Arbitrary-Length Analogs to de Bruijn Sequences

Abhinav Nellore
Oregon Health & Science University, Portland, OR, USA

Rachel Ward
The University of Texas at Austin, Austin, TX, USA

Abstract

Let $\widetilde{\alpha}$ be a length-L cyclic sequence of characters from a size-K alphabet $\mathcal{A}$ such that for every positive integer $m \leq L$, the number of occurrences of any length-m string on $\mathcal{A}$ as a substring of $\widetilde{\alpha}$ is $\lfloor L/K^m \rfloor$ or $\lceil L/K^m \rceil$. When $L = K^N$ for any positive integer N, $\widetilde{\alpha}$ is a de Bruijn sequence of order N, and when $L \neq K^N$, $\widetilde{\alpha}$ shares many properties with de Bruijn sequences. We describe an algorithm that outputs some $\widetilde{\alpha}$ for any combination of $K \geq 2$ and $L \geq 1$ in $O(L)$ time using $O(L \log K)$ space. This algorithm extends Lempel's recursive construction of a binary de Bruijn sequence. An implementation written in Python is available at `https://github.com/nelloreward/pkl`.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms; Mathematics of computing → Combinatorics on words

Keywords and phrases de Bruijn sequence, de Bruijn word, Lempel's D-morphism, Lempel's homomorphism

Digital Object Identifier 10.4230/LIPIcs.CPM.2022.9

Related Version *arXiv preprint*: `https://arxiv.org/abs/2108.07759`

Supplementary Material *Software (Source Code)*: `https://github.com/nelloreward/pkl`
archived at `swh:1:dir:28da4be18d03945d95e62f4acfc34f10e56b1772`

Acknowledgements We thank Arie Israel, Štěpán Holub, Joe Sawada, Jeffrey Shallit, and the anonymous reviewers for the 33rd Annual Symposium on Combinatorial Pattern Matching for their helpful feedback on various iterations of this manuscript. AN thanks the Oden Institute for Computational Engineering & Sciences at the University of Texas at Austin for hosting him as a visiting scholar as part of the J. Tinsley Oden Faculty Fellowship Research Program when the work contained here was initiated.

1 Introduction

1.1 Preliminaries

This paper is concerned with necklaces, otherwise known as circular strings or circular words. A *necklace* is a cyclic sequence of characters; each character has a direct predecessor and a direct successor, but no character begins or ends the sequence. So if 101 is said to be a necklace, 011 and 110 refer to the same necklace. In the remainder of this paper, the term *string* exclusively refers to a sequence of characters with a first character and a last character. A *substring* of a necklace is a string of contiguous characters whose length does not exceed the necklace's length. So the set of length-2 substrings of the necklace 101 is $\{10, 01, 11\}$. A *rotation* of a necklace is a substring whose length is precisely the necklace's length. A *prefix* of a string is any substring starting at the string's first character. So 011 can be called a rotation of the necklace 101, and 10 is a prefix of that rotation.

A *de Bruijn sequence* of order N on a size-K alphabet $\mathcal{A}$ is a length-K^N necklace that includes every possible length-N string on $\mathcal{A}$ as a substring [69, 17, 19, 18]. There are $(K!)^{K^{N-1}}/K^N$ distinct de Bruijn sequences of order N on $\mathcal{A}$ [19]. (See the appendix for a brief summary of the curious history of de Bruijn sequences.) An example for $\mathcal{A} = \{0, 1\}$ and $N = 4$ is the length-16 necklace

$$0000110101111001\,.$$

A de Bruijn sequence of order N on $\mathcal{A}$ is optimally short in the sense that its length is K^N, and there are K^N possible length-N strings on $\mathcal{A}$. But more is true: because any length-m string on $\mathcal{A}$ is a prefix of each of K^{N-m} strings on $\mathcal{A}$ when $m \leq N$, the sequence has precisely K^{N-m} occurrences of that length-m string as a substring. So in the example above, there are 8 occurrences of 0, 4 occurrences of 00, 2 occurrences of 000, and 1 occurrence of 0000. Note by symmetry, K^{N-m} is also the expected number of occurrences of any length-m string on $\mathcal{A}$ as a substring of a necklace of length K^N formed by drawing each of its characters uniformly at random from $\mathcal{A}$. More generally, by symmetry, L/K^m is the expected number of occurrences of any length-m string on $\mathcal{A}$ for $m \leq L$ as a substring of a necklace of arbitrary length L formed by drawing each of its characters uniformly at random from $\mathcal{A}$.

1.2 $P_L^{(K)}$-sequences

Consider a necklace defined as follows.

▶ **Definition 1** ($P_L^{(K)}$-sequence). *A $P_L^{(K)}$-sequence is a length-L necklace on a size-K alphabet $\mathcal{A}$ such that for every positive integer $m \leq L$, the number of occurrences of any length-m string on $\mathcal{A}$ as a substring of the necklace is $\lfloor L/K^m \rfloor$ or $\lceil L/K^m \rceil$.*

This paper proves by construction that a $P_L^{(K)}$-sequence exists for any combination of $K \geq 2$ and $L \geq 1$, giving an algorithm for sequence generation that runs in $O(L)$ time using $O(L \log K)$ space.

When $L = K^N$ for any positive integer N, $\lfloor L/K^m \rfloor = \lceil L/K^m \rceil = K^{N-m}$ for $m \leq N$, and a $P_L^{(K)}$-sequence collapses to a de Bruijn sequence of order N. When $L \neq K^N$, a $P_L^{(K)}$-sequence is a natural interpolative generalization of a de Bruijn sequence: it is a necklace for which the number of occurrences of any length-m string on $\mathcal{A}$ for $m \leq L$ as a substring differs by less than one from its expected value for a length-L necklace formed by drawing each of its characters uniformly at random from $\mathcal{A}$. When this expected value is an integer, $\lfloor L/K^m \rfloor = \lceil L/K^m \rceil$, and the number of occurrences of any length-m string on $\mathcal{A}$ as a substring of a given $P_L^{(K)}$-sequence is equal to the number of occurrences of any *other* length-m string on $\mathcal{A}$ as a substring of that sequence. When this expected value is not an integer, a $P_L^{(K)}$-sequence comes as close as it can to achieving the same end, as formalized in the proposition below.

▶ **Proposition 2.** *Consider a $P_L^{(K)}$-sequence $\widetilde{\alpha}$. Load across length-m strings on $\mathcal{A}$ for $m \leq L$ is balanced in $\widetilde{\alpha}$ as follows.*

1. *When L/K^m is an integer, each length-m string on $\mathcal{A}$ occurs precisely L/K^m times as a substring of $\widetilde{\alpha}$.*
2. *When L/K^m is not an integer, each of $L - K^m\lfloor L/K^m \rfloor$ length-m strings on $\mathcal{A}$ occurs precisely $\lceil L/K^m \rceil$ times as a substring of $\widetilde{\alpha}$, and each of $K^m\lceil L/K^m \rceil - L$ length-m strings on $\mathcal{A}$ occurs precisely $\lfloor L/K^m \rfloor$ times as a substring of $\widetilde{\alpha}$.*

Proof. Item 1 is manifestly true from $\lfloor L/K^m \rfloor = \lceil L/K^m \rceil$. To see why item 2 is true, consider the system of Diophantine equations

$$\begin{aligned} a\lfloor L/K^m \rfloor + b\lceil L/K^m \rceil &= L \\ a + b &= K^m \end{aligned} \quad . \tag{1}$$

Above, a represents the number of length-m strings on $\mathcal{A}$ for which there are $\lfloor L/K^m \rfloor$ occurrences each as a substring of $\widetilde{\alpha}$, and b represents the number of length-m strings on $\mathcal{A}$ for which there are $\lceil L/K^m \rceil$ occurrences each as a substring of $\widetilde{\alpha}$. The first equation says the total number of occurrences of strings as substrings of $\widetilde{\alpha}$ is L, and the second says there is a total of K^m length-m strings on $\mathcal{A}$. Note the equations hold only when L/K^m is nonintegral – that is, $\lfloor L/K^m \rfloor + 1 = \lceil L/K^m \rceil$. In this case, it is easily verified the unique solution to the system is $a = K^m \lceil L/K^m \rceil - L$ and $b = L - K^m \lfloor L/K^m \rfloor$. ◀

An example for $\mathcal{A} = \{0, 1\}$ and $L = 12$ is the sequence

$$000110111001\,. \tag{2}$$

To see why, note L/K^m for $L = 12$ and $K = 2$ is 6 for $m = 1$, 3 for $m = 2$, between 1 and 2 for $m = 3$, and between 0 and 1 for any $m \geq 4$. Further, the sequence (2) contains, as a substring, precisely

1. 6 occurrences of each string in the set $\{0, 1\}$;
2. 3 occurrences of each string in the set $\{00, 01, 10, 11\}$;
3. 2 occurrences of each string in the set $\{001, 011, 100, 110\}$, which is of size $L - K^m \lfloor L/K^m \rfloor = 12 - 2^3 \lfloor 12/2^3 \rfloor = 4$, and 1 occurrence of each string in the set $\{000, 010, 101, 111\}$, which is of size $K^m \lceil L/K^m \rceil - L = 2^3 \lceil 12/2^3 \rceil - 12 = 4$;
4. 1 occurrence of each string in the set
$$M := \{0001, 0011, 0110, 1101, 1011, 0111, 1110, 1100, 1001, 0010, 0100, 1000\}\,,$$
which is of size $L - K^m \lfloor L/K^m \rfloor = 12 - 2^4 \lfloor 12/2^4 \rfloor = 12$, and 0 occurrences of each of the set of length-4 strings on $\mathcal{A}$ not in M, which is of size $K^m \lceil L/K^m \rceil - L = 2^4 \lceil 12/2^4 \rceil - 12 = 4$; and
5. 0 or 1 occurrences of any length-m string for $4 < m \leq L$ due to item 4 above.

In distinct lines of work from the 1960s and 1970s, both Korobov [51, 52] and Stoneham [80, 77, 78, 79] explored the extent to which the repetends of base-K “decimal” forms of reduced proper fractions, when treated as necklaces, differed from expectation for digit content drawn uniformly at random from $[K]$. While the perspective differed from ours in that it did not a demand a particular necklace length L from the outset, the efforts did uncover that certain fractions in base-K decimal form yielded what we call $P_L^{(K)}$-sequences for particular combinations of K and L. Notably, in [80], Stoneham found that for $L + 1$ an odd prime and K a primitive root modulo $(L + 1)^2$, the repetend of the base-K decimal form of $1/(L + 1)$ is a $P_L^{(K)}$-sequence.

