Discrete Mathematics 342 (2019) 226–232

Contents lists available at ScienceDirect

Discrete Mathematics

journal homepage: www.elsevier.com/locate/disc

An efficient shift rule for the prefer-max De Bruijn sequence☆

Gal Amram [a], Yair Ashlagi [a], Amir Rubin [a], Yotam Svoray [a], Moshe Schwartz [b,*], Gera Weiss [a]

[a] Department of Computer Science, Ben-Gurion University of The Negev, Israel
[b] Department of Electrical and Computer Engineering, Ben-Gurion University of The Negev, Israel

ARTICLE INFO

Article history:
Received 14 September 2017
Received in revised form 19 July 2018
Accepted 19 September 2018
Available online xxxx

Keywords:
De Bruijn sequence
Ford sequence
Prefer-max sequence
Shift rule

ABSTRACT

A shift rule for the prefer-max De Bruijn sequence is formulated, for all sequence orders, and over any finite alphabet. An efficient algorithm for this shift rule is presented, which has linear (in the sequence order) time and memory complexity.

1. Introduction

A k-ary De Bruijn sequence of order n (denoted (n, k)-DB), is a word $\langle \nu_i \rangle_{i=0}^{k^n-1} \triangleq \nu_0 \nu_1 \dots \nu_{k^n-1}$ over the alphabet $[k] \triangleq \{0, 1, \dots, k-1\}$, i.e., $\nu_i \in [k]$ for all $i \in \mathbb{Z}_{k^n}$, such that all n-subwords $\nu_i \nu_{i+1} \dots \nu_{i+n-1}$ are distinct (note that $i \in \mathbb{Z}_{k^n}$ means that indices are taken modulo k^n). Of the many (n, k)-DB sequences that exist, a particular sequence stands out, featuring in many past works. Consider first the binary case, $k = 2$, start the sequence with 0^n, and add bit by bit, always preferring to append a 1, unless it creates an n-word that has already been seen previously. After obtaining a sequence of length 2^n, move the 0^n prefix to the end of the sequence. The result is an $(n, 2)$-DB sequence dubbed the prefer-one sequence or Ford sequence [10,6]. By complementing all the bits, we obtain the prefer-min $(n, 2)$-DB sequence. In the non-binary case, we can replace the prefer-one rule by a prefer-max (assuming a lexicographical order of the alphabet), resulting in the lexicographically largest (n, k)-DB sequence, and symmetrically (by complementation), a prefer-min (n, k)-DB sequence which is the lexicographically smallest (n, k)-DB sequence.

The greedy bit-by-bit algorithm of [10] is certainly an inefficient way of generating the prefer-max sequence, running in $\Theta(nk^n)$ time (integer operations), and requiring $\Theta(k^n)$ memory. Several suggestions have been made since to improve the efficiency of the sequence construction. Fredricksen and Kessler [8], and Fredricksen and Maiorana [9] showed that the prefer-max sequence is in fact a concatenation of certain (Lyndon) words. While seemingly an inefficient way to generate the prefer-max sequence, a later careful analysis [11] has shown that this decomposition allows us to generate the sequence of length k^n in $O(k^n)$ time.

However, another equally important way of generating sequences is of interest, namely, by using a shift rule. It is well known that any (n, k)-DB sequence $\langle \nu_i \rangle_{i=0}^{k^n-1}$ can be generated by a feedback shift register (FSR) of order n, i.e., there exists a

☆ This work was supported in part by the Israeli Science Foundation (ISF) under grant no. 130/14, and grant no. 856/16.

* Corresponding author.
E-mail addresses: galamra@cs.bgu.ac.il (G. Amram), ashlagi@post.bgu.ac.il (Y. Ashlagi), amirrub@cs.bgu.ac.il (A. Rubin), ysavorai@post.bgu.ac.il (Y. Svoray), schwartz@ee.bgu.ac.il (M. Schwartz), geraw@cs.bgu.ac.il (G. Weiss).

https://doi.org/10.1016/j.disc.2018.09.024
0012-365X/

shift-rule function $f\colon [k]^n \to [k]$ such that $\nu_{i+n} = f(\nu_i, \nu_{i+1}, \ldots, \nu_{i+n-1})$ for all $i \in \mathbb{Z}_{k^n}$. Several efficiently computable shift rules for De Bruijn sequences are known, requiring $O(n)$ time and memory to generate the next letter in the sequence, given the preceding n letters (see [5], as well as [12] for a comprehensive list). We also mention the recent [3], which describes an efficient shift rule for the k-ary "grandmama" sequence (which is defined by a co-lexicographic order, compared with the lexicographic order of the prefer-max sequence). However, with a single exception, they only generate non prefer-max sequences, and the only exception [7], addresses only the generation of binary prefer-one sequences.

