volume 8 of *LIPIcs*, pages 1–12. Schloss Dagstuhl, 2010.

2 Rajeev Alur, Emmanuel Filiot, and Ashutosh Trivedi. Regular transformations of infinite strings. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012*, pages 65–74. IEEE Computer Society, 2012.

3 Mikolaj Bojańczyk. Polyregular functions. *arXiv preprint arXiv:1810.08760*, 2018.

4 Michal P Chytil and Vojtěch Jákl. Serial composition of 2-way finite-state transducers and simple programs on strings. In *4th International Colloquium on Automata, Languages, and Programming, ICALP 1977*, pages 135–147. Springer, 1977.

5 Luc Dartois, Paulin Fournier, Ismaël Jecker, and Nathan Lhote. On reversible transducers. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, volume 80 of *LIPIcs*, pages 113:1–113:12. Schloss Dagstuhl, 2017.

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

7 Manfred Droste and Paul Gastin. Aperiodic weighted automata and weighted first-order logic. In *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019*, volume 138 of *LIPIcs*, pages 76:1–76:15. Schloss Dagstuhl, 2019.

8 Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic (TOCL)*, 2(2):216–254, 2001.

9 Joost Engelfriet, Hendrik Jan Hoogeboom, and Jan-Pascal Van Best. Trips on trees. *Acta Cybernetica*, 14(1):51–64, 1999.

10 Emmanuel Filiot and Pierre-Alain Reynier. Copyful streaming string transducers. In *International Workshop on Reachability Problems*, pages 75–86. Springer, 2017.

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

12 Oscar H Ibarra. Characterizations of some tape and time complexity classes of turing machines in terms of multihead and auxiliary stack automata. *Journal of Computer and System Sciences*, 5(2):88–117, 1971.

13 Nathan Lhote. Pebble minimization of polyregular functions. In *35th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'20*, pages 703–712. ACM, 2020.

14 Jacques Sakarovitch and Rodrigo De Souza. On the decidability of bounded valuedness for transducers. In *International Symposium on Mathematical Foundations of Computer Science*, pages 588–600. Springer, 2008.

15 Dana Scott. Some definitional suggestions for automata theory. *Journal of Computer and System Sciences*, 1(2):187–212, 1967.

16 John C Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3(2):198–200, 1959.

17 Andreas Weber and Helmut Seidl. On the degree of ambiguity of finite automata. *Theoretical Computer Science*, 88(2):325–349, 1991.