Constructing the first (and coolest) fixed-content universal cycle

Joe Sawada
University of Guelph, Canada
jsawada@uoguelph.ca

Aaron Williams*[1]
Williams College, USA
aaron.williams@williams.edu

Abstract

We explicitly construct the first universal cycles for strings with fixed-content—also known as strings with the same Parikh vector, or multiset permutations—using their shorthand encoding, which omits the final symbol as it is redundant. For example, 112312131132 is a universal cycle for content $S = \{1,1,2,3\}$. Its first three windows—112, 123, and 231—are the shorthand representatives of 1123, 1231, and 2311, respectively.

Our first construction $\mathcal{V}(S)$ applies the classic cycle-joining approach on the *first-inversion tree* of necklace cycles with content S. For example, when $S = \{1,1,2,3\}$ the root is the necklace cycle 1123 and its children are $1\underline{21}3$ and $11\underline{32}$ by swapping their first (i.e., leftmost) inversions. From this construction, we derive a successor rule to generate successive symbols of $\mathcal{V}(S)$ in $O(n)$-time, where $n = |S|$ is the cardinality of S.

Our second construction $\mathcal{U}(S)$ concatenates fixed-content necklaces together in a cool-lex order using the necklace-prefix algorithm. For example, $\mathcal{U}(S) = 1123 \cdot 1213 \cdot 1132$ for $S = \{1,1,2,3\}$. Central to this construction is the first shift Gray code for fixed-content necklaces, and a new efficient algorithm for generating these strings. From this construction, we can generate successive symbols of $\mathcal{U}(S)$ in $O(1)$-amortized time while using $O(n)$-space. We complete our investigation with a pleasant surprise: $\mathcal{V}(S) = \mathcal{U}(S)$.

Our new results simultaneously generalize universal cycle constructions of shorthand permutations by Ruskey, Holroyd, and Williams [*Algorithmica* 64 (2012)] and shorthand fixed-weight binary strings by Ruskey, Sawada, and Williams [*SIAM J. on Disc. Math.* 26 (2012)]. They also provide a prefix-shift Gray code for multiset permutations in which the first symbol moves into the last or second-last position, which tightens the previous prefix-shift Gray code by Williams [*Proc. Annu. ACM-SIAM Symp. Discrete Algorithms* (2009)]. Finally, we draw parallels between our constructions and the well-known *granddaddy de Bruijn sequence* for binary strings.

Funding *Joe Sawada*: Natural Sciences and Engineering Research Council of Canada (NSERC) grant RGPIN 400673-2012

1 Introduction

A *universal cycle* for a set $\mathbf{E}$ of length n strings[2] is a circular string of length $|\mathbf{E}|$ where each string in $\mathbf{E}$, or an encoding of it, appears exactly once as a substring [6]. Universal cycles generalize *de Bruijn sequences*, where $\mathbf{E}$ is the set of k-ary strings of length n [9, 51] (and see [10]). For example, 00010111 is a de Bruijn sequence for $k = 2$ and $n = 3$ since its substrings — 000, 001, 010, 101, 011, 111, 110, 100 — are precisely the $k^n = 2^3 = 8$ binary strings of length 3. When visualizing de Bruijn sequences and universal cycles it is common to picture a *window* moving through the circular string, where the width of the window is equal to the length of the encodings of the objects in $\mathbf{E}$. See [18, 19] and a new website [40] for recent surveys on the rich history of these objects.

This paper constructs the first fixed-content universal cycle. More precisely, we construct two different

[1] The corresponding author is Aaron Williams `aaron.williams@williams.edu`.
[2] Universal cycles can also be defined for objects other than strings (e.g., graphs [3]).

universal cycles with content S: $\mathcal{V}(S)$ and $\mathcal{U}(S)$. Then we show that the constructions are equivalent (i.e., $\mathcal{V}(S) = \mathcal{U}(S)$), and develop efficient algorithms for generating our universal cycle.

Throughout the article, S is a multiset of symbols referred to as the *content*. By convention, S has k distinct symbols $1, 2, \ldots, k$, and its cardinality (including repetitions) is n. For example, $k = 3$ and $n = 4$ when $S = \{1, 1, 2, 3\}$. The symbol $\cdot$ denotes concatenation and is optional, so $a \cdot b = ab$.

1.1 Fixed-Content Universal Cycles

De Bruijn sequences are examples of universal cycles that store the contents of $\mathbf{E}$ without any encoding, and additional examples exist for many other interesting sets [6, 11, 25, 26, 30, 31, 47]. But this is not possible for many other natural sets, including the permutations of $\{1, 2, \ldots, n\}$ in one-line notation, and the n-bit binary strings with *weight* w (i.e., w copies of 1). To illustrate this point, if the permutations of $n = 3$ (i.e., $\mathbf{E} = \{123, 132, 213, 231, 312, 321\}$) had a universal cycle, then $123abc$ would be such a cycle (as any cycle could be rotated to have 123 as its first substring). Since $23a$ is a window of the universal cycle, it must be that $23a \in \mathbf{E}$, and hence $a = 1$. Similarly, $b = 2$ and $c = 3$. But $123abc = 123123$ is not a universal cycle for $\mathbf{E}$. Likewise, there is no universal cycle for the binary strings of length $n = 4$ and weight $w = 2$ (i.e., $S = \{0011, 0101, 0110, 1001, 1010, 1100\}$).

