New Successor Rules to Efficiently Produce Exponentially Many Binary de Bruijn Sequences

Zuling Chang
School of Mathematics and Statistics
Zhengzhou University
450001 Zhengzhou, China
zuling_chang@zzu.edu.cn

Martianus Frederic Ezerman
School of Physical and Mathematical Sciences
Nanyang Technological University
21 Nanyang Link, Singapore 637371
fredezerman@ntu.edu.sg

Pinhui Ke
Key Laboratory of
Network Security and Cryptology
Fujian Normal University
350117 Fuzhou, China
keph@fjnu.edu.cn

Qiang Wang
School of Mathematics and Statistics
Carleton University
1125 Colonel By Drive
Ottawa ON K1S 5B6, Canada
wang@math.carleton.ca

Abstract

We propose a new general criteria to design successor rules for binary de Bruijn sequences and show that prior fast algorithms based on successor rules are special instances. We efficiently generate exponentially many binary de Bruijn sequences for any given order n. Producing the next bit in each such sequence takes $O(n)$ memory and $O(n)$ time. We devise computational routines to confirm the claims.

1 Introduction

A 2^n-periodic binary sequence is a *binary de Bruijn sequence* of order n if every binary n-tuple occurs exactly once within each period. There are $2^{2^{n-1}-n}$ such sequences [1]. They appear in many guises, drawing the attention of researchers from varied backgrounds and interests. Being balanced and having maximum period [2, 3] make these sequences applicable in coding and communication systems. A subclass with properly calibrated nonlinearity property can also be useful in cryptography.

Experts have been using tools from diverse branches of mathematics to study their generations and properties, see, *e.g.*, the surveys in [4] and [5] for further details. Of enduring interest are methods that excel in three measures: fast, with low memory requirement, and capable of generating a large number of sequences. Known constructions

come with some trade-offs with respect to these measures. Notable examples include Lempel's D-Morphism [6], an approach via preference functions described in [7] and in [3], greedy algorithms with specific preferences, *e.g.*, in [8] and, more recently, in [9], as well as various fast generation proposals, *e.g.*, those in [10] and in [11].

The most popular construction approach is the *cycle joining method* (CJM) [3]. It serves as the foundation of many techniques. A main drawback of the CJM, in its most general form, is the amount of computation to be done prior to actually generating the sequences. Given a feedback shift register, one must first determine its cycle structure before finding the conjugate pairs to build the so-called adjacency graph. Enumerating the spanning trees comes next. Once these general and involved steps have been properly done, then generating a sequence, either randomly or based on a predetermined rule, is very efficient in both time and memory. The main advantage, if carried out in full, is the large number of output sequences, as illustrated in [12, Table 3].

There are fast algorithms that can be seen as applications of the CJM on specially chosen conjugate pairs and designated initial states. They often produce a very limited number of de Bruijn sequences. One can generate a de Bruijn sequence, named the `granddady` in [10], in $O(n)$ time and $O(n)$ space per bit. A related de Bruijn sequence, named the `grandmama`, was built in [11]. Huang gave another early construction that joins the cycles of the *complementing circulating register* (CCR) in [13]. Etzion and Lempel proposed some algorithms to generate de Bruijn sequences based on the *pure cycling register* (PCR) and the *pure summing register* (PSR) in [14]. Their algorithms generate a number, exponential in n, of sequences at the expense of higher memory requirement.

Jansen, Franx, and Boekee established a requirement to determine some conjugate pairs in [15], leading to another fast algorithm. In [16], Sawada, Williams, and Wong proposed a simple de Bruijn sequence construction, which turns out to be a special case of the method in [15]. Gabric *et al.* generalized the last two works to form simple successor rule frameworks in [17]. Further generalization to k-ary de Bruijn sequences in [18] and [20] followed. Zhu *et al.* in [19], building upon the framework in [17], proposed two efficient generic successor rules based on the properties of the feedback function $f(x_0, x_1, \ldots, x_{n-1}) = x_0 + x_1 + x_{n-1}$ for $n \geq 3$. Each rule produces at least 2^{n-3} binary de Bruijn sequences.

Our Contributions

We generate de Bruijn sequences by using novel relations and orders on the cycles in combination with suitable successor rules. We define new classes of successor rules and, then, prove that they generate, respectively, a number, exponential in n, of de Bruijn sequences. In particular, the number of generated sequences based on the PCR of order n is $2(n-1)(n-2)\ldots 1 = 2 \cdot (n-1)!$. The cost to output the next bit is $O(n)$ time and $O(n)$ space. Nearly all known successor rules in the literature generate only a handful of de Bruijn sequences each. The few previously available approaches that can generate an exponential number of de Bruijn sequences require more space than the ones we are proposing.

A high level explanation of our approach is as follows. We begin with the set of cycles produced by any nonsingular feedback shift register. To join all of these cycles into a single

cycle, *i.e.*, to obtain a binary de Bruijn sequence, one needs a valid successor rule that assigns a uniquely identified state in one cycle to a uniquely identified state in another cycle and ensure that all of the cycles are joined in the end. If the cycles are represented by the vertices of an adjacency graph, then producing a de Bruijn sequence in the CJM corresponds to finding a spanning tree in the graph. We identify several new relations and orders on both the cycles and on the states in each cycle. These ensure the existence of spanning trees in the corresponding adjacency graphs.

We collect preliminary notions and several useful known results in Section 2. Section 3 presents our new general criteria. Section 4 shows how to apply the criteria on the cycles of the PCR, leading to scores of new successor rules. The last section summarizes our contributions and lists some future directions.

