

On Periodically Iterated Morphisms

Jörg Endrullis

Department of Computer Science
VU University Amsterdam
j.endrullis@vu.nl

Dimitri Hendriks

Department of Computer Science
VU University Amsterdam
r.d.a.hendriks@vu.nl

Abstract

We investigate the computational power of periodically iterated morphisms, also known as DOL systems with periodic control; we call them PerDOL systems for short. These systems give rise to a class of one-sided infinite sequences, called PerDOL words. We construct a PerDOL word with exponential subword complexity, thereby answering a question raised by Lepistö [23] on the existence of such words. We solve another open problem concerning the decidability of the first-order theories of PerDOL words [24]; we show it is already undecidable whether a certain letter occurs in a PerDOL word.

Categories and Subject Descriptors F.1.1 [Models of Computation]; F.4.1 [Mathematical Logic]; F.4.2 [Grammars and Other Rewrite Systems]; F.4.3 [Formal Languages]; G.2.1 [Combinatorics].

Keywords Periodically iterated morphisms, subword complexity, Fractran, automatic sequences, morphic words.

1. Introduction

Morphisms and automata for transforming and generating infinite words provide a fundamental tool for formal languages. A morphic word $\mathbf{u}$ is the image of a coding c (a letter-to-letter morphism) of the fixed point of iterating a morphism $h: A^* \rightarrow A^*$ (a map with the property $h(uv) = h(u)h(v)$ for all $u, v \in A^*$) on a starting letter $a \in A$, that is, $\mathbf{u} = c(h^\omega(a))$, where $h^\omega(a) = \lim_{n \rightarrow \infty} h^n(a)$. An important subclass is formed by the automatic words [2], that are obtained when $h(a)$ has the same length for every letter a . These classes have many applications and establish deep connections between theoretical computer science, number theory, and physics.

In this paper we investigate the computational power of a natural generalization of morphic words, namely words generated by periodically iterated morphisms [7, 11, 12, 23]. Instead of iterating a single morphism, multiple morphisms are alternately applied in a periodic fashion. For example, we consider the most famous word generated by such a procedure: the Oldenburger-Kolakoski word $K \in \{1, 2\}^\omega$ [22, 25],

$$K = 1\,2\,2\,1\,1\,2\,1\,2\,2\,1\,2\,2\,1\,1\,2\,1\,1\,2\,2\,1\,2\,1\,1\,2\,1\,2\,2\,1\,1\,2\,\dots,$$

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CSL-LICS 2014, July 14–18, 2014, Vienna, Austria.
Copyright © 2014 ACM 978-1-4503-2886-9...\$15.00.
<http://dx.doi.org/10.1145/2603088.2603152>

defined by $K(0) = 1$ and $K(n)$ = the length of the n -th run of K , where a ‘run’ is a maximal block of consecutive identical symbols. K can be generated by alternating morphisms h_1 and h_2 on the starting word 12, with $h_a(1) = a$ and $h_a(2) = aa$ ($a = 1, 2$). The first few iterations are

$$\begin{aligned} h_1(1) h_2(2) &= 12\mathbf{2} \\ h_1(1) h_2(2) h_1(2) &= 122\mathbf{11} \\ h_1(1) h_2(2) h_1(2) h_2(1) h_1(1) &= 12211\mathbf{21}. \end{aligned}$$

It is known that the Oldenburger-Kolakoski word is not purely morphic [12], 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 morphic image of a purely morphic word.

We shall use the ‘DOL’ terminology: DOL for purely morphic, CDOL for morphic, and PerDOL for words generated by periodically iterated morphisms, like the Oldenburger-Kolakoski word.

Well-studied properties of infinite words are their subword complexity and the decidability of their first- and second-order monadic logics. Subword complexity [1, 2, 19] is a natural characteristic of sequences, and can be used to prove that a sequence belongs to a certain class. The subword complexity of a sequence w is a function mapping n to the number of n -length factors in w . It is well-known that morphic words have at most quadratic subword complexity [13], whereas automatic words have at most linear subword complexity. Lepistö [23] proves that for all $x \in \mathbb{R}$ there is a PerD0L word with subword complexity in $\Omega(n^x)$; hence there are PerD0L words that are not CD0L. It remained an open problem whether PerD0L words can exhibit exponential subword complexity. We give a positive answer to this question.

The decidability of first- or second-order monadic logic of infinite words is important, for example, in model checking. A crucial result in this area is the decidability of the monadic-second order logic of ranked trees generated by order- n recursion schemes [26]. Also for morphic sequences it is known that their monadic second order logic is decidable [6]. It is natural to ask where the limits of decidability are. We find that the first-order logic of PerDOL words is not decidable.

The main results (Theorems 5.8 and 5.10) of this paper are:

- (A) For every computable word $w \in \Sigma^\omega$ there exists a *PerDOL* word u such that all prefixes of w occur in u as subwords between special marker symbols;
- (B) For every computable word $w \in \Sigma^\omega$ there exists a *PerDOL* word u such that w is the subsequence of u obtained from selecting all letters from Σ .

Our construction of PerDOL systems generating such words u makes use of Fractran [8, 9], which is a Turing complete programming language invented by Conway. Fractran turns out to be very suitable for our purposes, but note that we could have used any other Turing complete language.

Consequences of our main results (A) and (B) are:

- (1) There exist PerDOL words with exponential subword complexity (Theorem 5.11).
- (2) It is undecidable to determine whether a letter occurs (infinitely often) in a PerDOL word (Theorem 5.12).
- (3) The first-order theory of PerDOL words is undecidable (Corollary 5.13).
- (4) Equality of PerDOL words is undecidable (Corollary 5.14).
- (5) The set of PerDOL words is not closed under finite state transductions (Corollary 6.2).
- (6) There exist a PerDOL word $w \in \{0, 1\}^\omega$ such that any computable word u over a finite alphabet can be obtained from w by finite state transduction (Corollary 6.5).

All the above results concern PerDOL systems whose morphisms are non-erasing. We also study erasing PerDOL systems, and find:

- (7) It is undecidable to determine, given an erasing PerDOL system, if it generates an infinite word (Theorem 4.6).

Outline. In Section 2 we introduce PerDOL systems. In Section 3 we explain the workings of the Fractran algorithm, and show how to generate any computable infinite word. Then, as a steppingstone to our main results, we start with a proof of (7) in Section 4. This proof illustrates our key construction: encoding Fractran programs as PerDOL systems. We then modify and extend this encoding in Sections 5 and 6 to prove that PerDOL words can embed every computable word, in the sense of (A) and (B) above. We give a detailed example of the translation, and prove (1)–(6) listed above.

2. DOL Systems with Periodic Control

We use standard terminology and notation, 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 $\Sigma^\omega = \{x \mid x : \omega \rightarrow \Sigma\}$ with $\omega = \{0, 1, 2, \dots\}$, the set of non-negative integers. On the set of all words $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ we define the metric d for all $u, v \in \Sigma^\infty$ by $d(u, v) = 2^{-n}$, where n is the length of the longest common prefix of u and v .

We define $\Sigma_p = \{0, \dots, p-1\}$. 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 $x = uv$ 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 $h : \Sigma^* \rightarrow \Gamma^*$ 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 $k \in \omega$, 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 ‘DOL words with periodic control’ or just ‘PerDOL words’ for short, were introduced in [11]. These form a generalization of DOL words, also known as *purely morphic* words, which are obtained by iterating a single morphism.

