Efficient constructions of the Prefer-same and Prefer-opposite de Bruijn sequences

Evan Sala
School of Computer Science, University of Guelph, Canada

Joe Sawada
School of Computer Science, University of Guelph, Canada

Abbas Alhakim
Department of Mathematics, American University of Beirut, Lebanon

Abstract

The greedy Prefer-same de Bruijn sequence construction was first presented by Eldert et al. [*AIEE Transactions* 77 (1958)]. As a greedy algorithm, it has one major downside: it requires an exponential amount of space to store the length 2^n de Bruijn sequence. Though de Bruijn sequences have been heavily studied over the last 60 years, finding an efficient construction for the Prefer-same de Bruijn sequence has remained a tantalizing open problem. In this paper, we unveil the underlying structure of the Prefer-same de Bruijn sequence and solve the open problem by presenting an efficient algorithm to construct it using $O(n)$ time per bit and only $O(n)$ space. Following a similar approach, we also present an efficient algorithm to construct the Prefer-opposite de Bruijn sequence.

1 Introduction

Greedy algorithms often provide some of the nicest algorithms to exhaustively generate combinatorial objects, especially in terms of the simplicity of their descriptions. An excellent discussion of such algorithms is given by Williams [30] with examples given for a wide range of combinatorial objects including permutations, set partitions, binary trees, and de Bruijn sequences. A downside to greedy constructions is that they generally require exponential space to keep track of which objects have already been visited. Fortunately, most greedy constructions can also be constructed efficiently by either an iterative successor-rule approach, or by applying a recursive technique. Such efficient constructions often provide extra underlying insight into both the combinatorial objects and the actual listing of the object being generated.

A *de Bruijn* sequence of order n is a sequence of bits that when considered cyclicly contains every length n binary string as a substring exactly once; each such sequence has length 2^n. They have been studied as far back as 1894 with the work by Flye Sainte-Marie [13], receiving more significant attention starting in 1946 with the work of de Bruijn [7]. Since then, many different de Bruijn sequence constructions have been presented in the literature (see surveys in [15] and [20]). Generally, they fall into one of the following categories: (i) greedy approaches (ii) iterative successor-rule based approaches which includes linear (and non-linear) feedback shift registers (iii) string concatenation approaches (iv) recursive approaches. Underlying all of these algorithms is the fact that every de Bruijn sequence is in 1-1 correspondence with an Euler cycle in a related de Bruijn graph.

Perhaps the most well-known de Bruijn sequence is the one that is the lexicographically largest. It has the following greedy Prefer-1 construction [26].

Prefer-1 construction

1. Seed with 0^{n-1}
2. **Repeat** until no new bit is added: Append 1 if it does not create a duplicate length n substring; otherwise append 0 if it does not create a duplicate length n substring
3. Remove the seed

© Evan Sala, Joe Sawada and Abbas Alhakim;
licensed under Creative Commons License CC-BY
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

For example, applying this construction for $n = 4$ we obtain the string: ~~000~~ 1111011001010000. Like all greedy de Bruijn sequence constructions, this algorithm has a major downside: it requires an exponential amount of space to remember which substrings have already been visited. Fortunately, the resulting sequence can also be constructed efficiently by applying an $O(n)$ time per bit successor-rule which requires $O(n)$ space [14]. By applying a necklace concatenation approach, it can even be generated in amortized $O(1)$ time per bit and $O(n)$ space [17].

Two other interesting greedy constructions take into account the last bit generated. They are known as the Prefer-same and Prefer-opposite constructions and their resulting sequences are respectively the lexicographically largest and smallest with respect to a run-length encoding [3]. The Prefer-same construction was first presented by Eldert et al. [10] in 1958 and was revisited with a proof of correctness by Fredricksen [15] in 1982. Recently, the description of the algorithm was simplified [3] as follows:

Prefer-same construction

1. Seed with length $n-1$ string $\cdots 01010$
2. Append 1
3. **Repeat** until no new bit is added: Append the **same** bit as the last if it does not create a duplicate length n substring; otherwise append the opposite bit as the last if it does not create a duplicate length n substring
4. Remove the seed

For $n = 4$, the sequence generated by this Prefer-same construction is ~~010~~ 1111000011010010. It has run-length encoding 44211211 which is the lexicographically largest amongst all de Bruijn sequences for $n = 4$.

The Prefer-opposite construction is not greedy in the strictest sense since there is a special case when the current suffix is 1^{n-1}. Details about this special case are provided in the next section. The construction presented below produces a shift of the sequence produced by the original presentation in [1]. Here, the initial seed of 0^{n-1} is rotated to the end so the resulting sequence is the lexicographically smallest with respect to a run-length encoding.

Prefer-opposite construction

1. Seed with 0^{n-1}
2. Append 0
3. **Repeat** until no new bit is added:
   - **If** current suffix is 1^{n-1} **then**: append 1 if it is the first time 1^{n-1} has been seen; otherwise append 0
   - **Otherwise**: append the **opposite** bit as the last if it does not create a duplicate length n substring; otherwise append the same bit as the last
4. Remove the seed

For $n = 4$, the sequence generated by this Prefer-opposite construction is ~~000~~ 0101001101111000. The run-length encoding of this sequence is given by 111122143.

To simplify our discussion, let:

- $\mathcal{S}_n$ = the de Bruijn sequence of order n generated by the Prefer-same construction, and
- $\mathcal{O}_n$ = the de Bruijn sequence of order n generated by the Prefer-opposite construction.

Unlike the Prefer-1 sequence, and despite the vast research on de Bruijn sequences, $\mathcal{S}_n$ and $\mathcal{O}_n$ have no known efficient construction. For $\mathcal{S}_n$, finding an efficient construction has remained an elusive open problem for over 60 years. The closest attempt came in 1977 when Fredricksen and Kessler devised a construction based on lexicographic compositions [16] that we discuss further in Section 8. The main results of this paper are to solve these open problems by providing successor-rule based constructions for $\mathcal{S}_n$ and $\mathcal{O}_n$. They generate the respective sequences in $O(n)$ time per bit using only $O(n)$ space. The discovery of these efficient constructions hinged on the following idea:

> Every interesting de Bruijn sequence is the result of joining together smaller cycles induced by *simple* feedback shift registers.

The initial challenge was to find such a simple underlying feedback function. After careful study, the following function was revealed:

$$f(w_1w_2\cdots w_n) = w_1 \oplus w_2 \oplus w_n,$$

where $\oplus$ denotes addition modulo 2. We demonstrate this feedback function has nice run-length properties when used to partition the set of all binary strings of length n in Section 4.3. The next challenge was to find appropriate representatives for each cycle induced by f in order to apply the framework from [20] to join the cycles together.

Outline of paper. Before introducing our main results, we first provide an insight into greedy constructions for de Bruijn sequences that we feel has not been properly emphasized in the recent literature. In particular, we demonstrate how all such constructions, which are generalized by the notion of preference or look-up tables [2, 31], are in fact just special cases of a standard Euler cycle algorithm on the de Bruijn graph. This discussion is found in Section 2 which also outlines a second Euler cycle algorithm underlying the cycle joining approach applied in our main result. In Section 3, we present background on run-length encodings. In Section 4, we discuss feedback functions and de Bruijn successors and introduce the function $f(w_1w_2\cdots w_n) = w_1 \oplus w_2 \oplus w_n$ critical to our main results. In Section 5, we present two generic de Bruijn successors based on the framework from [20]. In Section 6 we present our first main result: an efficient successor-rule to generate $\mathcal{S}_n$. In Section 7 we present our second main result: an efficient successor-rule to generate $\mathcal{O}_n$. In Section 8 we discuss the lexicographic composition algorithm from [16] and a related open problem. In Section 9 we discuss implementation details and analyze the efficiency of our algorithms.

In Section 10 and Section 11 we detail the technical aspects required to prove our main results. Implementation of our algorithms, written in C, presented in this paper can be found in the appendices and are available for download at `http://debruijnsequence.org`.

Applications. One of the first instances of de Bruijn sequences is found in works of Sanskrit prosody by the ancient mathematician Pingala dating back to the 2nd century BCE. Since then, de Bruijn sequences and their related theory have a rich history of application. One of their more prominent applications, due to their random-like properties [22], is in the generation of pseudo-random bit sequences which are used in stream ciphers [25]. In particular, linear feedback shift register constructions (that omit the string of all 0s) allow for efficient hardware embeddings which have been classically applied to represent different maps in video games including Pitfall [4]. Another application uses de Bruijn sequences to crack cipher locks in an efficient manner [15]. More recently, the related de Bruijn graph has been applied to genome assembly [6, 27]. Given the vast literature on de Bruijn sequences and their various methods of construction, the more interesting new results may relate to sequences with specific properties. This makes the de Bruijn sequences $\mathcal{S}_n$ and $\mathcal{O}_n$ of special interest since they are, respectively, the lexicographically largest and smallest sequences with respect to a run-length encoding [3]. Moreover, recently it was noted they have a relatively small discrepancy when compared to the sequences generated by the Prefer-1 construction [19].

2 Euler cycle algorithms and the de Bruijn graph

The *de Bruijn graph* of order n is the directed graph $G(n) = (V, E)$ where V is the set of all binary strings of length n and there is a directed edge from $u = u_1u_2 \cdots u_n$ to $v = v_1v_2 \cdots v_n$ if $u_2 \cdots u_n = v_1 \cdots v_{n-1}$. Each edge e is labeled by v_n. Outputting the edge labels in a Hamilton cycle of $G(n)$ produces a de Bruijn sequence. Figure 1(a) illustrates a Hamilton cycle in the de Bruijn graph $G(3)$. Starting from 000, its corresponding de Bruijn sequence is 10111000.

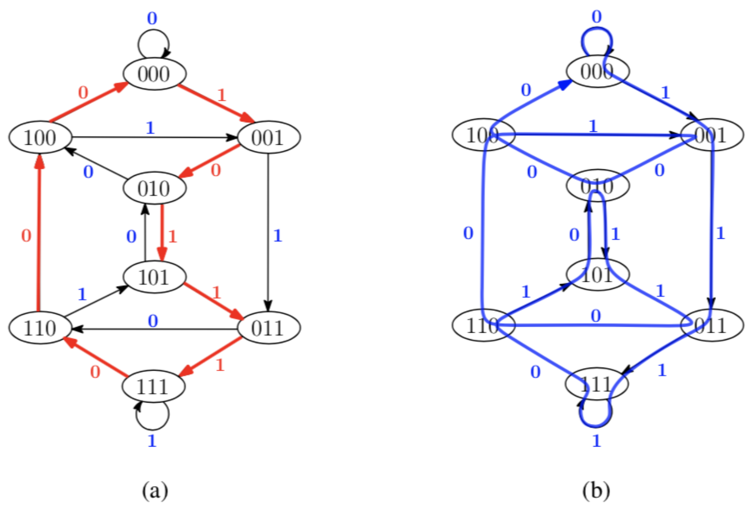


Figure 1 (a) A Hamilton cycle in $G(3)$ starting from 000 corresponding to the de Bruijn sequence 10111000 of order 3. (b) An Euler cycle in $G(3)$ starting from 000 corresponding to the de Bruijn sequence 0111101011001000 of order 4.

Each de Bruijn graph is connected and the in-degree and the out-degree of each vertex is two; the graph $G(n)$ is Eulerian. $G(n)$ is the line graph of $G(n-1)$ which means an Euler cycle in $G(n-1)$ corresponds to a Hamilton cycle in $G(n)$. Thus, the sequence of edge labels visited in an Euler cycle is a de Bruijn sequence. Figure 1(b) illustrates an Euler cycle in $G(3)$. The corresponding de Bruijn sequence of order four when starting from the vertex 000 is 0111101011001000.

Finding an Euler cycle in an Eulerian graph is linear-time solvable with respect to the size of the graph. However, since the graph must be stored, applying such an algorithm to find a de Bruijn sequence requires $O(2^n)$ space. One of the most well-known Euler cycle algorithms for directed graphs is the following due to Fleury [12] with details in [15]. The basic idea is to not burn bridges; in other words, do not visit (and use up) an edge if it leaves the remaining graph disconnected.

Fleury's Euler cycle algorithm (do not burn bridges)

1. Pick a root vertex and compute a spanning in-tree T
2. Make each edge of T (the bridges) the last edge on the adjacency list of the corresponding vertex
3. Starting from the root, traverse edges in a depth-first manner by visiting the first unused edge in the current vertex's adjacency list

Finding a spanning in-tree T can be done by reversing the direction of the edges in the Eulerian graph and computing a spanning out-tree with a standard depth first search on the resulting graph. The corresponding edges in the original graph will be a spanning in-tree. Using this approach, all de Bruijn sequences can be generated by considering all possible spanning in-trees.

Although not well documented, this algorithm is the basis for all greedy de Bruijn sequence constructions along with their generalizations using preference tables [2] or look-up tables [31]. Specifically, a preference table specifies the precise order that the edges are visited for each vertex when performing Step 3 in Fleury's Euler cycle algorithm. Thus given a preference table and a root vertex, Step 3 in the algorithm can be applied to construct a de Bruijn sequence if combining the last edge from each non-root vertex forms a spanning in-tree to the root. For example, the preference tables and corresponding spanning in-trees for the Prefer-1 (rooted at 000), the Prefer-same (rooted at 010), and the Prefer-opposite (rooted at 000) constructions are given in Figure 2 for $G(3)$. For the Prefer-1, the only valid root is 000. For the Prefer-same, either 010 or 101 could be chosen as root. The Prefer-opposite has a small nuance. By a strict greedy definition, the edges will not create a spanning in-tree for any root. But by changing the preference for the single string 111, a spanning in-tree is created when rooted at 000. This accounts for the special case required in the Prefer-opposite algorithm. Notice how these strings relate to the seeds in their respective greedy constructions. For the Prefer-same, a root of 101 could also have been chosen, and doing so will yield the complement of the Prefer-same sequence when applying this Euler cycle algorithm.