2 Preliminaries

2.1 Basic Definitions

An *n-stage shift register* is a circuit of n consecutive storage units, each containing a bit. The circuit is clock-regulated, shifting the bit in each unit to the next stage as the clock pulses. A shift register generates a binary code if one adds a feedback loop that outputs a new bit s_n based on the n bits $\mathbf{s}_0 = s_0, \ldots, s_{n-1}$, called an *initial state* of the register. The corresponding Boolean *feedback function* $f(x_0, \ldots, x_{n-1})$ outputs s_n on input $\mathbf{s}_0$. A *feedback shift register* (FSR) outputs a binary sequence $\mathbf{s} = \{s_i\} = s_0, s_1, \ldots, s_n, \ldots$ that satisfies the recursive relation $s_{n+\ell} = f(s_\ell, s_{\ell+1}, \ldots, s_{\ell+n-1})$ for $\ell = 0, 1, 2, \ldots$. For $N \in \mathbb{N}$, if $s_{i+N} = s_i$ for all $i \geq 0$, then $\mathbf{s}$ is *N-periodic* or *with period N* and one writes $\mathbf{s} = (s_0, s_1, s_2, \ldots, s_{N-1})$. The least among all periods of $\mathbf{s}$ is called the *least period* of $\mathbf{s}$.

We say that $\mathbf{s}_i = s_i, s_{i+1}, \ldots, s_{i+n-1}$ is *the i^{th} state* of $\mathbf{s}$. Its *predecessor* is $\mathbf{s}_{i-1}$ while its *successor* is $\mathbf{s}_{i+1}$. For $s \in \mathbb{F}_2$, let $\bar{s} := s + 1 \in \mathbb{F}_2$. Extending the definition to any binary vector or sequence $\mathbf{s} = s_0, s_1, \ldots, s_{n-1}, \ldots$, let $\overline{\mathbf{s}} := \overline{s_0}, \overline{s_1}, \ldots, \overline{s_{n-1}}, \ldots$. An arbitrary state $\mathbf{v} = v_0, v_1, \ldots, v_{n-1}$ of $\mathbf{s}$ has $\widehat{\mathbf{v}} := \overline{v_0}, v_1, \ldots, v_{n-1}$ and $\widetilde{\mathbf{v}} := v_0, \ldots, v_{n-2}, \overline{v_{n-1}}$ as its *conjugate* state and *companion* state, respectively. Hence, $(\mathbf{v}, \widehat{\mathbf{v}})$ is a *conjugate pair* and $(\mathbf{v}, \widetilde{\mathbf{v}})$ is a *companion pair*.

For any FSR, distinct initial states generate distinct sequences. There are 2^n distinct sequences generated from an FSR with feedback function $f(x_0, x_1, \ldots, x_{n-1})$. They are periodic if and only if f is *nonsingular*, *i.e.*, f expressible as $f(x_0, x_1, \ldots, x_{n-1}) = x_0 + h(x_1, \ldots, x_{n-1})$, for some Boolean function $h(x_1, \ldots, x_{n-1})$ whose domain is $\mathbb{F}_2^{n-1}$ [3, p. 116]. All feedback functions in this paper are nonsingular. An FSR is *linear* or an LFSR if its feedback function has the form $f(x_0, x_1, \ldots, x_{n-1}) = x_0 + c_1 x_1 + \ldots + c_{n-1} x_{n-1}$, with $c_i \in \mathbb{F}_2$, and its *characteristic polynomial* is $f(x) = x^n + c_{n-1} x^{n-1} + \cdots + c_1 x + 1 \in \mathbb{F}_2[x]$. Otherwise, it is *nonlinear* or an NLFSR. Further properties of LFSRs are treated in, *e.g.*, [22] and [23].

For an N-periodic sequence $\mathbf{s}$, the *left shift operator* L maps $(s_0, s_1, \ldots, s_{N-1}) \mapsto (s_1, s_2, \ldots, s_{N-1}, s_0)$, with the convention that L^0 fixes $\mathbf{s}$. The *right shift operator* R is

defined analogously. The set

$$[\mathbf{s}] := \{\mathbf{s}, L\mathbf{s}, \ldots, L^{N-1}\mathbf{s}\} = \{\mathbf{s}, R\mathbf{s}, \ldots, R^{N-1}\mathbf{s}\} \tag{1}$$

is a *shift equivalent class*. Sequences in the same shift equivalent class correspond to the same cycle in the state diagram of FSR [22]. We call a periodic sequence in a shift equivalent class a *cycle*. If an FSR with feedback function f generates r disjoint cycles $C_1, C_2, \ldots, C_r$, then its *cycle structure* is $\Omega(f) = \{C_1, C_2, \ldots, C_r\}$. A cycle can also be viewed as a set of consecutive n-stage states in the corresponding periodic sequence. Since the cycle are disjoint, we can write $\mathbb{F}_2^n = C_1 \cup C_2 \cup \ldots \cup C_r$. When $r = 1$, the corresponding FSR is of *maximal length* and its output is a de Bruijn sequence of order n.

The *weight* of an N-periodic cycle C, denoted by $\mathrm{wt}(C)$, is $|\{0 \le j \le N-1 : c_j = 1\}|$. Similarly, the weight of a state is the number of 1s in the state. The lexicographically least N-stage state in any N-periodic cycle is called its *necklace*. As discussed in, *e.g.*, [24] and [17], there is a fast algorithm that determines whether or not a state is a necklace in $O(N)$ time. In fact, one can efficiently sort all distinct states in C. The standard `python` implementation is `timsort` [25]. It was developed by Tim Peters based on McIlroy's techniques in [26]. In the worst case, its space and time complexities are $O(N)$ and $O(N \log N)$ respectively. A closely related proposal, by Buss and Knop, is in [27].

