# Comparison-Free Polyregular Functions

**Lê Thành Dũng (Tito) Nguyễn**
Laboratoire d'informatique de Paris Nord, Villetaneuse, France

**Camille Noûs**
Laboratoire Cogitamus, Université Volante, Sevran, France

**Pierre Pradic**
Department of Computer Science, University of Oxford, UK

## Abstract

This paper introduces a new automata-theoretic class of string-to-string functions with polynomial growth. Several equivalent definitions are provided: a machine model which is a restricted variant of pebble transducers, and a few inductive definitions that close the class of regular functions under certain operations. Our motivation for studying this class comes from another characterization, which we merely mention here but prove elsewhere, based on a $\lambda$-calculus with a linear type system.

As their name suggests, these *comparison-free polyregular functions* form a subclass of polyregular functions; we prove that the inclusion is strict. We also show that they are incomparable with HDT0L transductions, closed under usual function composition – but not under a certain "map" combinator – and satisfy a comparison-free version of the pebble minimization theorem.

On the broader topic of polynomial growth transductions, we also consider the recently introduced layered streaming string transducers (SSTs), or equivalently $k$-marble transducers. We prove that a function can be obtained by composing such transducers together if and only if it is polyregular, and that $k$-layered SSTs (or $k$-marble transducers) are closed under "map" and equivalent to a corresponding notion of $(k+1)$-layered HDT0L systems.

**2012 ACM Subject Classification** Theory of computation → Transducers

**Keywords and phrases** pebble transducers, HDT0L systems, polyregular functions

**Digital Object Identifier** 10.4230/LIPIcs.ICALP.2021.139

**Category** Track B: Automata, Logic, Semantics, and Theory of Programming

**Related Version** *Full Version with appendices*: `https://hal.archives-ouvertes.fr/hal-02986228`

**Funding** *Lê Thành Dũng (Tito) Nguyễn*: Partially funded by the French ANR project CoGITARe (ANR-18-CE25-0001).

**Acknowledgements** Thanks to Mikołaj Bojańczyk and Sandra Kiefer for inspiring discussions, to Gaëtan Douéneau-Tabot and Amina Doumane for explaining some features of their work to us, to Charles Paperman for his help with bibliography and to the reviewers for their feedback.

## 1 Introduction

The theory of transducers (as described in the surveys [21, 29]) has traditionally dealt with devices that take as input strings of length $n$ and output strings of length $O(n)$. However, several recent works have investigated function classes going beyond linear growth. We review three classes in this landscape below.

- *Polyregular functions* (§2.3) are thus named because they have (at most) polynomial growth and include regular functions (§2.2) (the most expressive of the traditional string-to-string transduction classes). They were defined in 2018 [4] by four equivalent computational models, one of which – the *pebble transducers* – is the specialization to strings of a tree transducer model that existed previously in the literature [28] (this specialization had been investigated earlier in [17, 14]). A subsequent work [8] gave a logical characterization based on Monadic Second-Order logic (MSO). They enjoy two nice properties:

  - *preservation of regular languages (by preimage)*: if $f : \Gamma^* \to \Sigma^*$ is polyregular and $L \subseteq \Sigma^*$ is regular, then $f^{-1}(L) \subseteq \Gamma^*$ is regular;
  - *closure under function composition*: if $f : \Gamma^* \to \Delta^*$ and $g : \Delta^* \to \Sigma^*$ are both polyregular, then so is $g \circ f : \Gamma^* \to \Sigma^*$.
- *HDT0L transductions* (§2.1) form another superclass of regular functions, whose output size may be at most exponential in the input size. They are older than polyregular functions, and we shall discuss their history in Section 2.1; suffice to say for now, they also admit various equivalent characterizations scattered in several papers [20, 22, 13]. These functions preserve regular languages by preimage, but are *not* closed under composition (the growth rate of a composition of HDT0L transductions may be a tower of exponentials).
- Very recently, the polynomially bounded HDT0L transductions (§2.3) have been characterized using two transducer models [13]. One of them, the *k-marble transducers* (where $k \in \mathbb{N}$ depends on the function to be computed), is obtained by putting a syntactic constraint on the model of *(unbounded) marble transducers* [13] which computes HDT0L transductions. But it can also be seen as a restricted variant of pebble transducers; it follows (although this is not explicitly stated in [13]) that a HDT0L transduction has polynomial growth if and only if it is polyregular. Moreover, as claimed in [13, Section 6], the functions computed by $k$-marble transducers are not closed under composition either, and thus form a strict subclass of polyregular functions.

**A new subclass of polyregular functions.** In this paper, we start by proving a few results on the above classes (Section 3). For instance, we supply a proof for the aforementioned claim of [13, Section 6], and show that the polyregular functions are exactly those computable by compositions of $k$-marble transducers. Those complements are not particularly difficult nor surprising and are included mostly for the sake of giving a complete picture.

But our main contribution is the introduction of a new class, giving its title to the paper; as we show, it admits three equivalent definitions:

- two ways to inductively generate the class (Sections 4 and 6 respectively):
  - by closing regular functions under a certain “composition by substitution” operation;
  - by combining regular functions and a certain kind of *squaring* functions (less powerful than the squaring plus underlining functions used to characterize general poyregular functions) with usual function composition;
- a restriction on pebble transducers (Section 5) – we disallow comparing the positions of a transducer’s multiple reading heads, hence the name *comparison-free polyregular functions* (henceforth abbreviated as *cfp*).

**Properties.** By the third definition above, comparison-free polyregular functions are indeed polyregular, while the second one implies that our new class contains the regular functions and is closed under composition. (In fact, in the proof that our first definition is equivalent to the second one, most of the work goes into showing that the former enjoys closure under composition.) We rule out inclusions involving the other classes that we mentioned by proving some *separation results* (Section 8): there exist

- comparison-free polyregular functions that are not HDT0L (we take one example from [13]),
- and polynomially bounded HDT0L transductions which are not comparison-free:
  - one of our examples follows from a precise characterization of cfp functions over unary input alphabets (extending a known result for regular functions with unary inputs [10]), which we give in Section 9;
  - another example shows that unlike (poly)regular functions, cfp functions are *not* closed under a certain counterpart of the “map” operation in functional programming.

We summarize the inclusions and separations between classes that we get in Figure 1.

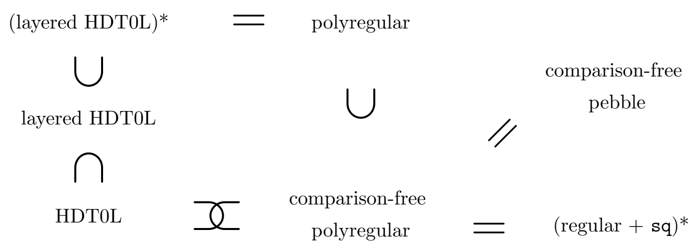

**Figure 1** Summary of the known relationships between superlinear transduction classes, taking our results into account. Inclusions $\subset$ are strict, and $\not\supset\!\subset$ means that there is no inclusion either way. Finally $C^*$ denotes the composition closure of the class $C$.

Finally, we show in Section 7 that the number of pebbles required to compute a function using a comparison-free transducer is related to its growth rate. The analogous result for pebble transducers was proved recently, with a whole paper dedicated to it [25]; we adapt its arguments to our setting, resulting in our longest and most technical proof. There is a similar property for $k$-marble transducers [13], but it is proved using very different tools.

**Motivations.** Although this is the first proper paper to introduce comparison-free pebble transducers, we were told that they had already been considered by several colleagues (Mikołaj Bojańczyk, personal communication). But in fact, the starting point in our investigation was a characterization of regular functions using a linear $\lambda$-calculus (in the sense of linear logic) that we had previously obtained [30]; this was part of a research programme relating automata and functional programming that we initiated in [31]. As we reported in a previous version of the present paper (version 1 of the full paper archived on HAL), by tweaking a parameter in this characterization, one gets the cfp functions instead; we initially defined the latter using composition by substitution, and only later realized the connection with pebble transducers. One interesting feature of the $\lambda$-calculus characterization is that it is trivially closed under composition, and this led us to take inspiration from the category-theoretic machinery that we used in [30] for our standalone composition proof in this paper.

## 2 Preliminaries

**Notations.** The set of natural numbers is $\mathbb{N} = \{0, 1, \dots\}$. We write $|w|$ for the length of a string $w \in \Sigma^*$; for $\Pi \subseteq \Sigma$, we write $|s|_\Pi$ for the number of occurrences of letters from $\Pi$ in $w$; and for $c \in \Sigma$, we abbreviate $|w|_{\{c\}}$ as $|w|_c$. The $i$-th letter of $w \in \Sigma^*$ is denoted by either $w_i$ or $w[i]$ (for $i \in \{1, \dots, |w|\}$). Given monoids $M$ and $N$, $\mathrm{Hom}(M, N)$ is the set of monoid morphisms. We write $\varepsilon$ for the empty word and $\underline{\Sigma} = \{\underline{a} \mid a \in \Sigma\}$ for a disjoint copy of the alphabet $\Sigma$ made of "underlined" letters.

### 2.1 HDT0L transductions and streaming string transducers

*L-systems* were originally introduced by Lindenmayer [26] in the 1960s as a way to generate formal languages, with motivations from biology. While this language-centric view is still predominant, the idea of considering variants of L-systems as specifications for string-to-string functions – whose range are the corresponding languages – seems to be old. For instance, in a paper from 1980 [18], one can find (multi-valued) string functions defined by ET0L systems.

More recently, Ferté, Marin and Sénizergues [20] provided alternative characterizations[1] (by catenative recurrent equations and higher-order pushdown transducers of level 2) of the string-to-string functions that *HDT0L systems* can express – what we call here *HDT0L transductions*. Later work by Filiot and Reynier [22] and then by Douéneau-Tabot, Filiot and Gastin [13] – that does not build on [36, 20] – proved the equivalence with, respectively, copyful SSTs (Definition 2.3) and unbounded marble transducers (not presented here).