Prefer-1

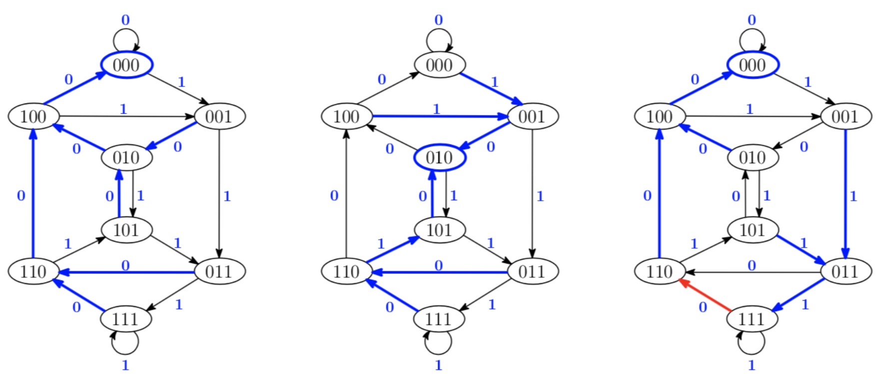
Vertex	Edge pref
000	1,0
010	1,0
100	1,0
110	1,0
001	1,0
011	1,0
101	1,0
111	1,0

Prefer-same

Vertex	Edge pref
000	0,1
010	0,1
100	0,1
110	0,1
001	1,0
011	1,0
101	1,0
111	1,0

Prefer-opposite

Vertex	Edge pref
000	1,0
010	1,0
100	1,0
110	1,0
001	0,1
011	0,1
101	0,1
111	**1,0**



(a) (b) (c)

Figure 2 (a) A preference table corresponding to the Prefer-1 greedy construction along with its corresponding spanning in-tree rooted at 000. (b) A preference table corresponding to the Prefer-same greedy construction along with its corresponding spanning in-tree rooted at 010. (c) A preference table corresponding to the Prefer-opposite greedy construction along with its corresponding spanning in-tree rooted at 000.

A second well-known Euler cycle algorithm for directed graphs, attributed to Hierholzer [23], is as follows:

Hierholzer's Euler cycle algorithm (cycle joining)

1. Start at an arbitrary vertex v visiting edges in a depth-first manner until returning to v, creating a cycle.
2. **Repeat until all edges are visited:** Start from any vertex u on the current cycle and visit remaining edges in a DFS manner until returning to u, creating a new cycle. Join the two cycles together.

This cycle-joining approach is the basis for all successor-rule constructions of de Bruijn sequences. A general framework for joining smaller cycles together based on an underlying feedback shift register is given for the binary case in [20], and then more generally for larger alphabets in [21]. It is the basis for the efficient algorithm presented in this paper, where the initial cycles are induced by a specific feedback function.

3 Run-length encoding

The sequences $\mathcal{S}_n$ and $\mathcal{O}_n$ both have properties based on a run-length encoding of binary strings. The *run-length encoding* (RLE) of a string $\omega = w_1w_2\cdots w_n$ is a compressed representation that stores consecutively the lengths of the maximal runs of each symbol. The *run-length* of ω is the length of its RLE. For example, the string 11000110 has RLE 2321 and run-length 4. Note that 00111001 also has RLE 2321. Since we are dealing with binary strings, we require knowledge of the starting symbol to obtain a given binary string from its RLE. As a further example:

$$\mathcal{S}_5 = 11111000001110110011010001001010 \ \text{ has RLE } \ 5531222113121111.$$

The following facts are proved in [3].

▶ **Proposition 1.** *The sequence $\mathcal{S}_n$ is the de Bruijn sequence of order n starting with 1 that has the lexicographically largest RLE.*

▶ **Proposition 2.** *The sequence $\mathcal{O}_n$ is the de Bruijn sequence of order n starting with 1 that has the lexicographically smallest RLE.*

Let $alt(n)$ denote the alternating sequence of 0s and 1s of length n that ends with 0: For example, $alt(6) = 101010$. The following facts are also immediate from [3].

▶ **Proposition 3.** *$\mathcal{S}_n$ has prefix 1^n and has suffix $alt(n{-}1)$.*

▶ **Proposition 4.** *$\mathcal{O}_n$ has length n prefix $010101\cdots$ and has suffix 10^{n-1}.*

The sequence based on lexicographic compositions [16] also has also has run-length properties: it is constructed by concatenating lexicographic compositions which are represented using a RLE. A brief discussion of this sequence is provided in Section 8.

4 Feedback functions and de Bruijn successors

Let $\mathbf{B}(n)$ denote the set of all binary strings of length n. We call a function $f : \mathbf{B}(n) \to \{0, 1\}$ a *feedback function*. Let $\omega = w_1w_2\cdots w_n$ be a string in $\mathbf{B}(n)$. A *feedback shift register* is a function $F : \mathbf{B}(n) \to \mathbf{B}(n)$ that takes the form $F(\omega) = w_2w_3\cdots w_n f(w_1w_2\cdots w_n)$ for a given feedback function f.

A feedback function $g : \mathbf{B}(n) \to \{0, 1\}$ is a *de Bruijn successor* if there exists a de Bruijn sequence of order n such that each string $\omega \in \mathbf{B}(n)$ is followed by $g(\omega)$ in the given de Bruijn sequence. Given a de Bruijn successor g and a seed string $\omega = w_1w_2\cdots w_n$, the following function $\mathrm{DB}(g, \omega)$ will return a de Bruijn sequence of order n with suffix ω:

```
function DB(g, ω)
    for i ← 1 to 2^n do
        x_i ← g(ω)
        ω ← w_2 w_3 ··· w_n x_i
    return x_1 x_2 ··· x_{2^n}
```

A *linearized de Bruijn sequence* is a linear string that contains every string in $\mathbf{B}(n)$ as a substring exactly once. Such a string has length 2^n+n-1. Note that the length n suffix of a de Bruijn sequence $\mathcal{D}_n = \text{DB}(g, w_1 \cdots w_n)$ is $w_1 \cdots w_n$. Thus, $w_2 \cdots w_n \mathcal{D}_n$ is a linearized de Bruijn sequence.

For each of the upcoming feedback functions, selecting appropriate representatives for the cycles they induce is an important step to developing efficient de Bruijn successors for $\mathcal{S}_n$ and $\mathcal{O}_n$. In particular, consider two representatives for a given cycle based on their RLE.

- *RL-rep*: The string with the lexicographically largest RLE; if there are two such strings, it is the one beginning with 1.
- *RL2-rep*: The string with the lexicographically smallest RLE; if there are two such strings, it is the one beginning with 0.

For our upcoming discussion, define the *period* of a string $\omega = w_1 w_2 \cdots w_n$ to be the smallest integer p such that $\omega = (w_1 \cdots w_p)^j$ for some integer j. If $j > 1$ we say that ω is *periodic*; otherwise, we say it is *aperiodic* (or primitive).

4.1 The pure cycling register (PCR)

The *pure cycling register*, denoted PCR, is the feedback shift register with the feedback function $f(\omega) = w_1$. Thus, $\text{PCR}(w_1 w_2 \cdots w_n) = w_2 \cdots w_n w_1$. It is well-known that the PCR partitions $\mathbf{B}(n)$ into cycles of strings that are equivalent under rotation. The following example illustrates the cycles induced by the PCR for $n = 5$ along with their corresponding RL-reps and RL2-reps.

Example 1 The PCR partitions $\mathbf{B}(5)$ into the following eight cycles $\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_8$ where the top string in bold is the RL-rep for the given cycle. The underlined string is the RL2-rep.

$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_4$	$\mathbf{P}_5$	$\mathbf{P}_6$	$\mathbf{P}_7$	$\mathbf{P}_8$
11010	**00101**	**11110**	**00001**	**11100**	**00011**	<u>**11111**</u>	<u>**00000**</u>
<u>10101</u>	<u>01010</u>	11101	00010	11001	00110		
01011	10100	11011	00100	<u>10011</u>	<u>01100</u>		
10110	01001	<u>10111</u>	<u>01000</u>	00111	11000		
01101	10010	01111	10000	01110	10001		

The PCR is the underlying feedback function used to construct the Prefer-1 greedy construction corresponding to the lexicographically largest de Bruijn sequence. It has also been applied in some of the simplest and most efficient de Bruijn sequence constructions [8, 20, 29]. In these constructions, the cycle representatives relate to the lexicographically smallest (or largest) strings in each cycle and they can be determined in $O(n)$ time using $O(n)$ space using standard techniques [5, 9]. We also apply these methods to efficiently determine the RL-reps and the RL2-reps.

Clearly 0^n and 1^n are both RL-reps. Consider a string $\omega = w_1 w_2 \cdots w_n$ in a cycle $\mathbf{P}$ with RLE $r_1 r_2 \cdots r_\ell$ where $\ell > 1$. If ω is an RL-rep, then $w_1 \neq w_n$ because otherwise $w_n w_1 \cdots w_{n-1}$ has a larger RLE than ω. All strings in $\mathbf{P}$ that differ in the first and last bits form an equivalence class under rotation with respect to their RLE. By definition, the RL-rep will be one that is lexicographically largest amongst all its rotations. As noted above, such a test can be performed in $O(n)$ time using $O(n)$ space. There is one special case to consider: when both a string beginning with 0 and its complement beginning with 1 belong to the same cycle. For example, consider 00101101 and 11010010 which both have RLE 211211. Note this RLE has period $p = 3$ and it is maximal amongst its rotations. By definition, the string beginning with 0 is not an RL-rep. It is not difficult to see that such a string occurs precisely when $w_1 = 0$, p is odd, and $p < \ell$, where p is the period of $r_1 r_2 \cdots r_\ell$.

▶ **Proposition 5.** *Let $\omega = w_1 w_2 \cdots w_n$ be a string with RLE $r_1 r_2 \cdots r_\ell$, where $\ell > 1$, in a cycle $\mathbf{P}$ induced by the PCR. Let p be the period of $r_1 r_2 \cdots r_\ell$. Then ω is the RL-rep for $\mathbf{P}$ if and only if*

1. $w_1 \neq w_n$,

2. *$r_1r_2\cdots r_\ell$ is lexicographically largest amongst all its rotations, and*
3. *either $w_1 = 1$ or $p = \ell$ or p is even.*

Moreover, testing whether or not ω is an RL-rep can be done in $O(n)$ time using $O(n)$ space.

In a similar manner we consider RL2-reps. Again 0^n and 1^n are both clearly RL2-reps. Consider a string $\omega = w_1w_2\cdots w_n$ in a cycle $\mathbf{P}$ with run length greater than one. If ω is an RL2-rep, then $w_1 \neq w_2$ because otherwise $w_2\cdots w_nw_1$ has a smaller RLE than ω. Thus, consider all strings $s_1s_2\cdots s_n$ in a cycle $\mathbf{P}$ such that $s_2 \neq s_1$. One of these strings is the RL2-rep. Now consider all left rotations of these strings taking the form $s_2\cdots s_ns_1$. Notice that a string in the latter set with the smallest RLE will correspond to the RL2-rep after rotating the string back to the right. As noted in the RL-case, the set of rotated strings form an equivalence class under rotation with respect to their RLE, since their first and last bits differ. Again, the same special case arises as with RL-reps: when both a string beginning with 0 and its complement beginning with 1 belong to the same cycle. For example, consider the cycle containing both 10100101 and 01011010. In each string the first two bits differ. The set of all strings in its cycle where the first two bits differ is $\{10100101, 01001011, 10010110, 01011010, 10110100, 01101001\}$. Rotating each string to the left we get the set $\{01001011, 10010110, 00101101, 10110100, 01101001, 11010010\}$. The corresponding RLEs for this latter set are $\{112112, 121121, 211211, 112112, 121121, 211211\}$. In this case there are two strings 0100100 and 10110100 that both have RLE 112112. Rotating these strings back to the right we have 10100101 and 01011010 which both have the lexicographically smallest RLE of 1112111 in their cycle induced by the PCR. By definition, the string beginning with 0 will be the RL2-rep. Thus ω is not an RL2-rep if $w_1 = 1$, p is odd, and $p < \ell$, where p is the period of the RLE $r_1r_2\cdots r_\ell$ for the string $w_2\cdots w_nw_1$.

▶ **Proposition 6.** *Let $\omega = w_1w_2\cdots w_n$ and let $r_1r_2\cdots r_\ell$ be the RLE of $w_2\cdots w_nw_1$, where $\ell > 1$, in a cycle $\mathbf{P}$ induced by the PCR. Let p be the period of $r_1r_2\cdots r_\ell$. Then ω is the RL2-rep for $\mathbf{P}$ if and only if*

1. *$w_1 \neq w_2$,*
2. *$r_1r_2\cdots r_\ell$ is lexicographically smallest amongst all its rotations, and*
3. *either $w_1 = 0$ or $p = \ell$ or p is even.*

Moreover, testing whether or not ω is an RL2-rep can be done in $O(n)$ time using $O(n)$ space.

4.2 The complementing cycling register (CCR)

The *complementing cycling register*, denoted CCR, is the FSR with the feedback function $f(\omega) = \overline{w_1}$, where $\overline{w_1}$ denotes the complement w_1. Thus, CCR$(w_1w_2\cdots w_n) = w_2\cdots w_n\overline{w_1}$. A string and its complement will belong to the same cycle induced by the CCR.

Example 2 The CCR partitions $\mathbf{B}(5)$ into the following four cycles $\mathbf{C}_1, \mathbf{C}_2, \mathbf{C}_3, \mathbf{C}_4$ where the top string in bold is the RL-rep for the given cycle. The underlined string is the RL2-rep.

$\mathbf{C}_1$	$\mathbf{C}_2$	$\mathbf{C}_3$	$\mathbf{C}_4$
10101	**11101**	**11001**	**11111**
01010	11010	10010	11110
	10100	00100	11100
	01000	01001	11000
	10001	10011	10000
	00010	00110	00000
	00101	01101	00001
	01011	11011	00011
	10111	10110	00111
	01110	01100	01111

The CCR has been applied to efficiently construct de Bruijn sequences in variety of ways [11, 24, 20]. An especially efficient construction applies a concatenation scheme to construct a de Bruijn sequence with discrepancy (maximum difference between the number of 0s and 1s in any prefix) bounded above by $2n$ [18, 19].