Given disjoint cycles C and C' in $\Omega(f)$ with the property that some state $\mathbf{v}$ in C has its conjugate state $\widehat{\mathbf{v}}$ in C', interchanging the successors of $\mathbf{v}$ and $\widehat{\mathbf{v}}$ joins C and C' into a cycle whose feedback function is

$$\widehat{f} := f(x_0, x_1, \ldots, x_{n-1}) + \prod_{i=1}^{n-1}(x_i + \overline{v_i}). \tag{2}$$

Similarly, if the companion states $\mathbf{v}$ and $\widetilde{\mathbf{v}}$ are in two distinct cycles, then interchanging their predecessors joins the two cycles. If this process can be continued until all cycles that form $\Omega(f)$ merge into a single cycle, then we obtain a de Bruijn sequence. The CJM is, therefore, predicated upon knowing the cycle structure of $\Omega(f)$ and is closely related to a graph associated to the FSR.

Given an FSR with feedback function f, its *adjacency graph* G_f, or simply G if f is clear, is an undirected multigraph whose vertices correspond to the cycles of $\Omega(f)$. The number of edges between two vertices is the number of shared conjugate (or companion) pairs, with each edge labelled by a specific pair. It is well-known that there is a bijection between the set of spanning trees of G and the set of all inequivalent de Bruijn sequences constructible by the CJM on input f.

A *pure cycling register* (PCR) *of order* n is an LFSR with feedback function and characteristic polynomial

$$f_{\mathrm{PCR}}(x_0, x_1, \ldots, x_{n-1}) = x_0 \text{ and } f_{\mathrm{PCR}}(x) = x^n + 1. \tag{3}$$

Let $\phi(\cdot)$ be the Euler totient function. The number of distinct cycles in $\Omega(f_{\mathrm{PCR}})$ is known, *e.g.*, from [3], to be

$$Z_n := \frac{1}{n}\sum_{d|n} \phi(d) 2^{\frac{n}{d}}. \tag{4}$$

By definition, all states in any given n-periodic cycle $C_{\mathrm{PCR}} := (c_0, c_1, \ldots, c_{n-1}) \in \Omega(f_{\mathrm{PCR}})$ have the same number of ones.

2.2 Jansen-Franx-Boekee (JFB) Algorithm

In [15], Jansen *et al.* proposed an algorithm to generate de Bruijn sequences by the CJM. Suppose that the FSR with a feedback function $f(x_0, x_1, \ldots, x_{n-1})$ is given. They defined the *cycle representative* of any cycle of the FSR to be its lexicographically smallest n-stage state. If the FSR is the PCR of order n, then it is clear that the cycle representative is its necklace. Based on the cycle representative, we can impose an order on the cycles. For arbitrary cycles C and C' in Ω_f, we say that $C \prec_{\mathrm{lex}} C'$ if and only if the cycle representative of C is lexicographically less than that of C'. This *lexicographic order* defines a total order on the cycles of the said PCR.

On current state $\mathbf{s}_i = s_i, s_{i+1}, \ldots, s_{i+n-1}$, the next state $\mathbf{s}_{i+1} = s_{i+1}, s_{i+2}, \ldots, s_{i+n}$ is produced based on the assignment rule in Algorithm 1. The correctness of the JFB Algorithm rests on the fact that the cycle representative in any cycle C_1 which does not contain the all zero state $0, \ldots, 0$ is unique. Its companion state is guaranteed to be in another cycle C_2 with $C_2 \prec_{\mathrm{lex}} C_1$. This ensures that we have a spanning tree and, hence, the resulting sequence must be de Bruijn.

Algorithm 1 Jansen-Franx-Boekee (JFB) Algorithm

1: **if** $\mathbf{s}_i = s_i, 0, \ldots, 0$ **then**
2: $\quad \mathbf{s}_{i+1} \leftarrow 0, \ldots, 0, s_i + 1$
3: **else**
4: $\quad$ **if** $s_{i+1}, \ldots, s_{i+n-1}, 0$ or $s_{i+1}, \ldots, s_{i+n-1}, 1$ is a cycle representative **then**
5: $\quad\quad \mathbf{s}_{i+1} \leftarrow s_{i+1}, \ldots, s_{i+n-1}, f(s_i, \ldots, s_{i+n-1}) + 1$
6: $\quad$ **else**
7: $\quad\quad \mathbf{s}_{i+1} \leftarrow s_{i+1}, \ldots, s_{i+n-1}, f(s_i, \ldots, s_{i+n-1})$

The main task of keeping track of the cycle representatives in Algorithm 1 may require a lot of time if the least periods of the cycles are large. For cases where all cycles produced by a given FSR have small least periods, *e.g.*, in the case of the PCR or the PSR of order n, the algorithm generates de Bruijn sequences very efficiently. The space complexity is $O(n)$ and the time complexity lies between $O(n)$ and $O(n \log n)$ to output the next bit.

Sawada *et al.* proposed a simple fast algorithm on the PCR to generate a de Buijn sequence [16]. Their algorithm is a special case of the JFB Algorithm. Later, in [17], Gabric and the authors of [16] considered the PCR and the complemented PCR, also known as the CCR, and proposed several fast algorithms to generate de Bruijn sequences by ordering the cycles lexicographically according to their respective necklace and co-necklace. They replace the generating algorithm by some *successor rule*.