▶ **Definition 2.1** (following [22]). *A* HDT0L system *consists of:*

- *an input alphabet $\Gamma$, an output alphabet $\Sigma$, and a working alphabet $\Delta$ (all* finite*);*
- *an initial word $d \in \Delta^*$;*
- *for each $c \in \Gamma$, a* monoid morphism $h_c \in \mathrm{Hom}(\Delta^*, \Delta^*)$*;*
- *a final morphism $h' \in \mathrm{Hom}(\Delta^*, \Sigma^*)$.*

*It defines the transduction taking $w = w_1 \dots w_n \in \Gamma^*$ to $h' \circ h_{w_1} \circ \dots \circ h_{w_n}(d) \in \Sigma^*$.*

(The definition of HDT0L systems given in [36, 20] makes slightly different choices of presentation[2].) To define the equivalent model of copyful streaming string transducers, we must first introduce the notion of register assignment.

▶ **Definition 2.2.** *Fix a finite alphabet $\Sigma$. Let $R$ and $S$ be two finite sets* disjoint from $\Sigma$*; we shall consider their elements to be "register variables".*

*For any word $\omega \in (\Sigma \cup R)^*$, we write $\omega^\dagger : (\Sigma^*)^R \to \Sigma^*$ for the map that sends $(u_r)_{r\in R}$ to $\omega$ in which every occurrence of a register variable $r \in R$ is replaced by $u_r$ – formally, we apply to $\omega$ the morphism $(\Sigma \cup R)^* \to \Sigma^*$ that maps $c \in \Sigma$ to itself and $r \in R$ to $u_r$.*

*A* register assignment[3] *$\alpha$ from $R$ to $S$ (over $\Sigma$) is a map $\alpha : S \to (\Sigma \cup R)^*$. It induces the action $\alpha^\dagger : \vec{u} \in (\Sigma^*)^R \mapsto (\alpha(s)^\dagger(\vec{u}))_{s\in S} \in (\Sigma^*)^S$ (which indeed goes "from $R$ to $S$").*

▶ **Definition 2.3** ([22]). *A (deterministic copyful)* streaming string transducer *(SST) with input alphabet $\Gamma$ and output alphabet $\Sigma$ is a tuple $\mathcal{T} = (Q, q_0, R, \delta, \vec{u}_I, F)$ where*

- *$Q$ is a finite set of* states *and $q_0 \in Q$ is the* initial state*;*
- *$R$ is a finite set of* register variables*, that we require to be* disjoint from $\Sigma$*;*
- *$\delta : Q \times \Gamma \to Q \times (R \to (\Sigma \cup R)^*)$ is the* transition function *– we abbreviate $\delta_{\mathrm{st}} = \pi_1 \circ \delta$ and $\delta_{\mathrm{reg}} = \pi_2 \circ \delta$, where $\pi_i$ is the projection from $X_1 \times X_2$ to its $i$-th component $X_i$;*
- *$\vec{u}_I \in (\Sigma^*)^R$ describes the* initial register values*;*
- *$F : Q \to (\Sigma \cup R)^*$ describes how to recombine the final values of the registers, depending on the final state, to produce the output.*

*The function $\Gamma^* \to \Sigma^*$ computed by $\mathcal{T}$ is*

$$w_1 \dots w_n \quad \mapsto \quad F(q_n)^\dagger \circ \delta_{\mathrm{reg}}(q_{n-1}, w_n)^\dagger \circ \dots \circ \delta_{\mathrm{reg}}(q_0, w_1)^\dagger(\vec{u}_I)$$

*where the sequence of states $(q_i)_{0\le i\le n}$ (sometimes called the* run *of the transducer over the input word) is inductively defined, starting from the fixed initial state $q_0$, by $q_i = \delta_{\mathrm{st}}(q_{i-1}, w_i)$.*

[1] Those characterizations had previously been announced in an invited paper by Sénizergues [36]. Some other results announced in [36] are proved in [9].

[2] The family $(h_c)_{c\in\Gamma}$ is presented as a morphism $H : \Gamma^* \to \mathrm{Hom}(\Delta^*, \Delta^*)$ (whose codomain is indeed a monoid for function composition). And an initial *letter* is used instead of an initial word; this is of no consequence regarding the functions that can be expressed (proof sketch: consider $\Delta' = \Delta \cup \{x\}$ with a new letter $x \notin \Delta$, take $x$ as the initial letter and let $h_c(x) = h_c(w)$, $h'(x) = h'(w)$).

[3] Some papers e.g. [11, 13] call register assignments *substitutions*. We avoid this name since it differs from its meaning in the context of our "composition by substitution" operation.

▶ **Example 2.4.** Let $\Sigma = \Gamma \cup \underline{\Gamma}$. We consider a SST $\mathcal{T}$ with $Q = \{q\}$, $R = \{X, Y\}$ and

$$\vec{u}_I = (\varepsilon)_{r \in R} \qquad F(q) = Y \qquad \forall c \in \Gamma,\ \delta(q, c) = (q,\ (X \mapsto cX,\ Y \mapsto \underline{c}XY))$$

If we write $(v, w)$ for the family $(u_r)_{r \in R}$ with $u_X = v$ and $u_Y = w$, then the action of the register assignments may be described as $(X \mapsto cX,\ Y \mapsto \underline{c}XY)^\dagger(v, w) = (c \cdot v,\ \underline{c} \cdot v \cdot w)$.

Let $\mathtt{1}, \mathtt{2}, \mathtt{3}, \mathtt{4} \in \Gamma$. After reading $\mathtt{1234} \in \Gamma^*$, the values stored in the registers of $\mathcal{T}$ are

$$(X \mapsto \mathtt{4}X,\ Y \mapsto \underline{\mathtt{4}}XY)^\dagger \circ \ldots \circ (X \mapsto \mathtt{1}X,\ Y \mapsto \underline{\mathtt{1}}XY)^\dagger(\varepsilon, \varepsilon) = (\mathtt{4321}, \underline{\mathtt{4}}\mathtt{321}\underline{\mathtt{3}}\mathtt{21}\underline{\mathtt{2}}\mathtt{1}\underline{\mathtt{1}})$$

Since $F(q) = Y$, the function defined by $\mathcal{T}$ maps $\mathtt{1234}$ to $\underline{\mathtt{4}}\mathtt{321}\underline{\mathtt{3}}\mathtt{21}\underline{\mathtt{2}}\mathtt{1}\underline{\mathtt{1}} \in (\Gamma \cup \underline{\Gamma})^* = \Sigma^*$.

This gives us an example of HDT0L transduction $\Gamma^* \to (\Gamma \cup \underline{\Gamma})^*$, since:

▶ **Theorem 2.5** ([22])**.** *A function $\Gamma^* \to \Sigma^*$ can be computed by a copyful SST if and only if it can be specified by a HDT0L system.*

▶ Remark 2.6. As observed in [22, Lemma 3.3], there is a natural translation from HDT0L systems to SSTs whose range is composed precisely of the *single-state* SSTs whose transitions and final output function *do not access the letters of their output alphabet* – those are called *simple SSTs* in [13, §5.1]. This involves a kind of reversal: the initial register values correspond to the final morphisms, while the final output function corresponds to the initial word. Thus, Theorem 2.5 is essentially a state elimination result; a direct translation from SSTs to single-state SSTs has also been given by Benedikt et al. [2, Proposition 8]. However, it does *not* preserve the subclass of *copyless* SSTs (this would contradict Proposition 3.7).

The lookahead elimination theorem for macro tree transducers [19, Theorem 4.21] arguably generalizes this to trees. Indeed, while those transducers are generally presented as a top-down model, their formal definition can also be read as bottom-up register tree transducers in the style of [7, §4], and top-down lookahead corresponds to bottom-up states.

## 2.2 Regular functions

▶ **Definition 2.7** (Alur and Černý [1])**.** *A register assignment $\alpha : S \to (\Sigma \cup R)^*$ from $R$ to $S$ is said to be* copyless *when each $r \in R$* occurs at most once among all the strings *$\alpha(s)$* for *$s \in S$, i.e. it does not occur at least twice in some $\alpha(s)$, nor at least once in $\alpha(s)$ and at least once in $\alpha(s')$ for some $s \neq s'$. (This restriction does not apply to the letters in $\Sigma$.)*

*A streaming string transducer is* copyless *if all the assignments in the image of its transition function are copyless. In this paper, we take computability by copyless SSTs as the definition of* regular functions *(but see Theorem 5.3 for another standard definition).*

▶ Remark 2.8. Thanks to Theorem 2.5, every regular function is a HDT0L transduction.

▶ Remark 2.9. The SST of Example 2.4 is *not* copyless: in a transition $\alpha = \delta_{\text{reg}}(q, c)$, the register $X$ appears twice, once in $\alpha(X) = cX$ and once in $\alpha(Y) = \underline{c}XY$; in other words, its value is *duplicated* by the action $\alpha^\dagger$. In fact, it computes a function whose output size is quadratic in the input size, while regular functions have linearly bounded output.

▶ **Example 2.10** (Iterated reverse [4, p. 1])**.** The following single-state SST is copyless:

$$\Gamma = \Sigma \text{ with } \# \in \Sigma \qquad Q = \{q\} \qquad R = \{X, Y\} \qquad \vec{u}_I = (\varepsilon)_{r \in R} \qquad F(q) = XY$$

$$\delta(q, \#) = (q,\ (X \mapsto XY\#,\ Y \mapsto \varepsilon)) \qquad \forall c \in \Sigma \setminus \{\#\},\ \delta(q, c) = (q,\ (X \mapsto X,\ Y \mapsto cY))$$

For $u_1, \ldots, u_n \in (\Sigma \setminus \{\#\})^*$, it maps $u_1\# \ldots \#u_n$ to $\mathtt{reverse}(u_1)\# \ldots \#\mathtt{reverse}(u_n)$.

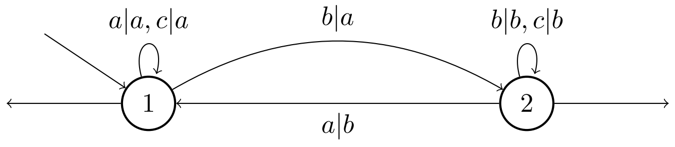

**Figure 2** An example of sequential transducer.

The concrete SSTs (copyless or not) that we have seen for now are all single-state. As a source of stateful copyless SSTs, one can consider the translations of *sequential transducers*. These are usual finite automata, whose transitions additionally produce a word catenated to the end of the would-be output function. For instance, the one in Figure 2 computes the function $\{a, b, c\}^* \to \{a, b\}^*$ that replaces each $c$ in its input by the closest non-$c$ letter on its left (or $a$ if no such letter exists). We do not give a detailed definition (which can be found e.g. in [33, Chapter V]) here, but for our purpose, it suffices to observe any sequential transducer can be translated into a copyless SST with the same set of states and a single register.

