# Fixed-Density Necklaces and Lyndon Words in Cool-lex Order

Joe Sawada
University of Guelph
**80** PUBLICATIONS **826** CITATIONS
SEE PROFILE

Aaron Williams
Williams College
**68** PUBLICATIONS **511** CITATIONS
SEE PROFILE

# Fixed-Density Necklaces and Lyndon Words in Cool-lex Order

J. Sawada[*] A. Williams[†]

July 6, 2009

**Abstract**

This paper creates a simple Gray code for fixed-density binary necklaces and Lyndon words. The Gray code is simultaneously a left-shift Gray code and a 2-transposition Gray code and is cyclic with these properties. It also has the advantage of using canonical representations (lexicographically minimum rotations) yielding the first known Gray code for Lyndon words ordered by density. This work solidifies the framework that produces loopless Gray code algorithms for generating combinations [15, 16], balanced parentheses and binary trees [17], and multiset permutations [23] in cool-lex order. A primary application of our new fixed-density Gray code is in the first explicit construction of fixed-weight de Bruijn cycles [24]. In particular, this paper presents a simple $O(n)$-time implementation that allows each bit of the de Bruijn cycles to be generated in amortized $O(1)$-time using $O(n)$ memory.

**Keywords:** Lyndon word, necklace, Gray code, cool-lex, fixed-density, de Bruijn cycle

## 1 Introduction

Combinatorial generation is the study of efficient algorithms for exhaustively generating every instance of a specific combinatorial object. The research area is fundamental to computer science, as evidenced by Knuth's devotion of over 400 pages to the subject in the upcoming volume of *The Art of Computer Programming* [9, 10, 11]. As evidenced in this paper, results in combinatorial generation often reveal (and rely upon) interesting mathematical properties of the combinatorial object itself. One of the most important aspects of combinatorial generation is to find orderings of the objects so that only a constant amount of change is required to go from one instance to the next. Such orderings are called *Gray codes*.

Fast and simple algorithms for generating necklaces and Lyndon words have been known for some time [4, 5, 13]. However, an open problem for many years was whether or not there existed a Gray code to list such objects. For fixed-density binary necklaces — those where the number of 1s is fixed — Gray codes were discovered separately by Wang and Savage [21] and Ueda [18]. Both algorithms were constructed by finding a Hamilton path in a graph whose vertices correspond to the necklaces of length $n$ and density $d$. Their algorithms did not use lexicographically minimal representations and thus did not apply to Lyndon words, and unfortunately the lists produced by these algorithms could not be concatenated together to find a Gray code for unrestricted binary

[*]Computing and Information Science, University of Guelph, Canada. Research supported by NSERC. `E-mail: jsawada@uoguelph.ca`

[†]Computer Science, University of Victoria, Canada. `haron@uvic.ca`

necklaces. Additionally, these algorithms did not obtain the optimal $O(1)$ amortized time implementations; however such an algorithm does exist for lexicographic order [14]. In 2006 this open problem was finally answered by Vajnovszki [19] for a binary alphabet and then generalized for alphabets of arbitrary size by Vajnovszki and Weston [22]. However, there remains three interesting open problems: (1) to find a Gray code for fixed-density Lyndon words, (2) to find a Gray code for fixed-density necklace using the lexicographically minimal rotation as the representative, and (3) to find a simple Gray code for unrestricted necklaces and Lyndon words that is ordered by density.

This paper answers these three problems simultaneously by using a subtle variation of lexicographic order known as *cool-lex order*. This order has previously been used to create Gray codes for combinations [15, 16] (fixed-density binary strings) and balanced parentheses [17] (fixed-density binary strings in which each prefix contains at least as many 1s as 0s). The focus of these previous papers was efficient generation algorithms, with Knuth describing the MMIX implementation for combinations as "incredibly efficient" [10] and Arndt rating the balanced parentheses algorithm as the fastest known [1]. In this paper we provide a simple $O(n)$-time/$O(1)$-additional variable implementation for fixed-density binary necklaces and Lyndon words. This result has added significance with respect to [24]. The existence of an amortized $O(1)$-time implementation is also conjectured.

Collectively, the aforementioned Gray codes for combinations, balanced parentheses, and fixed-density necklaces and Lyndon words are the beginnings of a *cool-lex hierarchy*, whereby Gray codes for a number of combinatorial objects can be obtained by using the same relative order, and efficient algorithms can be obtained by modifying the same basic program. A similar combinatorial generation hierarchy exists for the well-known binary reflected Gray code [7]. Within the binary reflected Gray code Degni and Drisko [2] conjecture a Gray code for binary necklaces and Vajnovszki proves a similar result without an algorithm [20].

The Gray codes developed in this paper are traditional 2-transposition Gray codes, but in addition, they are also *left-shift Gray codes* which means that successive strings differ by shifting a single bit to the left. We use an arrow to denote this operation on a word as follows: $ab\underleftarrow{cdef}gh = abfcdegh$. Our algorithms produce the following left-shift Gray codes for necklaces (left) and Lyndon words (right) where $n = 6$ and $d = 3$:

$$0\underleftarrow{0101}1, \underleftarrow{010}101, \underleftarrow{00110}1, 00\underleftarrow{0111} \qquad\qquad 00\underleftarrow{101}1, \underleftarrow{00110}1, 00\underleftarrow{0111} \tag{1}$$

Furthermore, the last string can always be transformed into the first string by using a left-shift or a single transposition, so the Gray codes are also *cyclic* (also known as *circular*). This property leads to a Gray code for unrestricted binary necklaces and Lyndon words ordered by density where a Hamming distance of one separates successive strings with differing density.

As an interesting application, the new fixed-density necklace Gray code can be applied in the first explicit and practical construction of fixed-density de Bruijn cycles (also known as fixed-weight de Bruijn cycles). This application is briefly illustrated below; details can be found in [24]. Consider the following string

$$00011100110101001011. \tag{2}$$

Its successive substrings of length five are given below and include those that wrap-around

$$00011, 00111, 01110, 11100, 11001, \ldots, 01011, 10110, 01100, 11000, 10001. \tag{3}$$

Interestingly, the $\binom{6}{3} = 20$ strings in (3) are all distinct. Furthermore, they represent all fixed-density binary strings with $n = 6$ and $d = 3$ except that the last (redundant) bit is omitted. This *fixed-density de Bruijn cycle* in (2) was obtained from (1) by concatenating the necklaces in reverse order and reducing the periodic string 010101 to its prime representative 01.

The rest of the paper is outlined as follows. To begin, we provide a background on necklaces and Lyndon words including new structural results in Section 2 that are necessary later on in the paper. In Section 3 we define cool-lex order as a variation of lexicographic order using suffixes of the form $01^i$. The ordering is then applied to fixed-density necklaces and Lyndon words in Section 4 where we prove that the resulting orderings are Gray codes. In Section 5, we provide an iterative approach for listing Gray codes of fixed-density necklaces and Lyndon words in cool-lex order. It is the first known fixed-density algorithm for Lyndon words. An optimization shows that the strings can be generated in $O(n)$ amortized time. We then show how we can easily append fixed-density lists together to obtain a Gray code for unrestricted binary necklaces and Lyndon words. We conclude in Section 6 with a number of open problems.