The general thinking behind the approach is as follows. Given an FSR with a feedback function $f(x_0, x_1, \ldots, x_{n-1})$, let A label some condition which guarantees that the resulting

sequence is de Bruijn. For any state $\mathbf{c} := c_0, c_1, \ldots, c_{n-1}$, the successor rule assigns

$$\rho_A(\mathbf{c}) = \begin{cases} f(\mathbf{c}) + 1, \text{ if } \mathbf{c} \text{ satisfies } A, \\ f(\mathbf{c}), \text{ otherwise.} \end{cases} \tag{5}$$

The usual successor of $\mathbf{c}$ is $c_1, \ldots, c_{n-1}, f(c_0, \ldots, c_{n-1})$. Every time $\mathbf{c}$ satisfies Condition A, however, its successor is *redefined* to be $c_1, \ldots, c_{n-1}, f(c_0, \ldots, c_{n-1}) + 1$. The last bit of the successor is *the complement* of the last bit of the usual successor under the feedback function f. The basic idea of a successor rule is to determine spanning trees in G_f by identifying a suitable Condition A. Seen in this light, the rule implements the CJM by assigning successors to carefully selected states.

3 New General Criteria for Successor Rules

New successor rules for de Bruijn sequences can be established by defining some relations or orders on the cycles of FSRs with special properties to construct spanning trees in G_f. This section proves a general criteria that such rules must meet. The criteria will be applied successfully, in latter sections, to the PCR and the PSR of any order n. The generality of the criteria allows for further studies to be conducted on the feasibility of using broader families of FSRs for fast generation of de Bruijn sequences.

We adopt set theoretic definitions and facts from [21]. Given Ω_f, we define a binary *relation* $\prec$ on $\Omega_f := \{C_1, C_2, \ldots, C_r\}$ as a set of ordered pairs in Ω_f. If $C \prec C$ for every $C \in \Omega_f$, then $\prec$ is said to be *reflexive*. Let $1 \leq i, j, k \leq r$. We say that $\prec$ is *transitive* if $C_i \prec C_j$ and $C_j \prec C_k$, together, imply $C_i \prec C_k$. It is *symmetric* if $C_i \prec C_j$ implies $C_j \prec C_i$ and *antisymmetric* if the validity of both $C_i \prec C_j$ and $C_j \prec C_i$ implies $C_i = C_j$.

The relation $\prec$ is called a *preorder* on Ω_f if it is reflexive and transitive. It becomes a *partial order* if it is an antisymmetric preorder. If $\prec$ is a partial order with either $C_i \prec C_j$ or $C_j \prec C_i$, for any C_i and C_j, then it is a *total order*. A totally ordered set Ω_f is called a *chain*. Hence, we can now say that $\prec_{\text{lex}}$ defined in Subsection 2.2 is a total order on the corresponding chain Ω_f.

Theorem 1. *Given an FSR with feedback function f, let $\prec$ be a transitive relation on $\Omega(f) := \{C_1, C_2, \ldots, C_r\}$ and let $1 \leq i, j \leq r$.*

1. *Let there be a unique cycle C with the property that $C \prec C'$ for any cycle $C' \neq C$,* i.e., *C is the unique smallest cycle in $\Omega(f)$. Let ρ be a successor rule that can be well-defined as follows. If any cycle $C_i \neq C$ contains a uniquely defined state whose successor can be assigned by ρ to be a state in a cycle $C_j \neq C_i$ with $C_j \prec C_i$, then ρ generates a de Bruijn sequence.*

2. *Let there be a unique cycle C with the property that $C' \prec C$ for any cycle $C' \neq C$,* i.e., *C is the unique largest cycle in $\Omega(f)$. Let ρ be a successor rule that can be well-defined as follows. If any cycle $C_i \neq C$ contains a uniquely defined state whose successor can be assigned by ρ to be a state in a cycle $C_j \neq C_i$ with $C_i \prec C_j$, then ρ generates a de Bruijn sequence.*

Proof. We prove the first case by constructing a rooted tree whose vertices are all of the cycles in $\Omega(f)$. This exhibits a spanning tree in the adjacency graph of the FSR according to the specified successor rule. The second case can be similarly argued.

Based on the condition set out in the first case, each $C_i \neq C$ contains a unique state whose assigned successor under ρ is in $C_j \neq C_i$, revealing that C_i and C_j are adjacent. Since $C_j \prec C_i$, we direct the edge **from** C_i **to** C_j. It is easy to check that, except for C whose outdegree is 0, each vertex has outdegree 1. Since $\prec$ is transitive, there is a unique path from the vertex to C. We have thus built a spanning tree rooted at C. □

There are two tasks to carry out in using Theorem 1. First, one must define a suitable transitive relation among the cycles to obtain the unique smallest or largest cycle C. The second task is to determine the unique state in each cycle. A sensible approach is to designate a state $\mathbf{v}$ as the *benchmark state* in each cycle C. We then uniquely define a state $\mathbf{w}$ in C with respect to the benchmark state. The cycle representative, *i.e.*, the necklace in the PCR, is the most popular choice for $\mathbf{v}$. In this paper we mainly use the necklace as the benchmark state in each cycle.