Definition 2.1. Let $H = \langle h_0, \dots, h_{p-1} \rangle$ be a tuple of morphisms $h_i : \Sigma^* \rightarrow \Sigma^*$. Define the map $H : \Sigma^* \rightarrow \Sigma^*$ by

$$H(a_0 a_1 \dots a_n) = u_0 u_1 \dots u_n$$

where $u_i = h_k(a_i)$, with $k \equiv i \pmod{p}$ and $k \in \Sigma_p$.

If $s \in \Sigma^*$ is such that $s \preceq H(s)$, then we call $\mathcal{H} = \langle \Sigma, H, s \rangle$ a *PerDOL system*. Then in the metric space (Σ^∞, d) the limit

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

exists, and we call $H^\omega(s)$ the *PerDOL 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 PerDOL word generated by p morphisms, and $x = uv$ 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 $i \equiv |u| \pmod{p}$.

We note that the map $H : \Sigma^* \rightarrow \Sigma^*$ can be viewed as a restricted form of a *general state machine* (gsm) [4] where the state transition function is independent of the input letter.

DOL words are generated by DOL systems $\langle \Sigma, h, s \rangle$, that is, PerDOL systems $\langle \Sigma, \langle h \rangle, s \rangle$ consisting of one single morphism h . Following [7], we call the image of a DOL word under a coding a *CDOL* word. CDOL words are also known as *morphic* or *substitutive* 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, since erasing morphisms may also 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 monotone function H .

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

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

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 PerDOL analogue of the usual condition for productivity of DOL systems. In Section 4 we show that productivity of PerDOL systems in general is undecidable. Productivity has been studied in the wider perspective of term rewriting systems in [14, 16].

Remark 2.3. Let $\langle \Sigma, h, s \rangle$ be a DOL 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 $h^\omega(s) = s x h(x) h^2(x) \dots$ is infinite. The generalization of this condition to PerDOL 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 $n \in \omega$, where $v_n \in \Sigma^*$ and $z_n \in \Sigma_p$ are defined by $z_0 = 0$ and

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

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 $H^\omega(v_0) = v_0 v_1 v_2 \dots$ to be infinite.

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

Proposition 2.5 ([13]). *The subword complexity of DOL words, and hence of CDOL words, is at most quadratic.*

We first consider an example of an erasing PerDOL 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:

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

where addition is computed modulo 3. Then $\mathcal{H}$ is productive (by Proposition 2.8) and generates the word

$$H^\omega(0) = 0121120101221201120212010120201001210 \dots$$

Definition 2.7. Let $\mathcal{H} = \langle \Sigma, \langle h_0, \dots, h_{p-1} \rangle, s \rangle$ be a PerD0L 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 $k_i \in \omega$ such that $k_i = |h_i(b)|$ for all $b \in \Sigma$. We say $\mathcal{H}$ is (*globally*) *uniform* if, for some $k \in \omega$, each h_i is k -uniform ($i \in \Sigma_p$).

Obviously, a globally k -uniform PerD0L 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 $\mathcal{H} = \langle \Sigma, \langle h_0, \dots, h_{p-1} \rangle, w \rangle$ be a locally uniform PerD0L 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 $w_{|w|}, w_{|w|+1}, w_{|w|+2}, \dots$ of finite words defined for $n \geq |w|$ by $w_{|w|} = H(w)$ and

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

We have $|w_{|w|}| = s(|w|)$ and by induction we get $|w_n| = s(n)$ for every $n \geq |w|$. The limit $\lim_{n \rightarrow \infty} w_n$ 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 $H = \langle h_0, h_1 \rangle$ with $h_0, h_1 : \Sigma_3^* \rightarrow \Sigma_3^*$ the morphisms defined by

$$h_0(b) = b(b+1)(b+2) \quad h_1(b) = (b+2)(b+1)b$$

for all $b \in \Sigma_3$, where addition is modulo 3. Then the PerD0L system $\langle \Sigma_3, H, 0 \rangle$ generates the infinite word

$$H^\omega(0) = 0120212012102010212012101201201210 \dots$$

This is the square-free Arshon word [3], which Berstel proved to be an example of a CD0L word that is not a D0L word [5]. That $H^\omega(0)$ can indeed be defined as a CD0L word follows from Proposition 2.10.

It is not hard to see that, when a word u is generated by a (globally) k -uniform PerD0L system, it is k -automatic [2], i.e., u is the image under a coding of the iterative fixed point of a k -uniform morphism.

Proposition 2.10. Fix $k \geq 2$, and let $\mathcal{H} = \langle \Sigma, H, s \rangle$ be a k -uniform PerD0L system. Then $H^\omega(s)$ is k -automatic.

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

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

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

$$g^n(t) = \langle 0, u(0) \rangle \langle 1, u(1) \rangle \dots \langle qk^n - 1, u(qk^n - 1) \rangle$$

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

One might wonder whether also locally uniform, productive PerD0L systems always generate morphic words. The following examples show that this is not the case.

Example 2.11 ([23]). 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 $H = \langle h_0, \dots, h_{p-1} \rangle$ is a PerD0L system, and h_i the morphisms

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

For example, $F_3 = 010100110001011001000110011100 \dots$. Lepistö [23] proves that F_p has more than quadratic subword complexity, for every $p \geq 2$. Hence, with Proposition 2.5, these PerD0L words F_p cannot be CD0L words. We note that, conversely, the existence of CD0L words that are not PerD0L words was shown in [11].

Example 2.12 ([7]). A Toeplitz word [21] over an alphabet Σ is generated by a seed word $u \in \Sigma(\Sigma \cup \{?\})^*$ with $? \notin \Sigma$, as follows. Start with the periodic $T_0(u) = u^\omega = uuu \dots$ and, for $i > 0$, $T_i(u) = F_u(T_{i-1}(u))$ where $F_u(x)$, for $x \in (\Sigma \cup \{?\})^\omega$, is obtained from u^ω by replacing the sequence of all occurrences of $?$ by x . Then define $T(u) = \lim_{i \rightarrow \infty} T_i(u)$.

For example, $u = 12????$ generates the infinite word $T(u) = 121211221112221 \dots$. Cassaigne and Karhumäki [7] show that all Toeplitz words are PerD0L 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 $a \in \{1, 2\}$. From [7, Theorem 5] it follows that the subword complexity $p_{T(u)}(n) \in \Theta(n^r)$ with $r = \frac{\log 5}{\log 5 - \log 3} \simeq 3.15066$, thus forming an alternative proof of what was established in [23]: there are PerD0L words that are not CD0L.