## 2 Necklaces and Lyndon words

Where possible, we generalize definitions and results for $k$-ary strings. However the main algorithmic results in Sections 4 and 5 apply only to binary strings. At the end of this section it is shown that the structural result given in Corollary 2 for fixed-density necklaces and Lyndon words does not hold for fixed-density bracelets.

A $k$-ary string $\boldsymbol{s} = s_1s_2\cdots s_m$ is said to be *lexicographically smaller* than $\boldsymbol{t} = t_1t_2\cdots t_n$, written $\boldsymbol{s} < \boldsymbol{t}$, if one of the following holds:

(1) $m < n$ and $s_1s_2\cdots s_m = t_1t_2\cdots t_m$ or
(2) there exists $1 \le i < m$ such that $s_1s_2\cdots s_i = t_1t_2\cdots t_i$ and $s_{i+1} < t_{i+1}$.

A *necklace* is defined to be the lexicographically smallest string in an equivalence class of strings under rotation. A *Lyndon word* is an aperiodic necklace. For any binary string $\boldsymbol{z}$ we say that the *density* of the string is the number of 1s it contains and we denote this number by $den(\boldsymbol{z})$. In this paper we are concerned with binary necklaces and Lyndon words of *fixed-density*. We will use the following notation to denote these objects:

- $\mathbf{N}(n, d)$: the set of binary necklaces of length $n$ and density $d$,
- $\mathbf{L}(n, d)$: the set of binary Lyndon words of length $n$ and density $d$.

As an example, (1) contains $\mathbf{N}(6, 3)$ (on the left) and $\mathbf{L}(6, 3)$ (on the right).

For several properties it will be useful to have the following notation for suffixes of these objects:

- $\mathbf{N_s}(n, d)$: the set of suffixes of strings in $\mathbf{N}(n, d)$,
- $\mathbf{L_s}(n, d)$: the set of all suffixes of strings in $\mathbf{L}(n, d)$.

The following characterization of necklaces and Lyndon words follows immediately from their definitions [12]:

OBSERVATION 1. *The following conditions characterize necklaces and Lyndon words:*

*(i)* $\alpha = \boldsymbol{pz}$ *is a necklace if and only if* $\boldsymbol{pz} \le \boldsymbol{zp}$ *for all* $\boldsymbol{p}$, $\boldsymbol{z}$
*(ii)* $\alpha = \boldsymbol{pz}$ *is a Lyndon word if and only if* $\boldsymbol{pz} < \boldsymbol{zp}$ *for all non-empty* $\boldsymbol{p}$, $\boldsymbol{z}$.

The following lemma that follows from this observation is proved in [8] :

LEMMA 1. *A word* $\alpha = \boldsymbol{pz}$ *is a Lyndon word if and only if it is strictly smaller than all of its proper right factors: for all non-empty* $\boldsymbol{p}, \boldsymbol{z}$ *we have* $\boldsymbol{pz} < \boldsymbol{z}$.

The proofs of the following results are presented in Appendix A. The primary structural result is given in Theorem 1. The corollaries that follow from this result are used to prove the correctness of our algorithms in the next sections.

LEMMA 2. *If $\alpha = \boldsymbol{pzp}$ is a necklace where $\boldsymbol{p}$ is non-empty then $\alpha = \boldsymbol{y}^t$ for some Lyndon word $\boldsymbol{y}$ where $t > 1$. Moreover $\boldsymbol{pzp} = \boldsymbol{ppz}$ and $\boldsymbol{p} = \boldsymbol{y}^s$ for some integer $s \geq 1$.*

LEMMA 3. *If both $\boldsymbol{pz}$ and $\boldsymbol{p}$ are Lyndon words then $\mathbf{p}^t\boldsymbol{z}$ is a Lyndon word where $t > 1$.*

THEOREM 1. *Let $\boldsymbol{pz}$ be a string and let $\boldsymbol{q}$ be a necklace such that $\boldsymbol{q}$ is not a prefix of $\boldsymbol{p}$ and $\boldsymbol{q} < \boldsymbol{p}$. If $\boldsymbol{pz} \in \mathbf{N}$ then $\boldsymbol{qz} \in \mathbf{L}$,*

COROLLARY 1. *Let $\boldsymbol{z}$ be a binary string where $j = d - den(\boldsymbol{z})$ and $i = n - |\boldsymbol{z}| - j$ for given integers $n$ and $d$.*

- *If $\boldsymbol{z} \in \mathbf{N}_s(n, d)$ , then $0^i1^j\boldsymbol{z} \in \mathbf{N}(n, d)$.*
- *If $\boldsymbol{z} \in \mathbf{L}_s(n, d)$ then $0^i1^j\boldsymbol{z} \in \mathbf{L}(n, d)$.*

COROLLARY 2. *Let $\boldsymbol{z}$ be a binary string that is empty or begins with 0 and let $0 \leq k < d - den(\boldsymbol{z})$ for a given integer $d$.*

- *If $01^k\boldsymbol{z} \in \mathbf{N}_s(n, d)$, then $01^{k+1}\boldsymbol{z} \in \mathbf{N}_s(n, d)$.*
- *If $01^k\boldsymbol{z} \in \mathbf{L}_s(n, d)$, then $01^{k+1}\boldsymbol{z} \in \mathbf{L}_s(n, d)$.*

A *bracelet* is the lexicographically smallest string in an equivalence class of strings under rotation and reversals. For example, the set of fixed-density binary bracelets of length 6 and density 3 are given below

$$\{000111, 001011, 010101\}.$$

Notice that $01\boldsymbol{z}$ is a suffix of 010101 for $\boldsymbol{z} = 01$ and $den(01\boldsymbol{z}) < 3$. However, $011\boldsymbol{z}$ is not a suffix of any of the fixed-density bracelets given above. (In particular, 001101 is not a bracelet because 001011 is a lexicographically smaller string that can be obtained by reversal and rotations.) Therefore, the structural result given in Corollary 2 does not hold for fixed-density bracelets.

## 3 Orderings: lex vs cool-lex

In this section we discuss lexicographic order and cool-lex order respectively for fixed-density binary strings. Table 1 provides a collective example/comparison of each ordering.

