On Periodically Iterated Morphisms *

Jörg Endrullis Dimitri Hendriks
VU University Amsterdam
Department of Computer Science
{ j.endrullis , r.d.a.hendriks }@vu.nl

Abstract

We investigate the computational power of periodically iterated morphisms, also known as D0L systems with periodic control, PD0L systems for short. These systems give rise to a class of one-sided infinite sequences, called PD0L words.

We construct a PD0L word with exponential subword complexity, thereby answering a question raised by Lepistö [22] on the existence of such words. We solve another open problem concerning the decidability of the first-order theories of PD0L words [23]; we show it is already undecidable whether a certain letter occurs in a PD0L word. This stands in sharp contrast to the situation for D0L words (purely morphic words), which are known to have at most quadratic subword complexity, and for which the monadic theory is decidable.

The main result of our paper, leading to these answers, is that every computable word $\boldsymbol{w} \in \Sigma^\omega$ can be embedded in a PD0L word $\boldsymbol{u} \in \Gamma^\omega$ with $\Gamma \supset \Sigma$ in the following two ways: (i) such that every finite prefix of $\boldsymbol{w}$ is a subword of $\boldsymbol{u}$, and (ii) such that $\boldsymbol{w}$ is obtained from $\boldsymbol{u}$ by erasing all letters from $\Gamma \setminus \Sigma$. The PD0L system generating such a word $\boldsymbol{u}$ is constructed by encoding a Fractran program that computes the word $\boldsymbol{w}$; Fractran is a programming language as powerful as Turing Machines.

As a consequence of (ii), if we allow the application of finite state transducers to PD0L words, we obtain the set of all computable words. Thus the set of PD0L words is not closed under finite state transduction, whereas the set of D0L words is. It moreover follows that equality of PD0L words (given by their PD0L system) is undecidable. Finally, we show that if erasing morphisms are admitted, then the question of productivity becomes undecidable, that is, the question whether a given PD0L system defines an infinite word.

1. Introduction

Morphisms for transforming and generating infinite words provide a fundamental tool for formal languages, and have been studied extensively; we refer to [2] and the bibliography therein.

In this paper we investigate the class of infinite words generated by periodically alternating morphisms [6, 10, 11, 22]. Instead of repeatedly applying a single morphism, one alternates several morphisms from a given (finite) set in a periodic fashion. Let us look at an example right away, and consider the most famous word generated by such a procedure, namely the Kolakoski word [21]

$$K = 1\,22\,11\,2\,1\,22\,1\,22\,11\,2\,11\,22\,1\,2\,11\,2\,1\,22\,11\,2\,\cdots$$

which is defined such that $K(0) = 1$ and $K(n)$ equals the length of the n-th run of K; here by a 'run' we mean a maximal subsequence of consecutive identical symbols. The Kolakoski word can be generated by alternating two morphisms on the starting word 12, h_0 for the even positions and h_1 for the odd positions, defined as follows:

$$h_0 : \begin{array}{l} 1 \to 1 \\ 2 \to 11 \end{array} \qquad h_1 : \begin{array}{l} 1 \to 2 \\ 2 \to 22 \end{array}$$

The first few iterations then are

$$\begin{aligned} 12 \\ h_0(1)\,h_1(2) &= 12\mathbf{2} \\ h_0(1)\,h_1(2)\,h_0(2) &= 122\mathbf{11} \\ h_0(1)\,h_1(2)\,h_0(2)\,h_1(1)\,h_0(1) &= 12211\mathbf{21} \end{aligned}$$

It is known that the Kolakoski word is not purely morphic [11], i.e, cannot be generated by iterating a single morphism. However it is an open problem whether it is a morphic word, i.e., the image of a purely morphic word under a coding (= letter-to-letter morphism). We shall use the 'D0L' terminology: D0L for purely morphic, CD0L for morphic, and PD0L for words generated by periodically alternating morphisms, like the Kolakoski word above.

A natural characteristic of sequences is their subword complexity [1, 2, 18]. The subword complexity of a sequence $\boldsymbol{u}$ is a function $\mathbb{N} \to \mathbb{N}$ mapping n to the number of n-length words that occur in $\boldsymbol{u}$. It is well-known that morphic words have at most quadratic subword complexity [12]. Lepistö [22] proves that for all $r \in \mathbb{R}$ there is a PD0L word whose subword complexity is in $\Omega(n^r)$; hence there are PD0L words that are not CD0L. It remained an open problem whether PD0L words can exhibit exponential subword complexity. This intriguing question formed the initial motivation for our investigations. We actually establish a stronger result from which the existence of such words can be derived, as we will describe next.

The main results of our paper can be stated as follows: *For every computable word $\boldsymbol{w} \in \Sigma^\omega$ there exists a PD0L word $\boldsymbol{u}$ such that*

I. all prefixes of $\boldsymbol{w}$ occur in $\boldsymbol{u}$ as subwords between special marker symbols (Theorem 5.9),

II. $\boldsymbol{w}$ is the subsequence of $\boldsymbol{u}$ obtained from selecting all letters from Σ (Theorem 5.12).

The construction of the PD0L systems generating such words $\boldsymbol{u}$ makes use of Fractran [7, 8], a Turing complete programming language invented by Conway, in the following way. First, in Section 3, we show how to employ Fractran to generate any com-

* This research has been funded by the Netherlands Organization for Scientific Research (NWO) under grant numbers 639.021.020 and 612.000.934.

putable infinite word. Then we encode Fractran programs as PD0L systems, and prove that the PD0L system correctly simulates the Fractran program and records its output, see Sections 4 and 5.

Consequences of I and II are as follows:

(1) There exist PD0L words with exponential subword complexity (Theorem 5.14).

(2) It is undecidable to determine, given a PD0L system $\mathcal{H}$ and a letter b, whether the letter b occurs (infinitely often) in the word generated by $\mathcal{H}$ (Theorem 5.15).

(3) The first-order theory of PD0L words is undecidable (Corollary 5.16).

(4) Equality of PD0L words is undecidable (Corollary 5.17).

(5) The set of PD0L words is not closed under finite state transductions (Corollary 5.13).

All the above results concern PD0L systems whose morphisms are non-erasing. But we also study erasing PD0L systems, and find

(6) It is undecidable to determine, on the input of an erasing PD0L system, whether it generates an infinite word (Theorem 4.6).

The outline of the paper is as follows. In Sections 2 and 3 we introduce the *dramatis personae* of our story: PD0L systems and Fractran programs. We explain the workings of the Fractran algorithm, and how to program in this language.

Then, as a steppingstone to our main result, we start with a proof of (6) in Section 4. This proof illustrates our key construction: encoding Fractran programs as PD0L systems. We then modify and extend this encoding in Section 5 to prove Theorems 5.9 and 5.12: PD0L words can embed every computable word, in the sense of I and II above. We give a detailed example of the translation, and prove (1)–(5) listed above.

PD0L systems resulting from encoding Fractran programs can be quite large. For example, the system obtained from a simple binary counter (computing an infinite word with exponential subword complexity) consists of

$$536393214598471230$$

morphisms. Due to space limitations we present a direct solution in Section 6, namely a PD0L system with 16 morphisms simulating such a counter.

2. D0L Systems with Periodic Control

We use standard terminology and notations, see, e.g., [2]. Let Σ be a finite alphabet. We denote by Σ^* the set of all finite words over Σ, by ε the empty word, and by $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ the set of finite non-empty words.

The set of infinite words over Σ is $\Sigma^\omega = \{\boldsymbol{x} \mid \boldsymbol{x} : \mathbb{N} \to \Sigma\}$. On the set of all words $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ we define the metric d for all $\boldsymbol{u}, \boldsymbol{v} \in \Sigma^\infty$ by $d(\boldsymbol{u}, \boldsymbol{v}) = 2^{-n}$, where n is the length of the longest common prefix of $\boldsymbol{u}$ and $\boldsymbol{v}$.

We let $\Sigma_p = \{0, \ldots, p-1\}$. We write $|x|$ for the length of $x \in \Sigma^\infty$, with $|x| = \infty$ if x is infinite. We call a word $v \in \Sigma^*$ a *factor* of $x \in \Sigma^\infty$ if $x = uvy$ for some $u \in \Sigma^*$ and $y \in \Sigma^\infty$, and say that v occurs at position $|u|$. For words $u, v \in \Sigma^*$, we write $u \prec v$ if u is a strict prefix of v, i.e., if $v = uu'$ for some $u' \in \Sigma^+$, and use $\preceq$ for its reflexive closure.

A *morphism* is a map $h : \Sigma^* \to \Gamma^*$ such that $h(uv) = h(u)h(v)$ for all $u, v \in \Sigma^*$, and can thus be defined by giving its values on the symbols of Σ. A morphism h is called *erasing* if $h(a) = \varepsilon$ for some $a \in \Sigma$, and *k-uniform*, with $k \in \mathbb{N}$, if $|h(a)| = k$ for all $a \in \Sigma$; h is a *coding* if it is 1-uniform.

Infinite sequences generated by periodically alternating morphisms, also called 'D0L words with periodic control' or just 'PD0L words' for short, were introduced in [10]. These form a generalization of D0L words, also known as *purely morphic* words, which are obtained by iterating a single morphism.

Definition 2.1. Let $H = \langle h_0, \ldots, h_{p-1} \rangle$ be a tuple of morphisms $h_i : \Sigma^* \to \Sigma^*$. We define the map $H : \Sigma^* \to \Sigma^*$ as follows:

$$H(a_0 a_1 \cdots a_n) = u_0 u_1 \cdots u_n$$
$$\text{where } u_i = h_k(a_i), \text{ with } k \equiv i \pmod p \text{ and } k \in \Sigma_p.$$

If $s \in \Sigma^*$ is such that $s \preceq H(s)$, then the triple $\mathcal{H} = \langle \Sigma, H, s \rangle$ is called a *PD0L system*. Then in the metric space $\langle \Sigma^\infty, d \rangle$ the limit

$$H^\omega(s) = \lim_{i \to \infty} H^i(s)$$

exists, and we call $H^\omega(s)$ the *PD0L word* generated by $\mathcal{H}$. We say that $\mathcal{H}$ is *productive* if $H^\omega(s)$ is infinite, and $\mathcal{H}$ is *erasing* if some of its morphisms h_i are erasing.

If x is a PD0L word generated by p morphisms, and $x = uvy$ for some $u, v \in \Sigma^*$ and $y \in \Sigma^\infty$, we say that the factor v of x occurs at *morphism index* i when $i \in \Sigma_p$ and $i \equiv |u| \pmod p$.

D0L words are generated by D0L systems $\langle \Sigma, h, s \rangle$, i.e., PD0L systems $\langle \Sigma, \langle h \rangle, s \rangle$ consisting of one single morphism h. Following [6], we call the image of a D0L word under a coding (a letter-to-letter morphism), a *CD0L* word, better known as morphic words.

In the literature, one typically requires the morphisms h_i to be non-erasing to ensure that the limit is infinite. We have taken a more general definition of PD0L-words, since also erasing morphisms may yield an infinite word in the limit. See Remark 2.3 below.

In the sequel it will be helpful to have a recursive definition of the map H.

Lemma 2.2. *Let $H = \langle h_0, h_1, \ldots, h_{p-1} \rangle$ be a tuple of morphisms. For $i \in \Sigma_p$ define $H_i = \langle h_i, \ldots, h_{p-1}, h_0, \ldots, h_{i-1} \rangle$ and the corresponding map $H_i : \Sigma^* \to \Sigma^*$ by*

$$H_i(\varepsilon) = \varepsilon$$
$$H_i(au) = h_i(a) H_{i+1}(u) \qquad (a \in \Sigma,\ u \in \Sigma^*)$$

where addition in the subscript of H is taken modulo p.

Then $H_0 = H$ with H the map defined in Definition 2.1, and $H_i(uv) = H_i(u) H_{i+|u|}(v)$ for all $u, v \in \Sigma^*$ and $i \in \Sigma_p$. □

Using this notation we now formulate the PD0L analogue of the usual condition for productivity of D0L systems. In Section 4 we show that productivity of PD0L systems in general is undecidable. Productivity has been studied in the wider perspective of term rewriting systems in [13, 15, 16, 25].

Remark 2.3. Let $\langle \Sigma, h, s \rangle$ be a D0L system. We say that h is *prolongable* on s if $h(s) = sx$ for some $x \in \Sigma^*$ and $h^i(x) \neq \varepsilon$ for all $i \geq 0$. Then $h^i(s) \prec h^{i+1}(s)$ for all $i \geq 0$, and hence the limit $h^\omega(s) = s\, x\, h(x)\, h^2(x) \cdots$ is infinite. The generalization of this condition to PD0L systems $\mathcal{H} = \langle \Sigma, H, v_0 \rangle$ is: (*) $H(v_0) = v_0 v_1$ for some $v_1 \in \Sigma^*$ such that $v_n \neq \varepsilon$ for all $n \in \mathbb{N}$, where $v_n \in \Sigma^*$ and $z_n \in \Sigma_p$ are defined by $z_0 = 0$ and

$$v_n = H_{z_{n-1}}(v_{n-1}) \qquad (n \geq 2)$$
$$z_n \equiv z_{n-1} + |v_{n-1}| \pmod p \qquad (n \geq 1)$$

Then $H^n(v_0) = H^{n-1}(v_0) v_n$ for all $n \geq 1$, and so (*) forms a necessary and sufficient condition for productivity of $\mathcal{H}$, that is, for the limit $H^\omega(v_0) = v_0 v_1 v_2 \cdots$ to be infinite.

Definition 2.4. The *subword complexity* of an infinite word $\boldsymbol{x} \in \Sigma^\omega$ is the function $p_{\boldsymbol{x}} : \mathbb{N} \to \mathbb{N}$ such that $p_x(n)$ is the number of factors (subwords) of $\boldsymbol{x}$ of length n.