As with the PCR, we discuss how to efficiently determine whether or not a given string is an RL-rep or an RL2-rep for a cycle $\mathbf{C}$ induced by the CCR. Consider a string $\omega = w_1w_2\cdots w_n$ in a cycle $\mathbf{C}$. If ω is an RL-rep, then $w_1 = w_n$ because otherwise $\overline{w_n}w_1\cdots w_{n-1}$, which is also in $\mathbf{C}$, has a larger RLE than ω. All strings in $\mathbf{C}$ that agree in the first and last bits form an equivalence class under rotation with respect to their RLE (that includes strings starting with both 0 and 1 for each RLE). By definition, the RL-rep will be one that is lexicographically largest amongst all its rotations. As noted in the previous subsection, such a test can be performed in $O(n)$ time using $O(n)$ space. There are no special cases to consider here since a string and its complement always belong to the same cycle. Thus, every RL-rep must begin with 1.

▶ **Proposition 7.** *Let $\omega = w_1w_2\cdots w_n$ be a string with RLE $r_1r_2\cdots r_\ell$ in a cycle $\mathbf{C}$ induced by the CCR. Then ω is the RL-rep for $\mathbf{C}$ if and only if*

1. *$w_1 = w_n = 1$ and*
2. *$r_1r_2\cdots r_\ell$ is lexicographically largest amongst all its rotations.*

Moreover, testing whether or not ω is an RL-rep can be done in $O(n)$ time using $O(n)$ space.

In a similar manner we consider RL2-reps. Again, consider a string $\omega = w_1w_2\cdots w_n$ in a cycle $\mathbf{C}$. If ω is an RL2-rep, then $w_1 \neq w_2$ because otherwise $w_2\cdots w_n\overline{w_1}$ has a smaller RLE than ω. Consider all such strings $w_2\cdots w_n\overline{w_1}$ in a cycle $\mathbf{C}$ such that $w_2 \neq w_1$. As noted in the RL-case, all such strings form an equivalence class under rotation with respect to their RLE. Clearly, such a string that has the lexicographically smallest RLE will be the RL2-rep. There are no special cases to consider here since a string and its complement always belong to the same cycle. Thus, every RL2-rep must begin with 0 and hence $w_2 = 1$.

▶ **Proposition 8.** *Let $\omega = w_1w_2\cdots w_n$ be a string with RLE $r_1r_2\cdots r_\ell$ in a cycle $\mathbf{C}$ induced by the CCR. Then ω is the RL2-rep for $\mathbf{C}$ if and only if*

1. *$w_1 = 0$,*
2. *$w_2 = 1$, and*
3. *$r_1r_2\cdots r_\ell$ is lexicographically smallest amongst all its rotations.*

Moreover, testing whether or not ω is an RL2-rep can be done in $O(n)$ time using $O(n)$ space.

4.3 The pure run-length register (PRR)

The feedback function of particular focus in this paper is $f(\omega) = w_1 \oplus w_2 \oplus w_n$. We will demonstrate that FSR based on this feedback function partitions $\mathbf{B}(n)$ into cycles of strings with the same run-length. Because of this property, we call this FSR the *pure run-length register* and denote it by PRR. Thus,

$$\mathrm{PRR}(w_1w_2\cdots w_n) = w_2\cdots w_n(w_1 \oplus w_2 \oplus w_n).$$

This follows the naming of the pure cycling register (PCR) and the pure summing register (PSR), which is based on the feedback function $f(\omega) = w_1 \oplus w_2 \oplus \cdots \oplus w_n$ [22].

Let $\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_t$ denote the cycles induced by the PRR on $\mathbf{B}(n)$. The following example illustrates how the cycles induced by the PRR relate to the cycles induced by the PCR and CCR.

Example 3 The PRR partitions $\mathbf{B}(6)$ into the following 12 cycles $\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_{12}$ where the top string in bold is the RL-rep for the given cycle. The underlined string is the RL2-rep. The cycles are ordered in non-increasing order with respect to the run-lengths of their RL-reps.

$\mathbf{R}_1$	$\mathbf{R}_2$	$\mathbf{R}_3$	$\mathbf{R}_4$	$\mathbf{R}_5$	$\mathbf{R}_6$	$\mathbf{R}_7$	$\mathbf{R}_8$	$\mathbf{R}_9$	$\mathbf{R}_{10}$
101010	**110101**	**001010**	**111010**	**110010**	**111101**	**000010**	**111001**	**000110**	**111110**
010101	101011	010100	110100	100100	111011	000100	110011	001100	111100
	010110	101001	101000	001001	110111	001000	100111	011000	111000
	101101	010010	010001	010011	101111	010000	001110	110001	110000
	011010	100101	100010	100110	011110	100001	011100	100011	100000
			000101	001101					000001
			001011	011011					000011
			010111	110110					000111
			101110	101100					001111
			011101	011001					011111

$\mathbf{R}_{11}$	$\mathbf{R}_{12}$
111111	**000000**

By omitting the last bit of each string, the columns are precisely the cycles of the PCR and CCR for $n = 5$. The cycles $\mathbf{R}_1, \mathbf{R}_4, \mathbf{R}_5, \mathbf{R}_{10}$ relating to the CCR start and end with the different bits. The remaining cycles relate to the PCR; each string in these cycles start and end with the same bit.

In the example above, note that all the strings in a given cycle $\mathbf{R}_i$ have the same run-length.

▶ **Lemma 9.** *All the strings in a given cycle $\mathbf{R}_i$ have the same run-length.*

Proof. Consider a string $\omega = w_1w_2\cdots w_n$ and the feedback function $f(\omega) = w_1 \oplus w_2 \oplus w_n$. It suffices to show that $w_2\cdots w_n f(\omega)$ has the same run-length as ω. This is easily observed since if $w_1 = w_2$ then $w_n = f(\omega)$ and if $w_1 \neq w_2$ then $w_n \neq f(\omega)$. ◀

Based on this lemma, if the strings in $\mathbf{R}_i$ have run length ℓ, we say that $\mathbf{R}_i$ has run-length ℓ. Each cycle $\mathbf{R}_i$ has another interesting property: either all the strings start and end with the same bit, or all the strings start and end with different bits. If the strings start and end with the same bit, then $\mathbf{R}_i$ must have odd run-length and if we remove the last bit of each string we obtain a cycle induced by the PCR of order $n-1$. In this case we say that $\mathbf{R}_i$ is a *PCR-related cycle*. If the strings start and end with the different bits, then $\mathbf{R}_i$ must have even run-length and if we remove the last bit of each string we obtain a cycle induced by the CCR of order $n-1$. In this case we say that $\mathbf{R}_i$ is a *CCR-related cycle*. These observations were first made in [28] and are illustrated in Example 3. Based on these observations, we can apply the RL-rep and RL2-rep testers for cycles induced by the PCR and CCR to determine whether or not a string ω is an RL-rep or an RL2-rep for a cycle $\mathbf{R}_i$. These testers will be critical to the efficiency of our upcoming de Bruijn successors.

▶ **Proposition 10.** *Let $\omega = w_1w_2\cdots w_n$ be a string in a cycle $\mathbf{R}$ induced by the PRR. Then ω is the RL-rep for $\mathbf{R}$ if and only if*

1. *$w_1 = w_n$ and $w_1w_2\cdots w_{n-1}$ is an RL-rep with respect to the PCR, or*
2. *$w_1 \neq w_n$ and $w_1w_2\cdots w_{n-1}$ is an RL-rep with respect to the CCR.*

Moreover, testing whether or not ω is an RL-rep for $\mathbf{R}$ can be done in $O(n)$ time using $O(n)$ space.

▶ **Proposition 11.** *Let $\omega = w_1w_2\cdots w_n$ be a string in a cycle $\mathbf{R}$ induced by the PRR. Then ω is the RL2-rep for $\mathbf{R}$ if and only if*

1. *$w_1 = w_n$ and $w_1w_2\cdots w_{n-1}$ is an RL2-rep with respect to the PCR, or*
2. *$w_1 \neq w_n$ and $w_1w_2\cdots w_{n-1}$ is an RL2-rep with respect to the CCR.*

Moreover, testing whether or not ω is an RL2-rep for $\mathbf{R}$ can be done in $O(n)$ time using $O(n)$ space.

5 Generic de Bruijn successors based on the PRR

In this section we provide two generic de Bruijn successors that are applied to derive specific de Bruijn successors for $\mathcal{S}_n$ and $\mathcal{O}_n$ in the subsequent sections. The results relate specifically to the PRR and we assume that $\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_t$ denote the cycles induced by the PRR on $\mathbf{B}(n)$.

Let $\omega = w_1 w_2 \cdots w_n$ be a binary string. Define the *conjugate* of ω to be $\hat{\omega} = \overline{w_1} w_2 \cdots w_n$. Similar to Hierholzer's cycle-joining approach discussed in Section 2, Theorem 3.5 from [20] can be applied to systematically join together the *ordered* cycles $\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_t$ given certain representatives α_i for each $\mathbf{R}_i$. This theorem is restated as follows when applied to the PRR and the function $f(\omega) = w_1 \oplus w_2 \oplus w_n$.

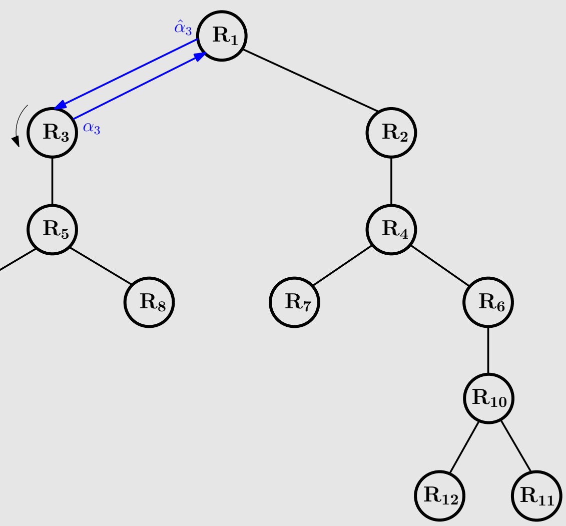
▶ **Theorem 12.** *For each $1 < i \leq t$, if the conjugate $\hat{\alpha}_i$ of the representative α_i for cycle $\mathbf{R}_i$ belongs to some $\mathbf{R}_j$ where $j < i$, then*

$$g(\omega) = \begin{cases} \overline{f(\omega)} & \text{if } \omega \text{ or } \hat{\omega} \text{ is in } \{\alpha_2, \alpha_3, \ldots, \alpha_t\}; \\ f(\omega) & \text{otherwise.} \end{cases}$$

is a de Bruijn successor.

Together, the ordering of the cycles and the sequence $\alpha_2, \alpha_3, \ldots, \alpha_t$ correspond to a rooted tree, where the nodes are the cycles $\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_t$ with $\mathbf{R}_1$ designated as the root. There is an edge between two nodes $\mathbf{R}_i$ and $\mathbf{R}_j$ where $i > j$, if and only if $\hat{\alpha}_i$ is in $\mathbf{R}_j$. Each edge represents the joining of two cycles similar to the technique used in Hierholzer's Euler cycle algorithm (see Section 2). An example of such a tree for $n = 6$ is given in the following example.

Example 4 Consider the cycles $\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_{12}$ for $n = 6$ from Example 3 along with their corresponding RL-reps α_i for each $\mathbf{R}_i$. For each $i > 1$, $\hat{\alpha}_i$ belongs to some $\mathbf{R}_j$ where $j < i$. Thus, we can apply Theorem 12 to obtain a de Bruijn successor $g(\omega)$ based on these representatives. The following tree illustrates the joining of these cycles based on g:



Starting with 101010 from $\mathbf{R}_1$, and repeatedly applying the function $g(\omega)$ we obtain the de Bruijn sequence:

$$101010010011\underline{000110}1100111001010110100010000101110111100000011111.$$

Note that the RL-rep of $\mathbf{R}_3$ is $\alpha_3 = 001010$ and its conjugate $\hat{\alpha}_3 = 101010$ is found in $\mathbf{R}_1$. The last string visited in each cycle $\mathbf{R}_i$, for $i > 1$, is its representative α_i.

The following observations, which will be applied later in our more technical proofs, follow from the tree interpretation of the ordered cycles rooted at $\mathbf{R}_1$ from Theorem 12 as illustrated in the previous example.

► **Observation 13.** *Let g be a de Bruijn successor from Theorem 12 based on representatives $\alpha_2, \alpha_3, \ldots, \alpha_t$. Let $\mathcal{D}_n = DB(g, w_1w_2\cdots w_n)$ and let $\mathcal{D}'_n = w_2\cdots w_n\mathcal{D}_n$ denote a linearized de Bruijn sequence. If the length n prefix of $\mathcal{D}'_n$ is in $\mathbf{R}_1$, then for each $1 < i \leq t$:*

1. *$\hat{\alpha}_i$ appears before all strings in $\mathbf{R}_i$,*
2. *the m strings of $\mathbf{R}_i$ appear in the following order: $PRR(\alpha_i), PRR^2(\alpha_i), \ldots, PRR^m(\alpha_i) = \alpha_i$,*
3. *if $\mathbf{R}_i$ and $\mathbf{R}_k$ are on the same level in the corresponding tree of cycles rooted at $\mathbf{R}_1$, then either every string in $\mathbf{R}_i$ comes before every string in $\mathbf{R}_k$ or vice-versa,*
4. *the strings in all descendant cycles of $\mathbf{R}_i$ appear after $\hat{\alpha}_i$ and before α_i, and*
5. *if $\hat{\alpha}_i = a_1a_2\cdots a_n$, then $a_2\cdots a_n g(\hat{\alpha}_i)$ is in $\mathbf{R}_i$.*

As an application of Theorem 12, consider the cycles $\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_t$ to be ordered in *non-increasing order* based on the run-length of each cycle. Such an ordering is given in Example 3 for $n = 6$. Using this ordering, let $\alpha_i = a_1a_2\cdots a_n$ be any string in $\mathbf{R}_i$, for $i > 1$, such that $a_1 = a_2$. Note that $\hat{\alpha}_i$ has run-length that is one *more* than the run-length of α_i and thus $\hat{\alpha}_i$ belongs to some $\mathbf{R}_j$ where $j < i$. Thus, Theorem 12 can be applied to describe the following generic de Bruijn successor based on the PRR.