### 3.1 Lexicographic order

This subsection discusses several variations of lexicographic order for fixed-density binary strings. In lexicographic order binary strings are recursively ordered from left to right with 0 coming before 1. Hence, the following formula can be used to recursively generate all fixed-density binary strings of length $n$ and density $d$ in lexicographic order:

$$\mathrm{L}(n, d) = 0\,\mathrm{L}(n-1, d),\ 1\,\mathrm{L}(n-1, d-1). \tag{4}$$

| $\mathrm{R}(n,d)$ | $\mathrm{R}(n,d)$ | $\mathrm{C}(n,d)$ | $\mathrm{C}(n,d)$ |
|---|---|---|---|
| 00011**1** | 00**0111** | 100**011** | $\underleftarrow{100}011$ |
| 00101**1** | 001**011** | 010**011** | $\underleftarrow{0100}11$ |
| 01001**1** | 010**011** | 001**011** | $\underleftarrow{00101}1$ |
| 10001**1** | 100**011** | 1001**01** | $\underleftarrow{100}101$ |
| 00110**1** | 0011**01** | 0101**01** | $\underleftarrow{0101}01$ |
| 01010**1** | 0101**01** | 1010**01** | $\underleftarrow{101}001$ |
| 10010**1** | 1001**01** | 1100**01** | $\underleftarrow{1100}01$ |
| 01100**1** | 0110**01** | 0110**01** | $\underleftarrow{01100}1$ |
| 10100**1** | 1010**01** | 0011**01** | $\underleftarrow{001101}$ |
| 11000**1** | 1100**01** | 10011**0** | $\underleftarrow{100}110$ |
| 00111**0** | 00111**0** | 01011**0** | $\underleftarrow{0101}10$ |
| 01011**0** | 01011**0** | 10101**0** | $\underleftarrow{101}010$ |
| 10011**0** | 10011**0** | 11001**0** | $\underleftarrow{1100}10$ |
| 01101**0** | 01101**0** | 01101**0** | $\underleftarrow{01101}0$ |
| 10101**0** | 10101**0** | 10110**0** | $\underleftarrow{101}100$ |
| 11001**0** | 11001**0** | 11010**0** | $\underleftarrow{1101}00$ |
| 01110**0** | 01110**0** | 11100**0** | $\underleftarrow{11100}0$ |
| 10110**0** | 10110**0** | 01110**0** | $\underleftarrow{011100}$ |
| 11010**0** | 11010**0** | 00111**0** | $\underleftarrow{001110}$ |
| 11100**0** | 11100**0** | 00**0111** | $\underleftarrow{000111}$ |

Table 1: Comparing reverse co-lex and cool-lex for binary strings of length 6 and density 3. The first column emphasizes the recursive construction of reverse co-lex by rightmost symbol, and the second column emphasizes the recursive construction of reverse co-lex by $01^i$ suffix. The third column emphasizes the recursive construction of cool-lex, and the fourth column emphasizes the iterative left-shift Gray code given by cool-lex.

There are four trivial variations of (4). These variations involve (independently) interchanging 0 and 1, and using 0 and 1 as suffixes instead of prefixes. The resulting orders are named *lexicographic* (prefix 0,1), *reverse lexicographic* (prefix 1,0), *co-lexicographic* (suffix 0,1), and *reverse co-lexicographic* (suffix 1,0). For brevity, *lexicographic* is shortened to *lex*. Reverse co-lex order of fixed-density binary strings is of particular interest in this paper and can be generated by the following:

$$\mathrm{R}(n,d) = \mathrm{R}(n-1,d)\ 1,\ \mathrm{R}(n-1,d-1)\ 0. \tag{5}$$

Instead of defining reverse co-lex order by its rightmost character (as above) one can equivalently use its rightmost 0. In other words, one can recursively partition the binary strings based on their suffix of the form $01^i$, and then recursively apply the same procedure with this suffix removed. In particular, reverse co-lex is obtained by ordering the $01^i$ suffixes by $i = d, d-1, \ldots, 0$. That is, (5) can be reexpressed as follows:

$$\mathrm{R}(n,d) = \mathrm{R}(n-d-1,0)01^d,\ \mathrm{R}(n-d,1)01^{d-1},\ \ldots,\ \mathrm{R}(n-2,d-1)01,\ \mathrm{R}(n-1,d)0. \tag{6}$$

Notice that (6) is the result of recursively expanding the first term of (5). The advantage of (6) is that it allow more flexibility in creating non-trivial variations of the original order.

Before concluding this subsection it is mentioned that any subset of fixed-density binary strings can be expressed in reverse co-lex order using (5) or (6), although some of the recursive sublists may be empty. Finally, the first term in (6) only contains the string $0^{n-d}1^d$. Therefore, (6) can be reexpressed as follows:

$$\mathrm{R}(n,d) = 0^{n-d}1^d,\ \mathrm{R}(n-d,1)01^{d-1},\ \ldots,\ \mathrm{R}(n-2,d-1)01,\ \mathrm{R}(n-1,d)0. \tag{7}$$

Any subset of fixed-density binary strings can be expressed in reverse co-lex order using (7), so long as the explicitly included string $0^{n-d}1^d$ is (recursively) included the subset.

## 3.2 Cool-lex order

This subsection discusses cool-lex order for fixed-density binary strings. Consider the following operation on a binary string.

$$\boxed{\text{Shift the bit following the leftmost } 10 \text{ (or the last bit if there is no such bit) into the first position.}}$$

Repeated application of this simple operation causes the bits of any binary string to be rearranged in all possible ways. That is, the above rule cyclicly generates all $\binom{n}{d}$ fixed-density binary strings of length $n$ and density $d$. The resulting order of fixed-density binary strings is known as cool-lex order due to its recursive similarity to lexicographic order. In particular, cool-lex order for fixed-density binary strings can be obtained by the following recursive expression:

$$\mathbf{C}(n,d) = \mathbf{C}(n-d,1)01^{d-1},\ \ldots,\ \mathbf{C}(n-2,d-1)01,\ \mathbf{C}(n-1,d)0,\ 0^{n-d}1^{d}. \tag{8}$$

Notice that (8) is identical to (7), except that $0^{n-d}1^{d}$ is last instead of first. Since $0^{n-d}1^{d}$ is recursively "misplaced" the resulting order is a non-trivial modification of reverse co-lex order. As mentioned, the advantage of cool-lex order is that fixed-density binary strings appear in Gray code order as illustrated in Table 1.

Any subset of fixed-density binary strings can be expressed in cool-lex order by using (8) although some of the recursive sublists may be empty, and one must ensure that $0^{n-d}1^{d}$ is (recursively) included in the subset. These two issues arise in next section when discussing fixed-density necklaces and Lyndon words.

# 4 Gray code

This section discusses a recursive definition of the Gray code for fixed-density necklaces and Lyndon words. The approach is to use cool-lex order with $01^{i}$ suffixes and decreasing $i$. This differs from the standard approach of maintaining the length of the longest Lyndon *prefix* when generating necklaces. **For our discussion we will focus only on necklaces, however the results apply to Lyndon words by simply interchanging these terms and $\mathbf{N}(n,d)$ with $\mathbf{L}(n,d)$ and so forth**.