## 2.3 Polynomial growth transductions

Next, we recall one way to define Bojańczyk's *polyregular functions* [4].

▶ **Definition 2.11** ([4]). *The class of* polyregular functions *is the smallest class of string-to-string functions closed under composition containing:*

- *the functions computed by sequential transducers (for instance, the one of Figure 2);*
- *the* iterated reverse *function of Example 2.10, over any finite alphabet containing #;*
- *the* squaring with underlining *functions* $\texttt{squaring}_\Gamma : \Gamma^* \to (\Gamma \cup \underline{\Gamma})^*$*, for any finite* $\Gamma$*, illustrated by* $\texttt{squaring}_\Gamma(1234) = \underline{1}234 1\underline{2}34 12\underline{3}4 123\underline{4}$*.*

As mentioned in the introduction, the intersection between the above class and HDT0L transductions has been recently characterized by Douéneau-Tabot et al. [13].

▶ **Theorem 2.12** ([13]). *Let* $f : \Gamma^* \to \Sigma^*$*. The following conditions are equivalent:*

- $f$ *is both a polyregular function and a HDT0L transduction;*
- $f$ *is a HDT0L transduction and has at most polynomial growth:* $f(|w|) = |w|^{O(1)}$*;*
- *there exists* $k \in \mathbb{N}$ *such that* $f$ *is computed by some* $k$-layered SST*, defined below.*

(Another equivalent model, the *k-marble transducers*, was mentioned in the introduction, but we will not use it in the rest of the paper.) Those $k$-layered SST propose a compromise between copyful and copyless SSTs: duplication is controlled, but not outright forbidden.

▶ **Definition 2.13** ([13]). *A register assignment* $\alpha : R \to (\Sigma \cup R)^*$ *is* $k$-layered *(for* $k \in \mathbb{N}$*) with respect to a partition* $R = R_0 \sqcup \ldots \sqcup R_k$ *when for* $0 \le i \le k$*,*

- *for* $r \in R_i$*, we have* $\alpha(r) \in (\Sigma \cup R_0 \cup \ldots \cup R_i)^*$*;*
- *each register variable in* $R_i$ *appears at most once among all the* $\alpha(r)$ *for* $r \in R_i$ *(however, those from* $R_0 \sqcup \ldots \sqcup R_{i-1}$ *may appear an arbitrary number of times).*

*A SST is* $k$-layered *if its registers can be partitioned in such a way that all assignments in the transitions of the SST are* $k$*-layered.*

Beware: with this definition, the registers of a $k$-layered SST are actually divided into $k + 1$ layers, not $k$. In particular, a SST is copyless if and only if it is 0-layered. (We chose this convention for backwards compatibility with [13]; see also Remark 5.4.)

For instance, the transducer of Example 2.4 is 1-layered with $R_0 = \{X\}$ and $R_1 = \{Y\}$. There also exist register assignments that cannot be made $k$-layered no matter the choice of partition, such as $X \mapsto XX$. Using such assignments, one can indeed build SSTs that compute functions $f$ such that e.g. $|f(w)| = 2^{|w|}$.

▶ Remark 2.14. There is arguably an old precursor to this recent characterization of HDT0L transductions with polynomial growth by a syntactic "layering" condition: Schützenberger's theorem on polynomially bounded $\mathbb{Z}$-rational series, which dates back to the 1960s (see for instance [3, Chapter 9, Section 2] – the preface of the same book describes this theorem as "one of the most difficult results in the area"). Let us give a brief exposition.

A *$\mathbb{Z}$-rational series* $f : \Sigma^* \to \mathbb{Z}$ is a function of the form $f : w \in \Sigma^* \mapsto X^T \cdot \Phi(w) \cdot Y$ where $X, Y \in \mathbb{Z}^R$ and $\Phi$ is a morphism from $\Sigma^*$ to the multiplicative monoid of $R$-indexed square matrices over $\mathbb{Z}$, where $R$ is a finite set. This data $(X, \Phi, Y)$ has a clear interpretation as a "simple SST" (cf. Remark 2.6) with register set $R$, whose register values are integers rather than strings. Schützenberger's theorem says that any $\mathbb{Z}$-rational series $f$ with polynomial growth (i.e. $|f(w)| = |w|^{O(1)}$ where $|\cdot|$ on the left is the absolute value) can be written as $f : w \mapsto X^T \cdot \Phi(w) \cdot Y$ where

**(i)** the image of $\Phi$ has a block triangular structure;
**(ii)** the projection of this image on each diagonal block is a finite monoid.

The first item gives us a partition of the register into layers where each layer "depends" only on the ones below them. The finiteness condition in the second item is equivalent to having bounded coefficients, which means that the register assignments within each layer are *bounded-copy*, while in a layered SST, they would be *copyless* instead – but bounded-copy SSTs are known to be equivalent to copyless SSTs (see e.g. [11]). The theorem also states a relationship between the number of blocks and the growth rate; compare this to Remark 7.2.

Via the canonical isomorphism $\{a\}^* \cong \mathbb{N}$, HDT0L transductions with unary output alphabet are the same thing as *$\mathbb{N}$-rational series*. The counterpart of Schützenberger's theorem over $\mathbb{N}$ is thus a corollary of the results of [13] on layered SSTs.

## 2.4 Transition monoids for streaming string transducers

To wrap up the preliminaries, let us recall some algebraic tools for working with SSTs (this technical section can be safely skipped on a first reading). Let us start by putting a monoid structure on register assignments (Definition 2.2).

▶ **Definition 2.15.** *Let $\mathcal{M}_{R,\Sigma} = R \to (\Sigma \cup R)^*$ for $R \cap \Sigma = \varnothing$. We endow it with the following composition operation, that makes it into a monoid:*

$$\alpha \bullet \beta = \alpha^{\odot} \circ \beta \quad \textit{where } \alpha^{\odot} \in \mathrm{Hom}((\Sigma \cup R)^*, (\Sigma \cup R)^*),\ \alpha^{\odot}(x) = \begin{cases} \alpha(x) & \textit{for } x \in R \\ x & \textit{for } x \in \Sigma \end{cases}$$

The monoid $\mathcal{M}_{R,\Sigma}$ thus defined is isomorphic to a submonoid of $\mathrm{Hom}((\Sigma \cup R)^*, (\Sigma \cup R)^*)$ with function composition. It admits a submonoid of *copyless* assignments.

▶ **Definition 2.16.** *We write $\mathcal{M}^{\mathrm{cl}}_{R,\Sigma}$ for the set of all $\alpha \in \mathcal{M}_{R,\Sigma}$ such that each letter $r \in R$ occurs at most once among all the $\alpha(r')$ for $r' \in R$.*

▶ **Proposition 2.17.** *$\mathcal{M}^{\mathrm{cl}}_{R,\Sigma}$ is a submonoid of $\mathcal{M}_{R,\Sigma}$. In other words, copylessness is preserved by composition (and the identity assignment is copyless).*

The following proposition ensures that this composition does what we expect. Recall from Definition 2.2 that $(-)^\dagger$ sends $\mathcal{M}_{R,\Sigma}$ to $(\Sigma^*)^R \to (\Sigma^*)^R$.

▶ **Proposition 2.18.** *For all $\alpha, \beta \in \mathcal{M}_{R,\Sigma}$, we have $(\alpha \bullet \beta)^\dagger = \beta^\dagger \circ \alpha^\dagger$.*

To incorporate information concerning the states of an SST, we define below a special case of the *wreath product* of transformation monoids.

▶ **Definition 2.19.** *Let $M$ be a monoid whose multiplication is denoted by $m, m' \mapsto m \cdot m'$. We define $M \wr Q$ as the monoid whose set of elements is $Q \to Q \times M$ and whose monoid multiplication is, for $\mu, \mu' : Q \to Q \times M$,*

$$(\mu \bullet \mu') : q \mapsto (\pi_1 \circ \mu' \circ \pi_1 \circ \mu(q),\ (\pi_2 \circ \mu(q)) \cdot (\pi_2 \circ \mu' \circ \pi_1 \circ \mu(q)))$$

*where $\pi_1 : Q \times M \to Q$ and $\pi_2 : Q \times M \to M$ are the projections.*

For instance, if $M$ is the trivial monoid with one element, $Q \wr M$ is isomorphic to $Q \to Q$ with *reverse* composition as the monoid multiplication: $f \bullet g = g \circ f$.

▶ **Proposition 2.20.** *Let $(Q, q_0, R, \delta, \vec{u}_I, F)$ be an SST that computes $f : \Gamma^* \to \Sigma^*$ (using the notations of Definition 2.3). For all $c \in \Gamma$, we have $\delta(-, c) \in \mathcal{M}_{R,\Sigma} \wr Q$, and the SST is copyless if and only if $\{\delta(-, c) \mid c \in \Gamma\} \subseteq \mathcal{M}^{\mathrm{cl}}_{R,\Sigma} \wr Q$. Furthermore, for all $w_1 \dots w_n \in \Gamma^*$,*

$$f(w_1 \dots w_n) \quad = \quad F(g(q_0))^\dagger(\alpha^\dagger(\vec{v})) \quad \text{where} \quad (g, \alpha) = \delta(-, w_1) \bullet \dots \bullet \delta(-, w_n)$$

Finally, it will sometimes be useful to consider monoids of assignments over an *empty* output alphabet. This allows us to keep track of how the registers are shuffled around by transitions.