4 Successor Rules from Pure Cycling Registers

In applying the criteria in Theorem 1 to the PCR of any order n, it is good to consider the positions of the states in each cycle *relative to its necklace* by ordering the states in several distinct manners. This general route is chosen since we can check whether or not a state is a necklace in $O(n)$ time and $O(n)$ space. If the relative position of a state to the necklace is efficient to pinpoint, then the derived successor rule also runs efficiently.

4.1 The Weight Relation on the Pure Cycling Register

The cycles of the PCR share a nice property. All of the states in any cycle C are shift-equivalent and share the same weight $\mathrm{wt}(C)$. Hence, we can define a *weight relation* on the cycles based simply on their respective weights. For cycles $C_i \neq C_j$, we say that $C_i \prec_{\mathrm{wt}} C_j$ if and only if $\mathrm{wt}(C_i) < \mathrm{wt}(C_j)$. The relation $\prec_{\mathrm{wt}}$ is not even a preorder, making it differs qualitatively from the lexicographic order.

Example 2. The PCR of order 6 generates $C_1 := (001001)$ and $C_2 := (000111)$, with $C_1 \succ_{\mathrm{lex}} C_2$, since the necklace 001001 in C_1 is lexicographically larger than the necklace 000111 in C_2. In the weight relation, $C_1 \prec_{\mathrm{wt}} C_2$ since $\mathrm{wt}(C_1) = 2 < 3 = \mathrm{wt}(C_2)$. □

The following successor rules rely on the weight relation.

Theorem 3. *For the PCR of order n, if a successor rule $\rho(x_0, x_1, \ldots, x_{n-1})$ satisfies one of the following conditions, then it generates a de Bruijn sequence.*

1. *For any $C_i \neq (0)$, the rule ρ exchanges the successor of a uniquely determined state $\mathbf{v}_i \in C_i$ with a state $\mathbf{w}_j$ in C_j, where $C_j \prec_{\mathrm{wt}} C_i$.*

2. *For any $C_i \neq (1)$, the rule ρ exchanges the successor of a uniquely determined state $\mathbf{v}_i \in C_i$ with a state $\mathbf{w}_j$ in C_j, where $C_i \prec_{\mathrm{wt}} C_j$.*

Proof. To prove the first case, note that $(0) \prec_{\mathrm{wt}} C_i$ for any $C_i \neq (0)$ in $\Omega(f_{\mathrm{PCR}})$. By the stated condition, C_i contains a unique state $\mathbf{v}_i$ such that its conjugate $\mathbf{w}_j := \widehat{\mathbf{v}}_i$ is in C_j and $\mathrm{wt}(C_j) < \mathrm{wt}(C_i)$. The successor rule ρ satisfies the criteria in Theorem 1. The proof for the second case is similar. □

Theorem 3 reduces the task to generate de Bruijn sequences by using ρ to performing one of two procedures. The first option is to find the *uniquely determined state* $\mathbf{v}_i \in C_i \neq (0)$ whose conjugate state $\widehat{\mathbf{v}}_i$ is guaranteed to be in C_j with $\mathrm{wt}(C_j) < \mathrm{wt}(C_i)$. The second option is to find the *uniquely determined state* $\mathbf{v}_i$ in each $C_j \neq (1)$ whose conjugate state $\widehat{\mathbf{v}}_i$ is guaranteed to be in C_j with $\mathrm{wt}(C_j) > \mathrm{wt}(C_i)$. If, for every C_i, its $\mathbf{v}_i$ can be determined quickly, then generating the de Bruijn sequence is efficient. Following the two cases in Theorem 3, the rule ρ comes in two forms. Let $\mathbf{c} := c_0, c_1, \ldots, c_{n-1}$.

First, let $\mathcal{A}$ be

> In $C := (0, c_1, \ldots, c_{n-1})$, the uniquely determined state $\mathbf{v}$ is $0, c_1, \ldots, c_{n-1}$. Its conjugate $\widehat{\mathbf{v}}$ has $\mathrm{wt}(\widehat{\mathbf{v}}) > \mathrm{wt}(\mathbf{v})$, which implies $\widehat{\mathbf{v}}$ is in C' with $C \prec_{\mathrm{wt}} C'$.

We confirm that the relevant requirement in Theorem 3 is met by

$$\rho_{\mathcal{A}}(\mathbf{c}) = \begin{cases} \overline{c_0}, \text{ if } 0, c_1, \ldots, c_{n-1} \text{ satisfies } \mathcal{A}, \\ c_0, \text{ otherwise.} \end{cases} \tag{6}$$

Second, let $\mathcal{B}$ be

> In $C := (c_1, \ldots, c_{n-1}, 1)$, the uniquely determined state $\mathbf{v}$ is $c_1, \ldots, c_{n-1}, 1$. Its companion $\widetilde{\mathbf{v}}$ has $\mathrm{wt}(\widetilde{\mathbf{v}}) < \mathrm{wt}(\mathbf{v})$, which means that $\widetilde{\mathbf{v}}_i$ is in C' with $C' \prec_{\mathrm{wt}} C$.

Hence, the successor rule

$$\rho_{\mathcal{B}}(\mathbf{c}) = \begin{cases} \overline{c_0}, \text{ if } c_1, \ldots, c_{n-1}, 1 \text{ satisfies } \mathcal{B}, \\ c_0, \text{ otherwise,} \end{cases} \tag{7}$$

fulfills the requirement in Theorem 3.

Based on $\mathcal{A}$ and $\mathcal{B}$, valid successor rules can be easily formulated once we manage to determine a unique state whose first bit is 0, respectively, whose last bit is 1, in each $C \neq (1)$, respectively, $C \neq (0)$.