► **Theorem 14.** *Let $\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_t$ be listed in non-increasing order with respect to the run-length of each cycle. Let $\alpha_i = a_1a_2\cdots a_n$ denote a representative in $\mathbf{R}_i$ such that $a_1 = a_2$, for each $1 < i \leq t$. Let $\omega = w_1w_2\cdots w_n$ and let $f(\omega) = w_1 \oplus w_2 \oplus w_n$. Then the function:*

$$g(\omega) = \begin{cases} \overline{f(\omega)} & \text{if } \omega \text{ or } \hat{\omega} \text{ is in } \{\alpha_2, \alpha_3, \ldots, \alpha_t\}; \\ f(\omega) & \text{otherwise.} \end{cases}$$

is a de Bruijn successor.

Now consider the cycles $\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_t$ to be ordered in *non-decreasing order* based on the run-length of each cycle. This means the first two cycles $\mathbf{R}_1$ and $\mathbf{R}_2$ will be the cycles containing 0^n and 1^n. But given this ordering, there is no way to satisfy Theorem 12 since the conjugate of any representative for $\mathbf{R}_2$ will not be found in $\mathbf{R}_1$. However, if we let $\mathbf{R}_t = \{1^n\}$, and order the remaining cycles in *non-decreasing order* based on the run-length of each cycle, then we obtain a result similar to Theorem 14. Observe, that this relates to the special case described for the Prefer-opposite greedy construction illustrated in Figure 2. Using this ordering, let $\alpha_i = a_1a_2\cdots a_n$ be any string in $\mathbf{R}_i$, for $1 < i < t$, such that $a_1 \neq a_2$. Such a string exists since $\mathbf{R}_1 = \{0^n\}$ and $\mathbf{R}_t = \{1^n\}$. This means $\hat{\alpha}_i$ has run-length that is one *less* than the run-length of α_i and thus $\hat{\alpha}_i$ belongs to some $\mathbf{R}_j$ where $j < i$. For the special case when $i = t$, the conjugate of 1^n clearly is found in some $\mathbf{R}_j$ where $j < t$. Thus, Theorem 12 can be applied again to describe another generic de Bruijn successor based on the PRR.

► **Theorem 15.** *Let $\mathbf{R}_t = \{1^n\}$ and let the remaining cycles $\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_{t-1}$ be listed in non-decreasing order with respect to the run-length of each cycle. Let $\alpha_i = a_1a_2\cdots a_n$ denote a representative in $\mathbf{R}_i$ such that $a_1 \neq a_2$, for each $1 < i < t$. Let $\omega = w_1w_2\cdots w_n$ and let $f(\omega) = w_1 \oplus w_2 \oplus w_n$. Then the function:*

$$g_2(\omega) = \begin{cases} \overline{f(\omega)} & \text{if } \omega \text{ or } \hat{\omega} \text{ is in } \{\alpha_2, \alpha_3, \ldots, \alpha_t\}; \\ f(\omega) & \text{otherwise.} \end{cases}$$

is a de Bruijn successor.

When Theorem 14 and Theorem 15 are applied naïvely, the resulting de Bruijn successors are not efficient since storing the set $\{\alpha_2, \alpha_3, \ldots, \alpha_t\}$ requires exponential space. However, if a membership tester for the set can be defined efficiently, then there is no need for the set to be stored. Such sets of representatives are presented in the next two sections.

6 A de Bruijn successor for $\mathcal{S}_n$

In this section we define a de Bruijn successor for $\mathcal{S}_n$. Recall the partition $\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_t$ of $\mathbf{B}(n)$ induced by the PRR. In addition to the RL-rep, we define a new representative for each cycle, called the LC-rep, where the LC stands for Lexicographic Compositions which are further discussed in Section 8. Then, considering these two representatives along with a small set of special strings, we define a third representative, called the same-rep. For each representative, we can apply Theorem 14 to produce a new de Bruijn successor. The definitions for these three representatives are as follows:

- *RL-rep*: The string with the lexicographically largest RLE; if there are two such strings, it is the one beginning with 1.
- *LC-rep*: The strings 0^n and 1^n for the classes $\{0^n\}$ and $\{1^n\}$ respectively. For all other classes, it is the string ω with RLE $21^{i-1}r_{i+1}\cdots r_\ell$ where $r_{i+1} \neq 1$ such that $\text{PRR}^{i+1}(\omega)$ is the RL-rep.
- *same-rep*: $\begin{cases} \text{RL-rep} & \text{if the RL-rep is } \textit{special} \\ \text{LC-rep} & \text{otherwise.} \end{cases}$

We say an RL-rep is *special* if it belongs to the set $\mathbf{SP}(n)$ defined as follows:

> $\mathbf{SP}(n)$ is the set of length n binary strings that begin and end with 0 and have RLE of the form $(21^{2x})^y1^z$, where $x \geq 0$, $y \geq 2$, and $z \geq 2$.

The RL-reps have already been illustrated in Section 4. There are relatively few special RL-reps and they all have odd run-length since they must begin and end with 0.

Example 5 The RLE of the special RL-reps for $n = 10, 11, 12, 13$.

$n = 10$: 2221111
$n = 11$: 2222111, 221111111, 211211111
$n = 12$: 2222211, 222111111
$n = 13$: 222211111, 22111111111, 21121111111

To illustrate an LC-rep, consider the string $\omega = \underline{11010}1111011$ with RLE 2111412. The string ω is an LC-rep since $PRR^5(\omega) = 111101110101$ which is an RL-rep with RLE $413\underline{1111}$. Note that another way to define the LC-rep is as follows: If the RLE of an RL-rep ends with i consecutive 1s, then the corresponding LC-rep is the string ω such that $PRR^{i+1}(\omega)$ is the RL-rep.

Let $\mathbf{RL}(n)$, $\mathbf{LC}(n)$, and $\mathbf{Same}(n)$ denote the sets of all length n RL-reps, LC-reps, and same-reps, respectively, **not including** the representative with run-length n. Consider the following feedback functions where $\omega = w_1w_2\cdots w_n$ and $f(\omega) = w_1 \oplus w_2 \oplus w_n$:

$$RL(\omega) = \begin{cases} \overline{f(\omega)} & \text{if } \omega \text{ or } \hat{\omega} \text{ is in } \mathbf{RL}(n); \\ f(\omega) & \text{otherwise.} \end{cases}$$

$$LC(\omega) = \begin{cases} \overline{f(\omega)} & \text{if } \omega \text{ or } \hat{\omega} \text{ is in } \mathbf{LC}(n); \\ f(\omega) & \text{otherwise.} \end{cases}$$

$$S(\omega) = \begin{cases} \overline{f(\omega)} & \text{if } \omega \text{ or } \hat{\omega} \text{ is in } \mathbf{Same}(n); \\ f(\omega) & \text{otherwise.} \end{cases}$$

▶ **Theorem 16.** *The feedback functions $RL(\omega)$, $LC(\omega)$ and $S(\omega)$ are de Bruijn successors.*

Proof. Let the partition $\mathbf{R}_1, \mathbf{R}_2, \dots, \mathbf{R}_t$ of $\mathbf{B}(n)$ induced by the PRR be listed in non-increasing order with respect to the run-length of each cycle. Observe that $\mathbf{R}_1$ is the cycle whose strings have run-length n, and thus any representative of $\mathbf{R}_1$ will have run-length n. By definition, this representative is not in the sets $\mathbf{RL}(n)$, $\mathbf{LC}(n)$, and $\mathbf{Same}(n)$. Now consider $\mathbf{R}_i$ for $i > 1$. Clearly the RL-rep for $\mathbf{R}_i$ will begin with 00 or 11 and by definition, the LC-rep for $\mathbf{R}_i$ also begins with 00 or 11. Together these results imply that each same-rep for $\mathbf{R}_i$ will also begin with 00 or 11. Thus, if follows directly from Theorem 14 that $RL(\omega)$, $LC(\omega)$ and $S(\omega)$ are de Bruijn successors. ◀

Recall that $alt(n)$ denotes the alternating sequence of 0s and 1s of length n that ends with 0. Let $\mathcal{X}_n = x_1x_2\cdots x_{2^n}$ be the de Bruijn sequence returned by DB$(S, 0alt(n{-}1))$; it will have suffix equal to the seed $0alt(n{-}1)$. Let $\mathcal{X}'_n$ denote the linearized de Bruijn sequence $alt(n{-}1)\mathcal{X}_n$. Our goal is to show that $\mathcal{X}_n = \mathcal{S}_n$. Our proof requires the following proposition that is proved later in Section 10.

▶ **Proposition 17.** *If β is a string in $\mathbf{B}(n)$ such that the run-length of β is one more than the run-length of $\hat{\beta}$ and neither β nor $\hat{\beta}$ are same-reps, then $\hat{\beta}$ appears before β in $\mathcal{X}'_n$.*

The following proposition follows from n applications of the successor S to the seed $0alt(n{-}1)$.

▶ **Proposition 18.** *$\mathcal{X}_n$ has prefix 1^n.*

▶ **Theorem 19.** *The de Bruijn sequences $\mathcal{S}_n$ and $\mathcal{X}_n$ are the same.*

Proof. Let $\mathcal{S}_n = s_1s_2\cdots s_{2^n}$, let $\mathcal{X}_n = x_1x_2\cdots x_{2^n}$. Recall that $\mathcal{X}_n$ ends with $alt(n{-}1)$. From Proposition 3 and Proposition 18, $x_1x_2\cdots x_n = s_1s_2\cdots s_n = 1^n$ and moreover $\mathcal{S}_n$ and $\mathcal{X}_n$ share the same length $n{-}1$ suffix. Suppose there exists some smallest t, where $n < t \leq 2^n{-}n$, such that $s_t \neq x_t$. Let $\beta = x_{t-n}\cdots x_{t-1}$ denote the length n substring of $\mathcal{X}_n$ ending at position $t{-}1$. Then $x_t \neq x_{t-1}$, because otherwise the RLE of $\mathcal{X}_n$ is lexicographically larger than that of $\mathcal{S}_n$, contradicting Proposition 1. We claim that $\hat{\beta}$ comes before β in $\mathcal{X}'_n$, by considering two cases, recalling $f(\omega) = w_1 \oplus w_2 \oplus w_n$:

- If $x_t = f(\beta)$, then by the definition of S, neither β nor $\hat{\beta}$ are in $\mathbf{Same}(n)$. By the definition of f and since $x_t \neq x_{t-1}$, the first two bits of β must differ from each other. Thus, the run-length of β is one more than the run-length of $\hat{\beta}$. Thus the claim holds by Proposition 17.
- If $x_t \neq f(\beta)$, then either β or $\hat{\beta}$ are in $\mathbf{Same}(n)$. Let $\beta = b_1b_2\cdots b_n$. Then PRR$(\beta) = b_2\cdots b_ns_t$ and PRR$(\hat{\beta}) = b_2\cdots b_nx_t$. From Lemma 9, the strings β and $b_2\cdots b_ns_t$ have the same run-length and the strings $\hat{\beta}$ and $b_2\cdots b_nx_t$ have the same run-length. Since $b_2\cdots b_nx_t$ has run-length one greater than that of $b_2\cdots b_ns_t$, it must be that $\hat{\beta}$ has run-length one greater than that of β. This means that $\hat{\beta}$ must begin with 10 or 01, and hence is not a same-rep, which can be inferred by definition. Thus β is a same-rep and the claim thus holds by Observation 13 (item 1).

Since $\hat{\beta}$ appears before β in $\mathcal{X}'_n$ then $\hat{\beta}$ must be a substring of $alt(n{-}1)x_1\cdots x_{t-2}$. Thus, either $x_{t-n+1}\cdots x_{t-1}x_t$ or $x_{t-n+1}\cdots x_{t-1}s_t$ must be in $alt(n{-}1)x_1\cdots x_{t-1}$ which contradicts the fact that both $\mathcal{X}_n$ and $\mathcal{S}_n$ are de Bruijn sequences. Thus, there is no $n < t \leq 2^n$ such that $s_t \neq x_t$ and hence $\mathcal{S}_n = \mathcal{X}_n$. ◀

7 A de Bruijn successor for $\mathcal{O}_n$

To develop an efficient de Bruijn successor for $\mathcal{O}_n$, we follow an approach similar to that for $\mathcal{S}_n$, except this time we focus on the lexicographically smallest RLEs and RL2-reps. Again, we consider three different representatives for the cycles $\mathbf{R}_1, \mathbf{R}_2, \dots, \mathbf{R}_t$ of $\mathbf{B}(n)$ induced by the PRR.

- *RL2-rep*: The string with the lexicographically smallest RLE; if there are two such strings, it is the one beginning with 0.
- *LC2-rep*: The strings 0^n and 1^n for the classes $\{0^n\}$ and $\{1^n\}$ respectively. For all other classes, it is the string ω with RLE $r_1 r_2 \cdots r_\ell$ such that $r_1 = 1$ and r_2 applications of the PRR starting with ω yields the RL2-rep.
- *opp-rep*: $\begin{cases} \text{RL2-rep} & \text{if the RL2-rep is } \textit{special2} \\ \text{LC2-rep} & \text{otherwise.} \end{cases}$

We say an RL2-rep is *special2* if it belongs to the set $\mathbf{SP2}(n)$ defined as follows:

> $\mathbf{SP2}(n)$ is the set of length n binary strings that begin with 1 and have RLE of the form $1x^zy$ where z is odd and $y > x$.

The RL2-reps have already been illustrated in Section 4. There are a relatively few special2 RL2-reps and they all have odd run-length.

Example 6 The RLEs of the special2 RL2-reps for $n = 10, 11, 12, 13$:

$n = 10$: 111111112, 1111114, 11116, 118, 12223, 127, 136, 145
$n = 11$: 111111113, 1111115, 11117, 119, 12224, 128, 137, 146
$n = 12$: 11111111112, 111111114, 1111116, 11118,11(10), 12225, 129, 138, 147, 156
$n = 13$: 11111111113, 111111115, 1111117, 11119,11(11), 12226, 12(10), 139, 148, 157

Except for the special cases 0^n and 1^n, the LC-rep will begin with 10 and 01. As an example, consider $\omega = 10000101001$ which has RLE $r_1r_2r_3r_4r_5r_6r_7 = 1411121$. It is an LC-rep since $r_2 = 4$ applications of the PRR to ω yields the RL2-rep 01010010000 with RLE 1111214. Note the last value of this RLE will correspond to r_2.