The main contribution of this paper is an efficient shift-rule function for the (n, k) prefer-max De Bruijn sequence, $k \geqslant 2$ (the case of $k = 1$ is trivial). The shift rule, given in Algorithm 1, is based on the decomposition of the prefer-max sequence found by [9], and operates in $O(n)$ time and memory. This closes a gap in the literature, since while efficient constructions for the entire prefer-max sequence are known, an efficient shift rule is only known in the binary case.

The paper is organized as follows. In Section 2 we provide the necessary notation used throughout the paper, and recall some relevant results. In Section 3 we provide a mathematical expression for the shift rule. We proceed in Section 4 to devise an efficient algorithm that implements the shift rule. We conclude in Section 5 with a short discussion of the results.

2. Preliminaries

Let us start by reviewing the necessary definitions and previous results, before presenting the new results. To avoid trivialities, we assume throughout the paper that $n, k \geqslant 2$. With our alphabet letters $[k]$ we associate a lexicographical order, $0 < 1 < \cdots < k-1$. This order is extended in the natural way to all finite words from $[k]^*$ by defining $x < y$ if either x is a prefix of y, or there exist (possibly empty) $u, v, w \in [k]^*$ and two letters $\sigma, \sigma' \in [k]$, $\sigma < \sigma'$, such that $x = u\sigma v$ and $y = u\sigma' w$.

Given a word $v \triangleq \sigma_0\sigma_1 \ldots \sigma_{n-1}$, with $\sigma_i \in [k]$, we define the *rotation* operator, $\mathcal{R}\colon [k]^n \to [k]^n$ as $\mathcal{R}(v) \triangleq \sigma_1\sigma_2 \ldots \sigma_{n-1}\sigma_0$. We say that two words $v, u \in [k]^n$ are *cyclically equivalent* if there exists $i \in \mathbb{Z}$ such that $v = \mathcal{R}^i(u)$. The equivalence classes under $\mathcal{R}$ are called *necklaces*. The number of necklaces, denoted by $Z_k(n)$, is known to be $Z_k(n) \triangleq \frac{1}{n}\sum_{d|n} \phi(d)k^{n/d}$, where ϕ is Euler's totient function (also the number of cycles in the pure cycling FSR, see [13]). Let $v \in [k]^n$ be a word. The cyclic order of v, denoted $o(v)$, is the smallest positive integer $o(v) \in \mathbb{N}$ such that $\mathcal{R}^{o(v)}(v) = v$ or, alternatively, it is the number of elements in the necklace containing v. If $o(v) = |v|$ we say that v is *primitive*. For any $v \in [k]^n$ there is a unique primitive word $w \in [k]^{o(v)}$ such that $v = w^{n/o(v)}$.

A primitive word that is lexicographically least in its necklace is called a *Lyndon word*. If $L \in [k]^+$ is a Lyndon word, we shall also find it useful to define L^m as an *expanded Lyndon word*[1], for all $m \in \mathbb{N}$. Additionally, we can arrange all the expanded Lyndon words of length n in increasing lexicographical order

$$L_0^{r_0} < L_1^{r_1} < \cdots < L_{Z_k(n)-1}^{r_{Z_k(n)-1}},$$

where $r_i \triangleq n/|L_i|$. The main result of [8,9] (rephrased to simplify the presentation) is that the prefer-min (n, k)-DB sequence is in fact the concatenation $L_0L_1 \cdots L_{Z_k(n)-1}$. We shall use this fact later on, and call it the FKM factorization. We also comment that by complementing all the letters via $\psi\colon [k] \to [k]$, $\psi(i) \triangleq k-1-i$, for all $i \in [k]$, the prefer-min (n, k)-DB sequence becomes the prefer-max (n, k)-DB sequence, and vice versa. We extend ψ to operate on words in the natural way, i.e., applying it to all letters of the word.

Example 1. Fix $n = 2$ and $k = 3$. We then have the following lexicographical order of expanded Lyndon words,

$$00 < 01 < 02 < 11 < 12 < 22$$

hence

$$L_0 = 0 \qquad L_1 = 01 \qquad L_2 = 02 \qquad L_3 = 1 \qquad L_4 = 12 \qquad L_5 = 2,$$

and indeed the prefer-min $(2, 3)$-DB sequence is $L_0L_1L_2L_3L_4L_5 = 001021122$. After complementing each letter we obtain $\psi(L_0L_1L_2L_3L_4L_5) = 221201100$, which is the prefer-max $(2, 3)$-DB sequence.