Proposition 2.5 ([12])**.** *The subword complexity of D0L words, and hence of CD0L words, is at most quadratic.*

We first consider an example of an erasing PD0L system.

Example 2.6. Let $\mathcal{H} = \langle \Sigma_3, H, 0\rangle$ with $H = \langle h_0, h_1, h_2\rangle$ defined for all $b \in \Sigma_3$ as follows, where addition runs modulo 3:

$$h_0(b) = b(b+1)(b+2) \qquad h_1(b) = \varepsilon \qquad h_2(b) = b+2$$

Then $\mathcal{H}$ is productive (by Proposition 2.8) and generates the word

$$H^\omega(0) = 012112010122120112021201020100120201001210\cdots$$

Definition 2.7. Let $\mathcal{H} = \langle \Sigma, \langle h_0, \ldots, h_{p-1}\rangle, s\rangle$ be a PD0L system. We say $\mathcal{H}$ is *locally uniform* if every morphism h_i is uniform, i.e, if for all $i \in \Sigma_p$ there is $k_i \in \mathbb{N}$ such that $k_i = |h_i(b)|$ for all $b \in \Sigma$. We say $\mathcal{H}$ is *(globally) uniform* if, for some $k \in \mathbb{N}$, each h_i is k-uniform ($i \in \Sigma_p$).

Obviously, a globally k-uniform PD0L system is productive if and only if $k \geq 2$. For locally uniform systems the condition is formulated as follows, and is easy to check.

Proposition 2.8. *Let $\mathcal{H} = \langle \Sigma, \langle h_0, \ldots, h_{p-1}\rangle, w\rangle$ be a locally uniform PD0L system, where h_i is k_i-uniform. Let $s(n)$ be defined by $s(0) = 0$ and $s(n+1) = s(n) + k_i$ with $i \equiv n \pmod p$. Then $\mathcal{H}$ is productive if and only if $s(n) > n$ for all $n \geq |w|$.*

Proof. The word $H^\omega(w)$ can be defined as the limit of the sequence $w_{|w|}, w_{|w|+1}, w_{|w|+2}, \ldots$ of finite words defined for $n \geq |w|$ by

$$w_{|w|} = H(w)$$

$$w_{n+1} = \begin{cases} w_n & \text{if } n \geq |w_n| \\ w_n\, h_i(w_n(n)) & \text{if } n < |w_n| \text{ and } n \equiv i \pmod p \end{cases}$$

We have $|w_{|w|}| = s(|w|)$ and by induction we get $|w_n| = s(n)$ for every $n \geq |w|$. The limit $\lim_{n\to\infty} w_n$ is infinite if and only if we never get to the clause $n \geq |w_n|$, which holds in turn if and only if $s(n) > n$ for all $n \geq |w|$. □

Example 2.9. Let $\Sigma = \{\mathsf{L}, \mathsf{O}, \mathsf{P}\}$, and define $\rho : \Sigma \to \Sigma$ by $\rho(\mathsf{L}) = \mathsf{O}$, $\rho(\mathsf{O}) = \mathsf{P}$, and $\rho(\mathsf{P}) = \mathsf{L}$. Let $H = \langle h_0, h_1\rangle$ with h_0, h_1 morphisms for all $a \in \Sigma$ defined by

$$h_0(a) = a\rho(a)\rho^2(a) \qquad h_1(a) = \rho^2(a)\rho(a)a$$

Then the PD0L system $\langle \Sigma, H, \mathsf{L}\rangle$ generates the infinite word

$$H^\omega(\mathsf{L}) = \mathsf{LOPLPOPLOPOLPLOLPOPLOPOLOPLOLPOPL}\cdots$$

This is the square-free Arshon word [3] (of rank 3), which Berstel proved to be an example of a CD0L word that is not a D0L word [4]; see Séébold [24] for a generalization. That $H^\omega(\mathsf{L})$ can indeed be defined as a CD0L word follows from Proposition 2.10.

It is not hard to see that, when a word $\boldsymbol{u}$ is generated by a (globally) k-uniform PD0L system, it is k-automatic [2], i.e., $\boldsymbol{u}$ is the image of a coding of the fixed point of a k-uniform morphism.

Proposition 2.10. *Let $k \geq 2$, and $\mathcal{H} = \langle \Sigma, H, s\rangle$ a k-uniform PD0L system. Then $H^\omega(s)$ is k-automatic.*

Proof. Let $H = \langle h_0, \ldots, h_{p-1}\rangle$, where every h_i is k-uniform. We define the (k-uniform) morphism $g : \Sigma_p \times \Sigma \to \Sigma_p \times \Sigma$ by

$$g(\langle i, a\rangle) = \langle ki, b_0\rangle\langle ki+1, b_1\rangle \cdots \langle ki+k-1, b_{k-1}\rangle$$

where addition in the first entries runs modulo p, and for $j \in \Sigma_k$, $b_j \in \Sigma$ is such that $h_i(a) = b_0 b_1 \cdots b_{k-1}$. Let $s = s_0 s_1 \cdots s_{q-1}$, $t = \langle 0, s_0\rangle\langle 1, s_1\rangle \cdots \langle q-1, s_{q-1}\rangle$, and $\boldsymbol{u} = H^\omega(s)$. Then

$$g^n(t) = \langle 0, \boldsymbol{u}(0)\rangle\langle 1, \boldsymbol{u}(1)\rangle \cdots \langle qk^n - 1, \boldsymbol{u}(qk^n - 1)\rangle$$

follows by induction on n. Hence $\tau(g^\omega(t)) = \boldsymbol{u}$ with τ the coding defined by $\tau(\langle i, a\rangle) = a$. □

One might wonder whether also locally uniform, productive PD0L systems always generate morphic words. Examples 2.11 and 2.12 show that this is not the case.

Example 2.11 ([22])**.** Define the word $F_p \in \{0,1\}^\omega$ for every $p \geq 2$ by $F_p = H^\omega(0)$ where $\langle \{0,1\}, H, 0\rangle$ with $H = \langle h_0, \ldots, h_{p-1}\rangle$ is a PD0L system, and h_i are morphisms defined by

$$h_0 : \begin{cases} 0 \to 01 \\ 1 \to 00 \end{cases} \qquad h_i : \begin{cases} 0 \to 1 \\ 1 \to 0 \end{cases} \quad \text{for } i \in \{1, \ldots, p-1\}$$

For example, the word F_3 starts like this:

$$010100110001011001000110011100010100001101010011\cdots$$

Lepistö [22] proves that F_p has more than quadratic subword complexity, for every $p \geq 2$. Hence, with Proposition 2.5, these PD0L words F_p cannot be CD0L words. We note that, conversely, the existence of CD0L words that are not PD0L words was shown in [10].

Example 2.12 ([6])**.** A Toeplitz word [20] over an alphabet Σ is generated by a seed word $u \in \Sigma(\Sigma \cup \{?\})^*$ with $? \notin \Sigma$, as follows. Start with the periodic u^ω and then replace its subsequence of ?'s by the sequence itself. For example $u = 12???$ generates the infinite word $T(u) = 121211221112221\cdots$. Cassaigne and Karhumäki [6] show that all Toeplitz words are PD0L words; e.g., $T(u) = H^\omega(1)$ where $H = \langle h_0, h_1, h_2\rangle$ and $h_0(a) = 12a$ and $h_1(a) = h_2(a) = a$ for all $a \in \{1, 2\}$. Moreover, from [6, Theorem 5] it follows that $p_{T(u)}(n) \in \Theta(n^r)$ with $r = \frac{\log 5}{\log 5 - \log 3} \simeq 3.15066$, thus forming an alternative proof of what was established in [22]: there are PD0L words that are not CD0L.

3. Fractran for Computing Streams

Fractran [7, 8] is a universal programming language invented by John Horton Conway. The simplicity of its execution algorithm, based on the unique prime factorization of integers, makes Fractran ideal for coding it into other formalisms.

A Fractran program F is a finite list of fractions

$$F = \frac{n_1}{d_1}, \ldots, \frac{n_k}{d_k} \tag{1}$$

with n_i, d_i positive integers. Let $f_i = \frac{n_i}{d_i}$. The action of F on an input integer $N \geq 1$ is to multiply N by the first 'applicable' fraction f_i, that is, the fraction f_i with i the least index such that the product $N' = N \cdot f_i$ is an integer again, and then to continue with N'. The program halts if there is no applicable fraction for the current integer N.

For example, consider the program

$$F = \frac{5}{2\cdot 3}, \frac{1}{2}, \frac{1}{3}$$

and the run of F on input $N = 2^3 3^5$:

$$2^3 3^5 \to 2^2 3^4 5^1 \to 2^1 3^3 5^2 \to 2^0 3^2 5^3 \to 2^0 3^1 5^3 \to 2^0 3^0 5^3\,.$$

Note that each multiplication by $\frac{5}{6}$ decrements the exponents of 2 and 3 while incrementing the exponent of 5. Once $\frac{5}{6}$ is no longer applicable, i.e., when one of the exponents of 2 and 3 in the prime factorization of the current integer N equals 0, the other is set to 0 as well. Hence, executing F on $N = 2^a\, 3^b$ halts after $\max(a, b)$ steps with $5^{\min(a,b)}$.

Thus the prime numbers that occur as factors in the numerators and denominators of a Fractran program can be regarded as registers, and if the current working integer is $N = 2^a\, 3^b\, 5^c\, \ldots$ we can say that register 2 holds a, register 3 holds b, and so on.

The real power of Fractran, however, comes from the use of prime exponents as *states*. To explain this, we temporarily let pro-

grams consist of multiple lines of the form

$$\alpha : \frac{n_1}{d_1} \to \alpha_1,\ \frac{n_2}{d_2} \to \alpha_2,\ \ldots,\ \frac{n_m}{d_m} \to \alpha_m \tag{2}$$

forming the instructions for the program in state α: multiply N with the first applicable fraction $\frac{n_i}{d_i}$ and proceed in state α_i, or terminate if no fraction is applicable. We call the states $\alpha_1, \ldots, \alpha_m$ in (2) the *successors* of α, and we say a state is *looping* if it is its own successor.

For example, the program P_{add} given by the lines

$$\alpha : \frac{2 \cdot 5}{3} \to \alpha,\ \frac{1}{1} \to \beta \qquad \text{and} \qquad \beta : \frac{3}{5} \to \beta$$

realizes addition; running P_{add} in state α on $N = 2^a 3^b$ terminates in state β with $2^{a+b}3^b$.

A program with n lines is called a *Fractran-n program*. A flat list of fractions $f_1, \ldots, f_k$ now is a shorthand for the Fractran-1 program $\alpha : f_1 \to \alpha, f_2 \to \alpha, \ldots, f_k \to \alpha$. Conway [8] explains how every Fractran-n program ($n \geq 2$) can be compiled into a Fractran-1 program, using the following steps:

(i) For every looping state α, introduce a 'mirror' state $\mathfrak{o}$, substitute $\mathfrak{o}$ for all occurrences of α in the right-hand sides of its program line, and add the line

$$\mathfrak{o} : \frac{1}{1} \to \alpha$$

(ii) Replace state identifiers α by 'fresh' prime numbers.

(iii) For every line of the form (2) append the following fractions:

$$\frac{n_1 \cdot \alpha_1}{d_1 \cdot \alpha},\ \frac{n_2 \cdot \alpha_2}{d_2 \cdot \alpha},\ \ldots,\ \frac{n_k \cdot \alpha_m}{d_m \cdot \alpha}$$

(preserving the order) to the list of fractions constructed so far.

Let us illustrate these steps on the adder P_{add} given above. Step (i) of splitting loops, results in

$$\alpha : \frac{2 \cdot 5}{3} \to \mathfrak{o},\ \frac{1}{1} \to \beta \qquad \beta : \frac{3}{5} \to \mathfrak{b}$$
$$\mathfrak{o} : \frac{1}{1} \to \alpha \qquad \mathfrak{b} : \frac{1}{1} \to \beta\,.$$

In step (ii), we introduce 'fresh' primes to serve as state indicators, e.g., $\langle \alpha, \mathfrak{o}, \beta, \mathfrak{b} \rangle = \langle 7, 11, 13, 17 \rangle$. Finally, step (iii), we replace lines by fractions, to obtain the Fractran-1 program

$$F_{\mathsf{add}} = \frac{2 \cdot 5 \cdot \mathfrak{o}}{3 \cdot \alpha},\ \frac{\alpha}{\mathfrak{o}},\ \frac{\beta}{\alpha},\ \frac{3 \cdot \mathfrak{b}}{5 \cdot \beta},\ \frac{\beta}{\mathfrak{b}}\,.$$

Then indeed the run of F_{add} on $2^a 3^b \alpha$ ends in $2^{a+b}3^b\beta$.

For 'sensible' programs any state indicator has value 0 ('off') or 1 ('on'), and the program is always in exactly one state at a time. Hence, if a program F uses primes $r_1, \ldots, r_p$ for storage, and primes $\alpha_1, \ldots, \alpha_q$ for control, at any instant the entire *configuration* of F (= register contents + state) is uniquely represented by the current working integer N

$$N = r_1^{e_1}\, r_2^{e_2} \cdots r_p^{e_p}\, \alpha_j$$

for some integers $e_1, \ldots, e_p \geq 0$ and $1 \leq j \leq q$.

The reason to employ two state indicators α and $\mathfrak{o}$ to break self-loops in step (i), is that each state indicator is consumed whenever it is tested, and so we need a secondary indicator $\mathfrak{o}$ to say "continue in the current state". This secondary indicator $\mathfrak{o}$ is swapped back to the primary indicator α in the next instruction, and the loop continues.

We now introduce some further notation. For partial functions $g : A \rightharpoonup B$ we write $g(x)\!\downarrow$ to indicate that g is defined on $x \in A$, and $g(x)\!\uparrow$ otherwise.