▶ **Proposition 2.21.** *Let $R$ and $\Sigma$ be disjoint finite sets. There is a monoid morphism $\mathcal{M}_{R,\Sigma} \to \mathcal{M}_{R,\varnothing}$, that sends the submonoid $\mathcal{M}^{\mathrm{cl}}_{R,\Sigma}$ to $\mathcal{M}^{\mathrm{cl}}_{R,\varnothing}$. For any $Q$, this extends to a morphism $\mathcal{M}_{R,\Sigma} \wr Q \to \mathcal{M}_{R,\varnothing} \wr Q$ that sends $\mathcal{M}^{\mathrm{cl}}_{R,\Sigma} \wr Q$ to $\mathcal{M}^{\mathrm{cl}}_{R,\varnothing} \wr Q$. We shall use the name $\mathtt{erase}_\Sigma$ for both morphisms ($R$ and $Q$ being inferred from the context).*

▶ Remark 2.22. Consider an SST with a transition function $\delta$. Let $\varphi_\delta \in \mathrm{Hom}(\Gamma^*, \mathcal{M}^{\mathrm{cl}}_{R,\varnothing} \wr Q)$ be defined by $\varphi_\delta(c) = \mathtt{erase}_\Sigma(\delta(-, c))$ for $c \in \Gamma$. The range $\varphi_\delta(\Gamma^*)$ is precisely the *substitution transition monoid (STM)* defined in [11, Section 3].

▶ **Proposition 2.23.** *For any finite $R$, the monoid $\mathcal{M}^{\mathrm{cl}}_{R,\varnothing}$ is* finite. *As a consequence, the substitution transition monoid of any* copyless *SST is finite.*

**Proof idea.** For all $\alpha \in \mathcal{M}^{\mathrm{cl}}_{R,\varnothing}$ and $r \in R$, observe that $|\alpha(r)| \leq |R|$. ◀

## 3 Complements on HDT0L systems, SSTs and polyregular functions

Before embarking on the study of our new comparison-free polyregular functions, we state some minor results that consolidate our understanding of pre-existing classes.

**Layered HDT0L systems.** Let us transpose the layering condition from SSTs to HDT0L systems. The hierarchy of models that we get corresponds *with an offset* to layered SSTs.

▶ **Definition 3.1.** *A HDT0L system $(\Gamma, \Sigma, \Delta, d, (h_c)_{c \in \Gamma}, h')$ is $k$-*layered *if its working alphabet can be partitioned as $\Delta = \Delta_0 \sqcup \dots \sqcup \Delta_k$ such that, for all $c \in \Gamma$ and $i \in \{0, \dots, k\}$:*

- *for $r \in \Delta_i$, we have $h_c(r) \in (\Delta_0 \sqcup \dots \sqcup \Delta_i)^*$;*
- *each letter in $\Delta_i$ appears at most once among all the $\alpha(r)$ for $r \in \Delta_i$ (but those in $\Delta_0 \sqcup \dots \sqcup \Delta_{i-1}$ may appear an arbitrary number of times).*

▶ **Theorem 3.2.** *For $k \in \mathbb{N}$, a function can be computed by a $k$-layered SST if and only if it can be specified by a $(k+1)$-layered HDT0L system.*

*In particular, regular functions correspond to 1-layered HDT0L systems.*

The obvious translation from HDT0L systems to SSTs preserves 1-layeredness and produces a single-state machine, so one may sacrifice copylessness to eliminate states for SSTs.

▶ **Corollary 3.3.** *Every regular function can be computed by a single-state 1-layered SST.*

The converse to this corollary does not hold: the single-state 1-layered SST of Example 2.4 computes a function which is not regular (cf. Remark 2.9).

**Polyregular functions vs layered SSTs.** By applying some results from [4], we can state a variant of Definition 2.11 which is a bit more convenient for us.

▶ **Proposition 3.4.** *Polyregular functions are the smallest class closed under composition that contains the regular functions and the squaring with underlining functions* $\mathtt{squaring}_\Gamma$.

This allows us to show that composing HDT0L transductions with at most polynomial growth yields the polyregular functions. One direction of this equivalence is proved by encoding $\mathtt{squaring}_\Gamma$ as a composition of two SSTs, one of which is Example 2.4. More precisely:

▶ **Theorem 3.5.** *Let $f : \Gamma^* \to \Sigma^*$. The following are equivalent:*
**(i)** *$f$ is polyregular;*
**(ii)** *$f$ can be obtained as a composition of layered SSTs;*
**(iii)** *$f$ can be obtained as a composition of single-state 1-layered SSTs.*

But layered SSTs by themselves are *strictly less expressive* than polyregular functions, as we shall see later in Theorem 8.1. Therefore, as promised in the introduction:

▶ **Corollary 3.6** (claimed in [13, Section 6])**.** *Layered SSTs are* not *closed under composition.*

**The importance of being stateful.** One interesting aspect of Theorem 3.2 is that 1-layered HDT0L systems can be seen, through Remark 2.6, as a kind of one-way transducer model for regular functions that does not use an explicit control state. This is in contrast with copyless SSTs, whose expressivity critically depends on the states (unlike copyful SSTs).

▶ **Proposition 3.7.** *The sequential (and therefore regular) function defined by the transducer of Figure 2 (Section 2.2)* cannot *be computed by a* single-state copyless SST*.*

In fact, the knowledgeable reader can verify that this counterexample belongs to the *first-order letter-to-letter sequential functions*, one of the weakest classical transduction classes.

**Closure under map.** The pattern of Example 2.10 (iterated reverse) can be generalized:

▶ **Definition 3.8.** *Let $f : \Gamma^* \to \Sigma^*$ and suppose that $\# \notin \Gamma \cup \Sigma$. We define the function* $\mathbf{map}(f) : w_1\#\dots\#w_n \in (\Gamma \cup \{\#\})^* \mapsto f(w_1)\#\dots\#f(w_n) \in (\Sigma \cup \{\#\})^*$.

▶ **Proposition 3.9.** *If $f$ is an HDT0L transduction, then so is* $\mathbf{map}(f)$*. For each $k \geq 1$, the functions that can be computed by $k$-layered HDT0L systems are also closed under* $\mathbf{map}$.

As an immediate corollary, closure under $\mathbf{map}$ holds for both regular and polyregular functions, but this was already known. In fact, $\mathtt{map}(f, [x_1, \dots, x_n]) = [f(x_1), \dots, f(x_n)]$ is an essential primitive in the *regular list functions* [6] and *polynomial list functions* [4, §4], two list-processing programming languages that characterize regular and polyregular functions respectively. We will come back to this point in Corollary 8.5 and the subsequent remark.

## 4 Composition by substitution

At last, we now introduce the class of *comparison-free polyregular functions*. The simplest way to define them is to start from the regular functions.

▶ **Definition 4.1.** *Let $f : \Gamma^* \to I^*$, and for each $i \in I$, let $g_i : \Gamma^* \to \Sigma^*$. The* composition by substitution *of $f$ with the family $(g_i)_{i\in I}$ is the function*

$$\mathrm{CbS}(f, (g_i)_{i\in I}) \quad : \quad w \;\mapsto\; g_{i_1}(w) \dots g_{i_k}(w) \quad \textit{where } i_1 \dots i_k = f(w)$$

*That is, we first apply $f$ to the input, then every letter $i$ in the result of $f$ is substituted by the image of the original input by $g_i$. Thus, $\mathrm{CbS}(f, (g_i)_{i\in I})$ is a function $\Gamma^* \to \Sigma^*$.*

▶ **Definition 4.2.** *The smallest class of string-to-string functions closed under* CbS *and containing all regular functions is called the class of* comparison-free polyregular functions.

▶ **Example 4.3.** The following variant of “squaring with underlining” (cf. Definition 2.11) is comparison-free polyregular: $\texttt{cfsquaring}_\Gamma : \mathtt{123} \in \Gamma^* \mapsto \mathtt{\underline{1}123\underline{2}123\underline{3}123} \in (\Gamma \cup \underline{\Gamma})^*$.

Indeed, it can be expressed as $\texttt{cfsquaring}_\Gamma = \mathrm{CbS}(f, (g_i)_{i\in I})$ where $I = \Gamma \cup \{\#\}$, the function $f : w_1 \dots w_n \mapsto w_1\# \dots w_n\#$ is regular (more than that, a morphism between free monoids) and $g_\# = \mathrm{id}$, $g_c : w \mapsto \underline{c}$ for $c \in \Gamma$ are also regular. Its growth rate is quadratic, while regular functions have at most linear growth. Other examples that also require a single composition by substitution are given in Theorem 8.1.

We can already justify the latter half of the name of our new class. Using the “polynomial list functions” mentioned at the end of the previous section, we prove:

▶ **Theorem 4.4.** *Polyregular functions are closed under composition by substitution.*

▶ **Corollary 4.5.** *Every comparison-free polyregular function is, indeed, polyregular.*

Fundamentally, Definition 4.2 is inductive: it considers the functions generated from the base case of regular functions by applying compositions by substitution. The variant below with more restricted generators is sometimes convenient.

▶ **Definition 4.6.** *A string-to-string function is said to be:*

- *of* rank at most $0$ *if it is regular;*
- *of* rank at most $k+1$ (for $k \in \mathbb{N}$) *if it can be written as $\mathrm{CbS}(f, (g_i)_{i\in I})$ where $f : \Gamma^* \to I^*$ is* regular *and each $g_i : \Gamma^* \to \Sigma^*$ is of rank at most $k$.*

▶ **Proposition 4.7.** *A function $f$ is comparison-free polyregular if and only if there exists some $k \in \mathbb{N}$ such that $f$ has rank at most $k$. In that case, we write $\mathrm{rk}(f)$ for the least such $k$ and call it the* rank *of $f$. If $(g_i)_{i\in I}$ is a family of comparison-free polyregular functions,*

$$\mathrm{rk}(\mathrm{CbS}(f, (g_i)_{i\in I})) \leq 1 + \mathrm{rk}(f) + \max_{i\in I} \mathrm{rk}(g_i)$$

A straightforward consequence of this definition is that, just like regular functions, cfp functions are closed under *regular conditionals* and concatenation.

▶ **Proposition 4.8.** *Let $f, g : \Gamma^* \to \Sigma^*$ be comparison-free polyregular functions and $L \subseteq \Gamma^*$ be a regular language. The function that coincides with $f$ on $L$ and with $g$ on $\Gamma^* \setminus L$ is cfp, and so is $w \in \Gamma^* \mapsto f(w) \cdot g(w)$; both have rank at most* $\max(\mathrm{rk}(f), \mathrm{rk}(g))$.