3. Shift-rule construction

In this section we state and prove our shift-rule construction. For ease of presentation, we work with the prefer-min sequence, while remembering that by simply complementing letters with ψ, this is equivalent to working with the prefer-max sequence.

We first require a definition that distinguishes another necklace member that is not necessarily the expanded Lyndon word $L_i^{r_i}$ defined above.

Definition 2. A word $v \in [k]^n$ is a necklace *head*, tested by the predicate $\mathsf{head}(v)$, if we can write v as $v = (k-1)^t w\sigma$, where $w \in [k]^{n-t-1}$, $\sigma \in [k-1]$ (i.e., $\sigma \neq k-1$), and $\mathcal{R}^t(v) = w\sigma(k-1)^t$ is an expanded Lyndon word.

[1] In some places, by abuse of notation, a lexicographically least representative of a necklace (which coincides with our definition of an expanded Lyndon word) is also called a necklace. We shall not do the same since we shall later require a different representative of a necklace, which might cause a confusion.

We briefly note that the necklace containing the single word $(k-1)^n$ does not formally have a necklace head, whereas all other necklaces have a unique necklace head. Additionally, by the above definition, if $(k-1)^t w\sigma$ is a necklace head, either $w = \varepsilon$ is empty, or it does not start with the letter $k-1$.

We now define our shift rule. Traditionally, a shift rule is a function that takes n consecutive letters in the sequence (i.e., the current state of an FSR generating the sequence) and its output is the next letter. However, we will find it more convenient to define a shift rule as providing the next state of the FSR. Specifically, let $\langle \nu_i \rangle_{i=0}^{k^n-1}$ be the prefer-min (n, k)-DB sequence. A shift rule for the sequence is a function $f : [k]^n \to [k]^n$ such that $f(\nu_i \nu_{i+1} \dots \nu_{i+n-1}) = \nu_{i+1}\nu_{i+2} \dots \nu_{i+n}$, for all $i \in \mathbb{Z}_{k^n}$.

Definition 3. Let $\mathsf{next}: [k]^n \to [k]^n$ be defined by

$$\mathsf{next}(\sigma w) \triangleq \begin{cases} w(\sigma+1) & \text{if } \sigma \neq k-1 \text{ and } \mathsf{head}(w\sigma), \\ w(\min S) & \text{if } \sigma = k-1 \text{ and } S \triangleq \left\{\sigma' \in [k-1] : \mathsf{head}(w\sigma')\right\} \neq \emptyset, \\ w\sigma & \text{otherwise,} \end{cases}$$

where $\sigma \in [k]$ and $w \in [k]^{n-1}$.

The main result of this section is the following theorem.

Theorem 4. next *is a shift rule for the prefer-min* (n, k)*-DB sequence.*

Before proceeding, we provide an example.

Example 5. Continuing our running example from Example 1, consider again the prefer-min $(2, 3)$-DB sequence 001021122. Take as an example the subword $\sigma w = 21$, i.e., $\sigma = 2$ and $w = 1$. In this case $\mathsf{next}(21)$ is computed using the second case of Definition 3, and $S = \{1\}$ since $\mathsf{head}(11)$ is true but $\mathsf{head}(10)$ is false. Thus, $\mathsf{next}(21) = 11$, which is consistent with the sequence.

In order to prove Theorem 4 we state a sequence of lemmas, building up to the main result.

Lemma 6. *A word $v \in [k]^+$ is an expanded Lyndon word if and only if $v \leqslant \mathcal{R}^i(v)$ for all $i \in \mathbb{Z}$ (i.e., it is lexicographically least in its necklace).*

Proof. Consider the (unique) decomposition $v = w^t$, where w is primitive. Note that $\mathcal{R}^i(v) = (\mathcal{R}^i(w))^t$. Thus, $v \leqslant \mathcal{R}^i(v)$ if and only if $w \leqslant \mathcal{R}^i(w)$, which holds for all $i \in \mathbb{Z}$ if and only if w is a Lyndon word. □

The first step we take is showing that increasing the rightmost letter that is not $k-1$ in an expanded Lyndon word maintains the expanded Lyndon property.