3. Fractran for Computing Infinite Words

Fractran [8, 9] 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

$$F = \frac{n_1}{d_1}, \dots, \frac{n_k}{d_k} \quad (1)$$

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 $N = 2^3 3^5$:

$$2^3 3^5 \rightarrow 2^2 3^4 5^1 \rightarrow 2^1 3^3 5^2 \rightarrow 2^0 3^2 5^3 \rightarrow 2^0 3^1 5^3 \rightarrow 2^0 3^0 5^3.$$

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 $N = 2^a 3^b 5^c \dots$, 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, Conway [9] first introduces programs that consist of multiple lines of the form

$$\alpha : \frac{n_1}{d_1} \rightarrow \alpha_1, \frac{n_2}{d_2} \rightarrow \alpha_2, \dots, \frac{n_m}{d_m} \rightarrow \alpha_m \quad (2)$$

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 $\alpha_1, \dots, \alpha_m$ 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} , with looping states α and β , given by the lines

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

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 $f_1, \dots, f_k$ now is a shorthand for the Fractran-1 program $\alpha : f_1 \rightarrow \alpha, f_2 \rightarrow \alpha, \dots, f_k \rightarrow \alpha$. Conway [9] explains how every Fractran- n program ($n \geq 2$) can be compiled into a Fractran-1 program, using the steps:

- (i) For every looping state α , introduce a ‘mirror’ state α' , substitute α' for all occurrences of α in the right-hand sides of its program line (leaving α unchanged in all other lines), and add the line

$$\alpha' : \frac{1}{1} \rightarrow \alpha$$

- (ii) Replace state identifiers α by ‘fresh’ prime numbers.
- (iii) For every line of the form (2) append the following fractions (preserving the order):

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

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

$$\begin{aligned} \alpha : \frac{2 \cdot 5}{3} \rightarrow \alpha', \frac{1}{1} \rightarrow \beta & \quad \beta : \frac{3}{5} \rightarrow \beta' \\ \alpha' : \frac{1}{1} \rightarrow \alpha & \quad \beta' : \frac{1}{1} \rightarrow \beta. \end{aligned}$$

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

$$F_{\text{add}} = \frac{2 \cdot 5 \cdot \alpha'}{3 \cdot \alpha}, \frac{\alpha}{\alpha'}, \frac{\beta}{\alpha}, \frac{3 \cdot \beta'}{5 \cdot \beta}, \frac{\beta}{\beta'}.$$

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 $r_1, \dots, r_p$ for storage, and primes $\alpha_1, \dots, \alpha_q$ for control, at any instant the entire *configuration* of F (= register contents + state) is uniquely represented by the current working integer N

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

with $e_1, \dots, e_p \geq 0$ and $1 \leq j \leq q$.

The reason to employ two state indicators α and α' 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 α' to say “continue in the current state”. This secondary indicator α' 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 $g : A \rightarrow B$ we write $g(x) \downarrow$ to indicate that g is defined on $x \in A$, and $g(x) \uparrow$ otherwise.

Definition 3.1. Let $F = f_1, \dots, f_k$ be a Fractran program with $f_i \in \mathbb{Q}_{>0}$. We define the partial function $\psi_F : \omega \rightarrow \omega$ 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, that is,

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

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

We overload notation and use $F : \omega \rightarrow \omega$ 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 $N, F(N), F^2(N), \dots$. 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 (implicit in [9], explicit in [15, 20]).

Proposition 3.2. *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 3.3. In some sense it does not matter which prime numbers are used in a Fractran program. We make this precise. For p a prime number, and n a positive integer, we let $v_p(n)$ denote the *p-adic valuation* of n , i.e., $v_p(n) = e$ with $e \in \omega$ maximal such that p^e divides n . For $\vec{p} = \langle p_1, p_2, \dots, p_t \rangle$ we write $v_{\vec{p}}(n)$ to denote $\langle v_{p_1}(n), v_{p_2}(n), \dots, v_{p_t}(n) \rangle$. Let F be a Fractran program with t distinct primes $\vec{p} = p_1, p_2, \dots, p_t$, let $\vec{q} = q_1, q_2, \dots, q_t$ 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 for all $M, N > 0$ 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 3.4. 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 $F(N), F^2(N), \dots$ 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 3.5. The Fractran program $\frac{3}{2}, \frac{5}{3}, \frac{3}{5}$ gives rise to the computation $3, 5, 3, 5, 3, 5, \dots$, and hence computes the infinite word $010101 \dots$ of alternating bits.

A word u is called *computable* if it is finite, or there exists a Turing machine M such that, for every natural number n , M halts on the input of the binary representation of n with output $u(n)$.

Proposition 3.6. *Every (finite or infinite) computable word over a finite alphabet can be computed by a Fractran program.*

Proof. It suffices to consider the alphabet $\{0, 1\}$; the construction immediately generalizes to arbitrary finite alphabets. In [15] it is shown that Fractran programs can simulate any Turing machine computation. By Remark 3.3 we may assume that this translation

does not employ the primes $\{2, 3, 5\}$. Then a straightforward adaptation of the proof in [15] 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 BIN that has every finite binary word as one of its factors. We represent a natural number n by the word $(n)_z$ that is obtained from the standard binary expansion of $n+1$ by dropping the leftmost 1, and reversing the result.

Definition 3.7. Let $\Sigma = \{0, 1\}$. For all $n \in \omega$ and $w \in \Sigma^*$, we define $(n)_z \in \Sigma^*$ and $[w]_z \in \omega$ by

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

and let BIN denote the word (similar to the Champernowne word):

$$\text{BIN} = (0)_z(1)_z(2)_z \cdots = 010010011000100010 \cdots$$

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

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

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 $(n)_z$ 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 $(\cdot)_z$ 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 $\alpha_2, \alpha_3, \alpha_4$, and α_5 , we introduce mirror states $\alpha'_2, \alpha'_3, \alpha'_4$, and α'_5 . Second, we assign the following prime numbers to the identifiers:

$$\begin{array}{cccccccccccccccc} \alpha_1 & \alpha_2 & \alpha'_2 & \alpha_3 & \alpha'_3 & \alpha_4 & \alpha'_4 & \alpha_5 & \alpha'_5 & \beta_0 & \beta_1 & r_1 & r_2 & r_3 \\ 2 & 7 & 11 & 13 & 17 & 19 & 23 & 29 & 31 & 3 & 5 & 37 & 41 & 43 \end{array}$$

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

$$\begin{aligned} F_{\text{BIN}} = & \frac{r_2 \alpha_5}{r_3 \alpha_1}, \frac{r_1 \alpha_2}{\alpha_1}, \frac{r_2 r_3 \alpha'_2}{r_1 \alpha_2}, \frac{\alpha_3}{\alpha_2}, \frac{\alpha_2}{\alpha'_2}, \frac{r_1 \alpha'_3}{r_3 \alpha_3}, \frac{\alpha_4}{\alpha_3}, \frac{\alpha_3}{\alpha'_3}, \\ & \frac{r_3 \alpha'_4}{r_2^2 \alpha_4}, \frac{\beta_0}{r_2 \alpha_4}, \frac{\beta_1}{r_3 \alpha_4}, \frac{\alpha_4}{\alpha'_4}, \frac{r_2 \alpha'_5}{r_3 \alpha_5}, \frac{\alpha_4}{\alpha'_5}, \frac{\alpha_5}{\alpha'_5}, \frac{\alpha_1}{\beta_0}, \frac{\alpha_1}{\beta_1} \end{aligned}$$

which is run on input $N = \alpha_1 = 2$ to force the program to start in the initial state.