Observe that except for the trivial cases when $d = 0$ or $d = n$, all binary necklaces start with 0 and end with 1. Thus, unlike the recursive expressions for fixed-density binary strings, we must limit the suffixes of the form $01^{i}$ to have non-empty sublists. For example, no fixed-density necklaces of length 8 and density 6 end with 0 or 01. Additionally, once we have determined a valid suffix, then the next set of valid $01^{i}$ substrings depends upon the suffix already constructed. Thus, recursive formulae must include the current suffix $\boldsymbol{z}$ as a parameter. These two observations lead to the following recursive formulae for generating fixed-density necklaces with suffix $\boldsymbol{z}$. We begin by presenting a reverse *co-lex* recurrence before giving our cool-lex result.

$$\mathbf{R}_{n,d}(\boldsymbol{z}) = 0^{m}1^{i}\boldsymbol{z}, \mathbf{R}_{n,d}(01^{i-1}\boldsymbol{z}), \mathbf{R}_{n,d}(01^{i-2}\boldsymbol{z}), \ldots, \mathbf{R}_{n,d}(01^{j}\boldsymbol{z}) \tag{9}$$

where $i = d - den(\boldsymbol{z})$, $j$ is the smallest integer such that $01^{j}\boldsymbol{z} \in \mathbf{N}_s(n,d)$, and $m = n - i - |\boldsymbol{z}|$. Observe that the entire list of fixed-density necklaces can be obtained by setting $\boldsymbol{z} = \epsilon$ (the empty string).

Within (9) notice that the explicitly included string $0^{m}1^{i}\boldsymbol{z}$ is guaranteed to be in $\mathbf{N}(n,d)$ by Corollary 1. From the definition of $j$, the listing $\mathbf{R}_{n,d}(\boldsymbol{z})$ will contain every string in $\mathbf{N}_s(n,d)$ with suffix $\boldsymbol{z}$ exactly once. Furthermore, by applying Corollary 2, each sublist given in (9) will be non-empty. These observations are summarized in the following lemma.

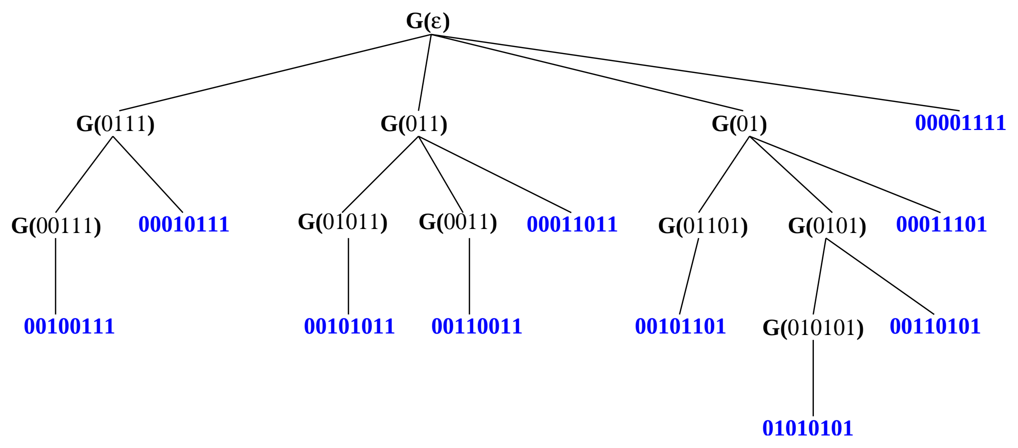

Figure 1: Computation tree for fixed-density necklaces of length $8$ and density $4$ where $\mathbf{G}$ represents $\mathbf{G}_{8,4}$.

LEMMA 4. *The listing $\mathbf{R}_{n,d}(\boldsymbol{z})$ contains every string in $\mathbf{N}_s(n,d)$ with suffix $\boldsymbol{z}$ exactly once, and no other string. Furthermore, each sublist in its recursive definition is non-empty.*

Clearly any reordering of the terms in (9) will also have the properties as stated in the previous lemma. In particular, the cool-lex reordering is generated by the following:

$$\mathbf{G}_{n,d}(\boldsymbol{z}) = \mathbf{G}_{n,d}(01^{i-1}\boldsymbol{z}), \mathbf{G}_{n,d}(01^{i-2}\boldsymbol{z}), \ldots, \mathbf{G}_{n,d}(01^{j}\boldsymbol{z}), 0^m1^i\boldsymbol{z} \tag{10}$$

where $i$, $j$, and $m$ are defined as in (9).

An example computation tree for $\mathbf{G}_{8,4}(\epsilon)$ is given in Figure 1. Observe that the necklaces (at the leaves) are in Gray code order, where each successive string differs from the previous by a left-shift. Formally, a *left shift* of the bit in position $j$ to position $i$ (where $i < j$) in the string $\boldsymbol{s} = s_1 s_2 \cdots s_n$ is defined as:

$$\overleftarrow{\mathsf{shift}}(\boldsymbol{s},\ j,\ i) = s_1 s_2 \cdots s_{i-1} s_j s_i s_{i+1} \cdots s_{j-1} s_{j+1} s_{j+2} \cdots s_n.$$

It turns out that the left-shifts given in Figure 1 can be simulated by at most two transpositions. Moreover, the same is true within all $\mathbf{G}_{n,d}(\boldsymbol{z})$, and so the listings are also 4-close Gray codes in the Hamming sense. This is stated formally in the upcoming Theorem 2. The proof given in Appendix A depends on a clear description of the first and last strings of the recurrence $\mathbf{G}_{n,d}(\boldsymbol{z})$ as described in the following lemma:

LEMMA 5. *The first and last strings in a non-empty $\mathbf{G}_{n,d}(\boldsymbol{z})$ where $i = d - den(\boldsymbol{z})$ and $m = n - i - |\boldsymbol{z}|$ are respectively*

$$\mathsf{first}(\mathbf{G}_{n,d}(\boldsymbol{z})) = 0^{m-h}10^h1^{i-1}\boldsymbol{z} \quad \textit{and} \quad \mathsf{last}(\mathbf{G}_{n,d}(\boldsymbol{z})) = 0^m1^i\boldsymbol{z},$$

*where $h$ is maximum value such that $0^{m-h}10^h1^{i-1}\boldsymbol{z} \in \mathbf{N}(n,d)$.*

THEOREM 2. *The cool-lex ordering $\mathbf{G}_{n,d}(\boldsymbol{z})$ is a left-shift, two-transposition cyclic Gray code for fixed-density necklaces with length $n$, density $d$, and suffix $\boldsymbol{z}$.*