Lemma 7. *If $w\sigma(k-1)^t \in [k]^n$ is an expanded Lyndon word and $\sigma \in [k-1]$ then $w(\sigma+1)(k-1)^t$ is also an expanded Lyndon word.*

Proof. If $w(\sigma+1)(k-1)^t$ starts with $k-1$ then it is equal to $(k-1)^n$ and the claim follows. Otherwise, write $w(\sigma+1)(k-1)^t = xy$ and we shall prove that $xy \leqslant yx$. If $|y| \leqslant t$ the claim trivially holds. Otherwise, for some word v, $y = v(\sigma+1)(k-1)^t$ and $w = xv$. By assumption, $xv\sigma(k-1)^t \leqslant v\sigma(k-1)^t x$. Since $|v| \leqslant |xv|$, $xv(\sigma+1)(k-1)^t \leqslant v(\sigma+1)(k-1)^t x$ as well. □

We now turn, in the following lemmas, to consider connections between successive expanded Lyndon words, $L_i^{r_i}$ and $L_{i+1}^{r_{i+1}}$.

Lemma 8. *Let $L_i^{r_i} = w\sigma(k-1)^t \in [k]^n$ be the ith expanded Lyndon word in increasing lexicographical order where $\sigma \neq k-1$. Then, the $(i+1)$th expanded Lyndon word, $L_{i+1}^{r_{i+1}}$, is $w(\sigma+1)x$ where $x \in [k]^t$ is the lexicographically smallest word for which $w(\sigma+1)x$ is an expanded Lyndon word.*

Proof. By Lemma 7, $w(\sigma+1)(k-1)^t$ is an expanded Lyndon word, i.e., $w(\sigma+1)(k-1)^t = L_j^{r_j}$ for some $j > i$. It then follows that $L_{i+1}^{r_{i+1}}$ must be of the form $w(\sigma+1)x$ as claimed. □

The following lemma combines the shift-rule function, next, and the lexicographical order of expanded Lyndon words. We use the notation $\mathsf{next}^j(\cdot)$, $j \in \mathbb{N}$, to denote the composition of next with itself j times.

Lemma 9. $\mathsf{next}^{|L_i|}(L_i^{r_i}) = L_{i+1}^{r_{i+1}}$, *for all* $i \in [Z_k(n) - 2]$.

Proof. Since $i \in [Z_k(n) - 2]$, $L_i^{r_i}$ is not the lexicographically last expanded Lyndon word and not the one before it, i.e.,

$$L_i^{r_i} \neq (k-1)^n, L_i^{r_i} \neq (k-2)(k-1)^{n-1}. \tag{1}$$

We can therefore write $L_i = w\sigma(k-1)^t$, $\sigma \in [k-1]$, so $L_i^{r_i} = w\sigma(k-1)^t L_i^{r_i-1}$. Using these notations

$$\mathsf{next}^{|L_i|}(L_i^{r_i}) = \mathsf{next}^{|w|+1+t}(w\sigma(k-1)^t L_i^{r_i-1}).$$

Our proof proceeds by establishing the following three facts:

(a) $\mathsf{next}^{|w|}(w\sigma(k-1)^t L_i^{r_i-1}) = \sigma(k-1)^t L_i^{r_i-1} w$
(b) $\mathsf{next}(\sigma(k-1)^t L_i^{r_i-1} w) = (k-1)^t L_i^{r_i-1} w(\sigma+1)$
(c) $\mathsf{next}^t((k-1)^t L_i^{r_i-1} w(\sigma+1)) = L_i^{r_i-1} w(\sigma+1)x$, such that $x \in [k]^t$ is the lexicographically smallest word for which $L_i^{r_i-1} w(\sigma+1)x$ is an expanded Lyndon word.

Combining the three facts together, we get that $\mathsf{next}^{|L_i|}(L_i^{r_i}) = L_i^{r_i-1} w(\sigma+1)x$, and by Lemma 8, we get the desired.

We first prove step (a). We contend that this step's claim holds since in the first $|w|$ applications of next only the third case of the definition of next (cf. Definition 3) takes place. To prove this, we need to show that for any decomposition $w = w_1\tau w_2$, $\tau \in [k]$, $w_1, w_2 \in [k]^*$, we have

$$\mathsf{next}(\tau w_2 \sigma(k-1)^t L_i^{r_i-1} w_1) = w_2\sigma(k-1)^t L_i^{r_i-1} w_1\tau.$$

Hence, we need to show that $w_2\sigma(k-1)^t L_i^{r_i-1} w_1\tau$ is not a necklace head, and that if $\tau = k-1$ then there is no $\sigma' \in [k-1]$ such that $w_2\sigma(k-1)^t L_i^{r_i-1} w_1\sigma'$ is a necklace head.