Definition 3.1. Let $F = f_1, \ldots, f_k$ be a Fractran program with $f_i = \frac{n_i}{d_i} \in \mathbb{Q}_{>0}$. We define the partial function $\psi_F : \mathbb{N} \rightharpoonup \mathbb{N}$ which, given an integer $N \geq 1$, selects the index of the first fraction applicable to N, and is undefined if no such fraction exists, i.e.,

$$\psi_F(N) = \min\ \{i \mid 1 \leq i \leq k,\ N \cdot f_i \in \mathbb{N}\}\ ,$$

where we stipulate $(\min \varnothing)\!\uparrow$. We write $\psi(n)$ for short when F is clear from the context.

We overload notation and use $F : \mathbb{N} \rightharpoonup \mathbb{N}$ to denote the *one-step computation* of the program F, defined for all $N \geq 1$ by

$$F(N) = N \cdot f_{\psi(N)}$$

where it is to be understood that $F(N)\!\uparrow$ whenever $\psi(N)\!\uparrow$. The *run of F on N* is the finite or infinite sequence $N, F(N), F^2(N), \ldots$. We say that F *halts* or *terminates* on N if the run of F on N is finite.

The halting problem for Fractran programs is undecidable.

Proposition 3.2 ([14, Theorem 2.2])**.** *The uniform halting problem for Fractran programs, that is, deciding whether a program halts for every starting integer $N \geq 0$, is Π_2^0-complete.*

Proposition 3.3 ([19, Theorem 68])**.** *The input-2 halting problem for Fractran programs, that is, deciding whether a program halts for the starting integer $N = 2$, is Σ_1^0-complete.*

Remark 3.4. In some sense it does not matter which prime numbers are used in a Fractran program. Let us make this precise. Let p be a prime number, and n a positive integer. Then let $v_p(n)$ denote the *p-adic valuation of n* i.e., $v_p(n) = a$ with $a \in \mathbb{N}$ maximal such that p^a divides n. For $\vec{p} = \langle p_1, p_2, \ldots, p_t \rangle$ we write $v_{\vec{p}}(n)$ to denote $\langle v_{p_1}(n), v_{p_2}(n), \ldots, v_{p_t}(n) \rangle$. Let F be a Fractran program with t distinct primes $\vec{p} = p_1, p_2, \ldots, p_t$, let $\vec{q} = q_1, q_2, \ldots, q_t$ be any vector of t distinct primes, and let G be the program obtained from F by uniformly substituting the q_i's for the p_i's. Then clearly, for all integers $M, N \geq 0$ such that $v_{\vec{p}}(M) = v_{\vec{q}}(N)$, we have $v_{\vec{p}}(F^i(M)) = v_{\vec{q}}(G^i(N))$ for all $i \geq 0$.

We employ Fractran programs to define finite or infinite words over the alphabet $\{0, 1\}$ by giving the primes 3 and 5 a special meaning, namely for indicating output 0 and 1, respectively. The construction easily generalizes to arbitrary finite alphabets.

Definition 3.5. Let F be a Fractran program. The finite or infinite *word W_F computed by F* is $W_F = W(2)$ where $W(N) = \varepsilon$ if the sequence $F(N), F^2(N), \ldots$ does not contain values divisible by 3 or 5, (note that this includes $W(N) = \varepsilon$ if $F(N)\!\uparrow$), and otherwise

$$W(N) = \begin{cases} 0\ W(F(N)) & \text{if } 3 \mid F(N), \\ 1\ W(F(N)) & \text{if } 5 \mid F(N) \text{ and } 3 \nmid F(N), \\ W(F(N)) & \text{otherwise.} \end{cases}$$

So the word W_F is infinite if and only if F does not terminate on input 2 and the run of F on N contains infinitely many numbers that are divisible by 3 or 5. The infinite word can be read off from the infinite run by dropping all entries neither divisible by 3 nor 5, and then mapping the remaining entries to 0 or 1, if they are divisible by 3 or 5 (and not 3), respectively.

Example 3.6. The Fractran program $\frac{3}{2}, \frac{5}{3}, \frac{3}{5}$ gives rise to the computation $3, 5, 3, 5, 3, 5, \ldots$, and hence computes the infinite word $010101\ldots$ of alternating bits.

Proposition 3.7. *Every (finite or infinite) computable, binary word can be computed by a Fractran program.*

Proof. In [14] it is shown that Fractran programs can simulate any Turing machine computation. By Remark 3.4 we may assume that this translation does not employ the primes $\{2, 3, 5\}$. Then a

straightforward adaptation of the proof in [14] yields the claim: we multiply the fractions corresponding to the Turing machine generating an output 0 or 1 by the primes 3 or 5, respectively, and make sure the thus introduced factor 3 or 5 is removed in the next step by putting fractions $\frac{1}{3}$ and $\frac{1}{5}$ in front of the program. □

We define a Fractran-n program and compile it to a Fractran-1 program F_{BIN} which computes an infinite word that has every finite binary word as one of its factors. For this we use the bijective 'z-representation' defined as follows.

Definition 3.8. Let $\Sigma = \{0, 1\}$. For all $n \in \mathbb{N}$ and $w \in \Sigma^*$, we define $(n)_{\mathsf{z}} \in \Sigma^*$ and $[w]_{\mathsf{z}} \in \mathbb{N}$ by

$$\begin{aligned} (0)_{\mathsf{z}} &= \varepsilon & [\varepsilon]_{\mathsf{z}} &= 0 \\ (2n+1)_{\mathsf{z}} &= 0(n)_{\mathsf{z}} & [0w]_{\mathsf{z}} &= 2[w]_{\mathsf{z}} + 1 \\ (2n+2)_{\mathsf{z}} &= 1(n)_{\mathsf{z}} & [1w]_{\mathsf{z}} &= 2[w]_{\mathsf{z}} + 2 \end{aligned}$$

and we let BIN denote the infinite word

$$\mathrm{BIN} = (0)_{\mathsf{z}}(1)_{\mathsf{z}}(2)_{\mathsf{z}} \cdots = 0\,1\,00\,10\,01\,11\,000\,100\,010 \cdots$$

We will now define a Fractran program that computes BIN; it will be the compilation of the following Fractran-7 program:

$$\begin{aligned} \alpha_1 &: \frac{r_2}{r_3} \to \alpha_5\,,\ \frac{r_1}{1} \to \alpha_2 & \beta_0 &: \frac{1}{1} \to \alpha_1 \\ \alpha_2 &: \frac{r_2 r_3}{r_1} \to \alpha_2\,,\ \frac{1}{1} \to \alpha_3 & \beta_1 &: \frac{1}{1} \to \alpha_1 \\ \alpha_3 &: \frac{r_1}{r_3} \to \alpha_3\,,\ \frac{1}{1} \to \alpha_4 \\ \alpha_4 &: \frac{r_3}{r_2^2} \to \alpha_4\,,\ \frac{1}{r_2} \to \beta_0\,,\ \frac{1}{r_3} \to \beta_1 \\ \alpha_5 &: \frac{r_2}{r_3} \to \alpha_5\,,\ \frac{1}{1} \to \alpha_4 \end{aligned}$$

We first explain its workings, and then compile it into a Fractran-1 program. Let e_1, e_2, e_3 be the register contents of the current integer N, i.e., such that $N = r_1^{e_1} r_2^{e_2} r_3^{e_3}$. In the run (= sequence of states) of the above program starting in α_1 with $e_1 = e_2 = e_3 = 0$, the subsequence of 'output' states β_0 and β_1 corresponds to the infinite word BIN. The idea is that r_1 holds the current value n for producing the factor $(n)_{\mathsf{z}}$ of BIN. State α_1 with $e_3 = 0$ increments e_1, and the program proceeds in state α_2. States α_2 and α_3 copy e_1 to e_2 and we continue in state α_4. State α_4 subtracts 2 from e_2 while incrementing e_3 as long as possible (corresponding to division of r_2 by 2 and storing the quotient in e_3), and then goes to output state β_0 if the remainder $e_2 \neq 0$, and to output state β_1 after decrementing e_3, otherwise (corresponding to the definition of $(\cdot)_{\mathsf{z}}$ above). After any of the two output states, the program returns to state α_1. State α_1 with a non-zero quotient r_3 copies e_3 to e_2 using state α_5, and then continues with state α_4.

We compile the above program into a flat list of fractions using the steps (i)–(iii) given above. For the looping states α_2, α_3, α_4, and α_5, we introduce mirror states $\mathfrak{o}_2$, $\mathfrak{o}_3$, $\mathfrak{o}_4$, and $\mathfrak{o}_5$. Second, we assign the following prime numbers to the identifiers:

α_1	α_2	$\mathfrak{o}_2$	α_3	$\mathfrak{o}_3$	α_4	$\mathfrak{o}_4$	α_5	$\mathfrak{o}_5$	β_0	β_1	r_1	r_2	r_3
2	7	11	13	17	19	23	29	31	3	5	37	41	43

Finally, with (iii), we obtain the following Fractran-1 program:

$$F_{\mathrm{BIN}} = \frac{r_2\alpha_5}{r_3\alpha_1}, \frac{r_1\alpha_2}{\alpha_1}, \frac{r_2 r_3 \mathfrak{o}_2}{r_1\alpha_2}, \frac{\alpha_3}{\alpha_2}, \frac{\alpha_2}{\mathfrak{o}_2}, \frac{r_1\mathfrak{o}_3}{r_3\alpha_3}, \frac{\alpha_4}{\alpha_3}, \frac{\alpha_3}{\mathfrak{o}_3}, \frac{r_3\mathfrak{o}_4}{r_2^2\alpha_4}, \frac{\beta_0}{r_2\alpha_4}, \frac{\beta_1}{r_3\alpha_4}, \frac{\alpha_4}{\mathfrak{o}_4}, \frac{r_2\mathfrak{o}_5}{r_3\alpha_5}, \frac{\alpha_4}{\alpha_5}, \frac{\alpha_5}{\mathfrak{o}_5}, \frac{\alpha_1}{\beta_0}, \frac{\alpha_1}{\beta_1}$$

which is run on input $N = \alpha_1 = 2$ to force the program to start in the initial state. We note that the huge number mentioned at the end of the introduction is the least common denominator of F_{BIN}.

Proposition 3.9. *The word computed by F_{BIN} is* BIN. □

4. Productivity for Erasing PD0L Systems

We show that the problem of deciding productivity of erasing PD0L systems is undecidable. The idea is to encode a given Fractran program F as a PD0L system $\mathcal{H}_F = \langle \Sigma, H, s \rangle$ such that $H^\omega(s)$ is infinite if and only if F does not terminate on input 2.

We consider Fractran programs of the form $\frac{n_1}{d}, \ldots, \frac{n_k}{d}$; every program can be brought into this form by taking d the least common denominator of the fractions.

Definition 4.1. Let $F = \frac{n_1}{d}, \ldots, \frac{n_k}{d}$ be a Fractran program. We define the PD0L system $\mathcal{H}_F = \langle \Gamma, H, \mathsf{s} \rangle$ where

$$\Gamma = \{\mathsf{s}, \textvisiblespace, \mathsf{a}, \mathsf{A}, \mathsf{b}, \mathsf{B}\}$$

and $H = \langle h_0, \ldots, h_{d-1} \rangle$ consisting of morphisms $h_i : \Gamma^* \to \Gamma^*$ defined for all $i \in \Sigma_d$ as follows:

$$h_i(\mathsf{s}) = \mathsf{s}\ \textvisiblespace^{d-1}\ \mathsf{a}\,\mathsf{a}\,\mathsf{b}\ \textvisiblespace^{d-1} \tag{3}$$

$$h_i(\textvisiblespace) = \varepsilon \tag{4}$$

$$h_i(\mathsf{a}) = \begin{cases} \mathsf{A}\ \textvisiblespace^{d-1} & \text{if } i = d-1 \\ \varepsilon & \text{otherwise} \end{cases} \tag{5}$$

$$h_i(\mathsf{b}) = \mathsf{B}\ \textvisiblespace^{d-1-i} \tag{6}$$

$$h_i(\mathsf{A}) = \begin{cases} \mathsf{a}^{n_{\psi(i)}} & \text{if } \psi(i) \text{ is defined} \\ \varepsilon & \text{otherwise} \end{cases} \tag{7}$$

$$h_i(\mathsf{B}) = \begin{cases} \mathsf{a}^{i \cdot \frac{n_{\psi(i)}}{d}}\ \mathsf{b}\ \textvisiblespace^{d-1} & \text{if } \psi(i) \text{ is defined} \\ \varepsilon & \text{otherwise} \end{cases} \tag{8}$$

Before we show that productivity of the PD0L system $\mathcal{H}_F$ coincides with F running forever on input 2, we give some intuition and an example to illustrate the working of $\mathcal{H}_F$.

The following trivial fact is useful to state separately.

Lemma 4.2. *Let $N, d, q, r \in \mathbb{N}$ such that $N = qd + r$, and F a Fractran program. Then $\psi_F(N) = \psi_F(r)$. If moreover $b \in \mathbb{N}$ divides d, then $b \mid N$ if and only if $b \mid r$.* □

Let F be a Fractran program with common denominator d, and (finite or infinite) run $N_0, N_1, N_2, \ldots$. Let $q_i \in \mathbb{N}$ and $r_i \in \Sigma_d$ such that $N_i = q_i d + r_i$, for all $i \geq 0$. We let x_n be the 'contribution' of the iteration H^{n+1}, i.e., x_n is such that $H^{n+1}(\mathsf{s}) = H^n(\mathsf{s})x_n$. Then $H^\omega(\mathsf{s}) = \mathsf{s}x_0x_1x_2\cdots$. We will display $H^\omega(\mathsf{s})$ in separate lines each corresponding to an x_n. The computation of the word $H^\omega(\mathsf{s})$ proceeds in two alternating phases: the transition from even to odd lines corresponds to division by d, and the transition from odd to even lines corresponds to multiplication by the currently applicable fraction $\frac{n_{\psi(N_i)}}{d}$. These phases are indicated by the use of lower- and uppercase letters, that is, $x_{2n} \in \{\textvisiblespace, \mathsf{a}, \mathsf{b}\}$ and $x_{2n+1} \in \{\textvisiblespace, \mathsf{A}, \mathsf{B}\}$, as can be seen from the definition of the morphisms. Now the intuition behind the alphabet symbols (in view of the defining rules of the morphisms) can be described as follows. We use s as the starting symbol, and the symbol $\textvisiblespace$ is used to shift the morphism index of subsequent letters.