Observe from the proof of Theorem 2 that we can determine exactly which bit to left shift to go from one string to the next in the cyclic Gray code. To perform the shift, we first need to know the length $k$ of the longest prefix of the form $0^*1^*$. From the proof we will either be left shifting the bit in position $k+2$, $k+1$, or $k$. Formally, the cool-lex shift rule given a necklace $\boldsymbol{s} = s_1 s_2 \cdots s_n$ is as follows:

$$\overleftarrow{\mathsf{cool}}(\boldsymbol{s}) = \begin{cases} \overleftarrow{\mathsf{shift}}(\boldsymbol{s},\ k,\ 1) & \text{if } k = n \\ \overleftarrow{\mathsf{shift}}(\boldsymbol{s},\ k+1,\ 1) & \text{if } k < n \text{ and } s_{k+2} = 0 \text{ or } \boldsymbol{w} \notin \mathbf{N}(n,d) \\ \overleftarrow{\mathsf{shift}}(\boldsymbol{s},\ k+2,\ j) & \text{if } k < n \text{ and } s_{k+2} = 1 \text{ and } \boldsymbol{w} \in \mathbf{N}(n,d) \end{cases}$$

where $\boldsymbol{w} = \overleftarrow{\mathsf{shift}}(\boldsymbol{s},\ k+2,\ k+1)$ and $j$ is the smallest value such that the shift yields a necklace. Notice that $\boldsymbol{w}$ is well-defined since $k < n$ implies $k+2 \leq n$. Therefore, $\overleftarrow{\mathsf{cool}}(\boldsymbol{s})$ is also well-defined. Notice that the left shifts can be thought of as *maximum* left shifts in the sense that the bit being shifted gets moved as far left as possible so the resulting string is still a necklace.

Assuming that $0^m 1^i$ is the longest prefix of length $k$ of the form $0^*1^*$ the next cool-lex shift can be computed by applying at most two swaps to the specified bit positions as outlined below:

| | |
|---|---|
| $\mathsf{swap}(g+1, m+1)$ | if $k = n$ |
| $\mathsf{swap}(m+1, k+1)$ | if $k < n$ and $s_{k+2} = 0$ or $\boldsymbol{w} \notin \mathbf{N}(n,d)$ |
| $\mathsf{swap}(k+1, k+2), \mathsf{swap}(h+1, m+1)$ | if $k < n$ and $s_{k+2} = 1$ and $\boldsymbol{w} \in \mathbf{N}(n,d)$ |

where $g$ is the minimum value such that $0^g 1 0^{n-d-g} 1^{d-1} \in \mathbf{N}(n,d)$, and $h$ is the minimum value such that $0^h 1 0^{m-h} 1^i 0 s_{k+2} s_{k+3} \cdots s_n \in \mathbf{N}(n,d)$.

The following section uses these rules and provides pseudocode for an iterative algorithm to produce a cool-lex Gray code for fixed-density necklaces. It is optimized to run in $O(n)$ amortized time.

## 5 Algorithms and pseudocode

In the previous section we provided a recursive cool-lex formulation to list all fixed-density necklaces or Lyndon words in Gray code order. We also described how to go from one string by making either 1 or 2 swaps. By applying these rules the function Next() illustrated in Figure 2 provides an iterative algorithm to generate these objects in cool-lex order. The algorithm must be seeded with the first string in the listing for a given $n$ and $d$, and the function TestString($\alpha$) tests if the given string belongs to the class of objects in question: necklaces, Lyndon words, balanced parentheses (when $n$ is even and $d = n/2$), or combinations.

If we let $i = \lceil \frac{n-d}{2} \rceil$ and let $j = \lfloor \frac{n-d}{2} \rfloor$, then it is not hard to see from Lemma 5 that the initial necklace will be $0^i 1 0^j 1^{d-1}$. For Lyndon words the first string is the same except for the special case when $d = 2$ and $n$ is even. In this case the seed string is $0^{i+1} 1 0^{i-1} 1$. For balanced parentheses the initial string is $010^{n-d-1} 1^{d-1}$ and for combinations the the initial string is $10^{n-d} 1^{d-1}$. For each object, the final string is always $0^{n-d} 1^d$, and this is the termination condition.

Notice that at the start of the function, we perform an $O(n)$ time search to determine the initial block of 0s and 1s in the string: $0^s 1^t$. Other than this, the running time of the algorithm depends on two factors: (1) the number of iterations of the for loop and (2) the running time of the function TestString($\alpha$). For combinations, this function always returns TRUE (in constant time), so the loop iterates only once. Since it is straightforward to efficiently maintain $s$ and $t$ parameters, as noted in [16], this algorithm can be made loop-free for combinations which means each new string can be generated in constant time. For balanced parentheses strings, the test can also be performed in constant time as outlined in [17] (in that paper the roles of 0 and 1 are reversed). For necklaces and Lyndon words, the function TestString($\alpha$) requires $O(n)$ time and without any extra optimization would have to be called $O(n)$ times in the worst case. Thus naïvely, the overall running time would be $O(n^2)$ per necklace

```
function Next() returns boolean
int $i, s, t$
    // Current string starts $0^s1^t$
    $s := t := 0$
    while $a_{s+1} = 0$ do $s := s + 1$
    while $a_{s+t+1} = 1$ do $t := t + 1$
    // Reached final string (termination)
    if $s + t = n$ then return FALSE
    // Move the 1 in position s+t+2 forward as much as possible
    if $a_{s+t+2} = 1$ then
        for $i := 0$ to $s$ do
            Swap($a_{i+1}, a_{s+1}$)
            Swap($a_{s+t+1}, a_{s+t+2}$)
            if TestString($\alpha$) then return TRUE
            Swap($a_{s+t+1}, a_{s+t+2}$)
            Swap($a_{i+1}, a_{s+1}$)
    // Shift the 0 in position s+t+1 to the front of the string
    Swap($a_{s+1}, a_{s+t+1}$)
    return TRUE
end.
```

Figure 2: Algorithm Next(), to list in cool-lex order the fixed-density binary strings of length $n > 1$ and density $0 < d < n$ tested by the function TestString - necklaces, Lyndon words, combinations.

generated. In the next subsection we describe an optimization to improve the algorithm to $O(n)$ amortized time.

## 5.1 Optimization

The purpose of the **for** loop in the function Next() is to determine the left most position that we can move the 1 from position $s + t + 2$ and still have a valid string, whether it is a necklace or a Lyndon word. By making observations about the valid positions that the 1 can be moved allows us to reduce the number of iterations required by the loop.

For any necklace, the number of consecutive 0's at the start of the string must be a largest consecutive block of 0's to appear anywhere in the string. Thus, since we are attempting to insert a 1 into the first block of $s$ 0's, the minimum position it can be placed is after position $\lfloor \frac{s}{2} \rfloor$. Next, by performing a simple linear scan, we determine the largest consecutive block of 0's other than the first block. By taking the maximum of these two values and adding one, we obtain the first possible position we can place a 1. If we let this position be denoted by $min$, then we can apply this optimization by inserting the following code fragment just before the **for** loop of Next() and modifying the loop counter $i$ to start at $min$ instead of 2.

```
$min := \lfloor \frac{s}{2} + 1 \rfloor$
for $i := s + t + 2$ to $n$ do
    if $a_i = 1$ then $count := 0$
    else $count := count + 1$
    if $count > min$ then $min := count$
```

Note that $a_{s+t+2} = 1$ when we require this fragment, so the counter $count$ for the number of consecutive 0's gets initialized to 0 on the first iteration.