Proposition 3.8. The word computed by F_{BIN} is BIN. ■

4. Productivity for Erasing PerDOL Systems

We show that the problem of deciding productivity of erasing PerDOL systems is undecidable. The idea is to encode a given Fractran program F as a PerDOL 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 $\frac{n_1}{d}, \dots, \frac{n_k}{d}$. Of course, every program can be brought into this form by taking d as the least common denominator of the fractions.

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

$$\Gamma = \{s, \square, \mathbf{a}, \mathbf{A}, \mathbf{b}, \mathbf{B}\}$$

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

$$\begin{aligned} h_i(s) &= s \square^{d-1} \mathbf{a} \mathbf{a} \mathbf{b} \square^{d-1} \\ h_i(\square) &= \varepsilon \\ h_i(\mathbf{a}) &= \begin{cases} \mathbf{A} \square^{d-1} & \text{if } i = d-1 \\ \varepsilon & \text{otherwise} \end{cases} \\ h_i(\mathbf{b}) &= \mathbf{B} \square^{d-1-i} \\ h_i(\mathbf{A}) &= \begin{cases} \mathbf{a}^{n_{\psi(i)}} & \text{if } \psi(i) \text{ is defined} \\ \varepsilon & \text{otherwise} \end{cases} \\ h_i(\mathbf{B}) &= \begin{cases} \mathbf{a}^{i \cdot \frac{n_{\psi(i)}}{d}} \mathbf{b} \square^{d-1} & \text{if } \psi(i) \text{ is defined} \\ \varepsilon & \text{otherwise} \end{cases} \end{aligned}$$

Before we show that productivity of the PerDOL 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 $N, d, q, r \in \omega$ such that $N = qd + r$, and F a Fractran program. Then $\psi_F(N) = \psi_F(r)$. If moreover $b \in \omega$ 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 $N_0, N_1, N_2, \dots$. Let $q_i \in \omega$ 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 $H^{n+1}(s) = H^n(s) x_n$. Then $H^\omega(s) = s x_0 x_1 x_2 \cdots$. We will display $H^\omega(s)$ in separate lines each corresponding to an x_n . The computation of the word $H^\omega(s)$ 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, $x_{2n} \in \{\square, \mathbf{a}, \mathbf{b}\}^*$ and $x_{2n+1} \in \{\square, \mathbf{A}, \mathbf{B}\}^*$, 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 $\square$ is used to shift the morphism index of subsequent letters.

In every even line x_{2i}

- (i) there is precisely one block of $\mathbf{a}$ ’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) $\mathbf{b}$ is a special marker for the end of a block of $\mathbf{a}$ ’s, so positioned at morphism index r_i , the remainder of dividing N_i by d .

- (iii) the number of $\mathbf{A}$'s corresponds to the quotient q_i , and every occurrence of $\mathbf{A}$ is positioned at morphism index r_i ;
- (iv) $\mathbf{B}$ (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 $\langle \frac{9}{2}, \frac{5}{3} \rangle$, or equivalently $F = \frac{27}{6}, \frac{10}{6}$ and its finite run 2, 9, 15, 25. Following Definition 4.1 we construct the PerDOL system $\mathcal{H}_F = \langle \Gamma, H, s \rangle$ with $H = \langle h_0, \dots, h_5 \rangle$ and

$$\begin{array}{ll}
h_i(\mathbf{s}) = \mathbf{s} \square^5 \mathbf{a} \mathbf{a} \mathbf{b} \square^5 & \\
h_i(\square) = \varepsilon & h_i(\mathbf{b}) = \mathbf{B} \square^{5-i} \\
h_0(\mathbf{a}) = \dots = h_4(\mathbf{a}) = \varepsilon & h_0(\mathbf{B}) = \mathbf{b} \square^5 \\
h_5(\mathbf{a}) = \mathbf{A} \square^5 & h_2(\mathbf{B}) = \mathbf{a}^9 \mathbf{b} \square^5 \\
h_0(\mathbf{A}) = h_2(\mathbf{A}) = h_4(\mathbf{A}) = \mathbf{a}^{27} & h_4(\mathbf{B}) = \mathbf{a}^{18} \mathbf{b} \square^5 \\
h_3(\mathbf{A}) = \mathbf{a}^{10} & h_3(\mathbf{B}) = \mathbf{a}^5 \mathbf{b} \square^5 \\
h_1(\mathbf{A}) = h_5(\mathbf{A}) = \varepsilon & h_1(\mathbf{B}) = h_5(\mathbf{B}) = \varepsilon
\end{array}$$

for $i \in \Sigma_6$. Then $H^\omega(\mathbf{s})$ 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 $H^\omega(\mathbf{s}) = \mathbf{s}x_0x_1 \cdots$ 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 $\mathbf{s}$ under H_0).

$$\begin{aligned}
x_0 &= \square^5 \text{ a a b } \square^5 \\
x_1 &= \text{B } \square^3 \\
x_2 &= \text{a a a a a a a a b } \square^5 \\
x_3 &= \text{A } \square^5 \text{ B } \square^2 \\
x_4 &= \text{a a a a a a a a a a a a a a b } \square^5 \\
x_5 &= \text{A } \square^5 \text{ A } \square^5 \text{ B } \square^2 \\
x_6 &= \text{a b } \square^5 \\
x_7 &= \text{A } \square^5 \text{ A } \square^5 \text{ A } \square^5 \text{ A } \square^5 \text{ B } \square^4 \\
x_8 &= \varepsilon
\end{aligned}$$

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

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

$$H(\mathbf{a}^N \mathbf{b} \mathbf{X}) = (\mathbf{A} \mathbf{X})^q \mathbf{B} \square^{d-1-r} \quad (3)$$

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

$$H_r((\mathbf{A} \mathbf{X})^q \mathbf{B} \square^{d-1-r}) = \mathbf{a}^{F(N)} \mathbf{b} \mathbf{X} \quad (4)$$

of length $F(N) + d$.

Proof. Equation (3) follows immediately by induction on q . To see that (4) 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

$$\begin{aligned} H_r((\mathbf{A} \times)^q \mathbf{B} \square^{d-1-r}) &= (\mathbf{a}^{n_{\psi(r)}})^q H_r(\mathbf{B} \square^{d-1-r}) \\ &= (\mathbf{a}^{n_{\psi(r)}})^q \mathbf{a}^{r \cdot \frac{n_{\psi(r)}}{d}} \mathbf{b} \times \end{aligned}$$

and hence $F(N) = N \cdot \frac{n_{\psi(N)}}{d} = q \cdot n_{\psi(N)} + r \cdot \frac{n_{\psi(N)}}{d}$.

Lemma 4.5. *For all Fractran programs F , the PerDOL 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 $N_0, N_1, N_2, \dots$ be the finite or infinite run of F on 2, i.e., $N_i = F^i(2)$, and let $t \in \omega \cup \{\infty\}$ denote its length. For all i with $0 \leq i < t$, let $q_i \in \omega$ and $r_i \in \Sigma_d$ be such that $N_i = q_id + r_i$.