1.3 $P_L^{(K)}$-sequences vs. other de Bruijn-like sequences

Two other arbitrary-length generalizations of de Bruijn sequences have appeared in the literature:

1. What we call a *Lempel-Radchenko sequence* is a length-L necklace on a size-K alphabet $\mathcal{A}$ such that every length-$\lceil \log_K L \rceil$ string on $\mathcal{A}$ has at most one occurrence as a substring of the necklace. As recounted by Yoeli in [87], according to Radchenko and Filippov in [66], the existence of binary Lempel-Radchenko sequences of any length was first proved by Radchenko in his unpublished 1958 University of Leningrad PhD dissertation [65]. Other binary-case existence proofs were furnished by (1) Yoeli himself in [85] and [86]; (2) Bryant, Heath, and Killik in [8] based on the work [42] of Heath and Gribble; and (3) Golomb, Welch, and Goldstein in [40]. Explicit constructions of arbitrary-length binary Lempel-Radchenko sequences were given by Etzion in 1986 [25]. In brief, Etzion’s approach is to join necklaces derived from the pure cycling register, potentially overshooting the target length L, and subsequently remove substrings as necessary in the resulting sequence according to specific rules to achieve the target length. This takes $o(\log L)$ time per bit generated and uses $O(\log^2 L)$ space.

The existence of Lempel-Radchenko sequences of any length for any alphabet size was proved in 1971 by Lempel in [56]. In the special case where the alphabet size is a power of a prime number, one of two approaches for sequence construction effective at any length L may be used: either (1) pursue the algebraic construction described by Hemmati and Costello in their 1978 paper [43], or (2) cut out a length-L stretch of contiguous sequence generated by a linear feedback function, as described in Chapter 7, Section 5 of Golomb's text [39]. In his 2000 paper [54], Landsberg built on Golomb's technique, explaining in the appendix how to use it to construct a Lempel-Radchenko sequence on an alphabet of arbitrary size. The idea is to decompose the desired alphabet size into a product of powers of pairwise-distinct primes, construct length-L sequences on alphabets of sizes equal to factors in this product with Golomb's technique, and finally write a particular linear superposition of the sequences. The time and space requirements of Hemmati and Costello's construction, when optimized, have gone unstudied in the literature. In general, Golomb's technique gives a length-L Lempel-Radchenko sequence in $O(L \log L)$ time using $O(\log L)$ space, and Landsberg's generalization multiplies these complexities by the number of factors in the prime power decomposition of the alphabet size. Etzion suggested in his 1986 paper [25] that, using results from [24], his algorithm generating a binary Lempel-Radchenko sequence could be extended to generate a Lempel-Radchenko sequence for any alphabet size, but he did not do so explicitly. It is reported on Joe Sawada's website [20] that in their recent unpublished manuscript [41], Gündoğan, Sawada, and Cameron extend Etzion's construction to arbitrary alphabet sizes, streamlining it so it generates each character in $O(\log L)$ time using $O(\log L)$ space. Sawada's website further includes an implementation in C.

2. A *generalized de Bruijn sequence* is a length-L Lempel-Radchenko sequence on a size-K alphabet $\mathcal{A}$ such that every length-$\lfloor \log_K L \rfloor$ string on $\mathcal{A}$ is a substring of the sequence. Generalized de Bruijn sequences were recently introduced by Gabric, Holub, and Shallit in [32, 37]. These papers also prove generalized de Bruijn sequences exist for any combination of $L \geq 1$ and $K \geq 2$. No work to date has given explicit constructions of arbitrary-length generalized de Bruijn sequences.

We prove the following.

▶ **Theorem 3.** *A $P_L^{(K)}$-sequence is a generalized de Bruijn sequence and therefore also a Lempel-Radchenko sequence.*

Proof. Let $\widetilde{\alpha}$ be a $P_L^{(K)}$-sequence. The proposition is true if and only if

1. every length-$\lceil \log_K L \rceil$ substring of $\widetilde{\alpha}$ occurs precisely once in the sequence, and
2. every length-$\lfloor \log_K L \rfloor$ string on $\mathcal{A}$ is a substring of $\widetilde{\alpha}$.

Item 1 is true because from Definition 1, $\widetilde{\alpha}$ has

$$\lfloor L/K^{\lceil \log_K L \rceil} \rfloor = \begin{cases} 1 & \text{when } \log_K L \text{ is an integer} \\ 0 & \text{otherwise} \end{cases}$$

or $\lceil L/K^{\lceil \log_K L \rceil} \rceil = 1$ occurrences of any length-$\lceil \log_K L \rceil$ string on $\mathcal{A}$ as a substring. Item 2 is true because from Definition 1, $\widetilde{\alpha}$ has

$$\lceil L/K^{\lfloor \log_K L \rfloor} \rceil = \begin{cases} 1 & \text{when } \log_K L \text{ is an integer} \\ k & \text{otherwise for } k \in \{2, \ldots, K\} \end{cases}$$

or $\lfloor L/K^{\lfloor \log_K L \rfloor} \rfloor \in \{1, \ldots, K-1\}$ occurrences of any length-$\lfloor \log_K L \rfloor$ string on $\mathcal{A}$ as a substring. ◀

$P_L^{(K)}$-sequences are more tightly constrained than generalized de Bruijn sequences and Lempel-Radchenko sequences. A length-L Lempel-Radchenko sequence imposes no requirements regarding presence or absence of particular strings as substrings; it simply requires that the number of distinct length-$\lceil \log_K L \rceil$ substrings is L. A length-L generalized de Bruijn sequence on $\mathcal{A}$ goes a step further, requiring not only this distinctness, but also the presence of every string on $\mathcal{A}$ smaller than $\lceil \log_K L \rceil$ as a substring. A $P_L^{(K)}$-sequence goes yet another step further, requiring not only this presence, but also specific incidences of strings as substrings that, as best as they can, try not to bias the sequence toward inclusion of any one length-m string over another. This requirement makes $P_L^{(K)}$-sequences, in general, more de Bruijn-like than Lempel-Radchenko sequences and generalized de Bruijn sequences.

An example (borrowed from [37]) of a Lempel-Radchenko sequence that is not a generalized de Bruijn sequence and therefore also not a $P_L^{(K)}$-sequence for $\mathcal{A} = \{0, 1\}$ and $L = 11$ is

$$10011110000\,. \tag{3}$$

In this case, $\lceil \log_K L \rceil = \lceil \log_2 11 \rceil = 4$, and indeed, there is precisely one occurrence in (3) of every length-4 substring of (3). But $\lfloor \log_K L \rfloor = \lfloor \log_2 11 \rfloor = 3$, and in (3) just 7 of 8 length-3 strings on $\mathcal{A}$ occur as substrings; the sequence is missing 101. An example of a generalized de Bruijn sequence that is not a $P_L^{(K)}$-sequence for $\mathcal{A} = \{0, 1, 2\}$ and $L = 12$ is

$$000111101011\,. \tag{4}$$

Again, $\lceil \log_K L \rceil = \lceil \log_2 12 \rceil = 4$ and $\lfloor \log_K L \rfloor = \lfloor \log_2 12 \rfloor = 3$. Now, not only is there precisely one occurrence in (4) of every length-4 substring of (4), but also all 8 length-3 strings on $\mathcal{A}$ occur as substrings. However, (4) should have $\lceil L/K \rceil = \lfloor L/K \rfloor = 12/2 = 6$ occurrences of each of 1 and 0 as substrings to be a $P_L^{(K)}$-sequence, and it has 5 occurrences of 0 and 7 occurrences of 1. This imbalance of 0s and 1s leads to further violations of constraints on $P_L^{(K)}$-sequences at other substring lengths. Another example of a generalized de Bruijn sequence that is not a $P_L^{(K)}$-sequence, this time on the nonbinary alphabet $\mathcal{A} = \{0, 1, 2\}$ and for $L = 20$, is

$$02220010121120111002\,. \tag{5}$$

(This sequence was constructed by Landsberg in [54] using Golomb's technique from [39] as an example of a Lempel-Radchenko sequence.) Note $\lceil \log_K L \rceil = \lceil \log_3 20 \rceil = 3$ and $\lfloor \log_K L \rfloor = \lfloor \log_3 20 \rfloor = 2$, every length-3 substring occurs exactly once, and every length-2 string on $\mathcal{A}$ is present as a substring. But (5) should have, for $m = 2$, precisely $\lceil L/K^m \rceil = 20/3^2 = 3$ or $\lfloor L/K^m \rfloor = 20/3^2 = 2$ occurrences of every length-2 string on $\mathcal{A}$ as a substring, and there is only 1 occurrence of 21 as a substring of (5).