## 5 Comparison-free pebble transducers

We now characterize our function class by a machine model that will explain our choice of the adjective "comparison-free", as well as the operational meaning of the notion of rank we just defined. It is based on the *pebble transducers* first introduced for trees by Milo, Suciu and Vianu [28] and later investigated in the special case of strings by Engelfriet and Maneth [17, 14]. However, the definition using composition by substitution will remain our tool of choice to prove further properties, so the next sections do not depend on this one.

▶ **Definition 5.1.** *Let $k \in \mathbb{N}$ with $k \geq 1$. Let $\Gamma, \Sigma$ be finite alphabets and $\triangleright, \triangleleft \notin \Gamma$.*

*A* $k$-pebble stack *on an input string $w \in \Gamma^*$ consists of an ordered list of $p$ positions in the string $\triangleright w \triangleleft$ (i.e. of $p$ integers between 1 and $|w|+2$) for some $p \in \{1, \dots, k\}$. We therefore write* $\mathrm{Stack}_k = \mathbb{N}^0 \cup \mathbb{N}^1 \cup \dots \cup \mathbb{N}^k$*, keeping in mind that given an input $w$, we will be interested in "legal" values bounded by $|w|+2$.*

*A* comparison-free $k$-pebble transducer ($k$-CFPT) *consists of a finite set of states $Q$, an initial state $q_I \in Q$ and a family of transition functions*

$$Q \times (\Gamma \cup \{\triangleright, \triangleleft\})^p \to Q \times (\mathbb{N}^p \to \mathrm{Stack}_k) \times \Sigma^* \quad \textit{for } 1 \leq p \leq k$$

*where the $\mathbb{N}^p$ on the left is considered as a subset of* $\mathrm{Stack}_k$*. For a given state and given letters $(c_1, \dots, c_p) \in (\Gamma \cup \{\triangleright, \triangleleft\})^p$, the allowed values for the stack update function $\mathbb{N}^p \to \mathrm{Stack}_k$ returned by the transition function are:*

| | | | | | |
|---|---|---|---|---|---|
| *(identity)* | $(i_1, \dots, i_p)$ | $\mapsto$ | $(i_1, \dots, i_p)$ | $\in$ | $\mathbb{N}^p$ |
| *(move left, only allowed when $c_p \neq \triangleright$)* | $(i_1, \dots, i_p)$ | $\mapsto$ | $(i_1, \dots, i_p - 1)$ | $\in$ | $\mathbb{N}^p$ |
| *(move right, only allowed when $c_p \neq \triangleleft$)* | $(i_1, \dots, i_p)$ | $\mapsto$ | $(i_1, \dots, i_p + 1)$ | $\in$ | $\mathbb{N}^p$ |
| *(push, only allowed when $p \leq k-1$)* | $(i_1, \dots, i_p)$ | $\mapsto$ | $(i_1, \dots, i_p, 1)$ | $\in$ | $\mathbb{N}^{p+1}$ |
| *(pop, only allowed when $p \geq 1$)* | $(i_1, \dots, i_p)$ | $\mapsto$ | $(i_1, \dots, i_{p-1})$ | $\in$ | $\mathbb{N}^{p-1}$ |

*(Note that the codomains of all these functions are indeed subsets of* $\mathrm{Stack}_k$*.)*

The run of a CFPT over an input string $w \in \Gamma^*$ starts in the initial configuration comprising the initial state $q_I$, the initial $k$-pebble stack $(1) \in \mathbb{N}^1$, and the empty string as an initial output log. As long as the current stack is non-empty a new configuration is computed by applying the transition function to $q$ and to $((\triangleright w \triangleleft)[i_1], \dots, (\triangleright w \triangleleft)[i_p])$ where $(i_1, \dots, i_p)$ is the current stack; the resulting stack update function is applied to $(i_1, \dots, i_p)$ to get the new stack, and the resulting output string in $\Sigma^*$ is appended to the right of the current output log. If the CFPT ever terminates by producing an empty stack, the *output associated to $w$* is the final value of the output log.

This amounts to restricting in two ways[4] the definition of *pebble transducers* from [4, §2]:

- in a general pebble transducer, one can *compare positions*, i.e. given a stack $(i_1, \dots, i_p)$, the choice of transition can take into account whether[5] $i_j \leq i_{j'}$ (for any $1 \leq j, j' \leq p$);
- in a "push", new pebbles are initialized to the leftmost position ($\triangleright$) for a CFPT, instead of starting at the same position as the previous top of the stack (the latter would ensure the equality of two positions at some point; it is therefore an implicit comparison that we must relinquish to be truly "comparison-free").

[4] There is also an inessential difference: the definition given in [4] does not involve end markers and handles the edge case of an empty input string separately. This has no influence on the expressiveness of the transducer model. Our use of end markers follows [15, 25].

[5] One would get the same computational power, with the same stack size, by only testing whether $i_j = i_p$ for $j \leq p-1$ as in [28] (this is also essentially what happens in the nested transducers of [25]).

This limitation is similar to (but goes a bit further than) the "invisibility" of pebbles in a transducer model introduced by Engelfriet et al. [16] (another difference, unrelated to position comparisons, is that their transducers use an unbounded number of invisible pebbles).

▶ Remark 5.2. Our definition guarantees that "out-of-bounds errors" cannot happen during the run of a comparison-free pebble transducer. The sequence of successive configurations is therefore always well-defined. But it may be infinite, that is, it may happen that the final state is never reached. Thus, a CFPT defines a *partial* function.

That said, the set of inputs for which a given pebble tree transducer does not terminate is always a regular language [28, Theorem 4.7]. This applies *a fortiori* to CFPTs. Using this, it is possible[6] to extend any partial function $f : \Gamma^* \rightharpoonup \Sigma^*$ computed by a $k$-CFPT into a total function $f' : \Gamma^* \to \Sigma^*$ computed by another $k$-CFPT for the same $k \in \mathbb{N}$, such that $f'(x) = f(x)$ for $x$ in the domain of $f$ and $f'(x) = \varepsilon$ otherwise. This allows us to *only consider CFPTs computing total functions* in the remainder of the paper.

A special case of particular interest is $k = 1$: the transducer has a single reading head, push and pop are always disallowed.

▶ **Theorem 5.3** ([1]). *Copyless SSTs and 1-CFPTs – which are more commonly called* two-way (deterministic) finite transducers (2DFTs) *– are equally expressive.*

Since we took copyless SSTs as our reference definition of regular functions, this means that 2DFTs characterize regular functions. But putting it this way is historically backwards: the equivalence between 2DFTs and MSO transductions came first [15] and made this class deserving of the name "regular functions" before the introduction of copyless SSTs.

▶ Remark 5.4. There are two different numbering conventions for pebble transducers. In [4, 25], 2DFTs are 1-pebble transducers, which is consistent with our choice. However, several other papers (e.g. [28, 17, 14, 16, 12]) consider that a 2DFT is a *0-pebble* transducer (likewise, in [13], 2DFTs are 0-marble transducers). This is because they think of a pebble automaton not as a restricted multi-head automaton, but as an enriched 2DFA that can drop stationary markers (called pebbles) on input positions, with a single moving head that is not a pebble.

Let us now show the equivalence with Definition 4.2. The reason for this is similar to the reason why $k$-pebble transducers are equivalent to the *$k$-nested transducers*[7] of [25], which is deemed "trivial" and left to the reader in [25, Remark 6]. But in our case, one direction (Theorem 5.6) involves an additional subtlety compared to in [25]; to take care of it, we use the fact that the languages recognized by pebble automata are regular (this is also part of [28, Theorem 4.7]) together with regular conditionals (Proposition 4.8).

▶ **Proposition 5.5.** *If $f$ is computed by a $k$-CFPT, and the $g_i$ are computed by $l$-CFPTs, then* $\mathrm{CbS}(f, (g_i)_{i\in I})$ *is computed by a* $(k + l)$*-CFPT.*

▶ **Theorem 5.6.** *If $f : \Gamma^* \to \Sigma^*$ is computed by a $k$-CFPT, for $k \geq 2$, then there exist a finite alphabet $I$, a regular function $h : \Gamma^* \to I^*$ and a family $(g_i)_{i\in I}$ computed by $(k - 1)$-CFPTs such that $f = \mathrm{CbS}(h, (g_i)_{i\in I})$.*

▶ **Corollary 5.7.** *For all $k \in \mathbb{N}$, the functions computed by $(k + 1)$-CFPTs are exactly the comparison-free polyregular functions of rank at most $k$.*

[6] Proof idea: do a first left-to-right pass to determine whether the input leads to non-termination of the original CFPT; if so, terminate immediately with an empty output; otherwise, move the first pebble back to the leftmost position and execute the original CFPT's behavior. This can be implemented by adding finitely many states, including those for a DFA recognizing non-terminating inputs.

[7] Remark: nested transducers should yield a machine-independent definition of polyregular functions as the closure of regular functions under a CbS-like operation that relies on *origin semantics* [29, §5].

## 6 Composition of basic functions

Another possible definition of cfp functions consists in swapping out $\texttt{squaring}_\Gamma$ for some other function in Proposition 3.4:

▶ **Theorem 6.1.** *The class of comparison-free polyregular functions is the smallest class closed under usual function composition and containing both all regular functions and the functions* $\texttt{cfsquaring}_\Gamma$ *(cf. Example 4.3) for all finite alphabets* $\Gamma$*.*

The hard part is to show that cfp functions are closed under composition. We exploit the following combinatorial phenomenon, often applied to the study of copyless SSTs: a copyless register assignment, i.e. an element of $\mathcal{M}^{\mathrm{cl}}_{R,\Delta}$ (cf. Section 2.4), can be specified by

- a "shape" described by an element of the *finite* monoid $\mathcal{M}^{\mathrm{cl}}_{R,\varnothing}$ (Proposition 2.23),
- plus finitely many "labels" in $\Sigma^*$ (where $\Sigma$ is the output alphabet) describing the constant factors that will be concatenated with the old register contents to give the new ones.