In every even line x_{2i}

(i) there is precisely one block of a's; this block is positioned at morphism index 0 and is of length N_i, representing the current value N_i in the run of F;

(ii) b is a special marker for the end of a block of a's, so positioned at morphism index r_i, the remainder of dividing N_i by d.

In every odd line x_{2i+1}

(iii) the number of A's corresponds to the quotient q_i, and every occurrence of A is positioned at morphism index r_i;

(iv) B (also at morphism index r_i) takes care of the multiplication of the remainder r_i with $\frac{n_{\psi(N_i)}}{d}$. Then $\psi(N_i) = \psi(r_i)$ ensures, via Lemma 4.2, that the morphism can select the right fraction to multiply with.

We illustrate the encoding by means of an example.

Example 4.3. Consider the Fractran program $\frac{9}{2}, \frac{5}{3}$, or equivalently

$$F = \frac{27}{6}, \frac{10}{6}$$

and its finite run $2, 9, 15, 25$. Following Definition 4.1 we construct the PD0L system $\mathcal{H}_F = \langle \Gamma, H, \mathsf{s} \rangle$ with $H = \langle h_0, \ldots, h_5 \rangle$ and

$$\begin{aligned}
h_i(\mathsf{s}) &= \mathsf{s}\,\mathsf{␣}^5\,\mathsf{a}\,\mathsf{a}\,\mathsf{b}\,\mathsf{␣}^5 \\
h_0(\mathsf{a}) &= \ldots = h_4(\mathsf{a}) = \varepsilon \\
h_5(\mathsf{a}) &= \mathsf{A}\,\mathsf{␣}^5 \\
h_i(\mathsf{b}) &= \mathsf{B}\,\mathsf{␣}^{5-k} \\
h_0(\mathsf{A}) &= h_2(\mathsf{A}) = h_4(\mathsf{A}) = \mathsf{a}^{27} \\
h_3(\mathsf{A}) &= \mathsf{a}^{10} \\
h_1(\mathsf{A}) &= h_5(\mathsf{A}) = \varepsilon \\
h_0(\mathsf{B}) &= \mathsf{b}\,\mathsf{␣}^5 \\
h_2(\mathsf{B}) &= \mathsf{a}^9\,\mathsf{b}\,\mathsf{␣}^5 \\
h_4(\mathsf{B}) &= \mathsf{a}^{18}\,\mathsf{b}\,\mathsf{␣}^5 \\
h_3(\mathsf{B}) &= \mathsf{a}^5\,\mathsf{b}\,\mathsf{␣}^5 \\
h_1(\mathsf{B}) &= h_5(\mathsf{B}) = \varepsilon
\end{aligned}$$

for $i \in \Sigma_6$. Then $H^\omega(\mathsf{s})$ is finite and the stepwise computation of this fixed point can be displayed as follows. To ease reading, we write below each letter its morphism index. Let z_n denote the morphism index of x_n. Moreover, the word $H^\omega(\mathsf{s}) = \mathsf{s}x_0x_1\cdots$ is broken into lines in such a way that every line x_{n+1} is the image of the previous line x_n under H_{z_n} (except for the line x_0, which is the tail of the image of s under $H = H_0$).

$$\begin{aligned}
&\mathsf{s} \\
&\;0 \\
x_0 &= \mathsf{␣}^5\,\mathsf{a}\,\mathsf{a}\,\mathsf{b}\,\mathsf{␣}^5 \\
&\;1\;0\;1\;2\;3 \\
x_1 &= \mathsf{B}\,\mathsf{␣}^3 \\
&\;2\;3 \\
x_2 &= \mathsf{a}\,\mathsf{a}\,\mathsf{a}\,\mathsf{a}\,\mathsf{a}\,\mathsf{a}\,\mathsf{a}\,\mathsf{a}\,\mathsf{a}\,\mathsf{b}\,\mathsf{␣}^5 \\
&\;0\;1\;2\;3\;4\;5\;0\;1\;2\;3\;4 \\
x_3 &= \mathsf{A}\,\mathsf{␣}^5\,\mathsf{B}\,\mathsf{␣}^2 \\
&\;3\;4\;3\;4 \\
x_4 &= \mathsf{a}\,\mathsf{a}\,\mathsf{a}\,\mathsf{a}\,\mathsf{a}\,\mathsf{a}\,\mathsf{a}\,\mathsf{a}\,\mathsf{a}\,\mathsf{a}\,\mathsf{a}\,\mathsf{a}\,\mathsf{a}\,\mathsf{a}\,\mathsf{b}\,\mathsf{␣}^5 \\
&\;0\;1\;2\;3\;4\;5\;0\;1\;2\;3\;4\;5\;0\;1\;2\;3\;4 \\
x_5 &= \mathsf{A}\,\mathsf{␣}^5\,\mathsf{A}\,\mathsf{␣}^5\,\mathsf{B}\,\mathsf{␣}^2 \\
&\;3\;4\;3\;4\;3\;4 \\
x_6 &= \mathsf{a}\,\mathsf{b}\,\mathsf{␣}^5 \\
&\;0\;1\;2\;3\;4\;5\;0\;1\;2\;3\;4\;5\;0\;1\;2\;3\;4\;5\;0\;1\;2\;3\;4\;5\;0\;1\;2 \\
x_7 &= \mathsf{A}\,\mathsf{␣}^5\,\mathsf{A}\,\mathsf{␣}^5\,\mathsf{A}\,\mathsf{␣}^5\,\mathsf{A}\,\mathsf{␣}^5\,\mathsf{B} \\
&\;1\;2\;1\;2\;1\;2\;1\;2\;1 \\
x_8 &= \varepsilon
\end{aligned}$$

Now we characterize the contribution of every iteration of H in the construction of the word $H^\omega(2)$. We employ the notations given in Lemma 2.2.

Lemma 4.4. *Let $F = \frac{n_1}{d}, \ldots, \frac{n_k}{d}$, and $N \geq 1$. Let $q \in \mathbb{N}$ and $r \in \Sigma_d$ be such that $N = qd + r$. Let $\mathsf{X} = \mathsf{␣}^{d-1}$. Then we have*

$$H(\mathsf{a}^N\,\mathsf{b}\,\mathsf{X}) = (\mathsf{A}\,\mathsf{X})^q\,\mathsf{B}\,\mathsf{␣}^{d-1-r} \tag{9}$$

of length $d(q+1) - r$. If, moreover, $F(N)$ is defined, then

$$H_r((\mathsf{A}\,\mathsf{X})^q\,\mathsf{B}\,\mathsf{␣}^{d-1-r}) = \mathsf{a}^{F(N)}\,\mathsf{b}\,\mathsf{X} \tag{10}$$

of length $F(N) + d$.

Proof. Equation (9) follows immediately by induction on q. To see that (10) holds for $F(N)\!\downarrow$, note that $\psi(N)$ is defined, and so is $\psi(r) = \psi(N)$, by Lemma 4.2. Hence we obtain

$$\begin{aligned}
H_r((\mathsf{A}\,\mathsf{X})^q\,\mathsf{B}\,\mathsf{␣}^{d-1-r}) &= (\mathsf{a}^{n_{\psi(r)}})^q\,H_r(\mathsf{B}\,\mathsf{␣}^{d-1-r}) \\
&= (\mathsf{a}^{n_{\psi(r)}})^q\,\mathsf{a}^{r\cdot\frac{n_{\psi(r)}}{d}}\,\mathsf{b}\,\mathsf{X}
\end{aligned}$$

and we conclude by $F(N) = N \cdot \frac{n_{\psi(N)}}{d} = q \cdot n_{\psi(N)} + r \cdot \frac{n_{\psi(N)}}{d}$. □

Lemma 4.5. *For all Fractran programs F, the PD0L system $\mathcal{H}_F$ is productive if and only if F does not terminate on input 2.*

Proof. Let F and $\mathcal{H}_F$ be as in Definition 4.1. Let $N_0, N_1, N_2, \ldots$ be the finite or infinite run of F on 2, i.e., $N_i = F^i(2)$, and let $t \in \mathbb{N} \cup \{\infty\}$ denote its length. For all i with $0 \leq i < t$, let $q_i \in \mathbb{N}$ and $r_i \in \Sigma_d$ be such that $N_i = q_i d + r_i$.

We define $x_n \in \Sigma^*$ and $z_n \in \Sigma_d$ for all $n \geq 0$, as follows. Let $\mathsf{X} = \mathsf{␣}^{d-1}$, $x_0 = \mathsf{a}\,\mathsf{a}\,\mathsf{b}\,\mathsf{X}$, $z_0 = 0$, and, for $n \geq 1$, let x_n and z_n be such that $H^{n+1}(\mathsf{s}) = H^n(\mathsf{s})x_n$ and $z_n \equiv |H^n(\mathsf{s})| \pmod d$. Then $H^\omega(\mathsf{s}) = \mathsf{s}\,\mathsf{X}\,x_0x_1x_2\cdots$, and the factor x_n is at morphism index z_n. With Lemma 2.2 we then have

$$x_n = H_{z_{n-1}}(x_{n-1}) \qquad z_n \equiv z_{n-1} + |x_{n-1}| \pmod d \tag{11}$$

for all $n \geq 1$. Now we prove by induction on $n \geq 0$ that

$$\begin{aligned}
x_n &= \mathsf{a}^{N_i}\,\mathsf{b}\,\mathsf{X} && z_n = 0 && \text{if } n = 2i < 2t, \\
x_n &= (\mathsf{A}\,\mathsf{X})^{q_i}\,\mathsf{B}\,\mathsf{␣}^{d-1-r_i} && z_n = r_i && \text{if } n = 2i+1 < 2t, \\
x_n &= \varepsilon && z_n = 0 && \text{if } n \geq 2t.
\end{aligned}$$

The base case is immediate. Let $n > 0$. If $n = 2i < 2t$ for some $i < t$, then $N_i = F(N_{i-1})$ is defined, and $x_n = H_{z_{n-1}}(x_{n-1}) = H_{r_{i-1}}((\mathsf{A}\,\mathsf{X})^{q_{i-1}}\,\mathsf{B}\,\mathsf{␣}^{d-1-r_{i-1}}) = \mathsf{a}^{N_i}\,\mathsf{b}\,\mathsf{X}$, and $z_n \equiv z_{n-1} + |x_{n-1}| \equiv r_{i-1} + d(q_{i-1}+1) - r_{i-1} \equiv 0 \pmod p$, both by (11), the induction hypothesis and Lemma 4.4. If $n = 2i+1 < 2t$ for some $i < t$, then $x_n = H_{z_{n-1}}(x_{n-1}) = H_0(\mathsf{a}^{N_i}\,\mathsf{b}\,\mathsf{X}) = (\mathsf{A}\,\mathsf{X})^{q_i}\,\mathsf{B}\,\mathsf{␣}^{d-1-r_i}$, and $z_n \equiv z_{n-1} + |x_{n-1}| \equiv 0 + N_i + d \equiv r_i \pmod p$, again by (11), the induction hypothesis and Lemma 4.4. Finally, if $n = 2t$, then $x_n = H_{z_{n-1}}(x_{n-1}) = H_{r_{t-1}}((\mathsf{A}\,\mathsf{X})^{q_{t-1}}\,\mathsf{B}\,\mathsf{␣}^{d-1-r_{t-1}}) = \varepsilon$, since F terminates on N_{t-1} (and so $\psi(N_{t-1})$ and $\psi(r_{t-1})$ are undefined), and $z_n \equiv z_{n-1} + |x_{n-1}| \equiv r_{t-1} + d(q_{t-1}+1) - r_{t-1} \equiv 0$. Clearly, then also $x_n = \varepsilon$ and $z_n = 0$ for all $n > 2t$. □

Hence, by Lemma 4.5 and Proposition 3.3, deciding productivity of PD0L systems is undecidable.

Theorem 4.6. *The problem of deciding on the input of a PD0L system $\mathcal{H}$ whether $\mathcal{H}$ is productive, is Π_1^0-complete.* □

5. Turing Completeness of Non-Erasing PD0L Systems

In this section we extend the encoding of Fractran from the previous section to show that every computable infinite word can be embedded in the following two ways.

Definition 5.1. Let Σ and $\Gamma \supset \Sigma$ be alphabets with letters $\mathsf{l}, \mathsf{r} \in \Gamma \setminus \Sigma$, and let $\boldsymbol{w} \in \Sigma^\omega$ and $\boldsymbol{u} \in \Gamma^\omega$ be infinite words. We say $\boldsymbol{w}$ is *prefix embedded* in $\boldsymbol{u}$ if the following three conditions are satisfied:

(i) for every finite prefix v of $\boldsymbol{w}$ there is an occurrence $\mathsf{l}\, v\, \mathsf{r}$ in $\boldsymbol{u}$,
(ii) for every occurrence of a word $\mathsf{l}\, v\, \mathsf{r}$ in $\boldsymbol{u}$ with $v \in (\Gamma \setminus \{\mathsf{r}\})^*$ we have that v is a prefix of $\boldsymbol{w}$, and
(iii) letters from Σ occur in $\boldsymbol{u}$ only in factors (subwords) of the form $\mathsf{l}\, v\, \mathsf{r}$ with $v \in (\Gamma \setminus \{\mathsf{r}\})^*$.

Definition 5.2. Let Σ and $\Gamma \supset \Sigma$ be alphabets, and let $\boldsymbol{w} \in \Sigma^\omega$ and $\boldsymbol{u} \in \Gamma^\omega$ be infinite words. We say $\boldsymbol{w}$ is *sparsely embedded* in $\boldsymbol{u}$ if $\boldsymbol{w}$ is obtained from $\boldsymbol{u}$ by erasing all letters in $\Gamma \setminus \Sigma$.