1.4 de Bruijn sequence constructions vs. de Bruijn-like sequence constructions

Unlike the current situation with de Bruijn-like sequences of arbitrary length, there is a veritable cornucopia of elegant constructions of de Bruijn sequences. Excellent summaries of many of these are provided on Sawada's website [20]. They include

1. greedy constructions. Prominent examples are the prefer-largest/prefer-smallest [58], prefer-same [23, 29, 3], and prefer-opposite [4] algorithms;
2. shift rules. A shift rule maps a length-N substring of a de Bruijn sequence of order N to the next length-N substring of the sequence. Shift rules are often simple, economical, and efficient; examples generating each character of a de Bruijn sequence in amortized constant time using $O(N)$ space are [73, 45, 26, 28] in the binary case and [74] in the K-ary case. See [5, 47, 35, 36, 84, 13, 88] for other specific rules;

3. concatenation rules. The best-known example, obtained by Fredricksen and Maiorana in 1978 [31], joins all Lyndon words on an ordered alphabet of size K whose lengths divide the desired order N in lexicographic order to form the lexicographically smallest (i.e., "granddaddy") de Bruijn sequence of that order on that alphabet. (Also see [27] for Ford's independent work generating this sequence.) The sequence is obtained in amortized constant time per character using $O(N)$ space with the efficient Lyndon word generation approach of Ruskey, Savage, and Wang [68], which builds on Fredricksen, Kessler, and Maiorana's papers [31, 30]. Dragon, Hernandez, Sawada, Williams, and Wong recently discovered that joining the Lyndon words in colexicographic order instead also outputs a particular de Bruijn sequence, the "grandmama" sequence [22, 21]. A generic concatenation approach using colexicographic order is developed in [33, 34];
4. recursive constructions. Broadly, these approaches are based on transforming a de Bruijn sequence into a de Bruijn sequence of higher order, where the transformation can be implemented recursively. They fall into two principal classes:
   a. the constructions of Mitchell, Etzion, and Paterson in [59], which interleave punctured and padded variants of a binary de Bruijn sequence of order N and modify the result slightly to obtain a binary de Bruijn sequence of order $2N$. If starting with a known binary de Bruijn sequence, this process takes amortized $O(1)$ time per output bit while using $O(1)$ additional space. The constructions are notable for being efficiently decodable – that is, the position of any given string on $\mathcal{A}$ occurring exactly once in the sequence as a substring can be retrieved in time polynomial in N;
   b. constructions based on Lempel's D-morphism (otherwise known as Lempel's homomorphism) [55], whose inverse lifts any length-L necklace $\widetilde{\beta}$ on a size-K alphabet $\mathcal{A}$ to up to K necklaces on $\mathcal{A}$. When $\widetilde{\beta}$ is a de Bruijn sequence of order N, the necklaces to which it is lifted may be joined to form a de Bruijn sequence of order $N + 1$. Efficient implementations constructing binary de Bruijn sequences of arbitrary order by repeated application of Lempel's D-morphism are given by Annexstein [6] as well as Chang, Park, Kim, and Song [12]; in general, a length-L binary de Bruijn sequence is generated in $O(L)$ time using $O(L)$ space. Lempel confined attention to the binary case in [55]. An extension to alphabets of arbitrary size was first written by Ronse in [67] and also developed by Tuliani in [81]; it was further generalized by Alhakim and Akinwande in [1]. See [38, 2] for other generalizations as well as [81] for a decodable de Bruijn sequence construction exploiting both interleaving and Lempel's D-morphism.

It is possible construction techniques for de Bruijn sequences have been more easily uncovered than for their arbitrary-length cousins as traditionally defined precisely because de Bruijn sequences are more tightly constrained. But $P_L^{(K)}$-sequences are similarly constrained.

1.5 Our contribution

This paper defines $P_L^{(K)}$-sequences. Further, it extends recursive de Bruijn sequence constructions based on Lempel's D-morphism [55, 67, 81, 1], giving an algorithm that outputs a $P_L^{(K)}$-sequence on the alphabet $\{0, \ldots, K-1\}$ for any combination of $L \geq 1$ and $K \geq 2$ in $O(L)$ time using $O(L \log K)$ space. The essence of our approach is to lengthen each of d_i longest runs of the same nonzero character by a single character at the ith step before lifting, where the $\{d_i\}$ are the digits of the desired length L of the $P_L^{(K)}$-sequence when expressed in base K – that is, for $L = \sum_{i=0}^{\lfloor \log_K L \rfloor} d_i K^{\lfloor \log_K L \rfloor - i}$. Finally, this paper is accompanied by Python code at `https://github.com/nelloreward/pkl` implementing our algorithm.

We were motivated to study arbitrary-length generalizations of de Bruijn sequences by [62], which introduces nength, an analog to the Burrows-Wheeler transform [10] for offline string matching in labeled digraphs. In a step preceding the transform, a digraph with edges labeled on one alphabet is augmented with a directed cycle that (1) includes every vertex of the graph and (2) matches a de Bruijn-like sequence on a different alphabet. This vests each vertex with a unique tag along the cycle. But if the de Bruijn-like sequence is an arbitrary Lempel-Radchenko or generalized de Bruijn sequence, some vertices may be significantly more identifiable than others when locating matches to query strings in the graph using its nength, biasing performance. So in general, it is reasonable to arrange that the directed cycle matches a $P_L^{(K)}$-sequence, which distributes identifiability across vertices as evenly as possible.

The remainder of this paper is organized as follows. The next section develops our algorithm for generating $P_L^{(K)}$-sequences, proving space and performance guarantees. The third and final section lists some open questions.

2 Generating $P_L^{(K)}$-sequences

2.1 Additional notation and conventions

In the development that follows, necklaces are represented by lowercase Greek letters adorned with tildes such as $\widetilde{\beta}$ and $\widetilde{\gamma}$, and strings are represented by unadorned lowercase Greek letters such as ω and ξ. A necklace or string's length or a set's size is denoted using $|\cdot|$. So $|\widetilde{\beta}|$ is the length of the necklace $\widetilde{\beta}$, and $|V|$ is the size of the set V. Necklaces and strings may be in indexed families, where for example in $\widetilde{\beta}_i$, i specifies the family member. Further, a necklace or string may be written as a function of another necklace or string. So $\omega(\widetilde{\beta})$ denotes that the string ω is a function of the necklace $\widetilde{\beta}$. When any function's argument is clear from context, that argument may be dropped with prior warning. So $\omega(\widetilde{\beta})$ may be written as, simply, ω.

The operation of *joining* two necklaces $\widetilde{\beta}$ and $\widetilde{\gamma}$ at a string ω to form a new necklace $\widetilde{\lambda}$ refers to cycle joining, described in Chapter 6 of Golomb's text [39]. $\widetilde{\lambda}$ is obtained by concatenating rotations of $\widetilde{\beta}$ and $\widetilde{\gamma}$ that share the prefix ω. So if $\widetilde{\beta} = 00101101$ and $\widetilde{\gamma} = 0110001$, joining $\widetilde{\beta}$ and $\widetilde{\gamma}$ at 110 gives $\widetilde{\lambda} = 110100101100010$. There may be more than one occurrence of ω as a substring of at least one of $\widetilde{\beta}$ and $\widetilde{\gamma}$, so there may be more than one way to join them at ω. Any way is permitted in such a case. Note joining $\widetilde{\beta}$ and $\widetilde{\gamma}$ at ω preserves length-m substring occurrence frequencies for $m \leq |\omega| + 1$.

For any positive integer j,

$$[j] := \{0, 1, \ldots, j-1\}\,.$$

While results are obtained for sequences on the alphabet $[K]$ here, they may be translated to any size-K alphabet $\mathcal{A}$ by appropriate substitution of characters. When a string or necklace is initially declared to be on the alphabet $[K]$, but an expression y for one of its characters is written such that $y \notin [K]$, that character should be interpreted as $y - K\lfloor y/K \rfloor$. This is simply the remainder of floored division of y by K. Put another way, expressions for characters of strings on $[K]$ respect arithmetic modulo K. For example, if the first character of a string on the alphabet $[2] = \{0, 1\}$ is specified as an expression that equals 9, that character is 1.

Individual characters comprising strings are often expressed in terms of variables, so a necklace or string may be written as a comma-separated list of characters enclosed by parentheses, where in the necklace case, $\circlearrowright$ is included as a subscript. For example, for $i = 3$, if $(i, i+1, 0, 1)$ is said to be on the alphabet $[4]$, it is the string 3001, while if $(i, i+1, 0, 1)_{\circlearrowleft}$ is said to be on $[4]$, it is the necklace 3001. Bracket notation is used to refer to a specific character of a string or necklace. So $\omega[i]$ refers to the character at index i of ω. Further, characters of a string are indexed in order, so $\omega[i+1]$ appears directly after $\omega[i]$ in ω. $\omega[0]$ and $\omega[|\omega|-1]$ refer, respectively, to the first and last characters of the string ω. For a necklace, the choice of the character at index 0 is arbitrary, but in a parenthetical representation of that necklace, the character at index 0 always comes first. So an arbitrary length-L necklace $\widetilde{\beta}$ always equals

$$\widetilde{\beta} = (\widetilde{\beta}[0], \widetilde{\beta}[1], \ldots, \widetilde{\beta}[L-1])_{\circlearrowleft}$$

but not necessarily

$$\widetilde{\beta} = (\widetilde{\beta}[1], \widetilde{\beta}[2], \ldots, \widetilde{\beta}[L-1], \widetilde{\beta}[0])_{\circlearrowleft}\,.$$

A valid character index of a string ω is confined to $[|\omega|]$, but a valid character index of a length-L necklace $\widetilde{\beta}$ is any integer j, with the stipulation

$$\widetilde{\beta}[j] = \widetilde{\beta}[j+L]\,.$$

A string or necklace on $[K]$ can be summed with any integer by adding that integer to each of its characters modulo K. So for an integer j and a length-L necklace $\widetilde{\beta}$,

$$\widetilde{\beta} + j = j + (\widetilde{\beta}[0], \widetilde{\beta}[1], \ldots, \widetilde{\beta}[L-1])_{\circlearrowleft} = (\widetilde{\beta}[0]+j, \widetilde{\beta}[1]+j, \ldots, \widetilde{\beta}[L-1]+j)_{\circlearrowleft}\,.$$

Finally, $\mathbf{i}_m$ is used as a shorthand for the length-m string $(i, i, \ldots, i)$, $\mathbf{i}_m^{++}$ is used as a shorthand for the length-m string $(i, i+1, \ldots, i+m-1)$, and $\widetilde{\mathbf{i}}_m^{++}$ is used as a shorthand for the length-m necklace $(i, i+1, \ldots, i+m-1)_{\circlearrowleft}$. In a slight abuse of notation, a variable representing a string such as ω, $\mathbf{i}_m$, or $\mathbf{i}_m^{++}$ can take the place of a character in a parenthetical representation of a string or necklace. So if $(\mathbf{2}_6^{++}, 3)$ is said to be a substring of a string on the alphabet $[4]$, that substring is 2301233.

2.2 Lempel's lift

Lempel's lift, defined below, realizes the simplest K-ary version of Lempel's D-morphism [55, 67, 81, 1] in inverse form.

▶ **Definition 4** (Lempel's lift). *Consider a length-L necklace $\widetilde{\beta}$ on the alphabet $[K]$. Lempel's lift of $\widetilde{\beta}$, denoted by $\{\widetilde{\lambda}_i(\widetilde{\beta})\}$, is the indexed family of necklaces on $[K]$ specified by*

$$\widetilde{\lambda}_i(\widetilde{\beta}) = i + \left(\widetilde{\beta}[0], \widetilde{\beta}[0] + \widetilde{\beta}[1], \ldots, \sum_{j=0}^{d(\widetilde{\beta})\cdot L-1} \widetilde{\beta}[j]\right)_{\circlearrowleft} \qquad i \in [p(\widetilde{\beta})]\,. \tag{6}$$

Above, $d(\widetilde{\beta})$ is the smallest positive integer such that $\left(\sum_{j=0}^{L-1} \widetilde{\beta}[j]\right) \cdot d(\widetilde{\beta})$ is divisible by K, and $p(\widetilde{\beta}) = K/d(\widetilde{\beta})$.