▶ **Proposition 6.2.** *There is a bijection*

$$\mathcal{M}^{\mathrm{cl}}_{R,\Delta} \quad\cong\quad \left\{ \left(\alpha, \vec{\ell}\right) \;\middle|\; \alpha \in \mathcal{M}^{\mathrm{cl}}_{R,\varnothing},\ \vec{\ell} \in \prod_{r\in R} (\Delta^*)^{|\alpha(r)|+1} \right\}$$

*through which* $\texttt{erase}_\Delta : \mathcal{M}^{\mathrm{cl}}_{R,\Delta} \to \mathcal{M}^{\mathrm{cl}}_{R,\varnothing}$ *can be seen as simply removing the "labels"* $\vec{\ell}$*.*

**Proof idea.** Let $\beta \in \mathcal{M}^{\mathrm{cl}}_{R,\Delta}$. For each $r \in R$, one can write $\beta(r) = w_0 r'_1 w_1 \dots r'_n w_n$ with $w_0, \dots, w_n \in \Delta^*$ and $r'_1, \dots, r'_n \in R$ such that $r'_1 \dots r'_n = \texttt{erase}_\Delta(\beta)(r) \in R^*$. ◀

This provides a clear way to represent a copyless register assignment inside the working memory of an SST: store the shape in the state and the labels in registers. Another important fact for us is that given two assignments $\beta, \beta' \in \mathcal{M}^{\mathrm{cl}}_{R,\Delta}$ the labels of $\beta \bullet \beta'$ can be obtained as a *copyless* recombination of the labels of $\beta$ and $\beta'$.

(There is a subtlety worth mentioning here: while the set of stateful transitions $\mathcal{M}^{\mathrm{cl}}_{R,\Delta} \wr Q$ also admits a "shape + labels" representation, its monoid multiplication does *not* have this copylessness property. This prevents a naive proof of the closure under composition of copyless SSTs from working. Nevertheless, the composition of two regular functions *is* always regular, and we rely on this fact to prove Theorem 6.1.)

The rest of the proof of Theorem 6.1 is relegated to the technical appendix.

## 7 Rank vs asymptotic growth

Our next result is the comparison-free counterpart to recent work on polyregular functions by Lhote [25], whose proof techniques (in particular the use of Ramsey's theorem) we reuse. Compare item (ii) below to the main theorem of [25] and item (iii) – which provides yet another definition of cfp functions – to [25, Appendix A].

▶ **Theorem 7.1.** *Let* $f : \Gamma^* \to \Sigma^*$ *and* $k \in \mathbb{N}$*. The following are equivalent:*

**(i)** *$f$ is comparison-free polyregular with rank at most $k$;*
**(ii)** *$f$ is comparison-free polyregular and* $|f(w)| = O(|w|^{k+1})$*;*
**(iii)** *there exists a regular function* $g : (\{0, \dots, k\} \times \Gamma)^* \to \Sigma^*$ *such that* $f = g \circ \texttt{cfpow}^{(k+1)}_\Gamma$*, with the following inductive definition:* $\texttt{cfpow}^{(0)}_\Gamma : w \in \Gamma^* \mapsto \varepsilon \in (\varnothing \times \Gamma)^*$ *and*

$$\texttt{cfpow}^{(n+1)}_\Gamma \quad:\quad w \mapsto (n, w_1) \cdot \texttt{cfpow}^{(n)}_\Gamma(w) \cdot \ldots \cdot (n, w_{|w|}) \cdot \texttt{cfpow}^{(n)}_\Gamma(w)$$

*To make (ii) $\implies$ (i) more precise, if $f$ is cfp with* $\mathrm{rk}(f) \geq 1$*, then it admits a sequence of inputs* $w_0, w_1, \ldots \in \Gamma^*$ *such that* $|w_n| \to +\infty$ *and* $|f(w_n)| = \Omega(|w_n|^{\mathrm{rk}(f)+1})$*.*

Note that $\texttt{cfpow}^{(2)}_{\Gamma}$ and $\texttt{cfsquaring}_{\Gamma}$ are the same up to a bijection $\{0,1\} \times \Gamma \cong \Gamma \cup \underline{\Gamma}$.

▶ Remark 7.2. The growth of an HDT0L transduction is also related, in a very similar way to item (ii) above, to the number of layers required in any SST that computes it [13, §5].

**Some proof elements.** Let us present a few definitions and lemmas to give an idea of the ingredients that go into the proof. Those technical details take up the rest of this section.

Lhote's paper [25] makes a heavy use of *factorizations* of strings that depend on a morphism to a finite monoid. This is also the case for our proof, but we have found that a slightly different definition of the kind of factorization that we want works better for us.

▶ **Definition 7.3** (similar but not equivalent to [25, Definition 19]). *An* $r$-split *of a string* $s \in \Gamma^*$ *according to a morphism* $\varphi : \Gamma^* \to M$ *is a tuple* $(u, v_1, \dots, v_r, w) \in (\Gamma^*)^{r+2}$ *such that:*

- $s = uv_1 \dots v_n w$ *with* $v_i$ *non-empty for all* $i \in \{1, \dots, r\}$*;*
- $\varphi(u) = \varphi(uv_1) = \dots = \varphi(uv_1 v_r)$*;*
- $\varphi(w) = \varphi(v_r w) = \dots = \varphi(v_1 \dots v_r w)$*.*

▶ **Proposition 7.4** (immediate from the definition). $(u, v_1, \dots, v_r, w)$ *is an* $r$*-split if and only if, for all* $i \in \{1, \dots, r\}$*,* $(uv_1 \dots v_{i-1}, v_i, v_{i+1} \dots v_r w)$ *is a 1-split.*

The difference with the $(1, r)$-*factorizations* of [25, Definition 19] is that we have replaced the equality and idempotency requirements on $\varphi(v_1), \dots, \varphi(v_n)$ by the "boundary conditions" involving $\varphi(u)$ and $\varphi(w)$ (actually, $(1, r+2)$-factorizations induce $r$-splits). This change allows us to establish a subclaim used in the proof of Lemma 7.7 in an elementary way.

The point of $r$-splits is that given a split of an input string according to the morphism that sends it to the corresponding transition in a SST, we have some control over what happens to the output of the SST if we pump a middle factor in the split. Furthermore, it suffices to consider a quotient of the transition monoid which is finite when the SST is copyless (this is similar to Proposition 2.23). More precisely, we have the key lemma below, which is used pervasively throughout our proof of Theorem 7.1:

▶ **Lemma 7.5.** *Let* $f : \Gamma^* \to \Sigma^*$ *be a regular function. There exist a morphism to a* finite *monoid* $\nu_f : \Gamma^* \to \mathcal{N}(f)$ *and, for each* $c \in \Sigma$*, a set of* producing triples $P(f, c) \subseteq \mathcal{N}(f)^3$ *such that, for any* 1-split *according to* $\nu_f$ *composed of* $u, v, w \in \Gamma^*$ *– i.e.* $\nu_f(uv) = \nu_f(u)$ *and* $\nu_f(vw) = \nu_f(w)$ *– we have:*

- *if* $(\nu_f(u), \nu_f(v), \nu_f(w)) \in P(f, c)$*, then* $|f(uvw)|_c > |f(uw)|_c$*;*
- *otherwise (when the triple is not producing),* $|f(uvw)|_c = |f(uw)|_c$*.*

*Furthermore, in the producing case, we get as a consequence that* $\forall n \in \mathbb{N}$*,* $|f(uv^n w)|_c \geq n$*.*

▶ **Definition 7.6.** *We fix once and for all a choice of* $\mathcal{N}(f)$*,* $\nu_f$ *and* $P(f, c)$ *for each* $c \in \Sigma$ *and regular* $f : \Gamma^* \to \Sigma^*$*. We say that a 1-split* $(u, v, w)$ *is* producing with respect to $(f, c)$ *when* $(\nu_f(u), \nu_f(v), \nu_f(w)) \in P(f, c)$*. For* $\Pi \subseteq \Sigma$*, we also set* $P(f, \Pi) = \bigcup_{c \in \Pi} P(f, c)$*.*

Something like Lemma 7.5 (but not exactly) appears in the proof of [25, Lemma 18]. We first apply it to prove the following lemma, which is morally a counterpart to the "$k = 1$ case" of the central Dichotomy Lemma from [25], with $r$-splits instead of $(k, r)$-factorizations.

▶ **Lemma 7.7.** *Let* $f : \Gamma^* \to \Sigma^*$ *be regular and* $\varphi : \Gamma^* \to M$ *be a morphism with* $M$ *finite. Suppose that* $\pi \circ \varphi = \nu_f$ *for some other morphism* $\pi : M \to \mathcal{N}(f)$*. Let* $r \geq 1$ *and* $\Pi \subseteq \Sigma$*.*

*We define* $L(f, \Pi, \varphi, r)$ *to be the set of strings that admit an* $r$*-split* $s = uv_1 \dots v_r w$ *according to* $\varphi$ *such that all the triples* $(uv_1 \dots v_{i-1}, v_i, v_{i+1} \dots v_r w)$ *are producing with respect to* $(f, \Pi)$ *– let us call this a producing* $r$*-split with respect to* $(f, \Pi, \varphi)$*.*

*Then* $L(f, \Pi, \varphi, r)$ *is a regular language, and* $\sup\{|f(s)|_\Pi \mid s \in \Gamma^* \setminus L(f, \Pi, \varphi, r)\} < \infty$*.*

Our proof of the above lemma uses the proposition below, analogous to [25, Claim 20]. Its statement is a bit stronger than necessary for this purpose, but it will be reused in the proof of Theorem 8.3; as for its proof, this is where a standard Ramsey argument occurs.

▶ **Proposition 7.8.** *Let $\Gamma$ be an alphabet, $M$ be a finite monoid and $\varphi : \Gamma^* \to M$ be a morphism. There exists $N \in \mathbb{N}$ such that any string $s = uvw \in \Gamma^*$ such that $|v| \geq N$ admits an $r$-split $s = u'v'_1 \dots v'_r w'$ according to $\varphi$ in which $u$ is a prefix of $u'$ and $w$ is a suffix of $w'$.*

To leverage Lemma 7.7, we combine it with an elementary property of composition by substitution that does not depend on the previous technical development. (Compare the assumptions of the lemma below with the conclusion of Lemma 7.7.)