For the first condition, assume to the contrary that the predicate $\mathsf{head}(w_2\sigma(k-1)^t L_i^{r_i-1} w_1\tau)$ is true. By definition, there should exist an integer $0 \leqslant m \leqslant |w_2|$ such that $w_2 = (k-1)^m w_3$ and

$$\mathcal{R}^m(w_2\sigma(k-1)^t L_i^{r_i-1} w_1\tau) = w_3\sigma(k-1)^t L_i^{r_i-1} w_1\tau(k-1)^m$$

is an expanded Lyndon word. However, we note that

$$w_3\sigma(k-1)^t L_i^{r_i-1} w_1\tau(k-1)^m = \mathcal{R}^{|w_1|+1+m}(L_i^{r_i}).$$

Since $0 < |w_1| + 1 + m < |L_i|$, this contradicts the cyclic order of $L_i^{r_i}$.

As for the second condition, where $\tau = k-1$, assume to the contrary that for some $\sigma' \in [k-1]$, the word $w_2\sigma(k-1)^t L_i^{r_i-1} w_1\sigma'$ is a necklace head. Again, there should exist an integer $0 \leqslant m \leqslant |w_2|$, such that $w_2 = (k-1)^m w_3$, and

$$\mathcal{R}^m(w_2\sigma(k-1)^t L_i^{r_i-1} w_1\sigma') = w_3\sigma(k-1)^t L_i^{r_i-1} w_1\sigma'(k-1)^m \tag{2}$$

is an expanded Lyndon word. Thus, on the right-hand side of (2), the rightmost letter that is not $k-1$, is σ'. By repeated applications of Lemma 7, we get that we can replace σ' by $k-1$ and still have an expanded Lyndon word, i.e.,

$$w_3\sigma(k-1)^t L_i^{r_i-1} w_1(k-1)^{m+1} = \mathcal{R}^{|w_1|+1+m}(L_i^{r_i})$$

is an expanded Lyndon word. As in the previous case, this contradicts the cyclic order of L_i.

The proof of step (b) is simpler. We need to show that we fall under the first case in the definition of next (cf. Definition 3), i.e., that $(k-1)^t L_i^{r_i-1} w\sigma$ is a necklace head. That is indeed true since

$$\mathcal{R}^t((k-1)^t L_i^{r_i-1} w\sigma) = L_i^{r_i-1} w\sigma(k-1)^t = L_i^{r_i}$$

is an expanded Lyndon word.

Finally, we address step (c), where we need to prove that $\mathsf{next}^t((k-1)^t L_i^{r_i-1} w(\sigma+1)) = L_i^{r_i-1} w(\sigma+1)x$, such that $x \in [k]^t$ is the lexicographically smallest word for which $L_i^{r_i-1} w(\sigma+1)x$ is an expanded Lyndon word. Note that by (1), $(k-1)^t L_i^{r_i-1} w(\sigma+1) \neq (k-1)^n$, so by Lemma 8 such an x exists. Additionally, for any $0 \leqslant i < t$ we have that $\mathsf{next}^i((k-1)^t L_i^{r_i-1} w(\sigma+1)) = (k-1)w'$, thus we never fall within the first case of next.

Next, we show that for any $0 \leqslant i < t, j = t-i-1, x = x_1\tau x_2, x_1 \in [k]^i, \tau \in [k]$, we have that

$$\mathsf{next}((k-1)^{j+1} L_i^{r_i-1} w(\sigma+1)x_1) = (k-1)^j L_i^{r_i-1} w(\sigma+1)x_1\tau.$$

We distinguish between two cases depending on τ. For the first case, let $\tau = k-1$. We contend that we do not fall within the second case of next. Assume to the contrary that there is some $\sigma' \in [k-1]$ such that $(k-1)^j L_i^{r_i-1} w(\sigma+1)x_1\sigma'$ is a necklace head. Thus, $L_i^{r_i-1} w(\sigma+1)x_1\sigma'(k-1)^j$ is an expanded Lyndon word. Looking at its suffix of length t, we get

$$x_1\sigma'(k-1)^j < x_1(k-1)x_2 = x_1\tau x_2 = x,$$

which is a contradiction to the minimality of x.