The crucial difference with the encoding of Section 4 is that we now use the knowledge about the remainder not only to select the correct fraction to multiply with, but also to recognize when the current value is divisible by 3 or 5, and correspondingly produce an output bit $\mathsf{0}$ or $\mathsf{1}$, cf. Definition 3.5. The process again proceeds in two phases, for division and multiplication, and we employ lower- and uppercase letters accordingly. We introduce letters l (and L) and r (and R) marking the beginning and the end of the prefix of the infinite word computed by the Fractran program. Furthermore, the symbol R produces the output bits depending on the current remainder r_i. In order to prevent that the output of R changes the morphism index of R, we introduce z (and Z) which compensate the production of R with an inverse length. The letter e (and E) marks the end of the line, and additionally e takes care of realignment after multiplication, such that the first a in each run stands on morphism index 0.

Definition 5.3. Let $F = \frac{n_1}{d}, \ldots, \frac{n_k}{d}$ be a Fractran program such that (without loss of generality) the common denominator d is divisible by 3 and 5. Define the PD0L system $\mathcal{H}_F = \langle \Gamma, H, \mathsf{s} \rangle$ with

$$\Gamma = \{\mathsf{s}, \textvisiblespace, \circ, \mathsf{a}, \mathsf{A}, \mathsf{b}, \mathsf{B}, \mathsf{z}, \mathsf{Z}, \mathsf{l}, \mathsf{L}, \mathsf{0}, \mathsf{1}, \mathsf{r}, \mathsf{R}, \mathsf{e}, \mathsf{E}, \mathsf{Q}\}$$

and $H = \langle h_0, \ldots, h_{d-1} \rangle$ consisting of (non-erasing) morphisms $h_i : \Gamma^* \to \Gamma^*$ defined for every $i \in \Sigma_d$ as follows:

$$h_i(\mathsf{s}) = \mathsf{s}\, \textvisiblespace^{d-1}\, \mathsf{a}\, \mathsf{a}\, \mathsf{b}\, \textvisiblespace^{d-1}\, \mathsf{z}\, \textvisiblespace^{d-2}\, \mathsf{l}\, \mathsf{r}\, \textvisiblespace^{d-1}\, \mathsf{e}$$

$$h_i(\textvisiblespace) = \textvisiblespace$$

$$h_i(\mathsf{a}) = \begin{cases} \mathsf{A}\, \textvisiblespace^{d-1} & \text{if } i = d-1, \\ \textvisiblespace^{d} & \text{otherwise.} \end{cases}$$

$$h_i(\mathsf{b}) = \mathsf{B}$$

$$h_i(\mathsf{z}) = \mathsf{Z}$$

$$h_i(\mathsf{l}) = \mathsf{L}$$

$$h_i(\mathsf{r}) = \mathsf{R}$$

$$h_i(\mathsf{e}) = \mathsf{E}\, \circ^{d-i}$$

$$h_i(\circ) = \textvisiblespace^{d}$$

$$h_i(\mathsf{A}) = \begin{cases} \mathsf{a}^{n_{\psi(i)}}\, \textvisiblespace & \text{if } \psi(i) \text{ is defined,} \\ \mathsf{Q} & \text{otherwise.} \end{cases}$$

$$h_i(\mathsf{B}) = \begin{cases} \mathsf{a}^{i \cdot \frac{n_{\psi(i)}}{d}}\, \mathsf{b} & \text{if } \psi(i) \text{ is defined,} \\ \mathsf{Q} & \text{otherwise.} \end{cases}$$

$$h_i(\mathsf{Z}) = \begin{cases} \mathsf{z}\, \textvisiblespace^{d-1} & \text{if } 3 \mid i \text{ or } 5 \mid i, \\ \mathsf{z} & \text{otherwise.} \end{cases}$$

$$h_i(\mathsf{L}) = \mathsf{l}$$

$$h_i(\mathsf{0}) = \mathsf{0}$$

$$h_i(\mathsf{1}) = \mathsf{1}$$

$$h_i(\mathsf{R}) = \begin{cases} \mathsf{0}\, \mathsf{r} & \text{if } 3 \mid i, \\ \mathsf{1}\, \mathsf{r} & \text{if } 5 \mid i, \\ \mathsf{r} & \text{otherwise.} \end{cases}$$

$$h_i(\mathsf{E}) = \mathsf{e}$$

$$h_i(\mathsf{Q}) = \mathsf{Q}$$

Remark 5.4. In Definition 5.3 we require that 3 and 5 divide the common denominator in order for Lemma 4.2 to apply. Informally speaking, via the remainder we can only observe factors that also divide the common denominator.

Remark 5.5. Let F and H be as in Definition 5.3. It can be shown that the symbol Q occurs in the word $H^\omega(\mathsf{s})$ if and only if the Fractran program F halts on input 2. This fact can be used to show that it is undecidable whether Q occurs in $H^\omega(\mathsf{s})$. However, we prove this differently, namely by applying Theorem 5.9 and using the fact that for non-terminating Fractran programs it is undecidable whether digit $\mathsf{1}$ occurs in the sequence computed by the program. See Theorem 5.15.

Time for an example.

Example 5.6. Consider the following Fractran program:

$$F = \frac{2}{7}, \frac{3 \cdot 7}{2 \cdot 5}, \frac{3}{2}, \frac{5}{3}, \frac{2}{1}$$

which has the infinite run 2, 3, 5, 10, 21, 6, 9, 15, 25, 50, 105, 30, 63, 18, 27, 45, 75, 125, 250, 525, 150, ... and computes the word

$$011000011000000011000000000011000000000000011 \cdots,$$

that is, $0^1 11\ 0^{3+1} 11\ 0^{6+1} 11\ 0^{9+1} 11\ 0^{12+1} \ldots = \prod_{i=0}^{\infty} 0^{3 \cdot i + 1} 11$. Writing the program with the common denominator 210 yields:

$$F = \frac{60}{210}, \frac{441}{210}, \frac{315}{210}, \frac{350}{210}, \frac{420}{210}$$

Let $\mathcal{H}_F$ be the PD0L encoding of F, as given in Definition 5.3. We consider the first steps of the iteration of the morphisms; for easier reading, we drop blocks of consecutive symbols $\textvisiblespace^{210}$ (they do not change the morphism index of other letters), and let $\mathsf{X} = \textvisiblespace^{209}$.

$$\begin{array}{rl}
 & \mathsf{s} \\
 & {\scriptstyle 0} \\
x_0 = & \mathsf{X}\, \mathsf{a}^2\, \mathsf{b}\, \mathsf{X}\, \mathsf{z}\, \textvisiblespace^{208}\, \mathsf{l}\, \mathsf{r}\, \mathsf{X}\, \mathsf{e} \\
 & {\scriptstyle 1\ 0\ 0\ 2\ 3\ 2\ 3\ 1\ 2\ 2} \\
x_1 = & \mathsf{X}\, \mathsf{B}\, \mathsf{X}\, \mathsf{Z}\, \textvisiblespace^{208}\, \mathsf{L}\, \mathsf{R}\, \mathsf{X}\, \mathsf{E}\, \circ^{208} \\
 & {\scriptstyle 3\ 2\ 3\ 2\ 3\ 1\ 2\ 3\ 2\ 3} \\
x_2 = & \mathsf{X}\, \mathsf{a}^3\, \mathsf{b}\, \mathsf{X}\, \mathsf{z}\, \textvisiblespace^{208}\, \mathsf{l}\, \mathsf{r}\, \mathsf{X}\, \mathsf{e} \\
 & {\scriptstyle 1\ 0\ 3\ 4\ 3\ 4\ 2\ 3\ 4\ 3} \\
x_3 = & \mathsf{X}\, \mathsf{B}\, \mathsf{X}\, \mathsf{Z}\, \textvisiblespace^{208}\, \mathsf{L}\, \mathsf{R}\, \mathsf{X}\, \mathsf{E}\, \circ^{207} \\
 & {\scriptstyle 4\ 3\ 4\ 3\ 4\ 2\ 3\ 4\ 3\ 4} \\
x_4 = & \mathsf{X}\, \mathsf{a}^5\, \mathsf{b}\, \mathsf{X}\, \mathsf{z}\, \textvisiblespace^{207}\, \mathsf{l}\, \mathsf{0}\, \mathsf{r}\, \mathsf{X}\, \mathsf{e} \\
 & {\scriptstyle 1\ 0\ 5\ 6\ 5\ 6\ 3\ 4\ 5\ 6\ 5} \\
x_5 = & \mathsf{X}\, \mathsf{B}\, \mathsf{X}\, \mathsf{Z}\, \textvisiblespace^{207}\, \mathsf{L}\, \mathsf{0}\, \mathsf{R}\, \mathsf{X}\, \mathsf{E}\, \circ^{205} \\
 & {\scriptstyle 6\ 5\ 6\ 5\ 6\ 3\ 4\ 5\ 6\ 5\ 6} \\
x_6 = & \mathsf{X}\, \mathsf{a}^{10}\, \mathsf{b}\, \mathsf{X}\, \mathsf{z}\, \textvisiblespace^{206}\, \mathsf{l}\, \mathsf{0}\, \mathsf{1}\, \mathsf{r}\, \mathsf{X}\, \mathsf{e} \\
 & {\scriptstyle 1\ 0\ 10\ 11\ 10\ 11\ 7\ 8\ 9\ 10\ 11\ 10} \\
x_7 = & \mathsf{X}\, \mathsf{B}\, \mathsf{X}\, \mathsf{Z}\, \textvisiblespace^{206}\, \mathsf{L}\, \mathsf{0}\, \mathsf{1}\, \mathsf{R}\, \mathsf{X}\, \mathsf{E}\, \circ^{200} \\
 & {\scriptstyle 11\ 10\ 11\ 10\ 11\ 7\ 8\ 9\ 10\ 11\ 10\ 11} \\
x_8 = & \mathsf{X}\, \mathsf{a}^{21}\, \mathsf{b}\, \mathsf{X}\, \mathsf{z}\, \textvisiblespace^{205}\, \mathsf{l}\, \mathsf{0}\, \mathsf{1}\, \mathsf{1}\, \mathsf{r}\, \mathsf{X}\, \mathsf{e} \\
 & {\scriptstyle 1\ 0\ 21\ 22\ 21\ 22\ 17\ 18\ 19\ 20\ 21\ 22\ 21} \\
x_9 = & \mathsf{X}\, \mathsf{B}\, \mathsf{X}\, \mathsf{Z}\, \textvisiblespace^{205}\, \mathsf{L}\, \mathsf{0}\, \mathsf{1}\, \mathsf{1}\, \mathsf{R}\, \mathsf{X}\, \mathsf{E}\, \circ^{189} \\
 & {\scriptstyle 22\ 21\ 22\ 21\ 22\ 17\ 18\ 19\ 20\ 21\ 22\ 21\ 22}
\end{array}$$

Note that the number of a's in rows x_{2i} precisely models the run of the Fractran program 2, 3, 5, 10, 21, ... By construction, the morphism index of R equals the remainder r_i, and consequently R produces the prefix 011 of the word computed by F.

Things become more interesting if the number of a's exceeds the denominator 210. We look at a few steps further down the sequence:

$$x_{36} = \mathsf{X}\,\mathsf{a}^{250}\,\mathsf{b}\,\mathsf{X}\,\mathsf{z}\,\llcorner^{191}\,\mathsf{l}_{01100001100000001}\,\mathsf{r}\,\mathsf{X}\,\mathsf{e}$$
$$x_{37} = \mathsf{X}\,\mathsf{A}\,\mathsf{X}\,\mathsf{B}\,\mathsf{X}\,\mathsf{Z}\,\llcorner^{191}\,\mathsf{L}_{01100001100000001}\,\mathsf{R}\,\mathsf{X}\,\mathsf{E}\,\circ^{170}$$
$$x_{38} = \mathsf{X}\,\mathsf{a}^{525}\,\mathsf{b}\,\mathsf{X}\,\mathsf{z}\,\llcorner^{190}\,\mathsf{l}_{01100001100000001}\,\mathsf{1}\,\mathsf{r}\,\mathsf{X}\,\mathsf{e}$$
$$x_{39} = \mathsf{X}\,(\mathsf{A}\,\mathsf{X})^{2}\,\mathsf{B}\,\mathsf{X}\,\mathsf{Z}\,\llcorner^{190}\,\mathsf{L}_{01100001100000001}\,\mathsf{1}\,\mathsf{R}\,\mathsf{X}\,\mathsf{E}\,\circ^{105}$$
$$x_{40} = \mathsf{X}\,\mathsf{a}^{150}\,\mathsf{b}\,\mathsf{X}\,\mathsf{z}\,\llcorner^{189}\,\mathsf{l}_{01100001100000001}\,\mathsf{1}\,\mathsf{0}\,\mathsf{r}\,\mathsf{X}\,\mathsf{e}$$

In line x_{36} we have 250 a's giving rise to only one A in the subsequent line x_{37}. Again, A, B, Z and R stand on index $40 \equiv 250 \pmod{210}$, and consequently both deduce that the first applicable fraction is $\frac{441}{210} = \frac{3\cdot 7}{2\cdot 5}$. The letter A represents the quotient from the division by 210, and hence produces 441 a's. The letter B is responsible for the multiplication of the remainder, and thus produces $40 \cdot \frac{3\cdot 7}{2\cdot 5} = 84$ a's. Thus we get $441 + 84 = 525$ a's in line x_{38}. Moreover, R produces a 1 since 250 is dividable by 5 but not 3, and Z produces an X-block ($\mathsf{X} = \llcorner^{209}$) to keep R and the remaining symbols on the correct index.