Observe that by applying this optimization that the first iteration of the **for** loop still may not produce a new necklace. For example, consider $\alpha = 000011011001$ where $s = 4$ and $t = 2$. In this case $min = 3$ yielding 001001101001 which is not a necklace. In fact a second iteration of the **for** loop may not even produce a new necklace. This could happen when $a_{s+t+3} = 0$, allowing the second block of zeros (of length 1) to be joined with the third block. For example, consider $\alpha = 000011010001$ where $s = 4$ and $t = 2$ . In this case $min = 4$ and $min + 1 = 5$, but by moving the 1 to position 5 we obtain 000011100001 which is not a necklace. Fortunately, a third iteration of the loop will ensure that the largest consecutive block of 0's appears only at the beginning of the string which means that the new string must be a necklace. Thus, by applying this optimization the **for** loop will iterate at most 3 times which yields the following theorem.

THEOREM 3. *The optimized version of Next() produces a Gray code of all binary necklaces (Lyndon words) with length $n$ and density $d$ in $O(n)$ amortized time.*

A loop-free approach implementation of this Gray code would be a considerable challenge because of the difficulty to determine whether moving a 1 forward can lead to a new necklace. Even a constant amortized time implementation would be a non-trivial improvement and leads us to the following conjecture:

CONJECTURE 1. *There exists a $O(1)$ amortized algorithm to list binary fixed-density necklaces and Lyndon words in cool-lex order.*

### 5.2 Unrestricted necklaces and Lyndon words in Gray code order

To generate all unrestricted necklaces or Lyndon words in Gray code order by density, we can simply append the lists obtained from the cool-lex fixed-density algorithm. This is because the last string in the listing $\mathbf{G}_{n,d}(\epsilon)$ differs from the first string in $\mathbf{G}_{n,d+1}(\epsilon)$ by flipping a single bit. In particular, the last string in any listing $\mathbf{G}_{n,d}(\epsilon)$ is always $0^{n-d}1^d(\epsilon)$, while as mentioned earlier the first string in $\mathbf{G}_{n,d+1}(\epsilon)$ is of the form $0^i10^j1^d$.

## 6 Summary and open problems

We applied cool-lex order to fixed-density necklaces and Lyndon words and proved that the resulting orders are cyclic Gray codes (thus answering Question 6 in [23] in the binary case) and yield the first Gray code for fixed-density Lyndon words. We give an $O(n)$ amortized time implementation and the resulting lists can be used to provide new Gray codes for unrestricted necklaces and Lyndon words with the strings ordered by density.

Applying cool-lex order to fixed-density bracelets does not produce a circular left-shift Gray code. (In fact, the fixed-density bracelets of length 6 and density 3 do not have a circular left-shift Gray, as can be seen at the end of Section 2.) The reason for this appears to be their failure of Corollary 2. On the other hand, balanced parentheses and combinations do satisfy Corollary 1 and Corollary 2. It will be interesting to further examine the connection between these structural results and the result of obtaining Gray codes from cool-lex order.

- ▷ Can the cool-lex order of fixed-density necklaces and Lyndon words be generated in $O(1)$ amortized time?
- ▷ Can the cool-lex order of fixed-density necklaces and Lyndon words be efficiently ranked?
- ▷ Can a variation of cool-lex be used to find a Gray code for fixed-density bracelets (unlabeled necklaces)?
- ▷ Can the multiset version of cool-lex be used to find a Gray code for fixed-content necklaces? [23]

## References

[1] J. Arndt, Matters computation, 2008.

[2] C. Degni and A. Drisko, Gray-ordered binary necklaces, Electron. J. Combin., Vol. 14 No. 1 (2007) Research Paper 7, 23 pp. (electronic).

[3] J.P. Duval, Factorizing words over an ordered alphabet, J. Algorithms Vol. 4 No. 4 (1983) 363-381.

[4] H. Fredricksen and I. J. Kessler, An algorithm for generating necklaces of beads in two colors, Discrete Math., Vol. 61 No. 2-3 (1986) 181-188.

[5] H. Fredricksen and J. Maiorana, Necklaces of beads in $k$ colors and $k$-ary de Bruijn sequences, Discrete Math., Vol. 23 No. 3 (1978) 207-210.

[6] E. Gilbert and J. Riordan, Symmetry types of periodic sequences, Illinois J. Math., 5 (1961) 657-665.

[7] F. Gray, Pulse Code Communication, U.S. Patent 2,632,058, 1947.

[8] M. Lothaire, Combinatorics on Words, Addison-Wesley, 1983, reprinted Cambridge University Press, 1997.

[9] D. E. Knuth, The Art of Computer Programming, Volume 4: Generating All Tuples and Permutations, Fascicle 2, Addison-Wesley, February 2005, 150 pages.

[10] D. E. Knuth, The Art of Computer Programming, Volume 4: Generating all Combinations and Partitions, Fascicle 3, Addison-Wesley, July 2005, 150 pages.

[11] D. E. Knuth, The Art of Computer Programming, Volume 4: Generating All Trees; History of Combinationatorial Generation,, Fascicle 4, Addison-Wesley, February 2006, 150 pages.

[12] F. Ruskey, Combinatorial Generation, unpublished manuscript.

[13] F. Ruskey, C. Savage, and T. Wang, Generating Necklaces, Journal of Algorithms, 13 (1992) 414430.

[14] F. Ruskey and J. Sawada, An efficient algorithm for generating necklaces of fixed density, SIAM J. Comput., 29 (1999) 671684.

[15] F. Ruskey and A. Williams, *Generating Combinations By Prefix Shifts*, Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Kunming, China, August 16-29, 2005, Proceedings. Lecture Notes in Computer Science 3595 Springer (2005).

[16] F. Ruskey and A. Williams, The coolest way to generate combinations, Discrete Mathematics, Special Issue on Generalizing de Bruijn Cycles and Gray Codes (Edited by G. Hurlbert, B. Jackson, and B. Stevens). 15 pages. In Press.

[17] F. Ruskey and A. Williams, *Generating Balanced Parentheses and Binary Trees by Prefix Shifts*, Computing: The Australasian Theory Symposium, CATS 2008, New South Wales, Australia, January 22-25, 2008, Proceedings. Theory of Computing 77 Conferences in Research and Practice in Information Technology (2008).

[18] T. Ueda, Gray codes for necklaces, Discrete Math., Vol. 219 No. 1-3 (2000) 235-248.

[19] V. Vajnovszki, Gray code order for Lyndon words, Discrete Math. Theor. Comput. Sci., Vol. 9 No. 2 (2007) 145-151.

[20] V. Vajnovszki, More restrictive Gray codes for necklaces and Lyndon words, Inform. Process. Lett. Vol. 106 No. 3 (2008), 96-99.

[21] T. Wang and C. Savage, A Gray code for necklaces of fixed density, SIAM J. Discrete Math., Vol. 9 No. 4 (1996) 654-673.

[22] M. Weston and V. Vajnovszki, Gray codes for necklaces and Lyndon words of arbitrary base, Pure Math. Appl. (PU.M.A.), Vol. 17 No. 1-2 (2006),175-182.