Now, for the case where $\tau \in [k-1]$. By the definition of x we know that $L_i^{r_i-1} w(\sigma+1)x = L_i^{r_i-1} w(\sigma+1)x_1\tau x_2$ is an expanded Lyndon word. Using Lemma 7, we get that $L_i^{r_i-1} w(\sigma+1)x_1\tau(k-1)^j$ is also an expanded Lyndon word. Therefore, $(k-1)^j L_i^{r_i-1} w(\sigma+1)x_1\tau$ is a necklace head. Left to be shown is that

$$\tau = \tau_{\min} \triangleq \min\left\{\tau' \in [k] \colon \mathsf{head}((k-1)^j L_i^{r_i-1} w(\sigma+1)x_1\tau')\right\}.$$

Assuming to the contrary that $\tau_{\min} < \tau$, then $(k-1)^j L_i^{r_i-1} w(\sigma+1)x_1\tau_{\min}$ is a necklace head, implying that $L_i^{r_i-1} w(\sigma+1)x_1\tau_{\min}(k-1)^j$ is an expanded Lyndon word. As in the previous case, since

$$x_1\tau_{\min}(k-1)^j < x_1\tau x_2 = x,$$

we get a contradiction to the minimality of x. □

Lemma 9 does not apply to the penultimate expanded Lyndon word, for which, by simple inspection of the definition of next we state

$$\mathsf{next}(L_{Z_k(n)-2}^{r_{Z_k(n)-2}}) = \mathsf{next}((k-2)(k-1)^{n-1}) = (k-1)^n. \tag{3}$$

We are now in a position to prove the main result.

Proof of Theorem 4. As a first technical step it is easy to verify that next is a shift rule generating some sequence, since indeed for every $\sigma w, \sigma \in [k], w \in [k]^{n-1}$, we have $\mathsf{next}(\sigma w) = w\sigma'$ for some $\sigma' \in [k]$.

In the next step, let us examine an unknown sequence $\langle \nu_i \rangle_{i=0}^{k^n-1}$, that is initialized with $\nu_0 \dots \nu_{n-1} = 0^n$, and whose following letters are generated using next. We define the numbers $s_i \triangleq \sum_{j=0}^{i-1} |L_i|$, for all $0 \leqslant i \leqslant Z_k(n) - 1$. We prove by induction that for all $i \in [Z_k(n)-1]$, $\nu_{s_i}\nu_{s_i+1}\dots\nu_{s_i+n-1} = L_i^{r_i}$. The proof is immediate since the induction base is our initialization of $\nu_0 \dots \nu_{n-1} = 0^n = L_0^{r_0}$, and the induction step is provided by Lemma 9, since

$$\nu_{s_{i+1}} \dots \nu_{s_{i+1}+n-1} = \mathsf{next}^{|L_i|}(\nu_{s_i} \dots \nu_{s_i+n-1}) = \mathsf{next}^{|L_i|}(L_i^{r_i}) = L_{i+1}^{r_{i+1}}.$$

By this induction, we already have the prefix of the generated sequence to be $L_0L_1 \dots L_{Z_k(n)-2}$, but we are missing the last Lyndon word, $L_{Z_k(n)-1} = k-1$. This is easily taken care of, since by (3),

$$\begin{aligned}\nu_{s_{Z_k(n)-2}+1} \dots \nu_{s_{Z_k(n)-2}+n} &= \mathsf{next}(\nu_{s_{Z_k(n)-2}} \dots \nu_{s_{Z_k(n)-2}+n-1}) \\ &= \mathsf{next}(L_{Z_k(n)-2}^{r_{Z_k(n)-2}}) = \mathsf{next}((k-2)(k-1)^{n-1}) \\ &= (k-1)^n,\end{aligned}$$

namely, the last letter is the last Lyndon word,

$$\nu_{s_{Z_k(n)-2}+n} = \nu_{s_{Z_k(n)-1}} = \nu_{k^n-1} = k-1 = L_{Z_k(n)-1}.$$

We also observe that the shift rule wraps around the end of the sequence. Indeed, by a simple inspection of Definition 3, for every $1 \leqslant i \leqslant n$,

$$\begin{aligned}\mathsf{next}(\nu_{k^n-i} \dots \nu_{k^n-1}\nu_0 \dots \nu_{n-1-i}) &= \mathsf{next}((k-1)^i 0^{n-i}) \\ &= (k-1)^{i-1}0^{n-i+1} \\ &= \nu_{k^n-i+1} \dots \nu_{k^n-1}\nu_0 \dots \nu_{n-i}.\end{aligned}$$

As the final step in the proof, by FKM [8,9] this sequence is exactly the prefer-min (n, k)-DB sequence. □

We conclude this section by reminding the reader that in order to generate the prefer-max (n, k)-DB sequence (instead of the prefer-min one), all that is required is to start the FSR with $(k-1)^n$, and to use the shift rule $\psi^{-1} \circ \mathsf{next} \circ \psi$, where ψ is the complement function defined in Section 2, and $\circ$ denotes function composition.