Let $\mathbf{RL2}(n)$, $\mathbf{LC2}(n)$, and $\mathbf{OPP}(n)$ denote the set of all length n RL2-reps, LC2-reps, and opp-reps, respectively, **not including** the representative 0^n. Consider the following feedback functions where $\omega = w_1 w_2 \cdots w_n$ and $f(\omega) = w_1 \oplus w_2 \oplus w_n$:

$$RL2(\omega) = \begin{cases} \overline{f(\omega)} & \text{if } \omega \text{ or } \hat{\omega} \text{ is in } \mathbf{RL2}(n); \\ f(\omega) & \text{otherwise.} \end{cases}$$

$$LC2(\omega) = \begin{cases} \overline{f(\omega)} & \text{if } \omega \text{ or } \hat{\omega} \text{ is in } \mathbf{LC2}(n); \\ f(\omega) & \text{otherwise.} \end{cases}$$

$$O(\omega) = \begin{cases} \overline{f(\omega)} & \text{if } \omega \text{ or } \hat{\omega} \text{ is in } \mathbf{OPP}(n); \\ f(\omega) & \text{otherwise.} \end{cases}$$

▶ **Theorem 20.** *The feedback functions $RL2(\omega)$, $LC2(\omega)$ and $O(\omega)$ are de Bruijn successors.*

Proof. Let the partition $\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_t$ of $\mathbf{B}(n)$ induced by the PRR be listed such that $\mathbf{R}_t = \{1^n\}$ and the remaining $t-1$ cycles are ordered in non-decreasing order with respect to the run-length of each cycle. This means that $\mathbf{R}_1 = \{0^n\}$ and its representative, which must be 0^n, is not in the sets $\mathbf{RL2}(n)$, $\mathbf{LC2}(n)$, and $\mathbf{OPP}(n)$ by their definition. Now consider $\mathbf{R}_i$ for $1 < i < t$. Clearly the RL2-rep for $\mathbf{R}_i$, which is a string with the lexicographically smallest RLE, will begin with 01 or 10. Similarly, the LC2-rep for $\mathbf{R}_i$ must begin with 01 or 10 by its definition. Together these results imply that each opp-rep for $\mathbf{R}_i$ will also begin with 01 or 10. Thus, if follows directly from Theorem 15 that $RL2(\omega)$, $LC2(\omega)$ and $O(\omega)$ are de Bruijn successors. ◀

Recall from Proposition 4 that the length n suffix of $\mathcal{O}_n$ is 10^{n-1}. Let $\mathcal{Y}_n = y_1 y_2 \cdots y_{2^n}$ be the de Bruijn sequence returned by DB$(O, 10^{n-1})$; it will have suffix 10^{n-1}. Let $\mathcal{Y}'_n$ denote the linearized de Bruijn sequence $0^{n-1}\mathcal{Y}_n$. Our goal is to show that $\mathcal{Y}_n = \mathcal{O}_n$. Our proof requires the following proposition that is proved later in Section 11.

▶ **Proposition 21.** *If β is a string in $\mathbf{B}(n)$ such that the run-length of β is one less than the run-length of $\hat{\beta}$ and neither β nor $\hat{\beta}$ are opp-reps, then $\hat{\beta}$ appears before β in $\mathcal{Y}'_n$.*

The following proposition follows from n applications of the successor O to the seed 10^{n-1}.

▶ **Proposition 22.** *$\mathcal{Y}_n$ has length n prefix $0101\cdots$.*

▶ **Theorem 23.** *The de Bruijn sequences $\mathcal{O}_n$ and $\mathcal{Y}_n$ are the same.*

Proof. Let $\mathcal{O}_n = o_1o_2\cdots o_{2^n}$, let $\mathcal{Y}_n = y_1y_2\cdots y_{2^n}$. Recall that $\mathcal{Y}_n$ ends with 0^{n-1}. From Proposition 4 and Proposition 22, $y_1y_2\cdots y_n = o_1o_2\cdots o_n = 0101\cdots$ and moreover $\mathcal{O}_n$ and $\mathcal{Y}_n$ share the same length $n{-}1$ suffix 0^{n-1}. Suppose there exists some smallest t, where $n < t \leq 2^n{-}n$, such that $o_t \neq y_t$. Let $\beta = y_{t-n}\cdots y_{t-1}$ denote the length n substring of $\mathcal{Y}_n$ ending at position $t{-}1$. Then $y_t = y_{t-1}$, because otherwise the RLE of $\mathcal{Y}_n$ is lexicographically smaller than that of $\mathcal{O}_n$, contradicting Proposition 2. We claim that $\hat{\beta}$ comes before β in $\mathcal{Y}'_n$, by considering two cases, recalling $f(\omega) = w_1 \oplus w_2 \oplus w_n$:

- If $y_t = f(\beta)$, then by the definition of O, neither β nor $\hat{\beta}$ are in $\mathbf{OPP}(n)$. By the definition of f and since $y_t = y_{t-1}$, the first two bits of β are the same. Thus, the run-length of β is one less than the run-length of $\hat{\beta}$. Thus the claim holds by Proposition 21.
- If $y_t \neq f(\beta)$, then either β or $\hat{\beta}$ are in $\mathbf{OPP}(n)$. Let $\beta = b_1b_2\cdots b_n$. Then $\text{PRR}(\beta) = b_2\cdots b_n o_t$ and $\text{PRR}(\hat{\beta}) = b_2\cdots b_n y_t$. From Lemma 9, this means β and $b_2\cdots b_n o_t$ have the same run length and $\hat{\beta}$ and $b_2\cdots b_n y_t$ have the same run length. Since $b_2\cdots b_n y_t$ has run-length one less than that of $b_2\cdots b_n o_t$, it must be that $\hat{\beta}$ has run-length one less than that of β. This means $\hat{\beta}$ must begin with 00 or 11 and hence is not an opp-rep, which can be inferred by definition. Thus β is an opp-rep and the claim holds by Observation 13 (item 1).

Since $\hat{\beta}$ appears before β in $\mathcal{Y}'_n$ then $\hat{\beta}$ must be a substring of $0^{n-1}y_1\cdots y_{t-2}$. Thus, either $y_{t-n+1}\cdots y_{t-1}y_t$ or $y_{t-n+1}\cdots y_{t-1}o_t$ must be in $0^{n-1}y_1\cdots y_{t-1}$ which contradicts the fact that both $\mathcal{Y}_n$ and $\mathcal{O}_n$ are de Bruijn sequences. Thus, there is no $n < t \leq 2^n$ such that $o_t \neq y_t$ and hence $\mathcal{O}_n = \mathcal{Y}_n$. ◀

8 Lexicographic compositions

As mentioned earlier, Fredricksen and Kessler devised a construction based on lexicographic compositions [16]. Let $\mathcal{L}_n$ denote the de Bruijn sequence of order n that results from this construction. The sequences $\mathcal{S}_n$ and $\mathcal{L}_n$ first differ at $n = 7$ (as noted below), and for $n \geq 7$ they were conjectured to match for a significant prefix [15, 16]:

$$S_7 = \begin{array}{l} 1111111000000011111011110011110100000100001100001011100011100100 \\ 0110111011000100111010110011001011011010\underline{011010100010100100101010}, \end{array}$$

$$L_7 = \begin{array}{l} 1111111000000011111011110011110100000100001100001011100011100100 \\ 0110111011000100111010110011001011011010\underline{100010100110100100101010}. \end{array}$$

After discovering the de Bruijn successor for $\mathcal{S}_n$, we observed that the de Bruijn sequence resulting from the de Bruijn successor $LC(\omega)$ corresponded to $\mathcal{L}_n$. Recall that $alt(n)$ denotes the alternating sequence of 0s and 1s of length n that ends with 0. Let $\mathcal{LC}_n$ be the de Bruijn sequence returned by $\text{DB}(LC, 0alt(n{-}1))$.

▶ **Conjecture 24.** *The de Bruijn sequences $\mathcal{LC}_n$ and $\mathcal{L}_n$ are the same.*

We have verified that $\mathcal{LC}_n$ is the same as $\mathcal{L}_n$ for all $n < 30$. However, as the description of the algorithm to construct $\mathcal{L}_n$ is rather detailed [16], we did not attempt to prove this conjecture.

9 Efficient implementation

Given a membership tester for $\mathbf{RL}(n)$, testing whether or not a string is an LC-rep or a same-rep can easily be done in $O(n)$ time and $O(n)$ space. Similarly, given the membership tester for $\mathbf{RL2}(n)$, testing whether or not a string is an LC2-rep or a opp-rep can easily be done in $O(n)$ time and $O(n)$ space. Thus, by applying Proposition 10 and Proposition 11, we can implement each of our six de Bruijn successors in $O(n)$ time using $O(n)$ space.

▶ **Theorem 25.** *The six de Bruijn successors $RL(\omega)$, $LC(\omega)$, $S(\omega)$, $RL2(\omega)$, $LC2(\omega)$ and $O(\omega)$ can be implemented in $O(n)$ time using $O(n)$ space.*

10 Proof of Proposition 17

Recall that $\mathcal{X}_n$ = DB$(S, 0alt(n-1))$ and $\mathcal{X}'_n = alt(n-1)\mathcal{X}_n$. We begin by restating Proposition 17 by reversing the roles of β and $\hat{\beta}$ in its original statement for convenience:

> If β is a string in $\mathbf{B}(n)$ such that the run-length of β is one less than the run-length of $\hat{\beta}$ and neither β nor $\hat{\beta}$ are same-reps, then β appears before $\hat{\beta}$ in $\mathcal{X}'_n$.

The first step is to further refine the ordering of the cycles $\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_t$ used in the proof of Theorem 16 to prove that S was a de Bruijn successor. In particular, let $\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_t$ be the cycles of $\mathbf{B}(n)$ induced by the PRR ordered in non-increasing order with respect to the run-lengths of each cycle, additionally refined so the cycles with the same run-lengths are ordered in decreasing order with respect to the RLE (RLE) of the RL-rep. If two RL-reps have the same RLE, then the cycle with RL-rep starting with 1 comes first. Note that this refinement satisfies the ordering of the cycles required in the proof.

Let $\alpha_i, \gamma_i, \sigma_i$ denote the same-rep, LC-rep, and the RL-rep, respectively, for $\mathbf{R}_i$. If β is in $\mathbf{R}_1$ then it has run-length n, which one more than that of $\hat{\beta}$. Thus assume β is in some $\mathbf{R}_i$, where $i > 1$, such that the run-length of β is one less than the run-length of $\hat{\beta}$ where neither β nor $\hat{\beta}$ are same-reps. The run-length constraint implies that the RLE of β must begin with a value greater than 1. We separate two special cases for β that are illustrated in Figure 3 (a): $\beta = \gamma_j$ when σ_j $(= \alpha_j)$ is special and $\beta = \sigma_i$ when $\overline{\sigma_i} = \sigma_j$ is special, in which case $\alpha_i = \gamma_i$. All other possible β are illustrated in Figure 3 (b). We begin by looking at an example for the special cases.

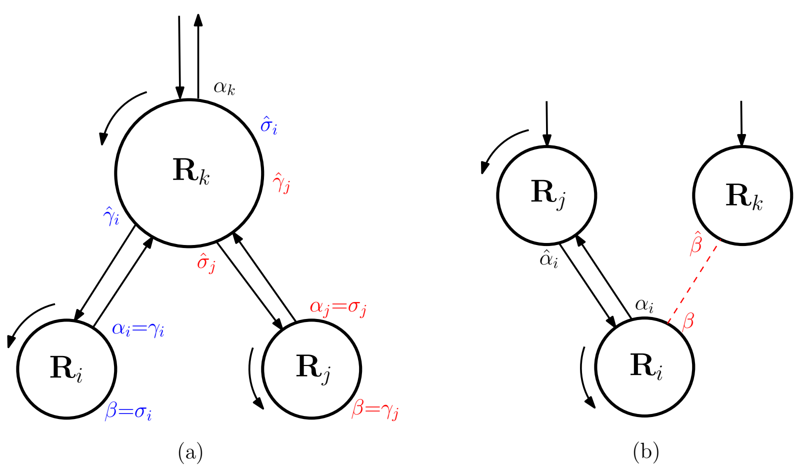


Figure 3 (a) The case when σ_j is special and $\beta = \gamma_j$ together with the case when $\overline{\sigma_i} = \sigma_j$ is special and $\beta = \sigma_i$. (b) All other possible β.

Example 7 Consider $\mathbf{R}_i$ and $\mathbf{R}_j$ with RL-reps $\sigma_i = 11010010101$ and $\sigma_j = 00101101010$. Both have RLE 211211111. Note that σ_i is in $\mathbf{SP}(11)$. The corresponding LC-reps are $\gamma_i = 00101011010$ and $\gamma_j = 11010100101$. The conjugates of all four strings belong to the same cycle $\mathbf{R}_k$. This only happens for these special RLEs. Below is the order that the strings from $\mathbf{R}_k$ appear in $\mathcal{X}'_{11}$, based on Observation 13 (item 2). In particular take notice of the positions of the four conjugates.

01010101011
10101010110
01010101101
10101011010 $\leftarrow$ $\hat{\gamma}_i$
01010110101
10101101010 $\leftarrow$ $\hat{\sigma}_j$
01011010101
10110101010
01101010101
11010101010 $\leftarrow$ σ_k, the RL-rep, with RLE 2111111111
10101010100
01010101001
10101010010
01010100101 $\leftarrow$ $\hat{\gamma}_j$
10101001010
01010010101 $\leftarrow$ $\hat{\sigma}_i$
10100101010
01001010101
10010101010
00101010101 $\leftarrow$ $\alpha_k = \gamma_k$, the same-rep and LC-rep for this cycle

The ordering of the four conjugates from this example are formalized in the second item of the following lemma. As a result of the lemma, and observing Figure 3 (a), we see that β comes before $\hat{\beta}$ in $\mathcal{X}'_n$ for the two special cases.

▶ **Lemma 26.** *Let $\mathbf{R}_i$ and $\mathbf{R}_j$ be cycles such that σ_i and σ_j have the same RLE, where $i < j$.*

(i) If σ_j is not special then $\hat{\alpha}_i$ and $\hat{\alpha}_j$ belong to the same cycle and appear in that relative order within $\mathcal{X}'_n$.