Now the division of 525 by 210 has quotient 2 and remainder 105, and so we have two A's in line x_{39}, and all A's, B, Z and R standing on index 105. The first applicable fraction for 105 is $\frac{60}{210} = \frac{2}{7}$, and correspondingly the two A's produce 60 a's each, and B produces $30 = 105 \cdot \frac{2}{7}$ a's, in total giving rise to 150 a's in line x_{40}. Now 525 is divisible by 3 and so R produces a 0.

We now start working towards a proof of Theorem 5.9. Let $F = \frac{n_1}{d}, \ldots, \frac{n_k}{d}$ be a Fractran program. We again employ the notation H_i as given in Lemma 2.2. Furthermore, we define relations $\rightsquigarrow, \sim \subseteq \Sigma^* \times \Sigma^*$ by

$$\rightsquigarrow = \{\langle u\,\llcorner^{d}\,v, uv\rangle \mid u, v \in \Sigma^*\} \qquad \sim = (\leftsquigarrow \cup \rightsquigarrow)^*$$

Then clearly we have $H_i(u) \sim H_i(v)$ for all $i \in \Sigma_d$, $u, v \in \Sigma^*$ with $u \sim v$. This allows us to prove properties of $H^\omega(\mathsf{s})$ reasoning modulo $\sim$. Below, we write $\llcorner^n$ with $n < 0$ to denote the block $\llcorner^m$ with $m \in \Sigma_d$ and $n \equiv m \pmod{d}$. For $N \geq 1$ we define

$$\begin{aligned} \kappa(N) &= \mathsf{0} && \text{if } 3 \mid N \\ \kappa(N) &= \mathsf{1} && \text{if } 3 \nmid N \text{ and } 5 \mid N \\ \kappa(N) &= \varepsilon && \text{otherwise} \end{aligned}$$

Lemma 5.7. *Let $F = \frac{n_1}{d}, \ldots, \frac{n_k}{d}$, and $N \geq 1$. Let $\kappa = \kappa(N)$ and $\mathsf{X} = \llcorner^{d-1}$. Let $q \in \mathbb{N}$ and $r \in \Sigma_d$ be such that $N = qd + r$, and let $v \in \{\mathsf{0}, \mathsf{1}\}^*$. Then we have*

$$\begin{aligned} &H_1(\mathsf{X}\,\mathsf{a}^{N}\,\mathsf{b}\,\mathsf{X}\,\mathsf{z}\,\llcorner^{d-2-|v|}\,\mathsf{l}\,v\,\mathsf{r}\,\mathsf{X}\,\mathsf{e}) \\ &= \mathsf{X}\,(\mathsf{A}\,\mathsf{X})^{q}\,\mathsf{B}\,\mathsf{X}\,\mathsf{Z}\,\llcorner^{d-2-|v|}\,\mathsf{L}\,v\,\mathsf{R}\,\mathsf{X}\,\mathsf{E}\,\circ^{d-r} \end{aligned} \tag{12}$$

of length equivalent to $-r$ modulo d.

Moreover, if in addition $F(N)$ is defined, then we have

$$\begin{aligned} &H_{r+1}(\mathsf{X}\,(\mathsf{A}\,\mathsf{X})^{q}\,\mathsf{B}\,\mathsf{X}\,\mathsf{Z}\,\llcorner^{d-2-|v|}\,\mathsf{L}\,v\,\mathsf{R}\,\mathsf{X}\,\mathsf{E}\,\circ^{d-r}) \\ &\sim \mathsf{X}\,\mathsf{a}^{F(N)}\,\mathsf{b}\,\mathsf{X}\,\mathsf{z}\,\llcorner^{d-2-|v\kappa|}\,\mathsf{l}\,v\,\kappa\,\mathsf{r}\,\mathsf{X}\,\mathsf{e} \end{aligned} \tag{13}$$

of length equivalent to $F(N)$ modulo d.

Proof. (12) follows immediately: Let $x, y \in \Gamma^*$ be arbitrary. Then $H_1(\mathsf{X}\,x) = \mathsf{X}\,H_0(x)$, and $H_0(\mathsf{a}^{N}\,y) = \mathsf{X}\,(\mathsf{A}\,\mathsf{X})^{q}H_0(\mathsf{a}^{r}\,y) = \mathsf{X}\,(\mathsf{A}\,\mathsf{X})^{q}H_r(y)$. Furthermore, $H_r(\mathsf{b}\,\mathsf{X}\,\mathsf{z}\,\llcorner^{d-2-|v|}\,\mathsf{l}\,v\,\mathsf{r}\,\mathsf{X}\,\mathsf{e}) = \mathsf{B}\,\mathsf{X}\,\mathsf{Z}\,\llcorner^{d-2-|v|}\,\mathsf{L}\,v\,\mathsf{R}\,H_r(\mathsf{e})$ and $H_r(\mathsf{e}) = \circ^{d-r}$.

To show (13), let $F(N)$ be defined. Then $\psi(N) = \psi(r)$ is also defined (Lemma 4.2). Hence we get, for $w \in \Gamma^*$ arbitrary:

$$\begin{aligned} &H_{r+1}(\mathsf{X}\,(\mathsf{A}\,\mathsf{X})^{q}\,\mathsf{B}\,\mathsf{X}\,w) \\ &= \mathsf{X}\,H_r((\mathsf{A}\,\mathsf{X})^{q}\,\mathsf{B}\,\mathsf{X}\,w) \\ &= \mathsf{X}\,(\mathsf{a}^{n_{\psi(r)}}\,\llcorner\,\mathsf{X})^{q}H_r(\mathsf{B}\,\mathsf{X}\,w) \\ &= \mathsf{X}\,(\mathsf{a}^{n_{\psi(r)}}\,\llcorner\,\mathsf{X})^{q}\,\mathsf{a}^{r\cdot\frac{n_{\psi(r)}}{d}}\,\mathsf{b}\,\mathsf{X}\,H_r(w) \\ &\sim \mathsf{X}\,\mathsf{a}^{F(N)}\,\mathsf{b}\,\mathsf{X}\,H_r(w) \end{aligned}$$

Finally, to compute $H_r(w)$ for $w = \mathsf{Z}\,\llcorner^{d-2-|v|}\,\mathsf{L}\,v\,\mathsf{R}\,\mathsf{X}\,\mathsf{E}\,\circ^{d-r}$, distinguish the following cases: If 3 or 5 divides r, then 3 or 5 divides also N, respectively, by Lemma 4.2. Hence we have $\kappa = \kappa(N) = \kappa(r)$ and $|\kappa| = 1$, and

$$\begin{aligned} H_r(w) &= \mathsf{z}\,\mathsf{X}\,\llcorner^{d-2-|v|}\,\mathsf{l}\,v\,H_r(\mathsf{R}\,\mathsf{X}\,\mathsf{E}\,\circ^{d-r}) \\ &= \mathsf{z}\,\mathsf{X}\,\llcorner^{d-2-|v|}\,\mathsf{l}\,v\,\kappa\,\mathsf{r}\,\mathsf{X}\,\mathsf{e}\,\llcorner^{d(d-r)} \\ &\sim \mathsf{z}\,\llcorner^{d-3-|v|}\,\mathsf{l}\,v\,\kappa\,\mathsf{r}\,\mathsf{X}\,\mathsf{e} \end{aligned}$$

as required. And , if $3 \nmid r$ and $5 \nmid r$, then $\kappa = \kappa(N) = \kappa(r) = \varepsilon$, and $H_r(w) \sim \mathsf{z}\,\llcorner^{d-2-|v|}\,\mathsf{l}\,v\,\mathsf{r}\,\mathsf{X}\,\mathsf{e}$, as required. □

Lemma 5.8. *Let F be a Fractran program computing an infinite word $\boldsymbol{w}$. Then the PD0L system $\mathcal{H}_F$ from Definition 5.3 generates a word that prefix embeds $\boldsymbol{w}$ (see Definition 5.1).*

Proof. Let $F = \frac{n_1}{d}, \ldots, \frac{n_k}{d}$ be a Fractran program, computing $\boldsymbol{w} \in \{\mathsf{0}, \mathsf{1}\}^\omega$, i.e., $\boldsymbol{w} = W_F$ with W_F as defined in Definition 3.5. Let $N_0, N_1, N_2, \ldots$ be the infinite run of F starting on $N_0 = 2$ (so with infinitely many N_i divisible by 3 or 5). Let $q_i \in \mathbb{N}$ and $r_i \in \Sigma_d$ be such that $N_i = q_i d + r_i$. Let $\mathcal{H}_F = \langle \Gamma, H, \mathsf{s}\rangle$ be the PD0L system defined in Definition 5.3.

We show that $\boldsymbol{u} = H^\omega(\mathsf{s})$ satisfies the conditions (i), (ii), and (iii) of Definition 5.1, by characterizing the contribution of every iteration of H. For every $i \in \mathbb{N}$ we let $v_i \in \Sigma^*$ be defined by $v_0 = \varepsilon$ and $v_{i+1} = v_i\kappa(N_i)$ so that $\boldsymbol{w} = \lim_{i\to\infty} v_i$. For all $n \geq 0$ let $x_n \in \Gamma^*$ and $z_n \in \Sigma_d$ be such that $H^{n+1}(\mathsf{s}) = H^n(\mathsf{s})x_n$ and $z_n \equiv |H^n(\mathsf{s})| \pmod{d}$. Then $H^\omega(\mathsf{s}) = \mathsf{s}\,x_0x_1\cdots$, and

$$x_n = H_{z_{n-1}}(x_{n-1}) \qquad z_n \equiv z_{n-1} + |x_{n-1}| \pmod{d} \tag{14}$$

Let us abbreviate $Y_i = \llcorner^{d-2-|v_i|}$. We prove that x_n and z_n satisfy

$$\begin{aligned} x_{2i} &\sim \mathsf{X}\,\mathsf{a}^{N_i}\,\mathsf{b}\,\mathsf{X}\,\mathsf{z}\,Y_i\,\mathsf{l}\,v_i\,\mathsf{r}\,\mathsf{X}\,\mathsf{e} \\ x_{2i+1} &\sim \mathsf{X}\,(\mathsf{A}\,\mathsf{X})^{q_i}\,\mathsf{B}\,\mathsf{X}\,\mathsf{Z}\,Y_i\,\mathsf{L}\,v_i\,\mathsf{R}\,\mathsf{X}\,\mathsf{E}\,\circ^{d-r_i} \\ z_{2i} &\equiv 1 \pmod{d} \\ z_{2i+1} &\equiv r_i + 1 \pmod{d} \end{aligned}$$

by induction on n. For the base case, we see $z_0 = |\mathsf{s}| = 1$ and $x_0 = \mathsf{X}\,\mathsf{a}\,\mathsf{a}\,\mathsf{b}\,\mathsf{X}\,\mathsf{z}\,\llcorner^{d-2}\,\mathsf{l}\,\mathsf{r}\,\mathsf{X}\,\mathsf{e}$, as required. So let $n \geq 1$. If $n = 2i$ for some $i \geq 1$, it follows from (14), Lemma 5.7 and the induction hypothesis that

$$\begin{aligned} x_n &= H_{z_{n-1}}(x_{n-1}) \\ &\sim H_{r_{i-1}+1}(\mathsf{X}\,(\mathsf{A}\,\mathsf{X})^{q_{i-1}}\,\mathsf{B}\,\mathsf{X}\,\mathsf{Z}\,Y_{i-1}\,\mathsf{L}\,v_{i-1}\,\mathsf{R}\,\mathsf{X}\,\mathsf{E}\,\circ^{d-r_{i-1}}) \\ &\sim \mathsf{X}\,\mathsf{a}^{N_i}\,\mathsf{b}\,\mathsf{X}\,\mathsf{z}\,Y_i\,\mathsf{l}\,v_i\,\mathsf{r}\,\mathsf{X}\,\mathsf{e} \end{aligned}$$

and $z_n \equiv z_{n-1} + |x_{n-1}| \equiv r_{i-1} + 1 - r_{i-1} \equiv 1 \pmod{d}$.

Similarly, if $n = 2i + 1$ for some $i \geq 0$, we obtain

$$\begin{aligned} x_n &= H_{z_{n-1}}(x_{n-1}) \\ &\sim H_1(\mathsf{X}\,\mathsf{a}^{N_i}\,\mathsf{b}\,\mathsf{X}\,\mathsf{z}\,Y_i\,\mathsf{l}\,v_i\,\mathsf{r}\,\mathsf{X}\,\mathsf{e}) \end{aligned}$$

$$\sim \mathsf{X}\,\mathsf{a}^{N_{i+1}}\,\mathsf{b}\,\mathsf{X}\,\mathsf{z}\,Y_{i+1}\,\mathsf{l}\,v_{i+1}\,\mathsf{r}\,\mathsf{X}\,\mathsf{e}$$

and $z_n \equiv z_{n-1} + |x_{n-1}| \equiv 1 + N_{i+1} \equiv r_i + 1 \pmod d$.

Knowing the exact shape (modulo $\sim$) of $\boldsymbol{u} = H^\omega(\mathsf{s})$, it is now easy to verify that $\boldsymbol{u}$ it satisfies (i), (ii), and (iii), taking into account that $\sqcup$ does not occur in any factor $\mathsf{l}\,v\,\mathsf{r}$ of $\boldsymbol{u}$ with $v \in (\Gamma \setminus \{\mathsf{r}\})^*$, by the definition of the morphisms. □

We are ready to collect our main results.

Theorem 5.9. *Every computable infinite word can be prefix embedded in a PD0L word (see Definition 5.1).*

Proof. Let $\boldsymbol{w} \in \{\mathsf{0}, \mathsf{1}\}^\omega$ be an infinite computable word. Then, by Proposition 3.7, $\boldsymbol{w}$ is computed by some Fractran program. By Lemma 5.8 the claim follows. □

Definition 5.10. Let F be a Fractran program, and $\mathcal{H}_F$ the PD0L system given in Definition 5.3 We define the PD0L system $\mathcal{H}'_F$ as the result of replacing in $\mathcal{H}_F$ the rules $h_i(\mathsf{0}) = \mathsf{0}$ and $h_i(\mathsf{1}) = \mathsf{1}$ by $h_i(\mathsf{0}) = \sqcup$ and $h_i(\mathsf{1}) = \sqcup$, for all $i \in \Sigma_d$.