We define $x_n \in \Sigma^*$ and $z_n \in \Sigma_d$ for all $n \geq 0$, as follows. Let $\mathbf{X} = \square^{d-1}$, $x_0 = \mathbf{a} \mathbf{a} \mathbf{b} \mathbf{X}$, $z_0 = 0$, and, for $n \geq 1$, let x_n and z_n be such that $H^{n+1}(\mathbf{s}) = H^n(\mathbf{s})x_n$ and $z_n \equiv |H^n(\mathbf{s})| \pmod{d}$. Then $H^\omega(\mathbf{s}) = \mathbf{s} \mathbf{X} x_0 x_1 x_2 \cdots$, and the factor x_n is at morphism index z_n . With Lemma 2.2 we have

$$x_n = H_{z_{n-1}}(x_{n-1}) \quad z_n \equiv z_{n-1} + |x_{n-1}| \pmod{d} \quad (5)$$

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

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

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 $x_n = H_{z_{n-1}}(x_{n-1}) = H_{r_{i-1}}((\mathbf{A} \mathbf{X})^{q_{i-1}} \mathbf{B} \square^{d-1-r_{i-1}}) = \mathbf{a}^{N_i} \mathbf{b} \mathbf{X}$, 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 (5), the induction hypothesis and Lemma 4.4. If $n = 2i + 1 < 2t$ for some $i < t$, then $x_n = H_{z_{n-1}}(x_{n-1}) = H_0(\mathbf{a}^{N_i} \mathbf{b} \mathbf{X}) = (\mathbf{A} \mathbf{X})^{q_i} \mathbf{B} \square^{d-1-r_i}$, and $z_n \equiv z_{n-1} + |x_{n-1}| \equiv 0 + N_i + d \equiv r_i \pmod{p}$, again by (5), the induction hypothesis and Lemma 4.4. Finally, if $n = 2t$, then $x_n = H_{z_{n-1}}(x_{n-1}) = H_{r_{t-1}}((\mathbf{A} \mathbf{X})^{q_{t-1}} \mathbf{B} \square^{d-1-r_{t-1}}) = \varepsilon$, 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 3.2, deciding productivity of PerDOL systems is undecidable.

Theorem 4.6. *The following problem is Π_1^0 -complete: Given a PerDOL system $\mathcal{H}$, is $\mathcal{H}$ productive?* ■

5. Turing Completeness of Non-Erasing PerDOL 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 $\mathbf{l}, \mathbf{r} \in \Gamma \setminus \Sigma$, and let $w \in \Sigma^\omega$ and $u \in \Gamma^\omega$ be infinite words. We say w is *prefix embedded* in u if the following three conditions are satisfied:

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

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

The 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 **0** or **1**, cf. Definition 3.4. 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 $F = \frac{n_1}{d}, \dots, \frac{n_k}{d}$ be a Fractran program such that (without loss of generality) the common denominator d is divisible by 3 and 5. Define the PerDOL system $\mathcal{H}_F = \langle \Gamma, H, s \rangle$ with

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

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

$$\begin{aligned} h_i(s) &= s \square^{d-1} \mathbf{a} \mathbf{a} \mathbf{b} \square^{d-1} \mathbf{z} \square^{d-2} \mathbf{l} \mathbf{r} \square^{d-1} \mathbf{e} \\ h_i(\mathbf{a}) &= \begin{cases} \mathbf{A} \square^{d-1} & \text{if } i = d-1, \\ \square^d & \text{otherwise.} \end{cases} \\ h_i(\mathbf{e}) &= \mathbf{E} \circ^{d-i} \quad h_i(\circ) = \square^d \quad h_i(\square) = \square \\ h_i(\mathbf{A}) &= \begin{cases} \mathbf{a}^{n_{\psi(i)}} \square & \text{if } \psi(i) \text{ is defined,} \\ \mathbf{Q} & \text{otherwise.} \end{cases} \\ h_i(\mathbf{B}) &= \begin{cases} \mathbf{a}^{i \cdot \frac{n_{\psi(i)}}{d}} \mathbf{b} & \text{if } \psi(i) \text{ is defined,} \\ \mathbf{Q} & \text{otherwise.} \end{cases} \\ h_i(\mathbf{Z}) &= \begin{cases} \mathbf{z} \square^{d-1} & \text{if } 3 \mid i \text{ or } 5 \mid i, \\ \mathbf{z} & \text{otherwise.} \end{cases} \\ h_i(\mathbf{R}) &= \begin{cases} \mathbf{0} \mathbf{r} & \text{if } 3 \mid i, \\ \mathbf{1} \mathbf{r} & \text{if } 5 \mid i, \\ \mathbf{r} & \text{otherwise.} \end{cases} \\ h_i(\mathbf{b}) &= \mathbf{B} \quad h_i(\mathbf{z}) = \mathbf{Z} \quad h_i(\mathbf{l}) = \mathbf{L} \quad h_i(\mathbf{r}) = \mathbf{R} \\ h_i(\mathbf{L}) &= \mathbf{l} \quad h_i(\mathbf{E}) = \mathbf{e} \quad h_i(\mathbf{Q}) = \mathbf{Q} \\ h_i(\mathbf{0}) &= \mathbf{0} \quad h_i(\mathbf{1}) = \mathbf{1} \end{aligned}$$

Remark 5.4. Let F and H be as in Definition 5.3. It can be shown that the symbol **Q** occurs in the word $H^\omega(s)$ 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 $H^\omega(s)$. However, we prove this differently, namely by applying Theorem 5.8 and using the fact that for non-terminating Fractran programs it is undecidable whether digit **1** occurs in the sequence computed by the program. See Theorem 5.12.

Example 5.5. 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

$$01100001100000001100000000001100000000000011 \dots,$$

that is, $0^1 11 \ 0^4 11 \ 0^7 11 \ 0^{10} 11 \dots = \prod_{i=0}^{\infty} 0^{3 \cdot i+1} 11$. 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 PerDOL 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 $\square^{210}$ as they do not change the morphism index of other letters, and let $X = \square^{209}$.

$$\begin{aligned} x_0 &= X \mathbf{a}^2 \mathbf{b} X \mathbf{z} \square^{208} \mathbf{l} \mathbf{r} X \mathbf{e} \\ x_1 &= X \mathbf{B} X \mathbf{Z} \square^{208} \mathbf{L} \mathbf{R} X \mathbf{E} \square^{208} \\ x_2 &= X \mathbf{a}^3 \mathbf{b} X \mathbf{z} \square^{208} \mathbf{l} \mathbf{r} X \mathbf{e} \\ x_3 &= X \mathbf{B} X \mathbf{Z} \square^{208} \mathbf{L} \mathbf{R} X \mathbf{E} \square^{207} \\ x_4 &= X \mathbf{a}^5 \mathbf{b} X \mathbf{z} \square^{207} \mathbf{l} \mathbf{0} \mathbf{r} X \mathbf{e} \\ x_5 &= X \mathbf{B} X \mathbf{Z} \square^{207} \mathbf{L} \mathbf{0} \mathbf{R} X \mathbf{E} \square^{205} \\ x_6 &= X \mathbf{a}^{10} \mathbf{b} X \mathbf{z} \square^{206} \mathbf{l} \mathbf{0} \mathbf{1} \mathbf{r} X \mathbf{e} \\ x_7 &= X \mathbf{B} X \mathbf{Z} \square^{206} \mathbf{L} \mathbf{0} \mathbf{1} \mathbf{R} X \mathbf{E} \square^{200} \\ x_8 &= X \mathbf{a}^{21} \mathbf{b} X \mathbf{z} \square^{205} \mathbf{l} \mathbf{0} \mathbf{1} \mathbf{1} \mathbf{r} X \mathbf{e} \end{aligned}$$