[23] A. Williams, Loopless generation of multiset permutations by prefix shifts, In SODA 09: The Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, New York, New York, USA, 2009.

[24] F. Ruskey and A. Williams, Fixed-weight de Bruijn cycles (submitted), In SODA 10: The Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, Austin, Texas, USA, 2010.

## Appendix A: Proofs

PROOF OF LEMMA 2: Since $\boldsymbol{p} < \boldsymbol{pzp}$, $\alpha$ is not a Lyndon word; it must be of the form $\boldsymbol{y}^t$ for some Lyndon word $y$ where $t > 1$. This clearly implies that $\boldsymbol{p} = \boldsymbol{y}^s$ for some integer $s \geq 1$ and thus $\boldsymbol{pzp} = \boldsymbol{ppz}$. □

PROOF OF LEMMA 3: It suffices to show that $\boldsymbol{ppz}$ is a Lyndon word by applying recursion (replace $\boldsymbol{z}$ with $\boldsymbol{pz}$). Let $\boldsymbol{p} = \boldsymbol{p}_1\boldsymbol{p}_2$ and let $\boldsymbol{z} = \boldsymbol{z}_1\boldsymbol{z}_2$ for non-empty $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{z}_1, \boldsymbol{z}_2$. We must show that $\boldsymbol{ppz}$ is strictly smaller than each of its non-trivial rotations. We will use Corollary 1: the Lyndon words $\boldsymbol{p}$ and $\boldsymbol{pz}$ are strictly smaller than each of their proper right factors:

(a) $\boldsymbol{ppz} < \boldsymbol{p}_2\boldsymbol{pzp}_1$ since $\boldsymbol{p} < \boldsymbol{p}_2$

(b) $\boldsymbol{ppz} < \boldsymbol{pzp}$ since $\boldsymbol{pz} < \boldsymbol{z}$

(c) $\boldsymbol{ppz} < \boldsymbol{p}_2\boldsymbol{zpp}_1$ since $\boldsymbol{p} < \boldsymbol{p}_2$

(d) $\boldsymbol{ppz} < \boldsymbol{zpp}$ since $\boldsymbol{ppz} < \boldsymbol{pzp}$ from (b) and $\boldsymbol{pzp} < \boldsymbol{zpp}$ using $\boldsymbol{pz} < \boldsymbol{zp}$.

(e) $\boldsymbol{ppz} < \boldsymbol{z}_2\boldsymbol{ppz}_1$ since $\boldsymbol{ppz} < \boldsymbol{pzp}$ from (b) and $\boldsymbol{pz} < \boldsymbol{z}_2$.

□

PROOF OF THEOREM 1: First we show that the result is true if $\boldsymbol{q}$ is a Lyndon word. By Theorem 1 we must show that $\boldsymbol{qz}$ is lexicographically smaller than all of its non-trivial rotations. If we consider all possible $\boldsymbol{q} = \boldsymbol{q}_1\boldsymbol{q}_2$, and $\boldsymbol{z} = \boldsymbol{z}_1\boldsymbol{z}_2$ for non-empty $\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{z}_2$. then each non-trivial rotation will either be of the form (i) $\boldsymbol{q}_2\boldsymbol{zq}_1$ or (ii) $\boldsymbol{z}_2\boldsymbol{qz}_1$.

(i) If $\boldsymbol{q}$ is a Lyndon word (Corollary 1, $\boldsymbol{q} < \boldsymbol{q}_2$ and hence $\boldsymbol{qz} < \boldsymbol{q}_2\boldsymbol{zq}_1$.

(ii) For this case we must be careful about comparing strings of different length with respect to the lexicographic order since $\boldsymbol{q} < \boldsymbol{p}$ does not necessarily implies that $\boldsymbol{qy} < \boldsymbol{p}$ for any $\boldsymbol{y}$. In particular, we must be careful of the case where $\boldsymbol{z}_2$ is prefix of $\boldsymbol{p}$ or $\boldsymbol{q}$ and vise-versa.

Since $\boldsymbol{pz}$ is a necklace we have $\boldsymbol{pz} \leq \boldsymbol{z}_2\boldsymbol{pz}_1$. If $\boldsymbol{p} < \boldsymbol{z}_2$ then since $\boldsymbol{q} < \boldsymbol{p}$ and $\boldsymbol{q}$ is not a prefix of $\boldsymbol{p}$, it must be that $\boldsymbol{q} < \boldsymbol{z}_2$ where $\boldsymbol{q}$ is not a prefix of $\boldsymbol{z}_2$. Thus $\boldsymbol{qz} < \boldsymbol{z}_2\boldsymbol{qz}_1$. Otherwise it must be the case that $\boldsymbol{z}_2$ is a prefix of $\boldsymbol{p}$. By Lemma 2 this means that $\boldsymbol{z}_2$ is of the form $\boldsymbol{y}^j$ for some Lyndon word $\boldsymbol{y}$. Now, since $\boldsymbol{q}$ is a Lyndon word where $\boldsymbol{q} < \boldsymbol{p}$ and $\boldsymbol{q}$ is not a prefix of $\boldsymbol{p}$ it must be that $\boldsymbol{q} < \boldsymbol{y}$. Thus $\boldsymbol{y}$ (and hence $\boldsymbol{z}_2$) is not a prefix of $\boldsymbol{q}$. Thus $\boldsymbol{qz} < \boldsymbol{z}_2\boldsymbol{qz}_1$.

We have shown that $\boldsymbol{qz} \in \mathbf{L}$ if $\boldsymbol{q}$ is a Lyndon word. If it is not a Lyndon word, it must be periodic: $\boldsymbol{q} = \boldsymbol{y}^t$ for some Lyndon word $\boldsymbol{y}$. Clearly $\boldsymbol{y} < \boldsymbol{q}$ so we can apply the result just proved to show that $\boldsymbol{yz} \in \mathbf{L}$. Now applying Lemma 3 we get that $\boldsymbol{y}^t\boldsymbol{z} = \boldsymbol{qz} \in \mathbf{L}$.

□

PROOF OF COROLLARY 1: We prove only the first result for necklaces as the proof for Lyndon words is similar.

Suppose $\boldsymbol{z} \in \mathbf{N}_s(n, d)$, then there exists some $\boldsymbol{p}$ such that $\boldsymbol{pz} \in \mathbf{N}(n, d)$. Let $\boldsymbol{q} = 0^i1^j$. If $\boldsymbol{p} = \boldsymbol{q}$ then we are done. Otherwise it must be the case that $i, j > 0$ and $\boldsymbol{p}$ must be some binary string containing $i$ 0s and $j$ 1s.

Clearly $\boldsymbol{q} < \boldsymbol{p}$ and $\boldsymbol{q}$ is not a prefix of $\boldsymbol{p}$. Thus by Theorem 1 we must have that $\boldsymbol{qz} \in \mathbf{L}(n,d)$ and hence it is also in $\mathbf{N}(n,d)$. $\square$

PROOF OF COROLLARY 2: We prove only the first result for necklaces as the proof for Lyndon words is similar.