The remainder of this subsection (i.e., Section 2.2) abbreviates functions of $\widetilde{\beta}$ given above by dropping it as an argument. For example, p is written rather than $p(\widetilde{\beta})$.

Observe that $\widetilde{\lambda}_i$ is a discrete integral of $\widetilde{\beta}$, with $i \in [p]$ the constant of integration. The number d specified in Definition 4 is the smallest positive integer $q \in \{1, 2, \ldots, K\}$ such that integrating a cycle of $\widetilde{\beta}$ a total of q times gives a cycle of $\widetilde{\lambda}_i$. Conversely, $\widetilde{\beta}$ is uniquely determined by a discrete derivative of $\widetilde{\lambda}_i$, which eliminates the constant of integration:

$$\widetilde{\beta}^d = (\lambda_i[1] - \lambda_i[0], \lambda_i[2] - \lambda_i[1], \ldots, \lambda_i[d \cdot L - 1] - \lambda_i[d \cdot L - 2], \lambda_i[0] - \lambda_i[d \cdot L - 1])_{\circlearrowleft} \quad i \in [p]\,.$$

Above, the power d on the LHS denotes $\widetilde{\beta}$ is concatenated with itself d times.

In the case $K = 2$, Lempel's lift of $\widetilde{\beta}$ is composed of one necklace if $\widetilde{\beta}$ has an odd number of 1s and two necklaces otherwise. For example, Lempel's lift of 01011 comprises only 0110110010, while Lempel's lift of 01010 comprises 01100 and 10011; further, the derivative of 0110110010 is $0101101011 = (01011)^2$, while the derivative of each of 01100 and 10011 is 01011. As a nonbinary example for $K = 5$, observe that Lempel's lift of the length-8 necklace 02134012 comprises the single length-40 necklace

0231001330143341134211244120440224032230 ,

whose derivative is $(02134012)^5$.

Note the sum of the lengths of the necklaces comprising Lempel's lift of $\widetilde{\beta}$ is $K \cdot L$. Other properties of the lift pertinent to constructing $P_L^{(K)}$-sequences are as follows.

▶ **Lemma 5.** *Suppose $\widetilde{\beta}$ is a length-L necklace on the alphabet $[K]$, and m is an integer satisfying $0 \le m < L$. Suppose ω is a length-m string on $[K]$, and ξ_ℓ is the length-$(m+1)$ string on $[K]$ given by*

$$\xi_\ell = \ell + \left(0, \omega[0], \omega[0] + \omega[1], \ldots, \sum_{j=0}^{m-1} \omega[j]\right) \qquad \ell \in [K]\,. \tag{7}$$

Then ω occurs t times as a substring of $\widetilde{\beta}$ if and only if ξ_ℓ occurs t times as a substring of the necklaces comprising Lempel's lift of $\widetilde{\beta}$. When $m = 0$, ω is the length-0 string occurring as a substring at every character of $\widetilde{\beta}$.

Proof. Start constructing a given $\widetilde{\lambda}_i$ by integrating $\widetilde{\beta}$ from its character index 0 up to character index $w < L$. If ω occurs as a substring of $\widetilde{\beta}$ at index w, it follows from (6) and (7) that ξ_y occurs as a substring of $\widetilde{\lambda}_i$ at its character index $w - 1$ for some $y \in [K]$, and vice versa. For $d > 1$, continue integrating $\widetilde{\beta}$ past its character index w for another L characters to encounter ω again. This time, how p is defined in terms of the sum of $\widetilde{\beta}$'s characters implies ω's presence as a substring of $\widetilde{\beta}$ at index w is a necessary and sufficient condition for ξ_{y+p}'s presence as a substring of $\widetilde{\lambda}_i$ at its character index $w - 1 + L$. More generally, ω occurs as a substring of $\widetilde{\beta}$ at its character index w if and only if ξ_{y+qp} occurs as a substring of $\widetilde{\lambda}_i$ at its character index $w - 1 + qL$ for $q \in [d]$, and all occurrences of ξ_ℓ in Lempel's lift of $\widetilde{\beta}$ for which the difference between ℓ and y is divisible by p are in $\widetilde{\lambda}_i$. An occurrence of ξ_ℓ at any other value of ℓ is easily seen from (6) to be at a corresponding character index $w - 1 + qL$ of $\widetilde{\lambda}_j$ for particular $j \in [p] \setminus \{i\}$ and $q \in [d]$. So there is an invertible map from the set of distinct occurrences of ω as a substring of $\widetilde{\beta}$ into the set of distinct occurrences of ξ_ℓ as a substring of Lempel's lift of $\widetilde{\beta}$ for $\ell \in [K]$, giving the lemma. ◀

▶ **Lemma 6.** *The number of occurrences of a given length-m string on the alphabet $[K]$ for $0 < m \le L$ as a substring in the family of necklaces comprising Lempel's lift of a $P_L^{(K)}$-sequence on $[K]$ is $\lfloor L/K^{m-1} \rfloor$ or $\lceil L/K^{m-1} \rceil$.*

Proof. From (7), choosing ξ_ℓ uniquely determines ω. So by Lemma 5, any length-m string on $[K]$ occurs $\lfloor L/K^{m-1} \rfloor$ or $\lceil L/K^{m-1} \rceil$ times as a substring in the family of necklaces comprising Lempel's lift of some length-L necklace $\widetilde{\beta}$ if and only if a certain length-$(m-1)$ string on $[K]$ occurs $\lfloor L/K^{m-1} \rfloor$ or $\lceil L/K^{m-1} \rceil$ times as a substring of $\widetilde{\beta}$ for $0 < m \le L$. This holds by definition when $\widetilde{\beta}$ is a $P_L^{(K)}$ sequence, for which every possible length-$(m-1)$ string on $[K]$ occurs $\lfloor L/K^{m-1} \rfloor$ or $\lceil L/K^{m-1} \rceil$ times for $0 < m \le L$, giving the lemma. ◀

2.3 Algorithm and analysis

In this subsection (Section 2.3), $\widetilde{\alpha}$ is reserved to denote a $P_L^{(K)}$-sequence. Moreover, when a function from Definition 4 is invoked, and it has $\widetilde{\alpha}$ as an argument, that function is abbreviated by dropping the $\widetilde{\alpha}$. For example, p now refers to $p(\widetilde{\alpha})$.

Lemma 6 suggests a way to obtain a $P^{(K)}_{K \cdot L}$-sequence from a $P^{(K)}_{L}$-sequence $\widetilde{\alpha}$: join the necklaces in Lempel's lift of $\widetilde{\alpha}$ strategically to ensure the numbers of occurrences of specific strings as substrings do not violate the parameters of Definition 1. Below, the procedure LiftAndJoin includes an explicit prescription, and Theorem 8 proves it works. They are preceded by a requisite lemma extending the discussion of cycle joining from Section 2.1.

Algorithm 1 Procedure LiftAndJoin referenced in the text.

```
// Returns the $P^{(K)}_{K \cdot L}$-sequence formed by joining the necklaces
// comprising Lempel's lift of an input $P^{(K)}_{L}$-sequence $\widetilde{\alpha}$ on the
// alphabet $[K]$, with $K$ a clarifying input.  Here, $N := \lceil \log_K L \rceil$.
procedure LiftAndJoin($\widetilde{\alpha}$, $K$)
    Construct Lempel's lift $\{\widetilde{\lambda}_i : i \in [p]\}$ of $\widetilde{\alpha}$.
    if $p = 1$ then                                                      // Case 1
        return $\widetilde{\lambda}_0$
    end if
    if $\mathbf{1}_N$ is a substring of $\widetilde{\alpha}$ then                                   // Case 2
        Find $k \in [K]$ such that $\mathbf{k}^{++}_N$ is a substring of each of $\widetilde{\lambda}_0$ and $\widetilde{\lambda}_1$.
        Initialize $\widetilde{\sigma}$ to $\widetilde{\lambda}_0$.
        for $j := 1$ to $p - 1$ do
            Set $\widetilde{\sigma}$ to the result of joining $\widetilde{\sigma}$ and $\widetilde{\lambda}_j$ at $\mathbf{s}^{++}_N$ for $s = k + j - 1$.
        end for
        return $\widetilde{\sigma}$
    end if
    Construct the join graph $G = (V, E)$ defined in Theorem 8.               // Case 3
    Initialize $\widetilde{\sigma}$ to the necklace represented by an arbitrary vertex $v \in V$.
    Starting at $v$, perform a depth-first traversal of the connected component
    $G_C = (V_C, E_C)$ of $G$ for which $v \in V_C$, where at each vertex in $V_C$
    reached by walking across a given edge in $E_C$, the necklace represented
    by that vertex is joined with $\widetilde{\sigma}$ at the string labeling that edge, and the
    result is assigned to $\widetilde{\sigma}$.
    if $G_C = G$ then                                                   // Case 3a
        return $\widetilde{\sigma}$
    end if
    Find $k \in [K]$ such that $\mathbf{k}^{++}_{N-1}$ is a substring of each of $\widetilde{\sigma}$ and $\widetilde{\sigma} + 1$.   // Case 3b
    Initialize $\widetilde{\zeta}$ to $\widetilde{\sigma}$.
    for $j := 1$ to $p/|V_C| - 1$ do
        Set $\widetilde{\zeta}$ to the result of joining $\widetilde{\zeta}$ and $\widetilde{\sigma} + j$ at $\mathbf{s}^{++}_{N-1}$ for $s = k + j - 1$.
    end for
    return $\widetilde{\zeta}$
end procedure
```

▶ **Lemma 7.** *Consider two necklaces $\widetilde{\beta}$ and $\widetilde{\gamma}$ on the alphabet $[K]$, and suppose the length-$(m-1)$ string ω is a substring of each of them. For every $k \in [K]$, suppose further that no length-m string (ω, k) is a substring of each of $\widetilde{\beta}$ and $\widetilde{\gamma}$, and no length-m string (k, ω) is a substring of each of $\widetilde{\beta}$ and $\widetilde{\gamma}$. Finally, suppose every length-$(m+1)$ string on $[K]$ occurs either zero times or one time as a substring of the family $\{\widetilde{\beta}, \widetilde{\gamma}\}$. Then*