4. An efficient shift-rule algorithm

Algorithms for implementing shift-rules for the prefer-min (or prefer-max) (n, k)-DB sequences are known [10,6]. These greedy algorithms require $\Theta(k^n)$ memory, and $\Theta(nk^n)$ time in the worst case (since they in fact need to generate the sequence until the position of the desired next letter). The main result of this section is an efficient algorithm, requiring $O(n)$ time and memory, that implements the shift rule we presented in the previous section. By quick inspection, the claim hinges on an efficient implementation of the head predicate, as well as finding min S in the second case of next.

Our algorithm uses two key components. The first is the renowned factorization due to Chen, Fox, and Lyndon [2], namely, that every word $w \in [k]^+$ has a unique decomposition $w = w_0w_1 \dots w_{m-1}$, such that w_i is a Lyndon word for all $0 \leqslant i \leqslant m-1$, and $w_0 \geqslant w_1 \geqslant \dots \geqslant w_{m-1}$. We shall call this the CFL factorization of w. The second key component is due to Duval [4], who showed that this unique decomposition may be computed for all $w \in [k]^+$ in $O(|w|)$ time and memory.

First, we address the efficiency of computing the predicate head.

Lemma 10. *For any $w \in [k]^n$ it is possible to compute $\mathsf{head}(w)$ in $O(n)$ time and memory.*

Proof. Let $i \in \mathbb{Z}$ be the largest integer such that $(k-1)^i$ is a prefix of w. We apply Duval's algorithm [4] to $\mathcal{R}^i(w)$ to obtain its CFL factorization $\mathcal{R}^i(w) = w_0w_1 \dots w_{m-1}$. Then $\mathsf{head}(w)$ is true if and only if $w_0 = w_{m-1}$. □

Next we recall some useful results already known in the literature. A word $w \in \Sigma^*$ is called a *pre-necklace* if there exists $w' \in \Sigma^*$ such that ww' is an expanded Lyndon word. By [1, Lemma 2.3], a pre-necklace must necessarily be a fractional power of a Lyndon word, i.e., $w = u^m v$, with u being a Lyndon word, $m \geqslant 1$, and v a proper prefix of u. Since the u^m part is a prefix of a CFL decomposition for w, this decomposition is unique and it is efficiently computable in $O(|w|)$ time and memory. Finally, we recall [1, Theorem 2.1], whose authors dubbed the "fundamental theorem of necklaces".

Theorem 11 (*Theorem 2.1 of [1]*). *Let $w = \tau_0\tau_1 \dots \tau_{n-1}$, with $\tau_i \in [k]$, be a pre-necklace with fractional-power decomposition $w = u^m v$. Then, $w\sigma$, $\sigma \in [k]$, is a pre-necklace if and only if $\sigma \geqslant \tau_{|v|}$. Furthermore, $w\sigma$ is a Lyndon word if and only if $\sigma > \tau_{|v|}$.*

We are now in a position to describe the algorithm for $\mathsf{next}(\sigma w)$ and prove its correctness.

Algorithm 1: $\mathsf{next}(\sigma w)$.

```
1:  if ((σ < k − 1) ∧ head(wσ)) then
2:      return w(σ + 1)
3:  else if σw = (k − 1)^n then
4:      return (k − 1)^{n−1}0
5:  else if ((σ = k − 1) ∧ head(w(k − 2))) then
6:      let w' = τ_0 ... τ_{n−t−1}, such that τ_i ∈ [k], τ_0 ≠ k − 1, w = (k − 1)^t w'
7:      let u^m v = w' be the fractional-power decomposition of w'
8:      σ' ← τ_{|v|}
9:      if head(wσ') then
10:         return wσ'
11:     else
12:         return w(σ' + 1)
13:     end if
14: else
15:     return wσ
16: end if
```

Theorem 12. *Algorithm 1 correctly computes the shift rule* next *from Definition 3 in $O(n)$ time and memory.*

Proof. We argue that Algorithm 1 computes the function next. We consider the three cases of Definition 3 separately. First, if $\sigma \in [k-1]$ and $\mathsf{head}(w\sigma)$, the algorithm returns $w(\sigma+1)$ in line 2 as required by the first case of Definition 3.