4.2 Under the Shift Order

Imposing a *shift order* on the states in a given cycle yields a lot of feasible successor rules. We call a state whose first entry is 0 a *leading zero state* or an LZ state in short. Analogously, a state whose last entry is 1 is said to be an *ending one state* or an EO state.

The necklace in a given cycle $(c_0, c_1, \ldots, c_{n-1}) \neq (1)$ must begin with 0, *i.e.*, its necklace is an LZ state. Here we define a *special left shift operator*, denoted by L_{lz}. Applied on a given LZ state $\mathbf{v} := 0, c_1, \ldots, c_{n-1}$ the operator L_{lz} outputs the first LZ state obtained by

consecutive left shifts on $\mathbf{v}$. More formally, $L_{\text{lz}}\,\mathbf{v} := \mathbf{v}$ if $c_1, \ldots, c_{n-1} = 1, \ldots, 1$. Otherwise, let $1 \le j < n$ be the least index such that $c_j = 0$. Then

$$L_{\text{lz}}\,\mathbf{v} := 0, c_{j+1}, \ldots, c_{n-1}, 0, c_1, \ldots, c_{j-1}.$$

Similarly, the necklace in any $C \neq (0)$ must end with 1, *i.e.*, it is an EO state. Given a state $\mathbf{u} := c_1, \ldots, c_{n-1}, 1$, the special operator L_{eo} fixes $\mathbf{u}$ if $c_1, \ldots, c_{n-1} := 0, \ldots, 0$. Otherwise, let $1 \le j < n$ be the least index such that $c_j = 1$. Then

$$L_{\text{eo}}\,\mathbf{u} := c_{j+1}, \ldots, c_{n-1}, 1, c_1, \ldots, c_{j-1}, 1.$$

In other words, $L_{\text{eo}}\,\mathbf{u}$ is the first EO state found upon consecutive left shifts on $\mathbf{u}$.

For these two special operators, the convention is to let

$$\begin{cases} L_{\text{lz}}^0\,\mathbf{v} = \mathbf{v}, \\ L_{\text{eo}}^0\,\mathbf{u} = \mathbf{u}, \end{cases} \quad \text{and} \quad \begin{cases} L_{\text{lz}}^k\,\mathbf{v} = L_{\text{lz}}^{k-1}(L_{\text{lz}}\,\mathbf{v}), \\ L_{\text{eo}}^k\,\mathbf{u} = L_{\text{eo}}^{k-1}(L_{\text{eo}}\,\mathbf{u}), \end{cases} \quad \text{for } k > 0.$$

Proposition 4. *With arbitrarily chosen $2 \le t \le n$, we let $1 = k_1 < k_2 < \cdots < k_t = n+1$ and $k_{t-1} < n$. For a state $\mathbf{c} := c_0, c_1, \ldots, c_{n-1}$, let $\mathbf{v} := 0, c_1, \ldots, c_{n-1}$ and $\mathbf{u} := c_1, \ldots, c_{n-1}, 1$. The following two successor rules generate de Bruijn sequences of order n.*

$$\rho_{\text{lz}}(\mathbf{c}) = \begin{cases} \overline{c_0}, \text{ if } k_i \le \mathrm{wt}(\overline{\mathbf{v}}) < k_{i+1} \\ \quad \text{for some } i \text{ and} \\ \quad L_{\text{lz}}^{k_i-1}\,\mathbf{v} \text{ is a necklace}, \\ c_0, \text{ otherwise.} \end{cases} \qquad \rho_{\text{eo}}(\mathbf{c}) = \begin{cases} \overline{c_0}, \text{ if } k_i \le \mathrm{wt}(\mathbf{u}) < k_{i+1} \\ \quad \text{for some } i \text{ and} \\ \quad L_{\text{eo}}^{k_i-1}\,\mathbf{u} \text{ is a necklace}, \\ c_0, \text{ otherwise.} \end{cases} \tag{8}$$

In Proposition 4 we let $k_t = n+1$ for consistency since $\mathrm{wt}(\overline{\mathbf{v}}) = n$ in $C = (0)$ and $\mathrm{wt}(\mathbf{u}) = n$ in $C = (1)$. Each of these special cycles has only a single state. The reason to have $k_{t-1} < n$ is then clear. The correctness of Proposition 4 comes from Theorem 3 and the fact that the state satisfying the respective conditions in ρ_{lz} and ρ_{eo} is uniquely determined in the corresponding cycle.

Proposition 5. *Each of the successor rules ρ_{lz} in (8) generates 2^{n-2} de Bruijn sequences of order n.*

Proof. We supply the proof for ρ_{lz} in (8), the other case being similar to argue. For each $1 \le \ell < n$, there exists at least one cycle of the PCR of order n having ℓ distinct LZ states. To verify existence, one can, *e.g.*, inspect the cycle $(\underbrace{00\ldots0}_{\ell}\underbrace{11\ldots1}_{n-\ell})$ for each $1 \le \ell < n$. On the other hand, taking all possible $2 \le t \le n$, there are 2^{n-2} distinct sets $\{1 = k_1, k_2, \ldots, k_{t-1}, k_t = n+1\}$ with $k_{t-1} < n$. Distinct sets provide distinct successor rules, producing 2^{n-2} inequivalent de Bruijn sequences in total. □

We are not quite done yet. Here are two more general successor rules whose validity can be routinely checked.