1. *every length-$(m+1)$ string on $[K]$ occurs either zero times or one time as a substring of the necklace $\widetilde{\sigma}$ formed by joining $\widetilde{\beta}$ and $\widetilde{\gamma}$ at ω, and*
2. *every length-w string for $w \leq m$ occurs the same number of times as a substring of $\{\widetilde{\beta}, \widetilde{\gamma}\}$ as it does as a substring of $\widetilde{\sigma}$.*

Proof. For $u, v, x, y \in [K]$, suppose the length-$(m-1)$ string ω occurs (1) in $\widetilde{\beta}$ as a substring of the length-$(m+1)$ string (u, ω, v), and (2) in $\widetilde{\gamma}$ as a substring of the length-$(m+1)$ string (x, ω, y). Join $\widetilde{\beta}$ and $\widetilde{\gamma}$ at these occurrences of ω to obtain the necklace $\widetilde{\sigma}$. The operation replaces (u, ω, v) and (w, ω, x) with (u, ω, y) and (x, ω, v) while affecting the occurrence frequencies of no other length-$(m+1)$ strings as substrings and no length-w strings as substrings for $w \leq m$. But (u, ω, y) cannot occur elsewhere as a substring of $\widetilde{\sigma}$ because if it does, then either (u, ω) or (ω, y) is a substring of each of $\widetilde{\beta}$ and $\widetilde{\gamma}$, a contradiction. By a parallel argument, (x, ω, v) cannot occur elsewhere in $\widetilde{\sigma}$. The lemma follows. ◀

▶ **Theorem 8.** *Given a $P_L^{(K)}$-sequence $\widetilde{\alpha}$ on the alphabet $[K]$, suppose $N = \lceil \log_K L \rceil$. Consider Lempel's lift $\{\widetilde{\lambda}_i : i \in [p]\}$ of $\widetilde{\alpha}$, and define the join graph $G = (V, E)$ as an undirected graph with p vertices such that*

1. *the vertex $v_i \in V$ represents $\widetilde{\lambda}_i$ for $i \in [p]$, and*
2. *an edge in E labeled by a length-N string of the form $(\mathbf{j}_{N-1}^{++}, k)$ or $(k, \mathbf{j}_{N-1}^{++})$ for $j, k \in [K]$ extends between vertex v_ℓ and vertex v_r if and only if that string occurs as a substring of each of $\widetilde{\lambda}_\ell$ and $\widetilde{\lambda}_r$ for $\ell, r \in [p]$.*

Then the length-KL necklace output by LiftAndJoin *with $\widetilde{\alpha}$ and K as inputs is a $P_{K \cdot L}^{(K)}$-sequence.*

Proof. Follow the logic of the LiftAndJoin pseudocode to prove it returns a $P_{K\cdot L}^{(K)}$-sequence. To start, line 2 constructs Lempel's lift of $\widetilde{\alpha}$, which is composed of p necklaces that together have precisely the same number of occurrences of any length-m string on $[K]$ as a substring that a $P_{K\cdot L}^{(K)}$-sequence does, according to Lemma 8. To join the necklaces, various cases are handled in order of increasing difficulty:

Case 1: (Lines 3-5) This is the most straightforward case, where Lempel's lift has precisely one necklace. By Lemma 6 and by definition of a $P_L^{(K)}$-sequence, the sole necklace is a $P_{K\cdot L}^{(K)}$-sequence, and it is returned (Line 4).

Case 2: (Line 6-13) In this case, $p > 1$ and $\mathbf{1}_N$ is a substring of $\widetilde{\alpha}$ so that by Lemma 5, $\mathbf{k}_{N+1}^{++}$ is a substring of $\widetilde{\lambda}_1$ for at least one $k \in [K]$. Consequently, $\mathbf{s}_N^{++}$ is a substring of each of $\widetilde{\lambda}_j$ and $\widetilde{\lambda}_{j-1}$ for $j \in [p] \setminus \{\underset{\sim}{0}\}$ and $s = k + j - 1$. Progressively joining a necklace under construction with the jth member $\widetilde{\lambda}_j$ of Lempel's lift of $\widetilde{\alpha}$ at $\mathbf{s}_N^{++}$ for $s = k + j - 1$ (Lines 8-11) and j running from 1 to $p - 1$ preserves occurrence frequencies of all strings on $[K]$ whose lengths do not exceed $N + 1$. Since by Lemma 6 a length-$(N + 1)$ string occurs either once or never as a substring of Lempel's lift of $\widetilde{\alpha}$, a string whose length exceeds $N + 1$ occurs either once or never as a substring of the joined necklace. So that joined necklace is a $P_{K\cdot L}^{(K)}$-sequence, and it is returned (Line 12). When $\widetilde{\alpha}$ is a de Bruijn sequence (i.e., for $L = K^N$), Case 2 is the K-ary extension [67, 81, 1] of the original join prescription of the paper [55] by Lempel introducing his D-morphism.

Case 3: (Lines 14-25) Because a length-N string on $[K]$ need not occur as a substring of $\widetilde{\alpha}$, $\mathbf{1}_N$ may not be a substring of $\widetilde{\alpha}$. This bars the availability of Lempel's join of Case 2. LiftAndJoin then looks for the closest alternative. By definition of a $P_L^{(K)}$-sequence, $\mathbf{1}_{N-1}$ is necessarily a substring of $\widetilde{\alpha}$, and so by Lemma 5, $\mathbf{j}_{N-1}^{++}$ is a substring of each necklace in Lempel's lift of $\widetilde{\alpha}$ for some $j \in [K]$. So Line 14 assembles the graph G encoding all possible joins at strings of the form $(\mathbf{j}_{N-1}^{++}, k)$ or $(k, \mathbf{j}_{N-1}^{++})$ for $j, k \in [K]$. Consider any connected component $G_C = (V_C, E_C)$ of G. A depth-first traversal of G_C prescribes a sequence of joins, which are performed to obtain a single necklace $\widetilde{\sigma}$ (Line 16). Two cases are then considered.

Case 3a: (Lines 17-19) In this case, there is just one connected component of G. Since each join was performed at a length-N string, by an argument parallel to that of Case 2, $\widetilde{\sigma}$ is a $P_{K\cdot L}^{(K)}$-sequence, and it is returned (Line 18).

Case 3b: (Lines 20-25) If there are multiple connected components of G, by symmetry, G_C is related to any other connected component by translation modulo K. More precisely, applying $v_k \to v_{k+j}$ to each vertex $v_k \in V_C$, $e_{k\ell} \to e_{k+j,\ell+j}$ to each edge $e_{k\ell} \in E_C$ extending between $v_k \in V_C$ and $v_\ell \in V_C$, and $\epsilon_{k\ell} \to \epsilon_{k+j,\ell+j} + j$ to each edge label $\epsilon_{k\ell}$ corresponding to $e_{k\ell} \in E_C$ gives a different connected component, where $j \in [p/|V_C|]$ and addition operations are performed modulo K. It follows that for every $j \in [p/|V_C|]$, $\widetilde{\sigma} + j$ gives the result of a sequence of joins prescribed by a different connected component of G. Because each join was performed at a length-N string, the necklaces $\{\widetilde{\sigma} + j : j \in [p/|V_C|]\}$ together have the same occurrence frequency of any length-m string on $[K]$ as does Lempel's lift of $\widetilde{\alpha}$ for $m \leq N + 1$. That occurrence frequency is 0 or 1 for $m = N + 1$, as it therefore also is for $m > N + 1$. Because possible joins at strings of the form $(\mathbf{s}_{N-1}^{++}, k)$ or $(k, \mathbf{s}_{N-1}^{++})$ for $s, k \in [K]$ were exhausted by prior joins, Lemma 7 guarantees that joins of the $\{\widetilde{\sigma} + j : j \in [p/|V_C|]\}$ at strings of the form $\mathbf{s}_{N-1}^{++}$ for $s \in [K]$ preserve the occurrence frequency of any length-m string on $[K]$ for $m \leq N$ while ensuring that when $m > N$, the occurrence frequency of a length-m

string remains either 0 or 1. So when all necklaces in $\{\widetilde{\sigma}+j : j \in [p/|V_C|]\}$ are joined as on Lines 21-24, the result is a $P_{K\cdot L}^{(K)}$-sequence, and it is returned (Line 25). Note the joins are performed in exact analogy to those of Case 2.

The output of LiftAndJoin is thus a $P_{K\cdot L}^{(K)}$-sequence. ◀

Repeated application of LiftAndJoin on a $P_L^{(K)}$-sequence $\widetilde{\alpha}$ outputs a $P_L^{(K)}$-sequence whose length multiplies the length of $\widetilde{\alpha}$ by a power of K. But this operation alone does not afford the expressive capacity to build up a $P_L^{(K)}$-sequence of arbitrary length starting from an $\widetilde{\alpha}$ of length less than K, in the same way that an arbitrary positive integer cannot be written as a power of K times a positive integer less than K. A mechanism for extending the length of $\widetilde{\alpha}$ by up to $K-1$ between applications of LiftAndJoin is required, where the length of the extension is determined by an appropriate digit from the base-K representation of L. The mechanism used in the iterative procedure GeneratePKL below, which outputs a $P_L^{(K)}$-sequence for any combination of $K \geq 2$ and $L \geq 1$, extends a given longest run of a nonzero character by a single character. Theorem 9 proves this approach works.