Now, assume the input σw falls within the third case of Definition 3. If $\sigma < k-1$ the claim is obvious as the condition in line 1 does not hold. If $\sigma = k-1$, then we have $S \triangleq \{\sigma'' \in [k-1]\colon \mathsf{head}(w\sigma'')\}$. By Lemma 7, $S \neq \emptyset$ if and only if $\mathsf{head}(w(k-2))$ holds. Thus, line 5 correctly checks whether the second case of Definition 3 applies. We therefore reach line 15 exactly when the third case of Definition 3 applies, and correctly return $w\sigma$.

We are left with the second case of Definition 3, where $\sigma = k-1$ and $S \neq \emptyset$. First, the special case of $\sigma w = (k-1)^n$, is handled correctly in line 4. Otherwise, w contains some letter other than $k-1$, and w' is well defined.

We now contend that $\min S \in \{\sigma', \sigma'+1\}$. Since $\mathsf{head}(w(k-2))$ holds, then $w'(k-2)(k-1)^t$ is an expanded Lyndon word, hence w' is a pre-necklace. Also, note that if $\sigma'' \in S$ then $w'\sigma''$ is a pre-necklace. By Theorem 11, if $\sigma'' < \sigma'$ then $w'\sigma''$ is not a pre-necklace. Hence, $\min S \geqslant \sigma'$. However, also by Theorem 11, $w(\sigma'+1)$ is a Lyndon word, thus $\sigma'+1 \in S$ and $\min S \leqslant \sigma'+1$. This leaves only two possible values for $\min S$, and consequently, the algorithm terminates in line 10 or in line 12, and returns the desired word.

Finally, as already noted, CFL factorization, head, as well as the fractional-power decomposition of line 7, may be computed in linear time and memory (all relying on the CFL factorization algorithm). Thus, the entire algorithm takes linear time and memory. □

5. Discussion

In this paper we studied the well known prefer-min and prefer-max (n, k)-DB sequences. We completed a gap in the literature by presenting a shift-rule for the sequences, as well as an efficient algorithm computing this shift rule. The algorithm receives as input a sub-sequence of n letters, and determines the next letter in $O(n)$ time and memory.

The shift rule we presented may be seen as an extension to the binary shift rule presented in [7]. Indeed, if we set $k = 2$ in our algorithm, the second case of Definition 3 becomes degenerate, we are left with the algorithm of [7]. This also explains the main difficulty in our solution, which is finding $\min S$ efficiently. The crux of solving this difficulty is the proof that we only need to choose between two carefully chosen values.

References

[1] K. Cattell, F. Ruskey, J. Sawada, M. Serra, Fast algorithms to generate necklaces, unlabeled necklaces, and irreducible polynomials over $GF(2)$, J. Algorithms 37 (2000) 267–282.
[2] K.T. Chen, R.H. Fox, R.C. Lyndon, Free differential calculus, IV, Ann. of Math. 68 (1958) 81–95.
[3] P.B. Dragon, O.I. Hernandez, J. Sawada, A. Williams, D. Wong, Constructing de Bruijn sequences with co-lexicographic order: The k-ary Grandmama sequence, European J. Combin. 72 (2018) 1–11.
[4] J.P. Duval, Factorizing words over an ordered alphabet, J. Algorithms 4 (1983) 363–381.
[5] T. Etzion, An algorithm for constructing m-ary de Bruijn sequences, J. Algorithms 7 (3) (1986) 331–340.
[6] L.R. Ford, A cyclic arrangement of m-tuples, Tech. Rep. P-1071, RAND Corp., 1957.
[7] H.M. Fredricksen, Generation of the Ford sequence of length 2^n, n large, J. Combin. Theory 12 (1972) 153–154.
[8] H.M. Fredricksen, I.J. Kessler, Lexicographic compositions and deBruijn sequences, J. Combin. Theory 22 (1977) 17–30.
[9] H.M. Fredricksen, J. Maiorana, Necklaces of beads in k colors and k-ary de Bruijn sequences, Discrete Math. 23 (1978) 207–210.
[10] M.H. Martin, A problem in arrangements, Bull. Amer. Math. Soc. 40 (1934) 859–864.
[11] F. Ruskey, C. Savage, T.M.Y. Wang, Generating necklaces, J. Algorithms 13 (3) (1992) 414–430.
[12] J. Sawada, A. Williams, D. Wong, A surprisingly simple De Bruijn sequence construction, Discrete Math. 339 (2016) 127–131.
[13] J.H. van Lint, R.M. Wilson, A Course in Combinatorics, second ed., Cambridge Univ. Press, 2001.