Proposition 6. *Let k be a nonnegative integer. For a state $\mathbf{c} := c_0, c_1, \ldots, c_{n-1}$, let $\mathbf{v} := 0, c_1, \ldots, c_{n-1}$ and $\mathbf{u} := c_1, \ldots, c_{n-1}, 1$. The following successor rules generate de Bruijn sequences of order n.*

$$\rho(\mathbf{c}) = \begin{cases} \overline{c_0}, \text{ if } L_{\mathrm{lz}}^k \mathbf{v} \text{ is a necklace,} \\ c_0, \text{ otherwise.} \end{cases} \qquad \rho(\mathbf{c}) = \begin{cases} \overline{c_0}, \text{ if } L_{\mathrm{eo}}^k \mathbf{u} \text{ is a necklace,} \\ c_0, \text{ otherwise.} \end{cases} \tag{9}$$

Proposition 7. *The number of distinct de Bruijn sequences of order n produced by each of the rules in (9) is*

$$\operatorname{lcm}(1, 2, \ldots, n-1) \geq (n-1)\binom{n-2}{\lfloor \frac{n-2}{2} \rfloor} \geq 2^{n-2}. \tag{10}$$

Proof. We supply the counting for the successor rule in (9). We know from the proof of Proposition 5 that, for each $1 \leq \ell < n$, there exists at least one cycle of the PCR of order n having ℓ distinct LZ states. For a given k, we construct the system of congruences

$$\{k \equiv a_i \pmod{i} \text{ for } i \in \{1, 2, \ldots, n-1\}\}. \tag{11}$$

The number of resulting distinct de Bruijn sequences of order n is equal to the number of solvable systems of congruences in (11). The sequences are distinct because different nonempty subsets of $\{a_1, \ldots, a_{n-1}\}$, whose corresponding systems are solvable, lead to different choices for the uniquely determined states in the respective cycles. By a generalized Chinese Remainder Algorithm in [28, Section 2.4], the number is $\operatorname{lcm}(1, 2, \ldots, n-1)$. From [29, Section 2] we get the lower bounds $(n-1)\binom{n-2}{\lfloor \frac{n-2}{2} \rfloor} \geq 2^{n-2}$. □

Proposition 6 includes the constructions of de Bruijn sequences from the PCR of order n in [17] as special cases. Taking $k \in \{0, 1, \operatorname{lcm}(1, 2, \ldots, n-1) - 1\}$ in the first rule in (9), for instance, outputs three sequences, including the PCR4 in [17] and `granddaddy`. Using the second rule in (9) with $k \in \{0, 1, \operatorname{lcm}(1, 2, \ldots, n-1) - 1\}$ yields sequences that include PCR3 (J1) in [17] and `grandmama`.

For the successor rules in Propositions 4 and 6, generating the next bit means checking if a state is a cycle's necklace by repeated simple left shifts. This can be done in $O(n)$ time and $O(n)$ space. We generalize Proposition 6 to define more successor rules.

Theorem 8. *Let $g(k) : \{1, 2, \ldots, n\} \mapsto \{0, 1, \ldots, k-1\}$ be an arithmetic function. As before, for any $\mathbf{c} := c_0, c_1, \ldots, c_{n-1}$, let $\mathbf{v} := 0, c_1, \ldots, c_{n-1}$ and $\mathbf{u} := c_1, \ldots, c_{n-1}, 1$. The following successor rules generate de Bruijn sequences of order n.*

$$\rho_{\mathrm{lz}}^g(\mathbf{c}) = \begin{cases} \overline{c_0}, \text{ if } L_{\mathrm{lz}}^{g(\mathrm{wt}(\overline{\mathbf{v}}))} \mathbf{v} \text{ is a necklace,} \\ c_0, \text{ otherwise.} \end{cases} \qquad \rho_{\mathrm{eo}}^g(\mathbf{c}) = \begin{cases} \overline{c_0}, \text{ if } L_{\mathrm{eo}}^{g(\mathrm{wt}(\mathbf{u}))} \mathbf{u} \text{ is a necklace,} \\ c_0, \text{ otherwise.} \end{cases} \tag{12}$$

For a cycle with $1 \leq \ell \leq n-1$ distinct LZ states, there are ℓ distinct ways to choose the uniquely determined state according to $g(\ell)$. The counting for ℓ distinct EO states is identical. It is then straightforward to confirm that each successor rule in Theorem 8 can generate $(n-1)!$ distinct de Bruijn sequences of order n by using all possible $g(\ell)$.

5 Conclusions

We have proposed a general design criteria for feasible successor rules. They perform the cycle joining method to output binary de Bruijn sequences. The focus of our demonstration is on their efficacy and efficiency when applied to the pure cycling register (PCR) of any order $n \geq 3$. Going beyond the often explored route of relying on the lexicographic ordering of the cycles, we have shown that many transitive relations can also be used to order the cycles. We have enumerated the respective output sizes of various specific successor rules that can be validly defined based on the general criteria. A straightforward complexity analysis has confirmed that generating the next bit in each resulting sequence is efficient.

We assert that the criteria we propose here can be applied to *all nonsingular FSRs*. If a chosen FSR has cycles with small least periods, then the complexity to produce the next bit can be kept low. Interested readers are invited to come up with feasible successor rules for their favourite FSRs. We intend to do the same and to further look into, among others, the cryptographic properties of the binary de Bruijn sequences produced by more carefully designed successor rules.

References

[1] N. G. de Bruijn, "A combinatorial problem," *Koninklijke Nederlandse Akademie v. Wetenschappen*, vol. 49, pp. 758–764, 1946.