Algorithm 2 Procedure GeneratePKL referenced in the text.

```
    // Returns a $P_L^{(K)}$-sequence on the alphabet $[K]$ given $K \geq 2$ and
    // $L \geq 1$ as inputs.  Here, $N := \lceil \log_K L \rceil$.
 1: procedure GeneratePKL($K$, $L$)
 2:     Compute the digits $\{d_i\}$ of $L$ in its base-$K$ representation as specified by
        $L = \sum_{i=0}^{N-1} d_i K^{N-i-1}$.
 3:     Initialize the necklace $\widetilde{\alpha}$ to $\widetilde{\mathbf{1}}_{d_0}^{++}$.
 4:     for $j := 1$ to $N-1$ do
 5:         Set $\widetilde{\alpha}$ to LiftAndJoin($\widetilde{\alpha}$, $K$).
 6:         if $d_j > 0$ then
 7:             Set $\widetilde{\alpha}$ to the extension of $\widetilde{\alpha}$ by $d_j$ characters obtained by replacing a substring
                $\mathbf{k}_{j-1}$ with $\mathbf{k}_j$ for every $k \in \{1, \ldots, d_j\}$.
 8:         end if
 9:     end for
10:     return $\widetilde{\alpha}$
11: end procedure
```

▶ **Theorem 9.** *GeneratePKL(K, L) outputs a $P_L^{(K)}$-sequence for any combination of $K \geq 2$ and $L \geq 1$.*

Proof. Use the notation $\widetilde{\alpha}_0$ to denote the value of $\widetilde{\alpha}$ after Line 3 of GeneratePKL is executed and the notation $\widetilde{\alpha}_j$ to denote the value of $\widetilde{\alpha}$ after step j of the **for** loop of GeneratePKL. Prove the theorem by induction, showing that if $\widetilde{\alpha}_{j-1}$ is a $P_{L_{j-1}}^{(K)}$-sequence of length L_{j-1}, and $\mathbf{0}_m$ occurs $\lfloor L_{j-1}/K^m \rfloor$ times as a substring of $\widetilde{\alpha}_{j-1}$ for all $m \leq L_{j-1}$, then $\widetilde{\alpha}_j$ is a $P_{L_j}^{(K)}$-sequence of length $L_j = K \cdot L_{j-1} + d_j$, and $\mathbf{0}_n$ occurs $\lfloor L_j/K^n \rfloor$ times as a substring of $\widetilde{\alpha}_j$ for all $n \leq L_j$. The base case for the induction holds: $\widetilde{\alpha}_0$, as initialized on Line 3, is the $P_{L_0}^{(K)}$-sequence $\widetilde{\mathbf{1}}_{d_0}^{++}$ of length $L_0 = d_0$, in which $\mathbf{0}_m$ occurs as a substring $\lfloor d_0/K^m \rfloor = 0$ times for $1 \leq m \leq d_0$ and $\lfloor d_0/K^m \rfloor = d_0$ times for $m = 0$. Now suppose that $\widetilde{\alpha}_{j-1}$ is a $P_{L_{j-1}}^{(K)}$-sequence of length L_{j-1}, and $\mathbf{0}_m$ occurs $\lfloor L_{j-1}/K^m \rfloor$ times as a substring of $\widetilde{\alpha}_{j-1}$ for all $m \leq L_{j-1}$. Then for every $k \in [K]$, $\mathbf{k}_{m+1}$ occurs $\lfloor L_{j-1}/K^m \rfloor = \lfloor (K \cdot L_{j-1})/K^{m+1} \rfloor$ times as a substring of LiftAndJoin$(\widetilde{\alpha}_{j-1}, K)$, obtained on Line 5. This follows from

1. Lemma 5, which says there are t occurrences of $\mathbf{0}_m$ as a substring of a necklace if and only if there are t occurrences of $\mathbf{k}_{m+1}$ as a substring in Lempel's lift of that necklace, and
2. how all joins of necklaces in Lempel's lift prescribed by LiftAndJoin, including those permitted by Lemma 7, do not affect occurrences of substrings of the form $\mathbf{k}_{m+1}$.

The extension performed on Line 7 increases the number of occurrences of $\mathbf{k}_m$, for $k = 1, 2, \ldots, d_j$, from $\lfloor (K \cdot L_{j-1})/K^{m+1} \rfloor$ to $\lceil (K \cdot L_{j-1})/K^{m+1} \rceil$ without affecting the numbers of occurrences of any other length-m strings as substrings for $m \leq j$. The longest string of 0s is never extended, and the number of occurrences of $\mathbf{0}_n$ remains $\lfloor (K \cdot L_{j-1})/K^{n+1} \rfloor$ for all $n \leq L_j$. So the resulting necklace $\widetilde{\alpha}_j$ is a $P_{L_j}^{(K)}$-sequence of length $L_j = K \cdot L_{j-1} + d_j$, and $\mathbf{0}_n$ occurs $\lfloor L_j/K^n \rfloor$ times as a substring of $\widetilde{\alpha}_j$ for all $n \leq L_j$.

The **for** loop thus encodes the recursion

$$L_j = K \cdot L_{j-1} + d_j \quad j \in [N-1] \setminus \{0\} \tag{8}$$

with initial condition $L_0 = d_0$. The formula $L = L_{N-1} = \sum_{i=0}^{N-1} d_i K^{N-i-1}$ follows, concluding the proof. ◀

In the binary case, GENERATEPKL and the joins it requires of its subroutine LIFTANDJOIN collapse to a particularly simple algorithm, which is given in the procedure GENERATEP2L below.

Algorithm 3 Procedure GENERATEP2L referenced in the text.

```
// Returns a $P_L^{(2)}$-sequence on the alphabet $\{0,1\}$ given $L \geq 1$ as
// an input.  Here, $N := \lceil \log_2 L \rceil$.
procedure GENERATEP2L($L$)
    Compute the digits $\{d_i\}$ of $L$ in its binary representation as specified by $L = \sum_{i=0}^{N-1} d_i 2^{N-i-1}$.
    Initialize the necklace $\widetilde{\alpha}$ to the single character 1.
    for $j := 1$ to $N - 1$ do
        Construct Lempel's lift $\{\widetilde{\lambda}_i : i \in [p]\}$ of $\widetilde{\alpha}$.
        if p = 1 then
            // $\widetilde{\alpha}$ has an odd number of 1s.
            Set $\widetilde{\alpha}$ to $\widetilde{\lambda}_0$.
        else if $\mathbf{1}_{j-1}$ is a substring of $\widetilde{\alpha}$ then
            Set $\widetilde{\alpha}$ to the result of joining $\widetilde{\lambda}_0$ and $\widetilde{\lambda}_1$ at $\mathbf{0}_{j-1}^{++}$.
        else if $\widetilde{\lambda}_0$ and $\widetilde{\lambda}_1$ can be joined at $(\mathbf{0}_{j-2}^{++}, k)$ or $(k, \mathbf{0}_{j-2}^{++})$ for $k \in [2]$ then
            Set $\widetilde{\alpha}$ to the result of joining $\widetilde{\lambda}_0$ and $\widetilde{\lambda}_1$ at $(\mathbf{0}_{j-2}^{++}, k)$ or $(k, \mathbf{0}_{j-2}^{++})$ for $k \in [2]$.
        else
            Set $\widetilde{\alpha}$ to the result of joining $\widetilde{\lambda}_0$ and $\widetilde{\lambda}_1$ at $\mathbf{0}_{j-2}^{++}$.
        end if
        if $d_j = 1$ then
            Set $\widetilde{\alpha}$ to the extension of $\widetilde{\alpha}$ by a single character obtained by replacing $\mathbf{1}_{j-1}$ with $\mathbf{1}_j$.
        end if
    end for
    return $\widetilde{\alpha}$
end procedure
```

Below is the final theorem of this paper, which proves complexity results.

▶ **Theorem 10.** *GENERATEPKL outputs a $P_L^{(K)}$-sequence in $O(L)$ time using $O(L \log K)$ space.*

Proof. The space required by GENERATEPKL is dominated by storage of the final $P_L^{(K)}$-sequence itself, which is $O(L \log K)$.

To see why the algorithm takes $O(L)$ time, consider first the case $L < K$. GENERATEPKL then initializes $\widetilde{\alpha}$ to the positive integers in order up to and including L (Line 3), which scales as L. It subsequently skips the **for** loop and returns $\widetilde{\alpha}$.

Now consider the opposite case $L \geq K$. Expressing L in base K (Line 2) scales as $\log_K L$, and initializing $\widetilde{\alpha}$ (Line 3) scales as K. Focus on Line 5's call of LIFTANDJOIN at step j of the **for** loop, where the length-L_{j-1} necklace $\widetilde{\alpha}_{j-1}$ is passed to LIFTANDJOIN in the notation of Theorem 9's proof. Constructing Lempel's lift of $\widetilde{\alpha}_{j-1}$ (Line 2 of LIFTANDJOIN) scales as $K \cdot L_{j-1}$, the total length of the necklaces constructed. Addressing Case 1 (Lines 3-5) takes constant time. Addressing Case 2 (Lines 6-13) involves searching $\widetilde{\alpha}_{j-1}$ for $\mathbf{1}_N$, which scales as L_{j-1}, and successively joining the necklaces comprising Lempel's lift, which scales as $K \cdot L_{j-1}$ if implemented as, e.g., a sequence of rotations and concatenations in which indexes of join substrings are tracked. Addressing Case 3 in its entirety (Lines 14-24) involves (1) constructing the join graph G, which is dominated by the $K \cdot L_{j-1}$ scaling of searching Lempel's lift for strings of the form $\mathbf{s}_{N-1}^{++}$ for $s \in [K]$, (2) performing a depth-first traversal of a connected component of the join graph, which takes time linear in a number of at most K vertices, and (3) joining necklaces, which also scales as $K \cdot L_{j-1}$. So LIFTANDJOIN is dominated by a $K \cdot L_{j-1}$ scaling at step j of the **for** loop of GeneratePKL. Refocusing on GeneratePKL, extending LiftAndJoin($\widetilde{\alpha}_{j-1}, K$) (Line 7) involves searching for longest runs of the same character and inserting characters as necessary, scaling as $K \cdot L_{j-1}$ if performed in one pass through the necklace. Therefore, step j of the **for** loop scales as $K \cdot L_{j-1}$, and from the recursion (8), executing all iterations of the **for** loop scales as L. The time taken by the **for** loop dominates that of Lines 2 and 3. So the overall scaling is L for the two cases $L \geq K$ and $L < K$, and GeneratePKL takes $O(L)$ time. ◀

3 Discussion

In this paper, we have introduced $P_L^{(K)}$-sequences as arbitrary-length analogs to de Bruijn sequences. We have shown by explicit construction that a $P_L^{(K)}$-sequence exists for any combination of $K \geq 2$ and $L \geq 1$, giving an $O(L)$-time, $O(L \log K)$-space algorithm extending Lempel's recursive construction of a binary de Bruijn sequence. An implementation of the algorithm in Python is available at `https://github.com/nelloreward/pkl`.