▶ **Lemma 7.9.** *Let $g : \Gamma^* \to I^*$ be a regular function and, for each $i \in I$, let $h_i : \Gamma^* \to \Sigma^*$ be comparison-free polyregular of rank at most $k$. Suppose that $\sup_{s \in \Gamma^*} |g(s)|_J < \infty$ where*

$$J = \begin{cases} \{i \in I \mid \mathrm{rk}(h_i) = k\} & \text{when } k \geq 1 \\ \{i \in I \mid |h_i(\Gamma^*)| = \infty\} & \text{when } k = 0 \end{cases}$$

*(Morally, regular functions with finite range play the role of "comparison-free polyregular functions of rank $-1$".) Then $\mathrm{rk}(\mathrm{CbS}(g, (h_i)_{i \in I})) \leq k$.*

The above lemma can be compared to [25, Claim 22], but it also seems to be related to the way the "nested transducer" $R_{k+1}$ is defined in the proof of the Dichotomy Lemma in [25]: indeed, $R_{k+1}$ can call either a $k$-nested subroutine or a $(k-1)$-nested one.

The remainder of the proof of Theorem 7.1 consists mainly of a rather technical induction on the rank, which we present in the appendix.

## 8 Separation results

Let us now demonstrate that the class of cfp functions is incomparable with the class of HDT0L transductions and is a strict subclass of polyregular functions.

▶ **Theorem 8.1.** *There exist comparison-free polyregular functions which are not HDT0L:*

**(i)** *the function $a^n \in \{a\}^* \mapsto (a^n b)^{n+1} \in \{a, b\}^*$ for $a \neq b$;*

**(ii)** *the function $w \in \Sigma^* \mapsto w^{|w|}$ for $|\Sigma| \geq 2$ (a simplification of Example 4.3);*

**(iii)** (from [13, §6]) *the cfp functions that map $a^n \# w \in \Sigma^*$ to $(w\#)^n$ for $a, \# \in \Sigma$, $a \neq \#$.*

▶ Remark 8.2. The first example in [13, §5] shows that $a^n \mapsto a^{n \times n}$ is HDT0L (via the equivalent model of marble transducers), hence the necessity of $|\Sigma| \geq 2$ above. More generally, Douéneau-Tabot has shown very recently that *every polyregular function with unary output alphabet is HDT0L* [12]. So polyregular functions with unary output coincide with *polynomial growth* $\mathbb{N}$*-rational series* (cf. Remark 2.14), and the latter admit several algebraic characterizations in the literature (see [32] and [3, Chapter 9, Exercise 1.2]).

▶ **Theorem 8.3.** *Some HDT0L transductions are polyregular but not comparison-free:*

**(i)** *$f : a^n \in \{a\}^* \mapsto ba^{n-1}b \dots baabab$ (with $f(\varepsilon) = \varepsilon$ and $f(a) = b$);*

**(ii)** $\mathbf{map}(a^n \mapsto a^{n \times n}) : a^{n_1} \# \dots \# a^{n_k} \mapsto a^{n_1 \times n_1} \# \dots \# a^{n_k \times n_k}$ *(cf. Definition 3.8).*

▶ Remark 8.4. The function $a^{n_1} \# \dots \# a^{n_k} \mapsto a^{n_1 \times n_1 + \dots + n_k \times n_k}$ obtained by erasing the #s in the output of $\mathbf{map}(a^n \mapsto a^{n \times n})$ is also *not* comparison-free. This result implies the second item of Theorem 8.3 by composition with the erasing morphism; we do not prove it here, but it appears in Douéneau-Tabot's aforementioned paper [12]. Therefore, according to [12], *not* every polyregular function with unary output is comparison-free.

To see why the first of the two functions in Theorem 8.3 is HDT0L, observe that it is Example 2.4 for $\Gamma = \{a\}$ (taking $b = \underline{a}$). As for the second one, combine Proposition 3.9 and the first observation in Remark 8.2.

The non-membership parts of Theorems 8.1 and 8.3 require more work. For the former, we use pumping arguments on HDT0L systems. Item (ii) of Theorem 8.3 is handled by first appealing to Theorem 7.1 to reduce to showing that $\mathbf{map}(a^n \mapsto a^{n\times n}) \neq \mathrm{CbS}(g, (h_i)_{i\in I})$ when $g$ and all the $h_i$ are *regular* functions; a combination of pumping and of a combinatorial argument then shows that inputs with $|I|$ occurrences of $\#$ suffice to discriminate the two sides of the inequality. This result also has the following consequence:

▶ **Corollary 8.5.** *Comparison-free polyregular functions are not closed under* **map**.

▶ Remark 8.6. Contrast with Proposition 3.9. The discussion that follows that proposition lends some significance to the above corollary: the latter rules out the obvious conjectures for a characterization of cfp functions in the style of regular/polynomial list functions.

As for item (i) of Theorem 8.3, it concerns a function whose domain consists of words over a unary alphabet, i.e., up to isomorphism, a *sequence*. This motivates the study of such sequences, which is the subject of the next section.

## 9 Comparison-free polyregular sequences

From now on, we identify $\mathbb{N}$ with the set of words $\{a\}^*$ and freely speak, for instance, of cfp sequences $\mathbb{N} \to \Gamma^*$ instead of cfp functions $\{a\}^* \to \Gamma^*$. It turns out that cfp sequences admit a characterization as finite combinations of what we call *poly-pumping sequences*.

▶ **Definition 9.1.** *A* poly-pumping sequence *is a function of the form* $[\![e]\!] : \mathbb{N} \to \Sigma^*$ *where*

- $e$ *is a* polynomial word expression *generated by* $e \ ::= \ w \mid e \cdot e' \mid e^*$ *where* $w \in \Sigma^*$*;*
- $[\![w]\!](n) = w$*,* $[\![e \cdot e']\!](n) = [\![e]\!](n)[\![e']\!](n)$ *and* $[\![e^*]\!](n) = ([\![e]\!](n))^n$*.*

*The* star-height *of a polynomial word expression is defined in the usual way.*

▶ **Theorem 9.2.** *Let* $s : \mathbb{N} \to \Sigma^*$ *and* $k \in \mathbb{N}$*. The sequence* $s$ *is comparison-free polyregular with* $\mathrm{rk}(s) \leq k$ *if and only if there exists* $p > 0$ *such that, for any* $m < p$*, there is a polynomial word expression* $e$ *of star-height at most* $k+1$ *such that* $\forall n \in \mathbb{N}$*,* $s((n+1)p+m) = [\![e]\!](n)$*.*

In short, the cfp sequences are exactly the *ultimately periodic combinations* of poly-pumping sequences. Our proof strategy is an induction on $k$.

The base case $k = 0$ says that regular sequences are ultimately periodic combinations of *pumping sequences* $n \mapsto u_0(v_1)^n \dots (v_l)^n u_l$. An essentially equivalent result is stated with a proof sketch using 2DFTs in [10, p. 90]; we propose an alternative proof using copyless SSTs. (Non-deterministic two-way transducers (2NFTs) taking unary inputs have also been studied [23]; furthermore, the notion of "$k$-iterative language" that appears in a pumping lemma for general 2NFTs [35] is related to the shape of the above pumping sequences.)

To make the inductive step go through, it is enough to synchronize the periods of the different poly-pumping sequences involved and to observe that $\mathrm{CbS}([\![e]\!], ([\![e'_i]\!])_{i\in I})$ is realized by an expression obtained by substituting the $e'_i$ for $i$ in $e$.

Coming back to Theorem 8.3, we show that $a^n \mapsto ba^{n-1}b \dots bab$ is not comparison-free polyregular by proving that its subsequences are *not* poly-pumping: for every poly-pumping sequence $s : \mathbb{N} \to \{a,b\}^*$, there is a uniform bound on the number of distinct contiguous subwords of the shape $baa \dots ab$ occuring in each $s(n)$ for $n \in \mathbb{N}$. Another consequence of Theorem 9.2 that we establish by induction over expressions contrasts with Corollary 8.5:

▶ **Corollary 9.3.** *If* $f : \Gamma^* \to \Sigma^*$ *and* $s : \mathbb{N} \to (\Gamma \cup \{\#\})^*$ *are cfp, so is* $\mathbf{map}(f) \circ s$*.*

## 10 Further topics

**Functional programming.** We mentioned in the introduction a forthcoming characterization of cfp functions using Church-encoded strings in a $\lambda$-calculus with linear types, in the vein of our previous results [31, 30]. Meanwhile, Corollary 8.5 could be understood as negative result in the search for another kind of functional programming characterization (cf. Remark 8.6).

It is also worth noting that the copying discipline of layered SSTs is very similar to what happens in the *parsimonious $\lambda$-calculus* [27]: a datum of type $!\tau$ cannot be duplicated into two copies of the same type $!\tau$, but it may yield an arbitrary number of copies of type $\tau$ without the modality "!". Since the function classes defined following the methodology of [31, 30] are automatically closed under composition, Theorem 3.5 leads us to conjecture that polyregular functions can be characterized in a variant of the parsimonious $\lambda$-calculus.

**First-order interpretations.** As we already said, regular and polyregular functions both admit logical characterizations using Monadic Second-Order Logic [15, 8]. The basic conceit behind these definitions is that a string $w$ may be regarded as a finite model $\mathfrak{M}(w)$ over a signature containing the order relation $\leq$ on positions and predicates encoding their labeling.

The classes obtained by replacing MSO with first-order logic (FO) are to (poly)regular functions what star-free languages are to regular languages, see [11, 4]. We expect that in the same way, replacing regular functions (i.e. MSO transductions) by FO transductions in Definition 4.2 and Theorem 6.1 results in the same class in both cases, which would then be the natural FO counterpart of cfp functions. Furthermore, we believe it can be defined logically. Given a finite model $\mathfrak{U} = (U, R, \ldots)$, we write $\mathfrak{U}^k$ for the $k^{\text{th}}$ power $(U^k, R_1, \ldots, R_k, \ldots)$ where $R_i(x_1, \ldots, x_m)$ of arity $m$ is defined as $R(\pi_i(x_1), \ldots, \pi_i(x_m))$ for $1 \leq i \leq k$.