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** is the remainder r_i , and consequently **R** produces the prefix **0 1 1** 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:

$$\begin{aligned} x_{36} &= X \mathbf{a}^{250} \mathbf{b} X \mathbf{z} \square^{191} \mathbf{l} \mathbf{01100001100000001} \mathbf{r} X \mathbf{e} \\ x_{37} &= X \mathbf{A} X \mathbf{B} X \mathbf{Z} \square^{191} \mathbf{L} \mathbf{01100001100000001} \mathbf{R} X \mathbf{E} \square^{170} \\ x_{38} &= X \mathbf{a}^{525} \mathbf{b} X \mathbf{z} \square^{190} \mathbf{l} \mathbf{01100001100000001} \mathbf{1} \mathbf{r} X \mathbf{e} \\ x_{39} &= X (\mathbf{A} X)^2 \mathbf{B} X \mathbf{Z} \square^{190} \mathbf{L} \mathbf{01100001100000001} \mathbf{1} \mathbf{R} X \mathbf{E} \square^{105} \\ x_{40} &= X \mathbf{a}^{150} \mathbf{b} X \mathbf{z} \square^{189} \mathbf{l} \mathbf{01100001100000001} \mathbf{1} \mathbf{0} \mathbf{r} X \mathbf{e} \end{aligned}$$

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 hence 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 divisible by 5 but not 3, and **Z** produces an X-block ($X = \square^{209}$) 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 5.8. Let $F = \frac{n_1}{d}, \dots, \frac{n_k}{d}$ 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

$$\rightsquigarrow = \{ \langle u \square^d v, uv \rangle \mid u, v \in \Sigma^* \} \quad \sim = (\rightsquigarrow \cup \rightsquigarrow^*)^*.$$

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 $H^\omega(s)$ reasoning modulo $\sim$. Below, we write $\square^n$ with $n < 0$ to denote the block $\square^m$ with $m \in \Sigma_d$ and $n \equiv m \pmod{d}$. For $N \geq 1$ we define

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

Lemma 5.6. *Let $F = \frac{n_1}{d}, \dots, \frac{n_k}{d}$, and $N \geq 1$. Let $\kappa = \kappa(N)$ and $X = \square^{d-1}$. Let $q \in \omega$ and $r \in \Sigma_d$ be such that $N = qd + r$, and let $v \in \{0, 1\}^*$. Then we have*

$$\begin{aligned} H_1(X \mathbf{a}^N \mathbf{b} X \mathbf{z} \square^{d-2-|v|} \mathbf{l} v \mathbf{r} X \mathbf{e}) \\ = X (\mathbf{A} X)^q \mathbf{B} X \mathbf{Z} \square^{d-2-|v|} \mathbf{L} v \mathbf{R} X \mathbf{E} \circ^{d-r} \end{aligned} \quad (6)$$

of length equivalent to $-r$ modulo d .

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

$$\begin{aligned} H_{r+1}(X (\mathbf{A} X)^q \mathbf{B} X \mathbf{Z} \square^{d-2-|v|} \mathbf{L} v \mathbf{R} X \mathbf{E} \circ^{d-r}) \\ \sim X \mathbf{a}^{F(N)} \mathbf{b} X \mathbf{z} \square^{d-2-|v|} \mathbf{l} v \mathbf{r} X \mathbf{e} \end{aligned} \quad (7)$$

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

Proof. (6) is immediate: For words $x, y \in \Gamma^*$ we have

$$\begin{aligned} H_1(Xx) &= XH_0(x) \\ H_0(\mathbf{a}^N y) &= X(\mathbf{A} X)^q H_0(\mathbf{a}^r y) \\ &= X(\mathbf{A} X)^q H_r(y). \end{aligned}$$

Also we have

$$H_r(\mathbf{b} X \mathbf{z} \square^{d-2-|v|} \mathbf{l} v \mathbf{r} X \mathbf{e}) = \mathbf{B} X \mathbf{Z} \square^{d-2-|v|} \mathbf{L} v \circ^{d-r}.$$

To show (7), let $F(N)$ be defined. Then $\psi(N) = \psi(r)$ is also defined (Lemma 4.2). Hence we get, for $w \in \Gamma^*$

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

Now let $w = \mathbf{Z} \square^{d-2-|v|} \mathbf{L} v \mathbf{R} X \mathbf{E} \circ^{d-r}$. In order to compute $H_r(w)$ we 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

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

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

Lemma 5.7. *Let F be a Fractran program computing an infinite word w . Then the PerDOL system $\mathcal{H}_F$ from Definition 5.3 generates a word that prefix embeds w (see Definition 5.1).*

Proof. It suffices to consider the alphabet $\{0, 1\}$; the construction immediately generalizes to arbitrary finite alphabets. Let $F = \frac{n_1}{d}, \dots, \frac{n_k}{d}$ be a Fractran program, computing $w \in \{0, 1\}^\omega$, that is, $w = W_F$ with W_F as defined in Definition 3.4. Let $N_0, N_1, N_2, \dots$ be the infinite run of F starting on $N_0 = 2$ (so with infinitely many N_i divisible by 3 or 5). Let $q_i \in \omega$ and $r_i \in \Sigma_d$ be such that $N_i = q_i d + r_i$. Let $\mathcal{H}_F = \langle \Gamma, H, s \rangle$ be the PerDOL system defined in Definition 5.3.

We show that $u = H^\omega(s)$ satisfies the conditions (i), (ii), and (iii) of Definition 5.1, by characterizing the contribution of every iteration of H . For every $i \in \omega$ we let $v_i \in \Sigma^*$ be defined by $v_0 = \varepsilon$ and $v_{i+1} = v_i \kappa(N_i)$ so that $w = \lim_{i \rightarrow \infty} v_i$. For all $n \geq 0$ let $x_n \in \Gamma^*$ and $z_n \in \Sigma_d$ be such that $H^{n+1}(s) = H^n(s) x_n$ and $z_n \equiv |H^n(s)| \pmod{d}$. Then $H^\omega(s) = s x_0 x_1 \dots$, and

$$x_n = H_{z_{n-1}}(x_{n-1}) \quad z_n \equiv z_{n-1} + |x_{n-1}| \pmod{d} \quad (8)$$

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

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

by induction on n . For the base case, we see $z_0 = |s| = 1$ and $x_0 = X \mathbf{a} \mathbf{a} \mathbf{b} X \mathbf{z} \square^{d-2} \mathbf{l} \mathbf{r} X \mathbf{e}$, as required. So let $n \geq 1$. If $n = 2i$ for some $i \geq 1$, it follows from (8), Lemma 5.6 and the induction hypothesis that

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

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

$$\begin{aligned} x_n &= H_{z_{n-1}}(x_{n-1}) \sim H_1(X \mathbf{a}^{N_i} \mathbf{b} X \mathbf{z} Y_i \mathbf{l} v_i \mathbf{r} X \mathbf{e}) \\ &\sim X \mathbf{a}^{N_{i+1}} \mathbf{b} X \mathbf{z} Y_{i+1} \mathbf{l} v_{i+1} \mathbf{r} X \mathbf{e} \end{aligned}$$

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 $u = H^\omega(s)$, it is now easy to verify that u satisfies conditions (i), (ii), and (iii) of Definition 5.1, taking into account that $\square$ does not occur in any factor $\mathbf{l} v \mathbf{r}$ of u with $v \in (\Gamma \setminus \{\mathbf{r}\})^*$, by the definition of the morphisms. ■

We are ready to collect our main results.