(ii) If σ_j is special, then $\hat{\gamma}_i$, $\hat{\sigma}_j$, $\hat{\gamma}_j$ and $\hat{\sigma}_i$ all belong to the same cycle and they appear in that relative order within $\mathcal{X}'_n$.

Proof. By the ordering of the cycles, since $i < j$ it must be that σ_i begins with 1 and σ_j begins with 0; they belong to PCR-related cycles. Note that $\sigma_i = \overline{\sigma_j}$ and similarly $\gamma_i = \overline{\gamma_j}$. Thus $\hat{\sigma}_i = \overline{\hat{\sigma}_j}$ and $\hat{\gamma}_i = \overline{\hat{\gamma}_j}$ and each pair, respectively, will belong to the same CCR-related cycle.
Case (i): If σ_j is not special, then $\alpha_i = \gamma_i$ and $\alpha_j = \gamma_j$. Suppose the RLE for σ_i is $r_1 r_2 \cdots r_m 1^v$, where $r_m \geq 2$. Note also that $r_1 \geq 2$ by the definition of RL-rep. Also $m + v$ is odd, since $\mathbf{R}_i$ is a PCR-related cycle. Then

- γ_i has RLE $21^{v-1} r_1 r_2 \ldots r_{m-1}(r_m{-}1)$ and $\text{PRR}^{v+1}(\gamma_i) = \sigma_i$,
- $\hat{\gamma}_i$ has RLE $1^{v+1} r_1 r_2 \ldots r_{m-1}(r_m{-}1)$,
- σ_k has RLE $r_1 r_2 \ldots r_{m-1}(r_m{-}1)1^{v+1}$ where $\hat{\gamma}_i \in \mathbf{R}_k$.

The third item is obtained by applying the definition of an RL-rep, using the fact that σ_i is an RL-rep. Since $\mathbf{R}_k$ is a CCR-related cycle (it has even run-length $m + v + 1$), its RL-rep σ_k begins with 1. Note then that $\text{PRR}^{v+1}(\hat{\gamma}_i) = \sigma_k$. As a special case, if σ_k has RLE 1^n, then $k = 1$ and moreover σ_i has RLE 21^{n-2}. Note $\sigma_i = \gamma_i = \alpha_i$. Since $\mathcal{X}_n$ begins with 1^n (Proposition 18), the first length n substring in $\mathcal{X}'_n$ is $0101 \cdots 01 = \hat{\alpha}_i$. Thus $\hat{\alpha}_i$ clearly comes before $\hat{\alpha}_j$ within $\mathcal{X}'_n$. For all remaining cases, if $r_m > 2$, then $\text{PRR}^{v+2}(\gamma_k) = \sigma_k$. Otherwise, if $m = 2$, let v' denote the length of the longest suffix of $r_1 r_2 \dots r_{m-1}(r_m{-}1)$ consisting only of 1s. Note this number is less than m since we already handled the special case where σ_i has RLE 21^{n-2}. In this case $\text{PRR}^{v+2+v'}(\gamma_k) = \sigma_k$. If there are $2z$ strings in $\mathbf{R}_k$, clearly $v + 2 + v'$ will be less than z. Moreover, $\text{PRR}^{z}(\hat{\gamma}_i) = \hat{\gamma}_j$ since $\hat{\gamma}_i = \overline{\hat{\gamma}_j}$. Thus by Observation 13 (item 2), $\hat{\gamma}_i$ comes before σ_k which comes before $\hat{\gamma}_j$ in $\mathcal{X}'_n$.

Case (ii): If σ_j is special, then $\alpha_i = \gamma_i$ and $\alpha_j = \sigma_j$. Since σ_j is special it begins and ends with 0 and it has RLE of the form $(21^{2x})^y 1^z$, where $x \geq 0$, $y \geq 2$, and $z \geq 2$. Thus:

- $\hat{\sigma}_j$ has RLE $1^{2x+2}(21^{2x})^{y-1}1^z$, and begins with 1,
- γ_j has RLE $21^{2x+z-1}(21^{2x})^{y-1}1$, and
- $\hat{\gamma}_j$ has RLE $1^{2x+z+1}(21^{2x})^{y-1}1$,
- σ_k has RLE $(21^{2x})^{y-1}1^{2x+z+2}$ where $\hat{\sigma}_j \in \mathbf{R}_k$, and begins with 1 since it is a CCR-related cycle.

The final item is obtained by applying the definition of an RL-rep, using the fact that σ_i is an RL-rep. Since $\mathbf{R}_k$ is a CCR-related cycle we have $\alpha_k = \gamma_k$ and based on the RLE of σ_k, the cycle will contain $2n - 2$ distinct strings. Thus for every string $\omega \in \mathbf{R}_k$, $\text{PRR}^{n-1}(\omega) = \overline{\omega}$. By the definition of LC-rep, $\text{PRR}^{(2x+z+2)+2x+1}(\gamma_k) = \sigma_k$. Note also that $\text{PRR}^{2x+2}(\hat{\sigma}_j) = \sigma_k$. Observe now that $\text{PRR}^{2x+z+1}(\hat{\gamma}_j)$ will have RLE $(21^{2x})^{y-1}1^{z+2x+2}$ and begin with 0; it is the complement of σ_k. Thus, $\text{PRR}^{2x+z+1}(\hat{\gamma}_i) = \sigma_k$. Putting it all together we have:

- $\text{PRR}^{2x+2}(\alpha_k) = \hat{\gamma}_i$
- $\text{PRR}^{2x+z+1}(\alpha_k) = \hat{\sigma}_j$
- $\text{PRR}^{2x+z+2+2x+1}(\alpha_k) = \sigma_k$
- $\text{PRR}^{n-1+2x+2}(\alpha_k) = \hat{\gamma}_j$
- $\text{PRR}^{n-1+2x+z+1}(\alpha_k) = \hat{\sigma}_i$

The result now follows from Observation 13 (item 2). ◀

▶ **Corollary 27.** *If $\mathbf{R}_i$ and $\mathbf{R}_j$ are cycles such that σ_i and σ_j have the same RLE, where $i < j$, then every string from $\mathbf{R}_i$ appears before every string from $\mathbf{R}_j$ in $\mathcal{X}'_n$.*

Proof. In case (ii) from Lemma , since σ_j is special then $\alpha_j = \sigma_j$ and $\alpha_i = \gamma_i$. Thus, an immediate consequence of Lemma is that $\hat{\alpha}_i$ appears before $\hat{\alpha}_j$ in $\mathcal{X}'_n$. Then by Observation 13 (item 5 and item 3), every string in $\mathbf{R}_i$ appears before every string in $\mathbf{R}_j$ in $\mathcal{X}'_n$. ◀

For all $\beta \in \mathbf{R}_i$ other than these two special cases, assume that $\hat{\alpha}_i$ belongs to $\mathbf{R}_j$ and $\hat{\beta}$ belongs to $\mathbf{R}_k$ – see Figure 3 (b). We will show that $j < k$ and subsequently that all strings in $\mathbf{R}_j$ come before all strings in $\mathbf{R}_k$ in $\mathcal{X}'_n$. Suppose the RLE for σ_i is $r_1 r_2 \cdots r_m 1^v$, where $r_m \geq 2$. Then

- γ_i has RLE $21^{v-1} r_1 r_2 \dots r_{m-1}(r_m{-}1)$,
- $\hat{\gamma}_i$ has RLE $1^{v+1} r_1 r_2 \dots r_{m-1}(r_m{-}1)$,
- σ_j has RLE $r_1 r_2 \dots r_{m-1}(r_m{-}1)1^{v+1}$ where $\hat{\gamma}_i \in \mathbf{R}_j$.

The third item is obtained by applying the definition of an RL-rep, using the fact that σ_i is an RL-rep. Moreover, note that σ_j begins with 1 if $\mathbf{R}_j$ corresponds to a CCR-related cycle. Otherwise, $\mathbf{R}_i$ must correspond to a CCR-related cycle which means that σ_i begins with 1 and hence again σ_j will begin with 1 based on the RLEs described above. We now consider two cases depending on whether or not $\beta = \overline{\alpha_i}$.

If $\beta = \overline{\alpha_i}$, then $\mathbf{R}_i$ must be a CCR-related cycle. Thus $\alpha_i = \gamma_i$ begins with 1 and hence β begins with 0. It is easy to see from the RLE of σ_j noted above that it will also begin with 1. Thus σ_k, which will have the same RLE as σ_j, begins with 0. From the ordering defined on the cycles, $j < k$. Thus by Lemma 10, $\hat{\alpha}_j$ appears before $\hat{\alpha}_k$ which implies that all strings in $\mathbf{R}_j$ appear before all strings in $\mathbf{R}_k$ in $\mathcal{X}'_n$ by Observation 13 (item 3). Furthermore, Observation 13 (item 4) implies that β will appear before $\hat{\beta}$ in $\mathcal{X}'_n$.

If $\beta \neq \overline{\alpha_i}$, then we first consider the case where either α_i or $\overline{\alpha_i}$ is special. We have already handled the two special cases where $\beta = \gamma_i$ or $\beta = \sigma_i$. Since the RLE for β cannot begin with 1, β must be of the form $(21^{2x})^{y-q}1^z(21^{2x})^q$, where $x \geq 0$, $y \geq 2$, $z \geq 2$, and $1 \leq q \leq y-2$. Thus $\hat{\beta}$ has RLE $1^{2x+2}(21^{2x})^{y-q-1}1^z(21^{2x})^q$. It is not hard to see σ_k will have a smaller RLE compared to σ_j which is detailed in the proof of Lemma 10. A similar analysis can be done when neither α_i nor $\overline{\alpha_i}$. For these cases, it is a relatively straightforward task to observe that the RLE for σ_k is less than the RLE for σ_j which means $j < k$. We can now apply the following lemma.

▶ **Lemma 28.** *If $\mathbf{R}_j$ and $\mathbf{R}_k$ have the same run-length where $j < k$ and σ_j and σ_k both begin with the same symbol, then every string in $\mathbf{R}_j$ appears in $\mathcal{X}'_n$ before any string in $\mathbf{R}_k$.*

Proof. The proof is by induction on the levels of the related tree of cycles rooted by $\mathbf{R}_1$, which is the unique cycle with run-length n. The base case trivially holds for cycles with run-length n since there is only one such cycle $\mathbf{R}_1$. Now assume that the result holds for all cycles at levels with run length greater than $\ell < n$, and consider two cycles $\mathbf{R}_j$ and $\mathbf{R}_k$ with run-length ℓ such that σ_j and σ_k both begin with the same symbol. By the ordering of the cycles the RLE of σ_j is greater than the RLE of σ_k. From Lemma 10, if σ_j is special, then $\hat{\alpha}_j$ belongs to the same cycle as $\hat{\gamma}_j$. Similarly for σ_k. Thus we need only focus on the RLE of the RL-reps σ_x and σ_y for the cycles $\mathbf{R}_x$ and $\mathbf{R}_y$ containing $\hat{\gamma}_j$ and $\hat{\gamma}_k$ respectively. From our earlier analysis (case (i) in the proof of Lemma 10), we analyzed the RLE of these strings, and it can be observed that the RLE for σ_x is greater than the RLE for σ_y since the RLE for σ_j is greater than the RLE for σ_k. Thus by the ordering of the cycles $x < y$. As noted earlier both $\mathbf{R}_x$ and $\mathbf{R}_y$ (any non-leaf in the related tree) must begin with 1. By induction, this means that the every string from the cycle containing $\mathbf{R}_x$ appears before every string from $\mathbf{R}_y$ in $\mathcal{X}'_n$, and hence by Observation 13 (item 4), we have our result. ◀

Recall that σ_j begins with 1 and $\sigma_j < \sigma_k$. Thus if σ_k begins with 1, then the above lemma implies that all strings in $\mathbf{R}_j$ appear before all strings in $\mathbf{R}_k$. Otherwise if σ_k begins with 0, then it must correspond to a PCR-related cycle. Consider $\mathbf{R}_{k'}$ containing RL-rep $\overline{\sigma_k}$ which begins with 1 and has the same RLE as σ_k. From the above lemma all strings in $\mathbf{R}_j$ will appear before all strings in $\mathbf{R}_{k'}$ which in turn come before all strings in $\mathbf{R}_k$ in $\mathcal{X}'_n$ by Corollary 27. By applying Observation 13 (item 4), as we did earlier, we have that all strings in $\mathbf{R}_i$ including β will appear before all strings in $\mathbf{R}_k$ including $\hat{\beta}$ in $\mathcal{X}'_n$. This completes the proof of Proposition 17.

11 Proof of Proposition 21

The proof of this proposition follows the same technical steps as the proof for Proposition 17. Recall that $\mathcal{Y}_n$ = DB$(O, 10^{n-1})$ and $\mathcal{Y}'_n = 0^{n-1}\mathcal{Y}_n$. We begin by restating Proposition 21 by reversing the roles of β and $\hat{\beta}$ in its original statement for convenience:

> If β is a string in $\mathbf{B}(n)$ such that the run-length of β is one more than the run-length of $\hat{\beta}$ and neither β nor $\hat{\beta}$ are opp-reps, then β appears before $\hat{\beta}$ in $\mathcal{Y}'_n$.

The first step is to further refine the ordering of the cycles $\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_t$ used in the proof of Theorem 20 to prove that O was a de Bruijn successor. To begin, recall that $\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_t$ was listed such that $\mathbf{R}_t = \{1^n\}$ with the remaining $t-1$ cycles ordered in non-decreasing order with respect to the run-length of each cycle. Thus, $\mathbf{R}_1 = \{0^n\}$. This listing is refined so that the cycles with the same run-lengths are ordered in increasing order with respect to the RLE of the RL2-rep. If two RL2-reps have the same RLE, then the cycle with RL2-rep starting with 0 comes first. Note that this refinement still satisfies the ordering of the cycles required in the proof.

Let $\alpha_i, \gamma_i, \sigma_i$ denote the opp-rep, LC2-rep, and the RL2-rep, respectively, for $\mathbf{R}_i$. The only strings in $\mathbf{R}_1$ and $\mathbf{R}_t$ are opp-reps. Thus assume β is in some $\mathbf{R}_i$, where $1 < i < t$, such that the run-length of β is one more than the run-length of $\hat{\beta}$ and neither β nor $\hat{\beta}$ are opp-reps. This run-length constraint implies that the RLE of β must begin with 1. As before we have two special cases that are illustrated in the following example. In general, the special cases can be visualized by Figure 3 (a).