[2] A. H. Chan, R. A. Games, and E. L. Key, "On the complexities of de Bruijn sequences," *J. Combinat. Theory, Ser. A*, vol. 33, no. 3, pp. 233 – 246, 1982.

[3] S. W. Golomb, *Shift Register Sequences*, 3rd ed. World Scientific, 2017.

[4] A. Ralston, "De Bruijn sequences - a model example of the interaction of discrete mathematics and computer science," *Math. Mag.*, vol. 55, no. 3, pp. 131–143, 1982.

[5] H. Fredricksen, "A survey of full length nonlinear shift register cycle algorithms," *SIAM Review*, vol. 24, no. 2, pp. 195–221, 1982.

[6] A. Lempel, "On a homomorphism of the de Bruijn graph and its applications to the design of feedback shift registers," *IEEE Trans. Comput.*, vol. C-19, no. 12, pp. 1204–1209, Dec 1970.

[7] A. Alhakim, "Spans of preference functions for de Bruijn sequences," *Discrete Appl. Math.*, vol. 160, no. 7, pp. 992 – 998, 2012.

[8] M. H. Martin, "A problem in arrangements," *Bull. Amer. Math. Soc.*, vol. 40, no. 12, pp. 859–865, Dec 1934.

[9] Z. Chang, M. F. Ezerman, and A. A. Fahreza, "On greedy algorithms for binary de Bruijn sequences," *Appl. Algebra Eng. Commun.*, vol. 33, pp. 523–550, Nov 2022.

[10] H. Fredricksen, "Generation of the Ford sequence of length 2^n, n large," *J. Combinat. Theory, Ser. A*, vol. 12, no. 1, pp. 153–154, Jan 1972.

[11] P. B. Dragon, O. I. Hernandez, J. Sawada, A. Williams, and D. Wong, "Constructing de Bruijn sequences with co-lexicographic order: The k-ary grandmama sequence,"

Eur. J. Comb., vol. 72, pp. 1–11, 2018.

[12] Z. Chang, M. F. Ezerman, S. Ling, and H. Wang, "On binary de Bruijn sequences from LFSRs with arbitrary characteristic polynomials," *Des. Codes Cryptogr.*, vol. 87, no. 5, pp. 1137–1160, May 2019.

[13] Y. Huang, "A new algorithm for the generation of binary de Bruijn sequences," *J. Algorithms*, vol. 11, no. 1, pp. 44–51, Mar 1990.

[14] T. Etzion and A. Lempel, "Algorithms for the generation of full-length shift-register sequences," *IEEE Trans. Inform. Theory*, vol. 30, no. 3, pp. 480–484, May 1984.

[15] C. Jansen, W. Franx, and D. Boekee, "An efficient algorithm for the generation of de Bruijn cycles," *IEEE Trans. Inform. Theory*, vol. 37, no. 5, pp. 1475–1478, 1991.

[16] J. Sawada, A. Williams, and D. Wong, "A surprisingly simple de Bruijn sequence construction," *Discrete Math.*, vol. 339, no. 1, pp. 127–131, Jan 2016.

[17] D. Gabric, J. Sawada, A. Williams, and D. Wong, "A framework for constructing de Bruijn sequences via simple successor rules," *Discrete Math.*, vol. 341, no. 11, pp. 2977–2987, Nov 2018.

[18] J. Sawada, A. Williams, and D. Wong, "A simple shift rule for k-ary de Bruijn sequences," *Discrete Math.*, vol. 340, no. 3, pp. 524–531, Mar 2017.

[19] Y. Zhu, Z. Chang, M. F. Ezerman, and Q. Wang, "An efficiently generated family of binary de Bruijn sequences," *Discrete Math.*, vol. 344, no. 6, 112368, Jun 2021.

[20] D. Gabric, J. Sawada, A. Williams, and D. Wong, "A successor rule framework for constructing k-ary de Bruijn sequences and universal cycles," *IEEE Trans. Inform. Theory*, vol. 66, no. 1, pp. 679–687, Jan 2020.

[21] P. Halmos, *Naive Set Theory*, ser. Undergraduate Texts in Mathematics. Springer, New York, 1974.

[22] S. W. Golomb and G. Gong, *Signal Design for Good Correlation: for Wireless Communication, Cryptography, and Radar*. Cambridge Univ. Press, New York, 2004.

[23] R. Lidl and H. Niederreiter, *Finite Fields*, ser. Encyclopaedia of Mathematics and its Applications. Cambridge Univ. Press, New York, 1997.

[24] K. S. Booth, "Lexicographically least circular substrings," *Inform. Process. Lett.*, vol. 10, no. 4-5, pp. 240–242, Jul 1980.

[25] T. Peters. [python-dev] Sorting. [Online]. Available: https://mail.python.org/pipermail/python-dev/2002-July/026837.html

[26] P. McIlroy, "Optimistic sorting and information theoretic complexity," in *Proc.* 4th *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1993, pp. 467–474.

[27] S. Buss and A. Knop, "Strategies for stable merge sorting," in *Proc.* 13th *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2019, pp. 1272–1290.

[28] C. Ding, D. Pei, and A. Salomaa, *Chinese Remainder Theorem: Applications in Computing, Coding, Cryptography*. World Scientific, Singapore, 1996.

[29] B. Farhi, "An identity involving the least common multiple of binomial coefficients and its application," *Amer. Math. Monthly*, vol. 116, no. 9, pp. 836–839, 2009.