Theorem 5.8. *Every computable infinite word can be prefix embedded in a PerDOL word (see Definition 5.1).*

Proof. Let w be an infinite computable word. Then, by Proposition 3.6, w is computed by some Fractran program. By Lemma 5.7 the claim follows. ■

Lemma 5.9. *Let F be a Fractran program computing an infinite word $w \in \{0, 1\}^\omega$, and let $\mathcal{H}_F$ be the PerDOL system given in Definition 5.3. Define $\mathcal{H}'_F$ to be the PerDOL system obtained from $\mathcal{H}_F$ by replacing in $\mathcal{H}_F$ the rules $h_i(\mathbf{0}) = \mathbf{0}$ and $h_i(\mathbf{1}) = \mathbf{1}$ by $h_i(\mathbf{0}) = \square$ and $h_i(\mathbf{1}) = \square$, for all $i \in \Sigma_d$. Then w is sparsely embedded in the PerDOL word generated by $\mathcal{H}'_F$.*

Proof. By an easy adaptation of the proof of Lemma 5.7, noting that every output is produced precisely once and is replaced by the symbol $\square$ in the next iteration. ■

Theorem 5.10. *Every computable infinite word over a finite alphabet can be sparsely embedded in a PerDOL word (see Definition 5.2).*

Proof. Analogous to the proof of Theorem 5.8; replace Lemma 5.7 by Lemma 5.9, and noting that this construction can be generalized to any finite alphabet. ■

We collect some further immediate consequences of Theorem 5.8. First of all, we have solved the open problem [23] on the existence of PerDOL words that have exponential subword complexity.

Theorem 5.11. *There is a PerDOL word u with subword complexity $p_u(n) \geq 2^n$, for all $n \in \omega$.*

Proof. Let $F = F_{\text{BIN}}$ be the Fractran program defined in Section 3, computing the word $W_F = \text{BIN}$ (Proposition 3.8). Furthermore, let $\mathcal{H}_F = \langle \Gamma, H, s \rangle$ be the PerDOL system of Definition 5.3. Then, by Lemma 5.7, $u = H^\omega(s)$ is the word we are looking for. ■

Lemma 5.7 also allows us to give a negative answer to [24, Problem 29 (3)].

Theorem 5.12. *The following problems are undecidable: Given a PerDOL system $\mathcal{H} = \langle \Gamma, H, s \rangle$, and a letter $b \in \Gamma$,*

- (i) *does b occur in $H^\omega(s)$?*
- (ii) *does b occur infinitely many times in $H^\omega(s)$?*

Proof. We show that the following problem is undecidable: given a Fractran program F computing an infinite word w over the alphabet $\{0, 1\}$, does the letter 1 occur in w ? This suffices since by Lemma 5.7, if u is the infinite word generated by $\mathcal{H}_F$, then the letter 1 occurs in u if and only if 1 occurs infinitely often in u if and only if 1 occurs in w .

We use the input-2 halting problem for Fractran programs which is Σ_1^0 -complete by Proposition 3.2. Let F be an arbitrary Fractran program. By Remark 3.3 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 5.12 it follows immediately that the first-order (and monadic second-order) theory of PerDOL words is undecidable, answering [24, Problem 28]; see [24] 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 [6].

Corollary 5.13. *The first-order theory of PerDOL words is undecidable.* ■

Also immediate from Theorem 5.12 is the undecidability of equivalence of PerDOL systems (equality of the limit words they generate). We note that equivalence of DOL systems is decidable [10], whereas that of CDOL words is an open problem.

Corollary 5.14. *Equality of PerDOL words (given by their PerDOL systems) is undecidable.*

Proof. We reduce problem (i) stated in Theorem 5.12 to equivalence of PerDOL systems, as follows. Let $\mathcal{H} = \langle \Sigma, H, s \rangle$ be a PerDOL 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 5.12 this is undecidable. ■

6. Finite State Transducers

We show that there are PerDOL words that can be transformed to any computable word using finite state transducers.

Definition 6.1. A *finite state transducer* $T = \langle \Sigma, \Gamma, Q, q_0, \delta, \lambda \rangle$ consists of a finite input alphabet Σ , a finite output alphabet Γ , a finite set of states Q , an initial state $q_0 \in Q$, a transition function $\delta : Q \times \Sigma \rightarrow Q$, and an output function $\lambda : Q \times \Sigma \rightarrow \Gamma^*$.

The output function λ is extended to the set of infinite words by

$$\lambda(q, aw) := \lambda(q, a) \cdot \lambda(\delta(q, a), w),$$

for all $q \in Q$, $a \in \Sigma$, and $w \in \Sigma^\omega$.

For $u \in \Sigma^\omega$, $v \in \Gamma^\omega$ we write $u \triangleright v$ if $v = \lambda(q_0, u)$ for some finite state transducer $\langle \Sigma, \Gamma, Q, q_0, \delta, \lambda \rangle$.

The *closure* of a set X of infinite words (over finite alphabets) *under finite state transduction* is the smallest set Y such that $X \subseteq Y$, and for all infinite words y, y' with $y \in Y$ and $y \triangleright y'$, we have $y' \in Y$.

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 again a morphic (or finite) word.

From Theorem 5.10 it follows that every computable word can be obtained by a finite state transduction of a PerDOL word, since erasing letters is a finite state transduction. Moreover, as there are computable words that are not PerDOL words [11], we know that the class of PerDOL words is not closed under finite state transduction, thus establishing a negative answer to Problem 29 (1) of [24].

Corollary 6.2.

- (i) *The set of PerDOL words is not closed under finite state transductions.*
- (ii) *The closure of the set of PerDOL words under finite state transduction consists of all computable words over finite alphabets.* ■

So every computable word can be obtained from some PerDOL word. Even stronger, there exist PerDOL words that can be transduced to any computable word, as we show next.

Let $\{\sigma_i \in A^\omega\}_{i \in \omega}$ be a countable set of infinite words over (a subset of) a finite alphabet A . Then we define the word $\text{zip}(\{\sigma_i\}_{i \in \omega}) \in A^\omega$ by

$$\text{zip}(\{\sigma_i\}_{i \in \omega})(2^{j+1}n + 2^j - 1) = \sigma_j(n),$$

for all $j, n \in \omega$. This construction can be illustrated as follows:

0	1	2	3	4	5	6	7	8	9	10	11	...
$\sigma_0(0)$	$\sigma_1(0)$	$\sigma_0(1)$	$\sigma_2(0)$	$\sigma_0(2)$	$\sigma_1(1)$	$\sigma_0(3)$	$\sigma_3(0)$	$\sigma_0(4)$	$\sigma_1(2)$	$\sigma_0(5)$	$\sigma_2(1)$	...