We conclude with several open questions suggested by our work:

1. What is the number of distinct $P_L^{(K)}$-sequences on $\mathcal{A}$ for every possible combination of K and L? As Gabric, Holub, and Shallit did in [32, 37] for generalized de Bruijn sequences, we have counted $P_L^{(2)}$-sequences for L up to 32 by exhaustive search. Table 1 displays our results, which can be reproduced using code at `https://github.com/nelloreward/pkl`. Note the counts do not increase monotonically with L.
2. Can the algorithm for $P_L^{(K)}$-sequence generation presented here or a variant be encoded in a shift rule? This would reduce the space it requires, perhaps at the expense of performance. An obstacle to deriving a shift rule from the algorithm that works for all values of L at a given alphabet size K is that it would have to account for cases like those of LiftAndJoin. See [61, 76, 2] for work along the lines of mathematically unrolling Lempel's recursion and generalizations.
3. Are there elegant constructions of $P_L^{(K)}$-sequences for any possible combination of K and L that extend constructions of de Bruijn sequences besides Lempel's recursive construction? There is a considerable body of literature on constructing universal cycles. (See, e.g., [46, 48, 71, 72, 83, 36, 75]). Introduced by Chung, Diaconis, and Graham in [14], a *universal cycle* is a length-L necklace in which every string in a size-L set S of length-m strings occurs as a substring. It is possible a set S curated to ensure the universal cycle is a $P_L^{(K)}$-sequence is compatible with existing universal cycle constructions or extensions.
4. Can an efficiently decodable $P_L^{(K)}$-sequence be constructed for any possible combination of K and L? Toward answering this question, it may be worth further investigating the efficient decoding of Lempel's recursive construction of a de Bruijn sequence. (See [64] for the binary case and [81] for the K-ary case.) Other efficiently decodable constructions of de Bruijn sequences are given in [50, 70].
5. What other properties that can be exhibited by a necklace are preserved under Lempel's D-morphism, and how can they be exploited to recursively construct other useful sequences? While this work was being prepared, Mitchell and Wild posted [60] to arXiv, which shows binary orientable sequences can be constructed recursively using Lempel's D-morphism. An *orientable sequence* is a necklace $\widetilde{\nu}$ for which each length-n substring has precisely one occurrence in precisely one of $\widetilde{\nu}$ and the reverse of $\widetilde{\nu}$ [9, 15]. It is perhaps unsurprising that Lempel's D-morphism, a kind of derivative, is so versatile and that orientability, $P_L^{(K)}$-sequence composition, and efficient decodability can be preserved by its inverse.

Table 1 Numbers of distinct $P_L^{(2)}$ sequences on $\mathcal{A}$ for various values of L.

L	Number of distinct $P_L^{(2)}$-sequences	L	Number of distinct $P_L^{(2)}$-sequences
1	2	17	32
2	1	18	36
3	2	19	68
4	1	20	57
5	2	21	138
6	3	22	123
7	4	23	252
8	2	24	378
9	4	25	504
10	3	26	420
11	6	27	1296
12	9	28	1520
13	12	29	2176
14	20	30	2816
15	32	31	4096
16	16	32	2048

References

1 Abbas Alhakim and Mufutau Akinwande. A recursive construction of nonbinary de Bruijn sequences. *Designs, Codes and Cryptography*, 60(2):155–169, 2011.

2 Abbas Alhakim and Maher Nouiehed. Stretching de Bruijn sequences. *Designs, Codes and Cryptography*, 85(2):381–394, 2017.

3 Abbas Alhakim, Evan Sala, and Joe Sawada. Revisiting the prefer-same and prefer-opposite de Bruijn sequence constructions. *Theoretical Computer Science*, 852:73–77, 2021.

4 Abbas M Alhakim. A simple combinatorial algorithm for de Bruijn sequences. *The American Mathematical Monthly*, 117(8):728–732, 2010.

5 Gal Amram, Yair Ashlagi, Amir Rubin, Yotam Svoray, Moshe Schwartz, and Gera Weiss. An efficient shift rule for the prefer-max de Bruijn sequence. *Discrete Mathematics*, 342(1):226–232, 2019.

6 Fred S. Annexstein. Generating de Bruijn sequences: An efficient implementation. *IEEE Transactions on Computers*, 46(2):198–200, 1997.

7 Charles Philip Brown. *Sanskrit Prosody and Numerical Symbols Explained.* Trübner & Company, 1869.

8 PR Bryant, FG Heath, and RD Killick. Counting with feedback shift registers by means of a jump technique. *IRE Transactions on Electronic Computers*, pages 285–286, 1962.

9 John Burns and Chris J Mitchell. *Coding Schemes for Two-Dimensional Position Sensing.* Hewlett-Packard Laboratories, Technical Publications Department, 1992.

10 Michael Burrows and David J Wheeler. A block-sorting lossless data compression algorithm. Technical report, Systems Research Center, 1994.

11 Camille Flye Sainte-Marie. `http://henripoincare.fr/s/correspondance/item/14609`. Accessed: 2021-07-23.

12 Taejoo Chang, Bongjoo Park, Yun Hee Kim, and Iickho Song. An efficient implementation of the D-homomorphism for generation of de Bruijn sequences. *IEEE Transactions on Information Theory*, 45(4):1280–1283, 1999.

13 Zuling Chang, Martianus Frederic Ezerman, Pinhui Ke, and Qiang Wang. General criteria for successor rules to efficiently generate binary de Bruijn sequences. *arXiv preprint*, 2019. arXiv:1911.06670.

14 Fan Chung, Persi Diaconis, and Ron Graham. Universal cycles for combinatorial structures. *Discrete Mathematics*, 110(1-3):43–59, 1992.

15 ZD Dai, KM Martin, MJB Robshaw, and PR Wild. Orientable sequences. In *Institute of Mathematics and Its Applications Conference Series*, volume 45, pages 97–97. Oxford University Press, 1993.

16 NG de Bruijn. In memoriam T. van Aardenne-Ehrenfest, 1905-1984. *Nieuw Archief voor Wiskunde*, 4(2):235–236, 1985.

17 Nicolaas Govert De Bruijn. A combinatorial problem. In *Proc. Koninklijke Nederlandse Academie van Wetenschappen*, volume 49, pages 758–764, 1946.

18 Nicolaas Govert de Bruijn. Acknowledgement of priority to C. Flye Sainte-Marie on the counting of circular arrangements of 2^n zeros and ones that show each n-letter word exactly once. *EUT report. WSK, Dept. of Mathematics and Computing Science*, 75, 1975.

19 Nicolaas Govert de Bruijn and Tanja van Aardenne-Ehrenfest. Circuits and trees in oriented linear graphs. *Simon Stevin*, 28:203–217, 1951.

20 De Bruijn sequence and universal cycle constructions. https://debruijnsequence.org/. Accessed: 2021-07-24.

21 Patrick Baxter Dragon, Oscar I Hernandez, Joe Sawada, Aaron Williams, and Dennis Wong. Constructing de Bruijn sequences with co-lexicographic order: The k-ary grandmama sequence. *European Journal of Combinatorics*, 72:1–11, 2018.

22 Patrick Baxter Dragon, Oscar I Hernandez, and Aaron Williams. The grandmama de Bruijn sequence for binary strings. In *LATIN 2016: Theoretical Informatics*, pages 347–361. Springer, 2016.

23 Cornelius Eldert, HM Gurk, HJ Gray, and Morris Rubinoff. Shifting counters. *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, 77(1):70–74, 1958.

24 Tuvi Etzion. An algorithm for constructing m-ary de Bruijn sequences. *Journal of algorithms*, 7(3):331–340, 1986.

25 Tuvi Etzion. An algorithm for generating shift-register cycles. *Theoretical computer science*, 44:209–224, 1986.

26 Tuvi Etzion and Abraham Lempel. Algorithms for the generation of full-length shift-register sequences. *IEEE Transactions on Information Theory*, 30(3):480–484, 1984.

27 LR Ford Jr. A cyclic arrangement of m-tuples. *Report No. P-1071, RAND Corp*, 1957.

28 Harold Fredricksen. A class of nonlinear de Bruijn cycles. *Journal of Combinatorial Theory, Series A*, 19(2):192–199, 1975.

29 Harold Fredricksen. A survey of full length nonlinear shift register cycle algorithms. *SIAM review*, 24(2):195–221, 1982.

30 Harold Fredricksen and Irving J Kessler. An algorithm for generating necklaces of beads in two colors. *Discrete mathematics*, 61(2-3):181–188, 1986.

31 Harold Fredricksen and James Maiorana. Necklaces of beads in k colors and k-ary de Bruijn sequences. *Discrete Mathematics*, 23(3):207–210, 1978.

32 Daniel Gabric, Štěpán Holub, and Jeffrey Shallit. Generalized de Bruijn words and the state complexity of conjugate sets. In *International Conference on Descriptional Complexity of Formal Systems*, pages 137–146. Springer, 2019.

33 Daniel Gabric and Joe Sawada. A de Bruijn sequence construction by concatenating cycles of the complemented cycling register. In *International Conference on Combinatorics on Words*, pages 49–58. Springer, 2017.

34 Daniel Gabric and Joe Sawada. Constructing de Bruijn sequences by concatenating smaller universal cycles. *Theoretical Computer Science*, 743:12–22, 2018.

35 Daniel Gabric, Joe Sawada, Aaron Williams, and Dennis Wong. A framework for constructing de Bruijn sequences via simple successor rules. *Discrete Mathematics*, 341(11):2977–2987, 2018.

36 Daniel Gabric, Joe Sawada, Aaron Williams, and Dennis Wong. A successor rule framework for constructing k-ary de Bruijn sequences and universal cycles. *IEEE Transactions on Information Theory*, 66(1):679–687, 2019.

37 Daniel Gabric, Štěpán Holub, and Jeffrey Shallit. Maximal state complexity and generalized de Bruijn words. *Information and Computation*, page 104689, 2021. `doi:10.1016/j.ic.2021.104689`.

38 R Games. A generalized recursive construction for de Bruijn sequences. *IEEE transactions on information theory*, 29(6):843–850, 1983.

39 Solomon W Golomb. *Shift Register Sequences: Secure and Limited-Access Code Generators, Efficiency Code Generators, Prescribed Property Generators, Mathematical Models*. World Scientific, 2017.

40 Solomon W Golomb, Lloyd R Welch, and Richard M Goldstein. Cycles from nonlinear shift registers. Technical report, Jet Propulsion Lab, Pasadena, CA, 1959.