Example 8 Consider $\mathbf{R}_i$ and $\mathbf{R}_j$ with RL2-reps $\sigma_i = 01100110000$ $\sigma_j = 10011001111$. Both have RLE 12224. Note that σ_j is in $\mathbf{SP2}(11)$. The corresponding LC2-reps are $\gamma_i = 10000110011$ and $\gamma_j = 01111001100$. The conjugates of all four strings belong to the same cycle $\mathbf{R}_k$. Below is the order that the strings from $\mathbf{R}_k$ appear in $\mathcal{Y}'_{11}$, based on Observation 13 (item 2). In particular take notice of the positions of the four conjugates.

00000011001
00000110011 $\leftarrow$ $\hat{\gamma}_i$
00001100111
00011001111 $\leftarrow$ $\hat{\sigma}_j$
00110011111
01100111111 $\leftarrow$ σ_k, the RL2-rep, with RLE 1226
11001111110
10011111100
00111111001
01111110011
11111100110
11111001100 $\leftarrow$ $\hat{\gamma}_j$
11110011000
11100110000 $\leftarrow$ $\hat{\sigma}_i$
11001100000
10011000000 $\leftarrow$ $\overline{\sigma_k}$
00110000001
01100000011
11000000110
10000001100 $\leftarrow$ $\alpha_k = \gamma_k$, the opp-rep and LC2-rep for this cycle

The ordering of the four conjugates from this example are formalized in the second item of the following lemma. As a result of the lemma, and observing Figure 3 (a), we see that β comes before $\hat{\beta}$ in $\mathcal{Y}'_n$ for the two special cases: $\beta = \gamma_j$ when $\sigma_j = \alpha_j$ is special and $\beta = \sigma_i$ when $\overline{\sigma_i} = \sigma_j$.

▶ **Lemma 29.** *Let $\mathbf{R}_i$ and $\mathbf{R}_j$ be cycles such that σ_i and σ_j have the same RLE, where $1 < i < j < t$.*

(i) If σ_j is not special then $\hat{\alpha}_i$ and $\hat{\alpha}_j$ belong to the same cycle and appear in that relative order within $\mathcal{Y}'_n$.

(ii) If σ_j is special, then $\hat{\gamma}_i$, $\hat{\sigma}_j$, $\hat{\gamma}_j$ and $\hat{\sigma}_i$ all belong to the same cycle and they appear in that relative order within $\mathcal{Y}'_n$.

Proof. By the ordering of the cycles, since $i < j$ it must be that σ_i begins with 0 and σ_j begins with 1; they belong to PCR-related cycles. Note that $\sigma_i = \overline{\sigma_j}$ and similarly $\gamma_i = \overline{\gamma_j}$. Thus $\hat{\sigma}_i = \overline{\hat{\sigma}_j}$ and $\hat{\gamma}_i = \overline{\hat{\gamma}_j}$ and each pair, respectively, will belong to the same CCR-related cycle.
Case (i): If σ_j is not special, then $\alpha_i = \gamma_i$ and $\alpha_j = \gamma_j$. Since $1 < i < j < t$, the run-lengths of $\mathbf{R}_i$ and $\mathbf{R}_j$ must be greater than one. Since there is only one cycle with run length n, the run-lengths of $\mathbf{R}_i$ and $\mathbf{R}_j$ must be less than n. Thus, suppose the RLE for σ_i is $1^v r_1 r_2 \cdots r_m$, where $v > 1$ and $r_1 \geq 2$. Note also that $r_m \geq 2$ since otherwise there is a string with RLE $1^{v+1} r_1 \cdots r_{m-1}$ in $\mathbf{R}_i$

which contradicts the fact that σ_i is the RL2-rep. Also $m+v$ is odd, since $\mathbf{R}_i$ is a PCR-related cycle. Then

- γ_i has RLE $1r_m1^{v-1}r_1r_2\cdots r_{m-1}$,
- $\hat{\gamma}_i$ has RLE $(r_m{+}1)1^{v-1}r_1r_2\cdots r_{m-1}$,
- σ_k has RLE $1^v r_1r_2\cdots(r_{m-1}+r_m)$ where $\hat{\gamma}_i \in \mathbf{R}_k$.

The third item is obtained by applying the definition of an RL2-rep, using the fact that σ_i is an RL2-rep. Since $\mathbf{R}_k$ is a CCR-related cycle (it has even run-length $m+v+1$), its RL2-rep σ_k begins with 0. Note then that $\text{PRR}^{r_m}(\hat{\gamma}_i)=\sigma_k$. From the discussion of LC2-reps, $\text{PRR}^{r_{m-1}+r_m}(\gamma_k)=\sigma_k$. If there are $2z$ strings in $\mathbf{R}_k$, clearly $r_{m-1}+r_m$ will be less than z. Moreover, $\text{PRR}^z(\hat{\gamma}_i)=\hat{\gamma}_j$ since $\hat{\gamma}_i=\overline{\hat{\gamma}_j}$. Thus by Observation 13 (item 2), $\hat{\gamma}_i$ comes before σ_k which comes before $\hat{\gamma}_j$ in $\mathcal{Y}'_n$.

Case (ii): If σ_j is special, then $\alpha_i=\gamma_i$ and $\alpha_j=\sigma_j$. Since σ_j is special it begins with 1 and has RLE of the form $1x^zy$, where z is odd and $y>x$. Thus:

- $\hat{\sigma}_j$ has RLE $(x{+}1)x^{z-1}y$, and begins with 0,
- γ_j has RLE $1yx^{z-1}$, and
- $\hat{\gamma}_j$ has RLE $(y{+}1)x^{z-1}$,
- σ_k has RLE $1x^{z-1}(y{+}x)$ where $\hat{\sigma}_j \in \mathbf{R}_k$, and begins with 0 since it is a CCR-related cycle.

The final item is obtained by applying the definition of an RL2-rep, using the fact that σ_i is an RL2-rep. Since $\mathbf{R}_k$ is a CCR-related cycle we have $\alpha_k=\gamma_k$ and based on the RLE of σ_k, the cycle will contain $2n-2$ distinct strings. Thus for every string $\omega \in \mathbf{R}_k$, $\text{PRR}^{n-1}(\omega)=\overline{\omega}$. From the discussion of LC2-reps, $\text{PRR}^{y+x}(\gamma_k)=\sigma_k$. Note also that $\text{PRR}^x(\hat{\sigma}_j)=\sigma_k$. Observe now that $\text{PRR}^y(\hat{\gamma}_j)$ will have RLE $1x^{z-1}(y{+}x)$ and begin with 1; it is the complement of σ_k. Thus, $\text{PRR}^y(\hat{\gamma}_i)=\sigma_k$. Putting it all together, recalling $\gamma_k=\alpha_k$, we have:

- $\text{PRR}^x(\alpha_k)=\hat{\gamma}_i$
- $\text{PRR}^y(\alpha_k)=\hat{\sigma}_j$
- $\text{PRR}^{x+y}(\alpha_k)=\sigma_k$
- $\text{PRR}^{(x+1)z+y+1}(\alpha_k)=\hat{\gamma}_j$
- $\text{PRR}^{xz+2y+1}(\alpha_k)=\hat{\sigma}_i$

The result now follows from Observation 13 (item 2). ◀

▶ **Corollary 30.** *If $\mathbf{R}_i$ and $\mathbf{R}_j$ are cycles such that σ_i and σ_j have the same RLE, where $i<j$, then every string from $\mathbf{R}_i$ appears before every string from $\mathbf{R}_j$ in $\mathcal{Y}'_n$.*

Proof. In case (ii) from Lemma , since σ_j is special then $\alpha_j=\sigma_j$ and $\alpha_i=\gamma_i$. Thus, an immediate consequence of Lemma is that $\hat{\alpha}_i$ appears before $\hat{\alpha}_j$ in $\mathcal{Y}'_n$. Then by Observation 13 (item 5 and item 3), every string in $\mathbf{R}_i$ appears before every string in $\mathbf{R}_j$ in $\mathcal{Y}'_n$. ◀

For all $\beta \in \mathbf{R}_i$ other than these two special cases, assume that $\hat{\alpha}_i$ belongs to $\mathbf{R}_j$ and $\hat{\beta}$ belongs to $\mathbf{R}_k$ – see Figure 3 (b). We will show that $j<k$ and subsequently that all strings in $\mathbf{R}_j$ come before all strings in $\mathbf{R}_k$ in $\mathcal{Y}'_n$. Suppose the RLE for σ_i is $1^v r_1r_2\cdots r_m$, where $r_1 \geq 2$. Since $1<i<t$, clearly $v>1$. Then

- γ_i has RLE $1r_m1^{v-1}r_1r_2\cdots r_{m-1}$,
- $\hat{\gamma}_i$ has RLE $(r_m{+}1)1^{v-1}r_1r_2\cdots r_{m-1}$,
- σ_j has RLE $1^v r_1r_2\cdots(r_{m-1}+r_m)$ where $\hat{\gamma}_i \in \mathbf{R}_j$.

The third item is obtained by applying the definition of an RL2-rep, using the fact that σ_i is an RL2-rep. Moreover, note that σ_j begins with 0 if $\mathbf{R}_j$ corresponds to a CCR-related cycle. Otherwise, $\mathbf{R}_i$ must correspond to a CCR-related cycle which means that σ_i begins with 0 and hence again σ_j will begin with 0 based on the RLEs described above. We now consider two cases depending on whether or not $\beta=\overline{\alpha_i}$.

If $\beta=\overline{\alpha_i}$, then $\mathbf{R}_i$ must be a CCR-related cycle. Thus $\alpha_i=\gamma_i$ begins with 0 and hence β begins with 1. It is easy to see from the RLE of σ_j noted above that it will also begin with 0. Thus σ_k, which will have the same RLE as σ_j, begins with 1. From the ordering defined on the cycles, $j < k$. Thus by Lemma 11, $\hat{\alpha}_j$ appears before $\hat{\alpha}_k$ which implies that all strings in $\mathbf{R}_j$ appear before all strings in $\mathbf{R}_k$ in $\mathcal{X}'_n$ by Observation 13 (item 3). Furthermore, Observation 13 (item 4) implies that β will appear before $\hat{\beta}$ in $\mathcal{X}'_n$.

If $\beta \neq \overline{\alpha_i}$, then we first consider the case where either α_i or $\overline{\alpha_i}$ is special. We have already handled the two special cases where $\beta = \gamma_i$ or $\beta = \sigma_i$. Since the RLE for β must begin with 1 and $\alpha \neq \beta$, β must be of the form $1^q y 1^{z-q+1}$ where z is odd and $y > z$ and $1 \leq q \leq z-2$. Thus $\hat{\beta}$ has RLE $21^{q-2}y1^{z-q+1}$. It is not hard to see σ_k will have a smaller RLE compared to σ_j which is detailed in the proof of Lemma 11. A similar analysis can be done when neither α_i nor $\overline{\alpha_i}$ are special. For these cases, it is a relatively straightforward task to observe that the RLE for σ_k is less than the RLE for σ_j which means $j < k$. We can now apply the following lemma.

▶ **Lemma 31.** *If $\mathbf{R}_j$ and $\mathbf{R}_k$ have the same run-length where $j < k$ and σ_j and σ_k both begin with the same symbol, then every string in $\mathbf{R}_j$ appears in $\mathcal{Y}'_n$ before any string in $\mathbf{R}_k$.*

Proof. The proof is by induction on the levels of the related tree of cycles rooted by $\mathbf{R}_1$. Recall $\mathbf{R}_1 = \{0^n\}$ and $\mathbf{R}_t = \{1^n\}$. The base case trivially holds for cycles with run-length 1, since we previously demonstrated that $\mathcal{Y}'_n$ begins with 0^n. Now assume the result holds for all cycles at levels with run length less than $\ell > 1$, and consider two cycles $\mathbf{R}_j$ and $\mathbf{R}_k$ with run-length ℓ such that σ_j and σ_k both begin with the same symbol. By the ordering of the cycles the RLE of σ_j is less than the RLE of σ_k. From Lemma 11, if σ_j is special, then $\hat{\alpha}_j$ belongs to the same cycle as $\hat{\gamma}_j$. Similarly for σ_k. Thus we need only focus on the RLE of the RL2-reps σ_x and σ_y for the cycles $\mathbf{R}_x$ and $\mathbf{R}_y$ containing $\hat{\gamma}_j$ and $\hat{\gamma}_k$ respectively. From our earlier analysis (case (i) in the proof of Lemma 11), we analyzed the RLE of these strings, and it can be observed that the RLE for σ_x is less than the RLE for σ_y since the RLE for σ_j is less than the RLE for σ_k. Thus by the ordering of the cycles $x < y$. As noted earlier both $\mathbf{R}_x$ and $\mathbf{R}_y$ (any non-leaf in the related tree) must begin with 0. By induction, this means that the every string from the cycle containing $\mathbf{R}_x$ appears before every string from $\mathbf{R}_y$ in $\mathcal{X}'_n$, and hence by Observation 13 (item 4), we have our result. ◀

Recall that σ_j begins with 0 and $\sigma_j < \sigma_k$. Thus if σ_k begins with 0, then the above lemma implies that all strings in $\mathbf{R}_j$ appear before all strings in $\mathbf{R}_k$. Otherwise if σ_k begins with 1, then it must correspond to a PCR-related cycle. Consider $\mathbf{R}_{k'}$ containing RL2-rep $\overline{\sigma_k}$ which begins with 0 and has the same RLE as σ_k. From the above lemma all strings in $\mathbf{R}_j$ will appear before all strings in $\mathbf{R}_{k'}$ which in turn come before all strings in $\mathbf{R}_k$ in $\mathcal{Y}'_n$ by Corollary 30. By applying Observation 13 (item 4), as we did earlier, we have that all strings in $\mathbf{R}_i$ including β will appear before all strings in $\mathbf{R}_k$ including $\hat{\beta}$ in $\mathcal{Y}'_n$. This completes the proof of Proposition 21.