Lemma 5.11. *Let F be a Fractran program computing an infinite word $\boldsymbol{w}$, and let $\boldsymbol{u} \in \Gamma^\omega$ be the PD0L word generated by the $\mathcal{H}'_F$ defined in Definition 5.10. Then $\boldsymbol{w}$ is sparsely embedded in $\boldsymbol{u}$.*

Proof. By an easy adaptation of the proof of Lemma 5.8, noting that every output $\mathsf{0}$ and $\mathsf{1}$ is produced precisely once and in the next iteration replaced by $\sqcup$. □

Theorem 5.12. *Every computable infinite word can be sparsely embedded in a PD0L word (see Definition 5.2).*

Proof. Analogous to the proof of Theorem 5.9, replacing Lemma 5.8 by Lemma 5.11. □

It is known that the set of morphic words is closed under finite state transductions [2, Theorem 7.9.1]. In particular, if we erase all occurrences of a certain letter from a morphic word, the result is a morphic (or finite) word. From Theorem 5.12 it follows that this is not the case for PD0L words, establishing a negative answer to Problem 29 (1) and (2) of [23].

Corollary 5.13. *The set of PD0L words is not closed under finite state transductions.*

Proof. There are computable streams that are not PD0L words [10]; hence the class of PD0L words is not closed under finite state transductions, by Theorem 5.12 (erasing letters is a finite state transduction). □

Finite state transducers play a central role in computer science. The transducibility relation via finite state transducers (FST) gives rise to a hierarchy of degrees of infinite words [17], analogous to the recursion theoretic hierarchy. But, in contrast to the latter, the FST-hierarchy does not identify all computable streams. An open problem in this area is the lack of methods for discriminating infinite words $\boldsymbol{u}, \boldsymbol{v}$, that is, to show that there exists no finite state transducer that transduces $\boldsymbol{u}$ to $\boldsymbol{v}$. Discriminating morphic words seems to require heavier machinery than arguments based on the pumping lemma.

We will now collect several immediate consequences of Theorem 5.9. First of all, we have solved the open problem [22] on the existence of PD0L words that have exponential subword complexity.

Theorem 5.14. *There is a PD0L word $\boldsymbol{u}$ such that $p_{\boldsymbol{u}}(n) \geq 2^n$.*

Proof. Let $F = F_{\mathrm{BIN}}$ be the Fractran program defined in Section 3, computing the word $W_F = \mathrm{BIN}$ (Proposition 3.9). Furthermore, let $\mathcal{H}_F = \langle \Gamma, H, \mathsf{s} \rangle$ be the PD0L system of Definition 5.3. Then, by Lemma 5.8, $\boldsymbol{u} = H^\omega(\mathsf{s})$ is the word we are looking for. □

Lemma 5.8 also allows us to give a negative answer to [23, Problem 29 (3)].

Theorem 5.15. *The following problems are undecidable:*

INPUT: *PD0L system $\mathcal{H} = \langle \Gamma, H, \mathsf{s} \rangle$, letter $b \in \Gamma$*

QUESTION: *(i) Does b occur in $H^\omega(\mathsf{s})$?*

(ii) Does b occur infinitely many times in $H^\omega(\mathsf{s})$?

Proof. We show that the following problem is undecidable: given a Fractran program F computing an infinite word $\boldsymbol{w}$ over the alphabet $\{0, 1\}$, does the letter 1 occur in $\boldsymbol{w}$? This suffices since by Lemma 5.8, if $\boldsymbol{u}$ is the infinite word generated by $\mathcal{H}_F$, then the letter 1 occurs in $\boldsymbol{u}$ if and only if 1 occurs infinitely often in $\boldsymbol{u}$ if and only if 1 occurs in $\boldsymbol{w}$.

We use the input 2 halting problem for Fractran programs which is Σ^0_1-complete by Proposition 3.3. Let F be an arbitrary Fractran program. By Remark 3.4 we can replace the primes in F to obtain a program F' that does not contain the primes $\{2, 3, 5\}$ such that F' halts on 7 if and only if F halts on 2. We now extend F' to F'' by adding in front the fraction $\frac{3\cdot 7}{2}$ and at the end the fractions $\frac{5}{3}$ and $\frac{1}{1}$. Then the first fraction of F'' starts F' on input 7 and ensures that the output is 0 for every step that F' is running, and only when F' terminates, the last two fractions of F'' switch the output to 1 and keep running forever. □

From Theorem 5.15 it follows immediately that the first-order (and monadic second-order) theory of PD0L words is undecidable, answering [23, Problem 28]; see [23] also for the definition of the first-order and monadic theory of a sequence. This again stands in contrast to the case for morphic sequences, which are known to have a decidable monadic second-order theory [5].

Corollary 5.16. *The first-order theory of PD0L words is undecidable.*

Also immediate from Theorem 5.15 is the undecidability of equivalence of PD0L systems (equality of the limit words they generate). We note that equivalence of D0L systems is decidable [9], whereas that of CD0L words is an open problem.

Corollary 5.17. *Equality of PD0L words (given by their PD0L systems) is undecidable.*

Proof. We reduce problem (i) stated in Theorem 5.15 to equivalence of PD0L systems, as follows. Let $\mathcal{H} = \langle \Sigma, H, s \rangle$ be a PD0L system and $b \in \Sigma$, and let $\mathcal{H}' = \langle \Sigma \cup \{b'\}, H', s' \rangle$ where $b' \notin \Sigma$ and H' and s' are obtained from H and s by replacing all occurrences of b by b', and letting $H'(b) = b$. Then b does not occur in the word generated by $\mathcal{H}$ if and only if $\mathcal{H}$ and $\mathcal{H}'$ generate the same word. By Theorem 5.15 this is undecidable. □

6. A Concrete PD0L Word with Exponential Subword Complexity

In this section we give a concrete example of a PD0L system which generates an infinite word with exponential subword complexity. The word embeds all prefixes of the word $\mathrm{BIN} = (0)_{\mathsf{z}}(1)_{\mathsf{z}}(2)_{\mathsf{z}}\cdots$ given in Definition 3.8. We refrain from proving that it indeed does have this property; the existence of such a PD0L word is already proved in the previous section, see Theorem 5.9.

We define a PD0L system $H = \langle h_0, h_1, \ldots, h_{15} \rangle$ consisting of 16 morphisms. We express morphism indices $i \in \Sigma_{16}$ by linear combinations

$$i = a(i) \cdot 2^3 + r(i) \cdot 2^2 + c(i) \cdot 2^1 + o(i) \cdot 2^0.$$

with $a(i), r(i), c(i), o(i) \in \{0, 1\}$ which we call *flags*. We use these flags to transmit information between symbols:

- $a(i) = 1$ stands for *active*,
- $r(i) = 1$ stands for *running*,
- $c(i) = 1$ stands for *carry flag*, and
- $o(i) = 1$ stands for *output one*.

The idea is to simulate a binary counter, using the representation of Definition 3.8. The counter repeatedly increments $(+1)$ the current value, and thereby brings $(n)_z$ to $(n+1)_z$. During an increment process we need to shift the activity from bit to bit. To this end, the activity flag $a(i)$ indicates whether a symbol at morphism index i is active.

We explain the increment process using the following example word. Here ... is the already produced prefix of BIN, and we assume for the moment that the symbols a, b and d each stand for a word of length 16, and c for a word of length 8.

$$\underset{a}{a}\ c\ \underset{a}{b}\ c\ c\ \underset{a}{b}\ c\ c\ c\ c\ \underset{a}{a}\ L\ \ldots\ R \tag{15}$$

Here a and b represent the bits 0 and 1, respectively, and we shall continue to call them bits. Ignoring the c's in between, (15) represents the word 0110 (in turn representing the integer 21). Apart from incrementing this initial word a b b a, it is at the same time 'copied' bit by bit to the word 0110 between symbols L and R. The least significant bit is left, and consequently the process of incrementing will proceed from left to right. The symbol c (being of length 8) swaps the value of $a(i)$ for the morphism index i of all subsequent letters. Note that between the n-th and $(n+1)$-th occurrence of bits (a or b), there are 2^{n-1} c's. Hence, if the first bit is active, then this is the only active bit.

We now describe the transition from (15) to its PD0L image (16). Note that starting from the first occurrence of c, every second occurrence in (15) has the activity flag set. When the symbol c is active, it will be eliminated, that is replaced by the symbol d (of assumed length 16), thus activating the next bit for the next iteration (16).

$$\underset{a}{B}\ \underset{a}{d}\ \underset{a}{b}\ \underset{a}{c}\ d\ b\ \underset{a}{c}\ d\ \underset{a}{c}\ d\ a\ L\ \ldots\ 0\ R \tag{16}$$

Note that a is replaced by B; uppercase letters are used for indicating the already processed bits during the increment loop. When the increment loop is finished, uppercase will be turned to lowercase, and the process restarts. The switch from a to B corresponds to incrementing. This is controlled by the carry flag indicating whether a bit has to be flipped. The carry flag is always set at the start of an increment loop. To keep this example simple we do not display this flag. At the end of this section we give the first iterations of the PD0L system displaying all flags.

Notice that in (16) the second bit b is the only active bit (ignoring B which we have already dealt with). Again, eliminating the active c's will shift the activity to the following bit:

$$\underset{a}{B}\ \underset{a}{d}\ \underset{a}{B}\ \underset{a}{d}\ \underset{a}{d}\ \underset{a}{b}\ \underset{a}{c}\ \underset{a}{d}\ \underset{a}{d}\ \underset{a}{d}\ \underset{a}{a}\ L\ \ldots\ 0\ 1\ R \tag{17}$$

After one more step we obtain:

$$\underset{a}{B}\ \underset{a}{d}\ \underset{a}{B}\ \underset{a}{d}\ \underset{a}{d}\ \underset{a}{B}\ \underset{a}{d}\ \underset{a}{d}\ \underset{a}{d}\ \underset{a}{d}\ \underset{a}{a}\ L\ \ldots\ 0\ 1\ 1\ \underset{a}{R} \tag{18}$$

and finally:

$$\underset{a}{B}\ \underset{a}{d}\ \underset{a}{B}\ \underset{a}{d}\ \underset{a}{d}\ \underset{a}{B}\ \underset{a}{d}\ \underset{a}{d}\ \underset{a}{d}\ \underset{a}{d}\ \underset{a}{A}\ L\ \ldots\ 0\ 1\ 1\ 0\ \underset{a}{R} \tag{19}$$

As soon as the most significant bit a is active, R becomes active as well. This can be used to recognize when the addition is finished, and then R unsets the bit $r(i)$ to restart the addition procedure.

The active bit makes use of the flag $o(i)$ to 'communicate' with the symbol R whether to output a 0 or 1. This actually means that R can produce the 0 or 1 only two iterations later; for simplicity we have in this intuitive explanation abstracted from this technicality and produce the 0's and 1's in the immediately following iteration (after a bit has become active).

There are more symbols and technical subtleties to be explained, but we leave this to the imagination of the reader. Enjoy!

The morphisms h_i are defined for all $i \in \Sigma_{16}$ as follows:

$$h_i(\mathsf{s}) = \mathsf{s}\,␣^{13} \bullet \mathsf{a}\,␣^{15}\,\mathsf{c}\,␣^{7}\,\mathsf{P}\,␣^{15}\,\mathsf{O}\,␣^{15}\,\mathsf{Z}_1\,␣^{14}\,\mathsf{L}\,\mathsf{R}_1\,␣^{8}$$

$$h_i(␣) = ␣$$

$$h_i(\bullet) = ␣^{2}$$

$$h_i(\circ) = ␣^{16}$$

$$h_i(\star) = ␣^{2}\,\circ$$

$$h_i(0) = 0$$

$$h_i(1) = 1$$

$$h_i(\mathsf{a}) = \begin{cases} \mathsf{a} & \text{if } \neg a(i) \\ ␣^{14} \star \mathsf{A} & \text{if } a(i) \wedge \neg c(i) \\ ␣^{14} \star ␣^{12} \bullet^{2} \mathsf{B} & \text{if } a(i) \wedge c(i) \end{cases}$$

$$h_i(\mathsf{A}) = \begin{cases} \mathsf{a} & \text{if } \neg r(i) \\ \mathsf{A} & \text{if } r(i) \end{cases}$$

$$h_i(\mathsf{b}) = \begin{cases} \mathsf{b} & \text{if } \neg a(i) \\ \mathsf{B} & \text{if } a(i) \wedge \neg c(i) \\ \mathsf{A} & \text{if } a(i) \wedge c(i) \end{cases}$$

$$h_i(\mathsf{B}) = \begin{cases} \mathsf{b} & \text{if } \neg r(i) \\ \mathsf{B} & \text{if } r(i) \end{cases}$$

$$h_i(\mathsf{c}) = \begin{cases} \mathsf{c} & \text{if } \neg a(i) \\ \mathsf{d}\,␣^{8} & \text{if } a(i) \end{cases}$$

$$h_i(\mathsf{d}) = \begin{cases} \mathsf{c}\,␣^{8} & \text{if } \neg r(i) \\ \mathsf{d} & \text{if } r(i) \end{cases}$$

$$h_i(\mathsf{P}) = \begin{cases} \mathsf{P} & \text{if } \neg c(i) \\ \mathsf{P} & \text{if } c(i) \wedge \neg a(i) \\ \mathsf{a} & \text{if } c(i) \wedge a(i) \end{cases}$$

$$h_i(\mathsf{o}) = \begin{cases} \mathsf{o} & \text{if } \neg c(i) \\ \mathsf{o} & \text{if } c(i) \wedge \neg a(i) \\ \mathsf{d} & \text{if } c(i) \wedge a(i) \end{cases}$$