41 Aysu Gündoğan, Ben Cameron, and Joe Sawada. Cut-down de Bruijn sequences, 2021. Unpublished manuscript.

42 FG Heath and MW Gribble. Chain codes and their electronic applications. *Proceedings of the IEE-Part C: Monographs*, 108(13):50–57, 1961.

43 Farhad Hemmati and Daniel J Costello. An algebraic construction for q-ary shift register sequences. *IEEE Transactions on Computers*, 100(12):1192–1195, 1978.

44 Patricia Hersh, Thomas Lam, Pavlo Pylyavskyy, and Victor Reiner. *The Mathematical Legacy of Richard P. Stanley*, volume 100. American Mathematical Soc., 2016.

45 Yuejiang Huang. A new algorithm for the generation of binary de Bruijn sequences. *Journal of Algorithms*, 11(1):44–51, 1990.

46 Glenn Hurlbert and Garth Isaak. Equivalence class universal cycles for permutations. *Discrete Mathematics*, 149(1-3):123–129, 1996.

47 Cees JA Jansen, Wouter G Franx, and Dick E Boekee. An efficient algorithm for the generation of DeBruijn cycles. *IEEE Transactions on Information Theory*, 37(5):1475–1478, 1991.

48 J Robert Johnson. Universal cycles for permutations. *Discrete Mathematics*, 309(17):5264–5270, 2009.

49 Subhash Kak. Yamatarajabhanasalagam: An interesting combinatoric sutra. *Indian Journal of History of Science*, 35(2):123–128, 2000.

50 Tomasz Kociumaka, Jakub Radoszewski, and Wojciech Rytter. Efficient ranking of Lyndon words and decoding lexicographically minimal de Bruijn sequence. *SIAM Journal on Discrete Mathematics*, 30(4):2027–2046, 2016.

51 Nikolai Mikhailovich Korobov. Trigonometric sums with exponential functions and the distribution of signs in repeating decimals. *Mathematical notes of the Academy of Sciences of the USSR*, 8(5):831–837, 1970.

52 Nikolai Mikhailovich Korobov. On the distribution of digits in periodic fractions. *Mathematics of the USSR-Sbornik*, 18(4):659, 1972.

53 Charles-Ange Laisant and Emile Michel Hyacinthe Lemoine. *L'Intermédiaire des Mathematiciens*, volume 1. Gauthier-Villars et Fils, 1894.

54 Max Landsberg. Feedback functions for generating cycles over a finite alphabet. *Discrete Mathematics*, 219(1-3):187–194, 2000.

55 Abraham Lempel. On a homomorphism of the de Bruijn graph and its applications to the design of feedback shift registers. *IEEE Transactions on Computers*, 100(12):1204–1209, 1970.

56 Abraham Lempel. m-ary closed sequences. *Journal of Combinatorial Theory, Series A*, 10(3):253–258, 1971.

57 Willem Mantel. Resten van wederkerige reeksen. *Niew Archief voor Wiskunde*, 1:172–184, 1897.

58 Monroe H Martin. A problem in arrangements. *Bulletin of the American Mathematical Society*, 40(12):859–864, 1934.

59 Chris J Mitchell, Tuvi Etzion, and Kenneth G Paterson. A method for constructing decodable de Bruijn sequences. *IEEE Transactions on Information Theory*, 42(5):1472–1478, 1996.

60 Chris J. Mitchell and Peter B. Wild. Constructing orientable sequences. *arXiv preprint*, 2021. arXiv:2108.03069.

61 Johannes Mykkeltveit, Man-Keung Siu, and Po Tong. On the cycle structure of some nonlinear shift register sequences. *Information and control*, 43(2):202–215, 1979.

62 Abhinav Nellore, Austin Nguyen, and Reid F. Thompson. An invertible transform for efficient string matching in labeled digraphs. In Paweł Gawrychowski and Tatiana Starikovskaya, editors, *32nd Annual Symposium on Combinatorial Pattern Matching (CPM 2021)*, volume 191 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:14, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CPM.2021.20.

63 Nicolaas de Bruijn - biography. https://mathshistory.st-andrews.ac.uk/Biographies/De_Bruijn/. Accessed: 2021-07-23.

64 Kenneth G Paterson and Matthew JB Robshaw. Storage efficient decoding for a class of binary de Bruijn sequences. *Discrete mathematics*, 138(1-3):327–341, 1995.

65 AN Radchenko. *Code Rings and Their Use in Contactless Coding Devices.* PhD thesis, University of Leningrad, USSR, 1958.

66 AN Radchenko and VI Filippov. Shifting registers with logical feedback and their use as counting and coding devices. *Automation and Remote Control (English translation of Soviet Journal Automatika i Telemekhanika)*, 20:1467–1473, November 1959.

67 Christian Ronse. Feedback shift registers. *Lecture Notes in Computer Science*, 169, 1984.

68 Frank Ruskey, Carla Savage, and Terry Min Yih Wang. Generating necklaces. *Journal of Algorithms*, 13(3):414–430, 1992.

69 C Flye Saint-Marie. Solution to question nr. 48. *L'Intermédiaire des Mathématiciens*, 1894.

70 Joe Sawada and Aaron Williams. Practical algorithms to rank necklaces, Lyndon words, and de Bruijn sequences. *Journal of Discrete Algorithms*, 43:95–110, 2017.

71 Joe Sawada, Aaron Williams, and Dennis Wong. The lexicographically smallest universal cycle for binary strings with minimum specified weight. *Journal of Discrete Algorithms*, 28:31–40, 2014.

72 Joe Sawada, Aaron Williams, and Dennis Wong. Generalizing the classic greedy and necklace constructions of de Bruijn sequences and universal cycles. *the electronic journal of combinatorics*, pages P1–24, 2016.

73 Joe Sawada, Aaron Williams, and Dennis Wong. A surprisingly simple de Bruijn sequence construction. *Discrete Mathematics*, 339(1):127–131, 2016.

74 Joe Sawada, Aaron Williams, and Dennis Wong. A simple shift rule for k-ary de Bruijn sequences. *Discrete Mathematics*, 340(3):524–531, 2017.

75 Joe Sawada and Dennis Wong. Efficient universal cycle constructions for weak orders. *Discrete Mathematics*, 343(10):112022, 2020.

76 Man-Keung Siu and Po Tong. Generation of some de Bruijn sequences. *Discrete Mathematics*, 31(1):97–100, 1980.

77 R Stoneham. On (j, ε)-normality in the rational fractions. *Acta Arithmetica*, 16:221–238, 1970.

78 R Stoneham. On absolute (j, ε)-normality in the rational fractions with applications to normal numbers. *Acta Arithmetica*, 22:277–286, 1973.

79 R Stoneham. Normal recurring decimals, normal periodic systems,(j, ε)-normality, and normal numbers. *Acta Arithmetica*, 28(4):349–361, 1976.

80 RG Stoneham. The reciprocals of integral powers of primes and normal numbers. *Proceedings of the American Mathematical Society*, 15(2):200–208, 1964.

81 Jonathan Tuliani. De Bruijn sequences with efficient decoding algorithms. *Discrete Mathematics*, 226(1-3):313–336, 2001.

82 Srisa Chandra Vasu et al. *The Ashtadhyayi of Panini*, volume 6. Satyajnan Chaterji, 1897.

83 Dennis Wong. A new universal cycle for permutations. *Graphs and Combinatorics*, 33(6):1393–1399, 2017.

84 Jun-Hui Yang and Zong-Duo Dai. Construction of m-ary de Bruijn sequences. In *International Workshop on the Theory and Application of Cryptographic Techniques*, pages 357–363. Springer, 1992.

85 Michael Yoeli. *Nonlinear Feedback Shift Registers.* International Business Machines Corporation, Development Laboratories, Data Systems Division, 1961.

86 Michael Yoeli. Binary ring sequences. *The American Mathematical Monthly*, 69(9):852–855, 1962.

87 Michael Yoeli. Counting with nonlinear binary feedback shift registers. *IEEE Transactions on Electronic Computers*, pages 357–361, 1963.

88 Yunlong Zhu, Zuling Chang, Martianus Frederic Ezerman, and Qiang Wang. An efficiently generated family of binary de Bruijn sequences. *Discrete Mathematics*, 344(6):112368, 2021.

A Appendix: A brief history of de Bruijn sequences

The earliest known recorded de Bruijn sequence is the Sanskrit sutra yamātārājabhānasalagām, a mnemonic encoding all possible length-3 combinations of short and long vowels [49]. Historians have had some trouble placing when it was first conceived, but it may be over 2,500 years old [7], having appeared in work by the ancient Indian scholar Pāṇini (dates of birth and death unavailable). Little is known of Pāṇini beyond his foundational work *Aṣṭādhyāī* codifying Sanskrit grammatical structure [82].

The question of whether and how many binary de Bruijn sequences of every order exist was first posed by A. de Rivière (dates of birth and death unavailable) as problem 48 of [53] in 1894. That same year, in response to the problem, the number of binary de Bruijn sequences of every order was counted in [69] by Camille Flye Sainte-Marie (1834–1926), a member of the Mathematical Society of France who was affiliated with the French military throughout his career [11]. His work was quickly forgotten.

Monroe Harnish Martin (1907–2007) was first to prove the existence of de Bruijn sequences of any order for any alphabet size in his 1934 paper [58] by explicit construction, shortly before arriving at the University of Maryland, where he spent the rest of his eminent career. Without knowing of Sainte-Marie's work, Nicolaas Govert de Bruijn (1918–2012) [63] also counted the number of binary de Bruijn sequences of every order in his 1946 work [17]. Tatyana van Aardenne-Ehrenfest and de

Bruijn were first to prove the formula for the number of de Bruijn sequences of any order for any alphabet size in their 1951 paper [19]. It is notable that after receiving her PhD from the University of Leiden in 1931, van Ardenne-Ehrenfest (1905–1984) made this and further significant contributions to the mathematics of sequences despite never holding paid employment as a mathematician and working as a homemaker [16].

Sainte-Marie's work was ultimately rediscovered by the well-known MIT combinatorialist Richard Peter Stanley (1944–) [44], who brought it to the attention of de Bruijn, and in 1975, de Bruijn issued an acknowledgement [18] of the work. In this acknowledgement, de Bruijn noted that as early as 1897, Willem Mantel (dates of birth and death unavailable) showed how to construct de Bruijn sequences of any order for any alphabet size that is prime [57], also in response to A. de Rivière's problem.