▶ **Conjecture 10.1.** *A function $f : \Gamma^* \to \Sigma^*$ is "FO comparison-free polyregular" if and only if there exists $k \in \mathbb{N}$ and a one-dimensional FO interpretation $\varphi$ such that for every $w \in \Gamma^*$ with $|w| \geq 2$, there is an isomorphism of structures $\mathfrak{M}(f(w)) \simeq \varphi\left(\mathfrak{M}(w)^k\right)$.*

On an intuitive level, this seems to capture the inability to compare the positions of two heads of comparison-free pebble transducers. However, as mentioned to us by M. Bojańczyk, the naive transposition of this conjecture to MSO fails because the direct product, generalized to Henkin structures, does not preserve *standard* second-order models.

**Integer sequences.** Recall from Remarks 8.2 and 8.4 that for unary outputs, polyregular and layered HDT0L transductions coincide, but comparison-free polyregular functions form a strictly smaller class (those results come from [12]). If we also restrict to unary inputs – in other words, if we consider sequences $\mathbb{N} \to \mathbb{N}$ – then we are fairly confident at this stage that the three classes collapse to a single one, and that this can be shown by routine methods:

▷ Claim 10.2. The classes of polyregular, comparison-free polyregular and layered HDT0L functions coincide on sequences of natural numbers.

Note that we already have a description of cfp integer sequences by specializing Theorem 9.2.

**Membership and equivalence.** We presented comparison-free polyregular functions as a strict subclass of polyregular functions. This leads to a natural *membership problem*, for which partial results were recently obtained by Douéneau-Tabot [12]:

▷ Problem 10.3. Is there an algorithm taking as input a (code for a) pebble transducer which decides whether the corresponding function $\Sigma^* \to \Gamma^*$ is comparison-free or not?

There are many similar problems of interest on the frontier between comparison-free and general polyregular functions. We hope that investigating such issues may also lead to machine/syntax-free characterizations of the containment between the two classes.

Finally, a major open problem on polyregular functions is the *equivalence problem*:

▷ Problem 10.4. Is there an algorithm taking as input two pebble transducers which decides whether they compute the same function?

Interestingly, a positive answer is known for HDT0L transductions. There is an short proof using Hilbert's basis theorem [24], which is now understood to be an example of a general approach using polynomial grammars (see e.g. [2, 5]). One could hope that a restriction to comparison-free pebble transducers also puts the equivalence problem within reach of known tools. Unfortunately, the extended polynomial grammars that would serve as the natural target for a reduction from 2-CFPT equivalence already have an undecidable zeroness problem (this was shown recently by Schmude [34]). This does *not* extend, however, to an undecidability proof for the CFPT equivalence problem, so the latter is still open.

## References

1 Rajeev Alur and Pavol Černý. Expressiveness of streaming string transducers. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, volume 8 of *LIPIcs*, pages 1–12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPIcs.FSTTCS.2010.1.

2 Michael Benedikt, Timothy Duff, Aditya Sharad, and James Worrell. Polynomial automata: Zeroness and applications. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, Reykjavik, Iceland, June 2017. IEEE. doi:10.1109/LICS.2017.8005101.

3 Jean Berstel and Christophe Reutenauer. *Noncommutative Rational Series with Applications*, volume 137 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2010.

4 Mikołaj Bojańczyk. Polyregular functions, 2018. arXiv:1810.08760.

5 Mikołaj Bojańczyk. The Hilbert method for transducer equivalence. *ACM SIGLOG News*, 6(1):5–17, 2019. doi:10.1145/3313909.3313911.

6 Mikołaj Bojańczyk, Laure Daviaud, and Shankara Narayanan Krishna. Regular and First-Order List Functions. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science - LICS '18*, pages 125–134, Oxford, United Kingdom, 2018. ACM Press. doi:10.1145/3209108.3209163.

7 Mikołaj Bojańczyk and Amina Doumane. First-order tree-to-tree functions. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany (online conference), July 8-11, 2020*, pages 252–265. ACM, 2020. doi:10.1145/3373718.3394785.

8 Mikołaj Bojańczyk, Sandra Kiefer, and Nathan Lhote. String-to-String Interpretations With Polynomial-Size Output. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 106:1–106:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.106.

9 Michaël Cadilhac, Filip Mazowiecki, Charles Paperman, Michał Pilipczuk, and Géraud Sénizergues. On polynomial recursive sequences. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 117:1–117:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.117.

10 Christian Choffrut. Sequences of words defined by two-way transducers. *Theoretical Computer Science*, 658:85–96, 2017. doi:10.1016/j.tcs.2016.05.004.

11 Luc Dartois, Ismaël Jecker, and Pierre-Alain Reynier. Aperiodic String Transducers. *International Journal of Foundations of Computer Science*, 29(05):801–824, August 2018. doi:10.1142/S0129054118420054.

12 Gaëtan Douéneau-Tabot. Pebble transducers with unary output, 2021. arXiv:2104.14019.

13 Gaëtan Douéneau-Tabot, Emmanuel Filiot, and Paul Gastin. Register Transducers Are Marble Transducers. In Javier Esparza and Daniel Kráľ, editors, *45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020)*, volume 170 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:14, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2020.29.

14 Joost Engelfriet. Two-way pebble transducers for partial functions and their composition. *Acta Informatica*, 52(7-8):559–571, 2015. doi:10.1007/s00236-015-0224-3.

15 Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic*, 2(2):216–254, April 2001. doi:10.1145/371316.371512.

16 Joost Engelfriet, Hendrik Jan Hoogeboom, and Bart Samwel. XML navigation and transformation by tree-walking automata and transducers with visible and invisible pebbles. *Theoretical Computer Science*, 850:40–97, January 2021. doi:10.1016/j.tcs.2020.10.030.

17 Joost Engelfriet and Sebastian Maneth. Two-way finite state transducers with nested pebbles. In Krzysztof Diks and Wojciech Rytter, editors, *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26-30, 2002, Proceedings*, volume 2420 of *Lecture Notes in Computer Science*, pages 234–244. Springer, 2002. doi:10.1007/3-540-45687-2_19.

18 Joost Engelfriet, Grzegorz Rozenberg, and Giora Slutzki. Tree transducers, L systems, and two-way machines. *Journal of Computer and System Sciences*, 20(2):150–202, 1980. doi:10.1016/0022-0000(80)90058-6.

19 Joost Engelfriet and Heiko Vogler. Macro tree transducers. *Journal of Computer and System Sciences*, 31(1):71–146, 1985. doi:10.1016/0022-0000(85)90066-2.

20 Julien Ferté, Nathalie Marin, and Géraud Sénizergues. Word-Mappings of Level 2. *Theory of Computing Systems*, 54(1):111–148, January 2014. doi:10.1007/s00224-013-9489-5.

21 Emmanuel Filiot and Pierre-Alain Reynier. Transducers, Logic and Algebra for Functions of Finite Words. *ACM SIGLOG News*, 3(3):4–19, August 2016. doi:10.1145/2984450.2984453.

22 Emmanuel Filiot and Pierre-Alain Reynier. Copyful streaming string transducers. *Fundamenta Informaticae*, 178(1-2):59–76, January 2021. doi:10.3233/FI-2021-1998.

23 Bruno Guillon. Input- or output-unary sweeping transducers are weaker than their 2-way counterparts. *RAIRO – Theoretical Informatics and Applications*, 50(4):275–294, 2016. doi:10.1051/ita/2016028.

24 Juha Honkala. A short solution for the HDT0L sequence equivalence problem. *Theoretical Computer Science*, 244(1-2):267–270, 2000. doi:10.1016/S0304-3975(00)00158-4.

25 Nathan Lhote. Pebble minimization of polyregular functions. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 703–712. ACM, 2020. doi:10.1145/3373718.3394804.

26 Aristid Lindenmayer. Mathematical models for cellular interactions in development II. Simple and branching filaments with two-sided inputs. *Journal of Theoretical Biology*, 18(3):300–315, March 1968. doi:10.1016/0022-5193(68)90080-5.

27 Damiano Mazza. Simple Parsimonious Types and Logarithmic Space. In *24th EACSL Annual Conference on Computer Science Logic (CSL 2015)*, pages 24–40, 2015. doi:10.4230/LIPIcs.CSL.2015.24.

28 Tova Milo, Dan Suciu, and Victor Vianu. Typechecking for XML transformers. *Journal of Computer and System Sciences*, 66(1):66–97, 2003. Journal version of a PODS 2000 paper. doi:10.1016/S0022-0000(02)00030-2.

29 Anca Muscholl and Gabriele Puppis. The Many Facets of String Transducers. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*, volume 126 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:21. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.STACS.2019.2.

30 Lê Thành Dũng Nguyễn, Camille Noûs, and Pierre Pradic. Implicit automata in typed $\lambda$-calculi II: streaming transducers vs categorical semantics, 2020. arXiv:2008.01050.

31 Lê Thành Dũng Nguyễn and Pierre Pradic. Implicit automata in typed $\lambda$-calculi I: aperiodicity in a non-commutative logic. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 135:1–135:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.135.

32 Christophe Reutenauer. Sur les séries associées à certains systèmes de Lindenmayer. *Theoretical Computer Science*, 9:363–375, 1979. doi:10.1016/0304-3975(79)90036-7.

33 Jacques Sakarovitch. *Elements of Automata Theory.* Cambridge University Press, 2009. Translated by Reuben Thomas. doi:10.1017/CBO9781139195218.

34 Janusz Schmude. On polynomial grammars extended with substitution, 2021. arXiv:2102.08705.

35 Tim Smith. A pumping lemma for two-way finite transducers. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 523–534. Springer, 2014. doi:10.1007/978-3-662-44522-8_44.

36 Géraud Sénizergues. Sequences of level 1, 2, 3, ..., $k$, ... In Volker Diekert, Mikhail V. Volkov, and Andrei Voronkov, editors, *Computer Science - Theory and Applications, Second International Symposium on Computer Science in Russia, CSR 2007, Ekaterinburg, Russia, September 3-7, 2007, Proceedings*, volume 4649 of *Lecture Notes in Computer Science*, pages 24–32. Springer, 2007. doi:10.1007/978-3-540-74510-5_6.