References

1. A. Alhakim. A simple combinatorial algorithm for de Bruijn sequences. *The American Mathematical Monthly*, 117(8):728–732, 2010.
2. A. Alhakim. Spans of preference functions for de Bruijn sequences. *Discrete Applied Mathematics*, 160(7-8):992 – 998, 2012.
3. A. Alhakim, E. Sala, and J. Sawada. Revisiting the prefer-same and prefer-opposite de Bruijn sequence constructions. *Theoretical Computer Science*, 2020 (to appear).
4. J. Aycock. *Retrogame Archeology*. Springer International Publishing, 2016.
5. K. S. Booth. Lexicographically least circular substrings. *Inform. Process. Lett.*, 10(4/5):240–242, 1980.
6. P. E. C. Compeau, P. A. Pevzner, and G. Tesler. How to apply de Bruijn graphs to genome assembly. *Nature Biotechnology*, 29(11):987–991, 2011.
7. N. G. de Bruijn. A combinatorial problem. *Indagationes Mathematicae*, 8:461–467, 1946.

8 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. *European J. Combin.*, 72:1–11, 2018.

9 J. P. Duval. Factorizing words over an ordered alphabet. *Journal of Algorithms*, 4(4):363–381, 1983.

10 C. Eldert, H. Gray, H. Gurk, and M. Rubinoff. Shifting counters. *AIEE Trans.*, 77:70–74, 1958.

11 T. Etzion. Self-dual sequences. *Journal of Combinatorial Theory, Series A*, 44(2):288 – 298, 1987.

12 M. Fleury. Deux problemes de geometrie de situation. *Journal de mathematiques elementaires*, 42:257–261, 1883.

13 C. Flye Sainte-Marie. Solution to question nr. 48. *L'intermédiaire des Mathématiciens*, 1:107–110, 1894.

14 H. Fredricksen. Generation of the Ford sequence of length 2^n, n large. *J. Combin. Theory Ser. A*, 12(1):153–154, 1972.

15 H. Fredricksen. A survey of full length nonlinear shift register cycle algorithms. *Siam Review*, 24(2):195–221, 1982.

16 H. Fredricksen and I. Kessler. Lexicographic compositions and de Bruijn sequences. *J. Combin. Theory Ser. A*, 22(1):17 – 30, 1977.

17 H. Fredricksen and J. Maiorana. Necklaces of beads in k colors and k-ary de Bruijn sequences. *Discrete Math.*, 23:207–210, 1978.

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

19 D. Gabric and J. Sawada. Investigating the discrepancy property of de Bruijn sequences. *Submitted manuscript*, 2020.

20 D. Gabric, J. Sawada, A. Williams, and D. Wong. A framework for constructing de Bruijn sequences via simple successor rules. *Discrete Mathematics*, 341(11):2977 – 2987, 2018.

21 D. Gabric, J. Sawada, A. Williams, and D. Wong. A successor rule framework for constructing k -ary de Bruijn sequences and universal cycles. *IEEE Transactions on Information Theory*, 66(1):679–687, 2020.

22 S. W. Golomb. *Shift Register Sequences*. Aegean Park Press, Laguna Hills, CA, USA, 1981.

23 C. Hierholzer. Deux problemes de geometrie de situation. *Journal de mathematiques elementaires*, 42:257–261, 1873.

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

25 A. Klein. *Stream Ciphers*. Springer-Verlag London, 2013.

26 M. H. Martin. A problem in arrangements. *Bull. Amer. Math. Soc.*, 40(12):859–864, 1934.

27 P. A. Pevzner, H. Tang, and M. S. Waterman. An eulerian path approach to dna fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001.

28 E. Sala. Exploring the greedy constructions of de Bruijn sequences. Master's thesis, University of Guelph, 2018.

29 J. Sawada, A. Williams, and D. Wong. A surprisingly simple de Bruijn sequence construction. *Discrete Math.*, 339:127–131, 2016.

30 A. Williams. The greedy Gray code algorithm. In F. Dehne, R. Solis-Oba, and J.-R. Sack, editors, *Algorithms and Data Structures*, pages 525–536, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

31 S. Xie. Notes on de Bruijn sequences. *Discrete Applied Mathematics*, 16(2):157 – 177, 1987.

A Implementation of the de Bruijn successors $RL(\omega)$, $LC(\omega)$, and $S(\omega)$

```
#include<stdio.h>
#include<math.h>
#define N_MAX 50
int n;

// =============================================================================
// Compute the RLE of a[1..m] in run[1..r], returning r = ruh length
// =============================================================================
int RLE(int a[], int run[], int m) {
    int i,j,r,old;

    old = a[m+1];
    a[m+1] = 1 - a[m];
    r = j = 0;
    for (i=1; i<=m; i++) {
        if (a[i] == a[i+1]) j++;
        else {  run[++r] = j+1;  j = 0;  }
    }
    a[m+1] = old;
    return r;
}
// ===============================================================================
// Check if a[1..n] is a "special" RL representative.  It must be that a[1] = a[n]
// and the RLE of a[1..n] is of the form (21^j)^s1^t where j is even, s >=2, t>=2
// ===============================================================================
int Special(int a[]) {
    int i,j,r,s,t,run[N_MAX];

    if (a[1] != 0  || a[n] != 0) return 0;
    r = RLE(a,run,n);

    // Compute j of prefix 21^j
    if (run[1] != 2) return 0;
    j = 0;
    while (run[j+2] == 1 && j+2 <= r) j++;

    // Compute s of prefix (21^j)^s
    s = 1;
    while (s <= r/(1+j) -1 && run[s*(j+1)+1] == 2) {
        for (i=1; i<=j; i++) if (run[s*(j+1)+1+i] != 1) return 0;
        s++;
    }

    // Test remainder of string is (21^j)^s is 1^t
    for (i=s*(j+1)+1; i<=r; i++) if (run[i] != 1) return 0;
    t = r - s*(1+j);

    if (s >= 2 && t >= 2 && j%2 == 0) return 1;
    return 0;
}
// =============================================================================
// Apply PRR^{t+1} to a[1..n] to get b[1..n], where t is the length of the
// prefix before the first 00 or 11 in a[2..n] up to n-2
// =============================================================================
int Shift(int a[], int b[]) {
    int i,t = 0;
    while (a[t+2] != a[t+3] && t < n-2) t++;
    for (i=1; i<=n; i++) b[i] = a[i];
    for (i=1; i<=n; i++) b[i+n] = (b[i] + b[i+1] + b[n+i-1]) % 2;
    for (i=1; i<=n; i++) b[i] = b[i+t+1];
    return t;
}
// =============================================================================
// Test if b[1..len] is the lex largest rep (under rotation), if so, return the
// period p; otherwise return 0. Eg. (411411, p=3)(44211, p=5) (411412, p=0).
// =============================================================================
int IsLargest(int b[], int len) {
    int i, p=1;
    for (i=2; i<=len; i++) {
        if (b[i-p] < b[i]) return 0;
        if (b[i-p] > b[i]) p = i;
    }
    if (len % p != 0) return 0;
    return p;
}
```

```
// ==========================================================================
// Membership testers not including the cycle containing 0101010...
// ==========================================================================
int RLrep(int a[]) {
    int p,r,rle[N_MAX];

    r = RLE(a,rle,n-1);
    p = IsLargest(rle,r);

    // PCR-related cycle
    if (a[1] == a[n]) {
        if (r == n-1 && a[1] == 1) return 0;  // Ignore root a[1..n] = 1010101..
        if (r == 1) return 1;  // Special case: a[1..n] = 000..0 or 111..1
        if (p > 0 && a[1] != a[n-1] && (p == r || a[1] == 1 || p%2 == 0)) return 1;
    }
    // CCR-related cycle
    if (a[1] != a[n]) {
        if (p > 0 && a[1] == 1 && (a[n-1] == 1)) return 1;
    }
    return 0;
}
// ==========================================================================
int LCrep(int a[]) {
    int b[N_MAX];

    if (a[1] != a[2]) return 0;
    Shift(a,b);
    return RLrep(b);
}
// ==========================================================================
int SameRep(int a[]) {
    int b[N_MAX];

    Shift(a,b);
    if (Special(a) || (LCrep(a) && !Special(b))) return 1;
    return 0;
}
// ==========================================================================
// Repeatedly apply the Prefer-Same or LC or RL successor rule starting with 1^n
// ==========================================================================
void DB(int type) {
    int i,j,v,a[N_MAX],REP;

    for (i=1; i<=n; i++) a[i] = 1;  // Initial string

    for (j=1; j<=pow(2,n); j++) {
        printf("%d", a[1]);

        v = (a[1] + a[2] + a[n]) % 2;
        REP = 0;
        // Membership testing of a[1..n]
        if (type == 1 && SameRep(a)) REP = 1;
        if (type == 2 && LCrep(a)) REP = 1;
        if (type == 3 && RLrep(a)) REP = 1;


        // Membership testing of conjugate of a[1..n]
        a[1] = 1 - a[1];
        if (type == 1 && SameRep(a)) REP = 1;
        if (type == 2 && LCrep(a)) REP = 1;
        if (type == 3 && RLrep(a)) REP = 1;

        // Shift String and add next bit
        for (i=1; i<n; i++) a[i] = a[i+1];
        if (REP) a[n] = 1 - v;
        else a[n] = v;
    }
}
//-----------------------------------------------------
int main() {
    int type;

    printf("Enter (1) Prefer-same (2) LC (3) RL: ");  scanf("%d", &type);
    printf("Enter n: ");   scanf("%d", &n);

    DB(type);
}
```

B Implementation of the de Bruijn successors $RL2(\omega)$, $LC2(\omega)$, and $O(\omega)$

```
#include<stdio.h>
#include<math.h>
#define N_MAX 50
int n;

// ============================================================================
// Compute the RLE of a[s..m] in run[1..r], returning r = run length
// ============================================================================
int RLE(int a[], int run[], int s, int m) {
    int i,j,r,old;

    old = a[m+1];
    a[m+1] = 1 - a[m];
    r = j = 0;
    for (i=s; i<=m; i++) {
        if (a[i] == a[i+1]) j++;
        else {  run[++r] = j+1;  j = 0;  }
    }
    a[m+1] = old;
    return r;
}
// ==============================================================================
// Check if a[1..n] is a "special" RL representative: the RLE of a[1..n] is of
// the form 1 x^j y where y > x and j is odd. Eg. 12224, 1111113 (PCR-related)
// ==============================================================================
int Special(int a[]) {
    int i,r,rle[N_MAX];

    r = RLE(a,rle,1,n);
    if (r%2 == 0) return 0;
    for (i=3; i<r; i++) if (rle[i] != rle[2]) return 0;
    if  (a[1] == 1 && a[2] == 0 && i == r && rle[r] > rle[2]) return 1;
    return 0;
}
// ============================================================================
// Apply PRR^{t} to a[1..n] to get b[1..n], where t is the length of the
// prefix in a[1..n] before the first 01 or 10 in a[2..n]
// ============================================================================
int Shift(int a[], int b[]) {
    int i,t=1;

    while (a[t+1] == a[t+2] && t < n-1) t++;
    for (i=1; i<=n; i++) b[i] = a[i];
    for (i=1; i<=n; i++) b[i+n] = (b[i] + b[i+1] + b[n+i-1]) % 2;
    for (i=1; i<=n; i++) b[i] = b[i+t];
    return t;
}
// ============================================================================
// Test if b[1..len] is the lex smallest rep (under rotation), if so, return the
// period p; otherwise return 0. Eg. (114114, p=3)(11244, p=5)(124114, p=0).
// ============================================================================
int IsSmallest(int b[], int len) {
    int i, p=1;
    for (i=2; i<=len; i++) {
        if (b[i-p] > b[i]) return 0;
        if (b[i-p] < b[i]) p = i;
    }
    if (len % p != 0) return 0;
    return p;
}
// ============================================================================
// Membership testers with special case for 111111...1  (run length for a[2..n])
// ============================================================================
int RL2rep(int a[]) {
    int p,r,rle[N_MAX];

    r = RLE(a,rle,2,n);
    if (r == 1) return 1;        // Special case: a[1..n] = 000..0 or 111..1
    if (a[1] == a[2]) return 0;
    p = IsSmallest(rle,r);

    if (a[1] == a[n] && p > 0 && (p == r || a[1] == 0 || p%2 == 0)) return 1;  //PCR-related
    if (a[1] != a[n] && p > 0 && a[1] == 0) return 1;  // CCR-related
    return 0;
}
```

```c
// ===========================================================================
int LC2rep(int a[]) {
    int t,b[N_MAX];

    if (a[1] == a[2]) return 0;
    t = Shift(a,b);
    return RL2rep(b);
}
// ===========================================================================
int OppRep(int a[]) {
    int b[N_MAX];

    Shift(a,b);
    if (Special(a) || (LC2rep(a) && !Special(b))) return 1;
    return 0;
}
// ===========================================================================
// Repeatedly apply the Prefer Opp or LC or RL successor rule starting with 1^n
// ===========================================================================
void DB(int type) {
    int i,j,v,a[N_MAX],REP;

    // Initial string
    for (i=1; i<=n; i+=2) a[i] = 0;
    for (i=2; i<=n; i+=2) a[i] = 1;

    for (j=1; j<=pow(2,n); j++) {
        printf("%d", a[1]);

        v = (a[1] + a[2] + a[n]) % 2;
        REP = 0;
        // Membership testing of a[1..n]
        if (type == 1 && OppRep(a)) REP = 1;
        if (type == 2 && LC2rep(a)) REP = 1;
        if (type == 3 && RL2rep(a)) REP = 1;

        // Membership testing of conjugate of a[1..n]
        a[1] = 1 - a[1];
        if (type == 1 && OppRep(a)) REP = 1;
        if (type == 2 && LC2rep(a)) REP = 1;
        if (type == 3 && RL2rep(a)) REP = 1;

        // Shift String and add next bit
        for (i=1; i<n; i++) a[i] = a[i+1];
        if (REP) a[n] = 1 - v;
        else a[n] = v;
    }
}
//------------------------------------------------------
int main() {
    int type;

    printf("Enter (1) Prefer-opposite (2) LC2 (3) RL2: ");  scanf("%d", &type);
    printf("Enter n: ");   scanf("%d", &n);

    DB(type);
}
```