$$h_i(\mathsf{O}) = \begin{cases} \mathsf{O} & \text{if } \neg a(i) \\ \mathsf{o}\,␣^{15}\,\mathsf{O} & \text{if } a(i) \wedge \neg c(i) \\ \mathsf{d}\,␣^{15}\,\mathsf{P}\,␣^{15}\,\mathsf{O} & \text{if } a(i) \wedge c(i) \end{cases}$$

$$h_i(\mathsf{Z}) = \begin{cases} \mathsf{Z} & \text{if } \neg r(i) \\ \mathsf{Z}\,␣^{15} & \text{if } r(i) \wedge \neg a(i) \\ \mathsf{Z}_3\,␣^{15} & \text{if } r(i) \wedge a(i) \end{cases}$$

$$h_i(\mathsf{Z}_1) = \mathsf{Z}$$

$$h_i(\mathsf{Z}_2) = \mathsf{Z}_1$$

$$h_i(\mathsf{Z}_3) = \mathsf{Z}_2$$

$$h_i(\mathsf{L}) = \mathsf{L}$$

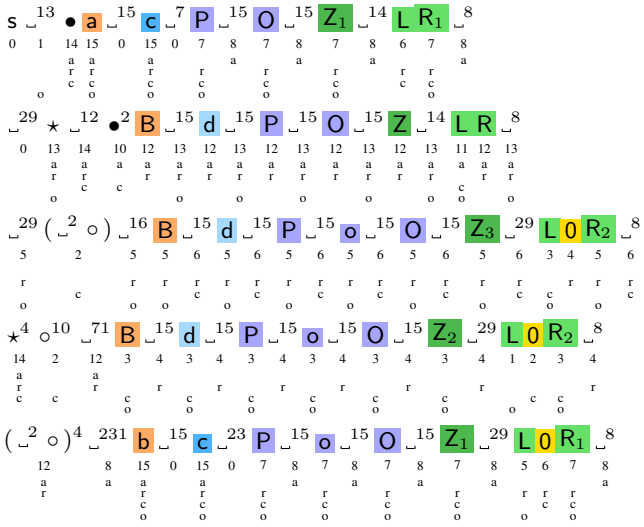
$$h_i(\mathsf{R}) = \begin{cases} \mathsf{R} & \text{if } \neg r(i) \\ \mathsf{0\,R} & \text{if } r(i) \wedge \neg o(i) \wedge \neg a(i) \\ \mathsf{0\,R_3}\ {\scriptstyle\sqcup^8}\ \star^4 \circ^{10} & \text{if } r(i) \wedge \neg o(i) \wedge a(i) \wedge \neg c(i) \\ \mathsf{0\,R_3}\ {\scriptstyle\sqcup^8}\ \star^4 \circ^8 & \text{if } r(i) \wedge \neg o(i) \wedge a(i) \wedge c(i) \\ \mathsf{1\,R} & \text{if } r(i) \wedge o(i) \wedge \neg a(i) \\ \mathsf{1\,R_3}\ {\scriptstyle\sqcup^8}\ \star^4 \circ^{10} & \text{if } r(i) \wedge o(i) \wedge a(i) \wedge \neg c(i) \\ \mathsf{1\,R_3}\ {\scriptstyle\sqcup^8}\ \star^4 \circ^8 & \text{if } r(i) \wedge o(i) \wedge a(i) \wedge c(i) \end{cases}$$

$$h_i(\mathsf{R_1}) = \mathsf{R}$$

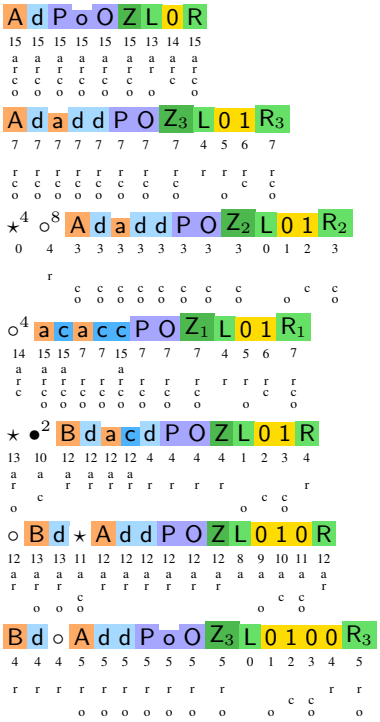
$$h_i(\mathsf{R_2}) = \mathsf{R_1}$$

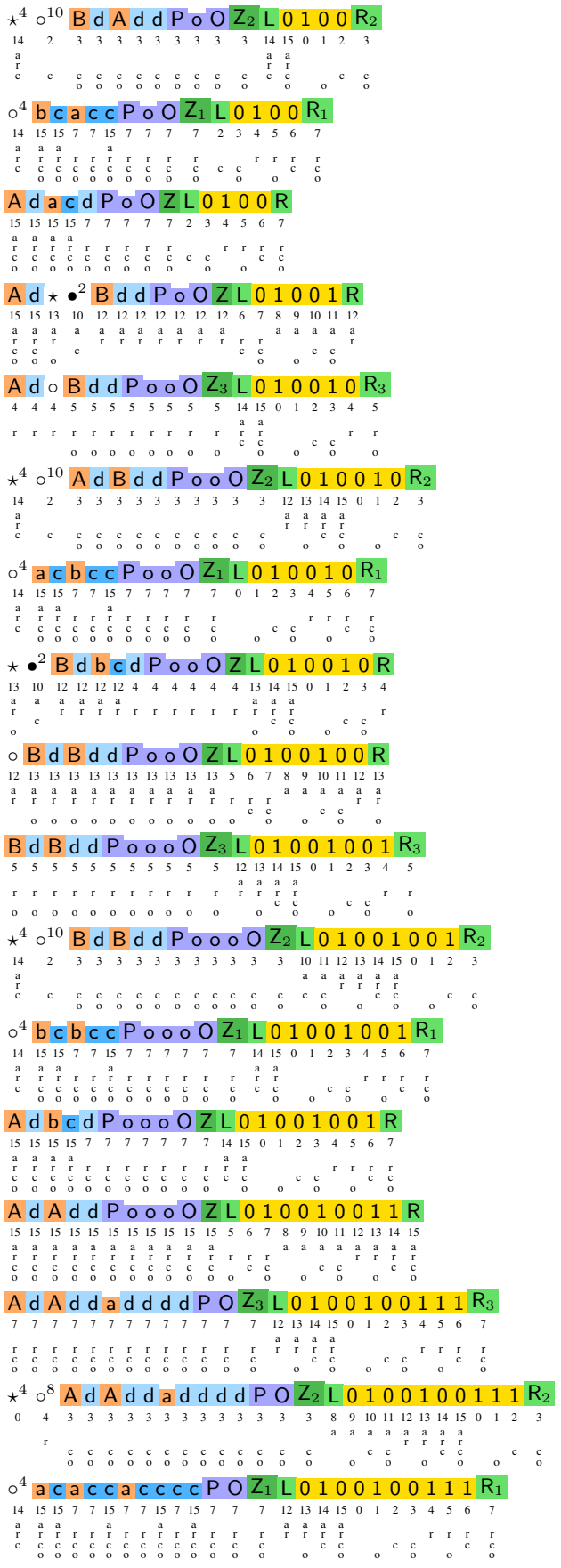
$$h_i(\mathsf{R_3}) = \mathsf{R_2}$$

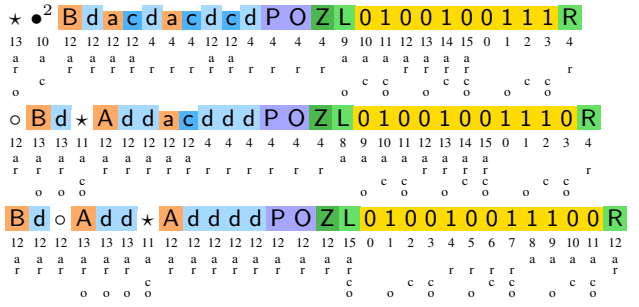
The PD0L word $H^\omega(\mathsf{s})$ starts as follows:



For compactness, we continue without displaying the symbols $\sqcup$. The length n of blocks $\sqcup^n$ matters only modulo 16, and can be deduced from the morphism indexes of the surrounding letters.







7. Discussion

In Section 6 we have encoded the state of a binary counter using a binary encoding. In comparison with the binary counter obtained from the Fractran encoding, this yields an enormous simplification concerning the number of required morphisms. Moreover, we have illustrated a construction which allows for shifting the activity from one letter to the next in each iteration of the morphism, and how the letters can 'communicate' computation results to the following letter. It would be interesting to investigate whether Turing machines can be encoded in a similar way. The crucial difference would be that for Turing machines we need to shift the activity left or right depending on the outcome of the current step; the binary counter always shifts the activity to the right. It is unclear to us whether the encoding from Section 6 can be extended in this direction. Compared to our Fractran encoding of Section 5, such an encoding of Turing machines could lead to significantly less morphisms (but with a slightly larger alphabet).

References

[1] J.-P. Allouche. Sur la Complexité des Suites Infinies. *Journées Montoises*, 1(2):133–143, 1994.

[2] J.-P. Allouche and J. Shallit. *Automatic Sequences: Theory, Applications, Generalizations*. Cambridge University Press, New York, 2003.

[3] S. Arshon. Démonstration de l'Existence de Suites Asymétriques Infinies. *Matematicheskii Sbornik*, 44:769–777, 1937. In Russian. French summary: 777–779.

[4] J. Berstel. Mots Sans Carré et Morphismes Itérés. *Discrete Mathematics*, 29:235–244, 1980.

[5] O. Carton and W. Thomas. The Monadic Theory of Morphic Infinite Words and Generalizations. *Information and Computation*, 176(1):51–65, 2002.

[6] J. Cassaigne and J. Karhumäki. Toeplitz Words, Generalized Periodicity and Periodically Iterated Morphisms. *European Journal of Combinatorics*, 18(5):497–510, 1997.

[7] J. H. Conway. Unpredictable Iterations. In *Proc. of the 1972 Number Theory Conference*, pages 49–52. University of Colorado, 1972.

[8] J. H. Conway. Fractran: A Simple Universal Programming Language for Arithmetic. In *Open Problems in Communication and Computation*, pages 4–26. Springer, 1987.

[9] K. Čulik II and T. Harju. The ω-Sequence Problem for D0L Systems Is Decidable. *Journal of the Association for Computing Machinery*, 31(2):282–298, 1984.

[10] K. Čulik II and J. Karhumäki. Iterative Devices Generating Infinite Words. In *Proc. 9th Ann. Symp. on Theoretical Aspects of Computer Science (STACS 1992)*, volume 577 of *Lecture Notes in Computer Science*, pages 531–543. Springer, 1992.

[11] K. Čulik II, J. Karhumäki, and A. Lepistö. Alternating Iteration of Morphisms and Kolakovski [*sic*] Sequence. In G. Rozenberg and A. Salomaa, editors, *Lindermayer Systems, Impacts on Theoretical Computer Science, Computer Graphics and Developmental Biology*, pages 93–106. Springer, 1992.

[12] A. Ehrenfeucht, K. P. Lee, and G. Rozenberg. Subword Complexity of Various Classes of Deterministic Languages without Interaction. *Theoretical Computer Science*, 1:59–75, 1975.

[13] J. Endrullis, C. Grabmayer, and D. Hendriks. Data-Oblivious Stream Productivity. In *Proc. 15th Int. Conf. on Logic for Programming, Artifical Intelligence and Reasoning (LPAR 2008)*, volume 5330 of *Lecture Notes in Computer Science*, pages 79–96. Springer, 2008.

[14] J. Endrullis, C. Grabmayer, and D. Hendriks. Complexity of Fractran and Productivity. In *Proc. 22nd Int. Conf. on Automated Deduction (CADE 2009)*, volume 5663 of *Lecture Notes in Computer Science*, pages 371–387. Springer, 2009.

[15] J. Endrullis, C. Grabmayer, D. Hendriks, A. Isihara, and J.W. Klop. Productivity of Stream Definitions. *Theoretical Computer Science*, 411:765–782, 2010.

[16] J. Endrullis and D. Hendriks. Lazy Productivity via Termination. *Theoretical Computer Science*, 412(28):3203–3225, 2011.

[17] J. Endrullis, D. Hendriks, and J.W. Klop. Degrees of Streams. *Journal of Integers*, 11B(A6):1–40, 2011. Proceedings of the Leiden Numeration Conference 2010.

[18] S. Ferenczi. Complexity of Sequences and Dynamical Systems. *Discrete Mathematics*, 206(1-3):145–154, 1999.

[19] C. Grabmayer, J. Endrullis, D. Hendriks, J.W. Klop, and L.S. Moss. Automatic Sequences and Zip-Specifications. In *Proc. Symp. on Logic in Computer Science (LICS 2012)*. IEEE Computer Society, 2012.

[20] K. Jacobs and M. Keane. 0-1-Sequences of Toeplitz Type. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 13(2):123–131, 1969.

[21] W. Kolakovski. Self Generating Runs. *The American Mathematical Monthly*, 72, 1965. Problem 5304.

[22] A. Lepistö. On the Power of Periodic Iteration of Morphisms. In *Proc. 20th Int. Coll. on Automata, Languages and Programming (ICALP 1993)*, volume 700 of *Lecture Notes in Computer Science*, pages 496–506. Springer, 1993.

[23] A. A. Muchnik, Y. L. Pritykin, and A. L. Semenov. Sequences Close to Periodic. *Russian Mathematical Surveys*, 64(5):805–871, 2009.

[24] P. Séébold. On Some Generalizations of the Thue-Morse Morphism. *Theoretical Computer Science*, 292(1):283–298, 2003.

[25] B. A. Sijtsma. On the Productivity of Recursive List Definitions. *ACM Transactions on Programming Languages and Systems*, 11(4):633–649, 1989.