Let $j = d - den(\boldsymbol{z}) - k$ and let $i = n - |\boldsymbol{z}| - j - k - 1$. Suppose that $01^k\boldsymbol{z} \in \mathbf{N}_s(n,d)$ and let $\boldsymbol{p} = 0^i1^j01^k$ and $\boldsymbol{q} = 0^i1^{j-1}01^{k+1}$ . From Corollary 1 we know that $\boldsymbol{pz} \in \mathbf{N}(n,d)$. Since $\boldsymbol{z}$ is empty or begins with 0 then if $j > k$ the rotation $01^k\boldsymbol{z}01^j$ would be strictly smaller than $\boldsymbol{pz}$, contradicting the fact that it is a necklace. Thus $j \leq k$. Clearly this implies that $\boldsymbol{q}$ is a Lyndon word such that $\boldsymbol{q} < \boldsymbol{p}$ where $\boldsymbol{q}$ is not a prefix of $\boldsymbol{p}$. Thus, from Theorem 1 we have $\boldsymbol{qz} \in \mathbf{L}(n,d)$ which implies that $01^{k+1}\boldsymbol{z} \in \mathbf{N}_s(n,d)$. $\square$

PROOF OF LEMMA 5: The expression for $\mathsf{last}(\mathbf{G}_{n,d}(\boldsymbol{z}))$ follows immediately from (10). Let $j$ be the smallest integer such that $01^j\boldsymbol{z} \in \mathbf{N}_s(n,d)$. The expression for $\mathsf{first}(\mathbf{G}_{n,d}(\boldsymbol{z}))$ is proven in two cases depending on the value of $j$.

If $j = i$, then $\mathbf{G}_{n,d}(\boldsymbol{z}) = 0^m1^i\boldsymbol{z}$ by (10) and $h = 0$ is the maximum value such that $0^{m-h}10^h1^{i-1}\boldsymbol{z} \in \mathbf{N}(n,d)$.

If $j < i$, then $\mathsf{first}(\mathbf{G}_{n,d}(\boldsymbol{z})) = \mathsf{first}(\mathbf{G}_{n,d}(01^{i-1}\boldsymbol{z}))$ by (10). Thus we have only a single 1 left to place. The remaining argument depends on the value of $h$:

- If $h = 1$ then $\mathbf{G}_{n,d}(01^{i-1}\boldsymbol{z}) = 0^{m-1}101^{i-1}\boldsymbol{z}$ by (10). Thus $\mathsf{first}(\mathbf{G}_{n,d}(\boldsymbol{z})) = 0^{m-h}10^h1^{i-1}\boldsymbol{z}$.
- If $h > 1$ then $\mathsf{first}(\mathbf{G}_{n,d}(01^{i-1}\boldsymbol{z})) = \mathsf{first}(\mathbf{G}_{n,d}(001^{i-1}\boldsymbol{z}))$ by (10) and $d - den(01^{i-1}\boldsymbol{z}) = 1$. By repeating the same argument, $\mathsf{first}(\mathbf{G}_{n,d}(01^{i-1}\boldsymbol{z})) = \mathsf{first}(\mathbf{G}_{n,d}(0^h1^{i-1}\boldsymbol{z}))$. Finally, $\mathbf{G}_{n,d}(0^h1^{i-1}\boldsymbol{z}) = 0^{m-h}10^h1^{i-1}\boldsymbol{z}$ by (10).

$\square$

PROOF OF THEOREM 2: We assume that $\mathbf{G}_{n,d}(\boldsymbol{z})$ has at least 2 strings, otherwise the result is trivial. Observe that since $\mathbf{R}_{n,d}(\boldsymbol{z})$ is an ordering of the fixed-densiy necklaces with suffix $\boldsymbol{z}$ (Lemma 4), then so is $\mathbf{G}_{n,d}(\boldsymbol{z})$. Thus, we simply need to show that successive strings $\boldsymbol{s}$ and $\boldsymbol{t}$ in the listing differ by a left shift, and that this shift can be obtained by at most two transpositions. We prove the cyclic property as a special case at the end of the proof.

Let $\boldsymbol{z}'$ denote the longest common suffix of $\boldsymbol{s}$ and $\boldsymbol{t}$ that begins with 0. Clearly $\boldsymbol{z}$ is a suffix of $\boldsymbol{z}'$. Thus, from the recurrence of $\mathbf{G}_{n,d}(\boldsymbol{z}')$, the string $\boldsymbol{s}$ must the last string in one of the recursive sub-lists, and $\boldsymbol{t}$ must be the first string in the following sub-list. Specifically, $\boldsymbol{s} = \mathsf{last}(\mathbf{G}_{n,d}(01^j\boldsymbol{z}'))$ for some $j \geq 0$. From Lemma 5, this means that $\boldsymbol{s} = 0^{m-1}1^{i-j}01^j\boldsymbol{z}'$ where $i = d - den(\boldsymbol{z}')$ and $m = n - i - |\boldsymbol{z}'|$. There are two cases for $\boldsymbol{t}$ depending on whether or not it is the last string of the recurrence:

**Case 1**: $\boldsymbol{t} = \mathsf{first}(\mathbf{G}_{n,d}(01^{j-1}\boldsymbol{z}'))$. From Lemma 5, $\boldsymbol{t} = 0^{m-h-1}10^h1^{i-j}01^{j-1}\boldsymbol{z}'$ for some $h$. Thus to go from $\boldsymbol{s}$ to $\boldsymbol{t}$ we can left shift the first 1 in the block $0^j$ to position $m - h$. From Lemma 5 this position is clearly defined. Also observe that this shift can be made by performing two swaps of the bits at positions $m$ and $m - h$ and positions $m + i - j$ and $m + i - j + 1$.

**Case 2**: $\boldsymbol{t} = 0^m1^i\boldsymbol{z}'$. In this case we left shift the 0 after the first block of 1s to the front of the string. Observe that this can be accomplished by the single swap of bits in position $m$ and $m + i - j$. It is important to note that if we are in this case then there is no necklace with suffix $0^{j-1}\boldsymbol{z}'$. This allows us to clearly differentiate which case we are in strictly from the content of $\boldsymbol{s}$.

To prove the cyclic property we consider the case where $\boldsymbol{s}$ is the last string in the listing and $\boldsymbol{t}$ is the first. Clearly

$s = 0^m 1^i \boldsymbol{z}$ where $i = d - den(\boldsymbol{z})$ and $m = n - i - |\boldsymbol{z}|$. The first string $\boldsymbol{t} = 0^{m-h}10^h1^{i-1}\boldsymbol{z}$. Again, we can get from $\boldsymbol{s}$ to $\boldsymbol{t}$ by a left shift of the 1 in position $m + i$ to position $m - h + 1$ which can be accomplished by the single swap of bits in position $m - h + 1$ and $m + 1$. $\square$