Theorem 6.3. *There exist a computable word $w \in \{0, 1\}^\omega$ such that the closure of $\{w\}$ under finite state transduction is the set of all computable words over finite alphabets.*

Proof. Let $M_0, M_1, \dots$ be a computable enumeration of all Turing machines over $\{0, 1\}$. For a Turing machine M_i we define an infinite word σ_i over $\{0, 1, \star\}$ as follows: $\sigma_i = N(0)$ where $N : \omega \rightarrow \{0, 1, \star\}^\omega$ is defined by $N(n) = \star^s b N(n+1)$ if

M_i terminates on n with output b in s steps, and $N(n) = \star^\omega$ if M_i does not terminate on input n . The word σ_i is computable because a universal Turing machine can simulate M_i and compute $\sigma_i(0), \sigma_i(1), \sigma_i(2), \dots$ step by step; for every simulation step one output is produced, either $\star$ or an element in $\{0, 1\}$.

Now define the word $w = \text{zip}(\{\sigma_i\}_{i \in \omega})$. A universal Turing machine can, on the input of an index m compute the unique j and n such that $m = 2^{j+1}n + 2^j - 1$ and then output $w(m) = \sigma_j(n)$. Hence w is computable.

Finally, it is clear that $w \triangleright \sigma_i$ for all $i \in \omega$. Moreover, every computable word u can be obtained from some σ_i by erasing all occurrences of the symbol $\star$. Hence $w \triangleright \sigma_i \triangleright u$. ■

Remark 6.4. In the proof of Theorem 6.3 we enumerate all Turing machines (not only the total ones) and deal with partiality by the use of the $\star$ symbol (“wait for output”). We note that the word $\text{zip}(\{\tau_i\}_{i \in \omega})$, with $\{\tau_i\}_{i \in \omega}$ the set of all computable words, is not computable itself, for otherwise the ‘complement’ of the diagonal $1 - \tau_0(0), 1 - \tau_1(1), 1 - \tau_2(2), \dots$ would also be computable.

The reducibility relation $\triangleright$ induces a partial order on ‘degrees’ of infinite words, where a degree is an equivalence class with respect to the equivalence relation $\triangleleft \cap \triangleright$, see [18]. By Theorem 6.3 and Corollary 6.2 we obtain:

Corollary 6.5. *There is a PerDOL word in the supremum degree (the least upper bound with respect to $\triangleright$) of the computable words.* ■

7. Discussion

PerDOL systems resulting from encoding Fractran programs can be quite large. For example, the system obtained from the binary counter F_{BIN} on page 5 (which computes an infinite word with exponential subword complexity) consists of

536393214598471230¹ morphisms,

this number being the least common denominator of F_{BIN} . In our report [17] we present a direct solution, namely a PerDOL system with 16 morphisms simulating such a counter.

So for one morphism the maximal subword complexity is at most quadratic [13], and for 16 morphisms it is exponential [17]. For other numbers of morphisms we only know lower bounds on the maximal subword complexity, exhibited by words generated by locally uniform PerDOL systems [7, 23]. One might ask

- (i) What is the maximal subword complexity for PerDOL systems with p morphisms, for $p \neq 1$, and $p \neq 16$? (We conjecture that for prime numbers p the subword complexity cannot be exponential.)
- (ii) What is the maximal subword complexity for locally uniform PerDOL systems (see Definition 2.7)?

References

- [1] J.-P. Allouche. Sur la complexité des suites infinies. *Journées Monitoises*, 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.
- [4] J.-M. Autebert and J. Gabarró. Iterated GSMs and Co-CFL. *Acta Informatica*, 26(8):749–769, 1989.
- [5] J. Berstel. Mots sans carré et morphismes itérés. *Discrete Mathematics*, 29:235–244, 1980.
- [6] O. Carton and W. Thomas. The monadic theory of morphic infinite words and generalizations. *Information and Computation*, 176(1):51–65, 2002.
- [7] J. Cassaigne and J. Karhumäki. Toeplitz words, generalized periodicity and periodically iterated morphisms. *European Journal of Combinatorics*, 18(5):497–510, 1997.
- [8] J. H. Conway. Unpredictable Iterations. In *Proceedings of the 1972 Number Theory Conference*, pages 49–52. University of Colorado, 1972.
- [9] J. H. Conway. Fractran: A simple universal programming language for arithmetic. In *Open Problems in Communication and Computation*, pages 4–26. Springer, 1987.
- [10] K. Culik II and T. Harju. The ω -sequence problem for DOL systems is decidable. *Journal of the Association for Computing Machinery*, 31(2):282–298, 1984.
- [11] K. Culik II and J. Karhumäki. Iterative Devices Generating Infinite Words. In *Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science (STACS 1992)*, volume 577 of *Lecture Notes in Computer Science*, pages 531–543. Springer, 1992.
- [12] K. Culik II, J. Karhumäki, and A. Lepistö. Alternating iteration of morphisms and Kolakowski [sic] sequence. In *Lindenmayer Systems, Impacts on Theoretical Computer Science, Computer Graphics and Developmental Biology*, pages 93–106. Springer, 1992.
- [13] 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.
- [14] J. Endrullis, C. Grabmayer, and D. Hendriks. Data-Oblivious Stream Productivity. In *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2008)*, volume 5330 of *Lecture Notes in Computer Science*, pages 79–96. Springer, 2008.
- [15] J. Endrullis, C. Grabmayer, and D. Hendriks. Complexity of Fractran and Productivity. In *Proceedings of the 22nd International Conference on Automated Deduction (CADE 2009)*, volume 5663 of *Lecture Notes in Computer Science*, pages 371–387. Springer, 2009.
- [16] J. Endrullis, C. Grabmayer, D. Hendriks, A. Ishihara, and J. W. Klop. Productivity of stream definitions. *Theoretical Computer Science*, 411:765–782, 2010.
- [17] J. Endrullis and D. Hendriks. On periodically iterated morphisms. *CoRR*, abs/1207.2336, 2012.
- [18] 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.
- [19] S. Ferenczi. Complexity of sequences and dynamical systems. *Discrete Mathematics*, 206(1-3):145–154, 1999.
- [20] C. Grabmayer, J. Endrullis, D. Hendriks, J. W. Klop, and L. S. Moss. Automatic Sequences and Zip-Specifications. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science (LICS 2012)*, pages 335–344. IEEE, 2012.
- [21] K. Jacobs and M. Keane. 0-1-sequences of Toeplitz type. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 13(2):123–131, 1969.
- [22] W. Kolakoski. Self generating runs. *The American Mathematical Monthly*, 72, 1965. Problem 5304.
- [23] A. Lepistö. On the Power of Periodic Iteration of Morphisms. In *Proceedings of the 20th International Colloquium on Automata, Languages and Programming (ICALP 1993)*, volume 700 of *Lecture Notes in Computer Science*, pages 496–506. Springer, 1993.
- [24] A. A. Muchnik, Y. L. Pritykin, and A. L. Semenov. Sequences close to periodic. *Russian Mathematical Surveys*, 64(5):805–871, 2009.
- [25] R. Oldenburger. Exponent trajectories in dynamical systems. *Transactions of the American Mathematical Society*, 46:453–466, 1939.
- [26] C.-H. L. Ong. On Model-Checking Trees Generated by Higher-Order Recursion Schemes. In *Proceedings of the 21th Annual IEEE Symposium on Logic in Computer Science (LICS 2006)*, pages 81–90. IEEE, 2006.

¹This is the least common denominator of F_{BIN} .