Fortunately, permutations and fixed-weight binary strings have a simple alternate encoding: The *shorthand representation* of a string omits its final symbol. This is a suitable choice since permutations and fixed-weight strings are determined by their length $n-1$ prefixes, as the final symbol is redundant.

Example 1 Consider the set $\mathbf{E}_1 = \{12, 13, 21, 23, 31, 32\}$ of shorthand permutations for $n = 3$. Observe that 231321 is a universal cycle for $\mathbf{E}_1$.

Example 2 Consider the set $\mathbf{E}_2 = \{0001, 0010, 0100, 1000, 0011, 0110, 0101, 1001, 1010, 1100\}$ of shorthand fixed-weight strings of length $n = 5$ and weight $w = 2$. Observe that 1010011000 is a universal cycle for $\mathbf{E}_2$.

In this paper, we consider the natural generalization of permutations and fixed-weight binary strings, namely *strings with fixed-content*. These objects are also known strings with the same *Parikh vector* and as *multiset permutations*. Owing to the latter term, we let $\boldsymbol{Perm}(S)$ denote the strings with content S. For example, the set of strings with content $S = \{1, 1, 2, 3\}$ is

$$\boldsymbol{Perm}(S) = \{1123, 1132, 1213, 1231, 1312, 1321, 2113, 2131, 2311, 3112, 3121, 3211\}.$$

We let $\boldsymbol{Short}(S)$ denote the shorthand representation of strings with content S. For example, the set of shorthand representations of strings with content $S = \{1, 1, 2, 3\}$ is

$$\boldsymbol{Short}(S) = \{112, 113, 121, 123, 131, 132, 211, 213, 231, 311, 312, 321\}.$$

As with permutations and fixed-weight binary strings, the final symbol of a string with content S is redundant, so the shorthand encoding can be used in a universal cycle without any loss of information. Such a cycle can be referred to as *universal cycle of* $\boldsymbol{Short}(S)$, or as a *shorthand universal cycle of* $\boldsymbol{Perm}(S)$, or simply as *fixed-content universal cycle over* S. Regardless of the name, the windows provide the strings in $\boldsymbol{Short}(S)$, which provide a simple encoding of the strings in $\boldsymbol{Perm}(S)$.

Example 3 Observe that 112312131132 is a universal cycle of $\boldsymbol{Short}(S)$ with $S = \{1,1,2,3\}$. It can also be described as a shorthand universal cycle for $\boldsymbol{Perm}(S)$, or as a fixed-content universal cycle over S.

1.1.1 First Symbol or Missing Symbol

If $\beta = b_1 \cdot b_2 \cdots b_{n-1} \in \boldsymbol{Short}(S)$, then b_1 is β's *first symbol*. The *missing symbol* from β is b_n for the unique b_n with $\beta \cdot b_n \in \boldsymbol{Perm}(S)$. We also use $z = b_n$ for the missing symbol to emphasize that it is not included in β. For example, if $S = \{1,1,2,3\}$ and $\beta = 213 \in \boldsymbol{Short}(S)$, then β's first symbol is $b_1 = 2$ and its missing symbol is $z = b_4 = 1$. When viewed symbol-by-symbol, fixed-content universal cycles repeatedly select between the first symbol and the missing symbol of the current window, as formalized by the following lemma.

▶ **Lemma 1.** *Let $\mathcal{W}$ be a fixed-content universal cycle over S and $\beta = b_1 \cdot b_2 \cdots b_{n-1} \in \boldsymbol{Short}(S)$ with missing symbol z. The window β appears exactly once in $\mathcal{W}$ and it is immediately followed by its first symbol b_1 or its missing symbol z. Equivalently, if β' is the window following β in $\mathcal{W}$, then*

Case 0: $\beta' = b_2 \cdots b_{n-1} \cdot z$ or
Case 1: $\beta' = b_2 \cdots b_{n-1} \cdot b_1$.

Proof. Since z is missing from $\beta \in \boldsymbol{Short}(S)$, we have $\beta \cdot z = b_1 \cdot b_2 \cdots b_{n-1} \cdot z \in \boldsymbol{Perm}(S)$ and $S = \{b_1, b_2, \ldots, b_{n-1}, z\}$. Since β' follows β, we have $\beta' = b_2 \cdots b_{n-1} \cdot x \in \boldsymbol{Short}(S)$ for some symbol $x \in S$. Thus, the claim holds by the following (where $-$ denotes multiset difference),

$$x \in S - \{b_2, \ldots, b_{n-1}\} = \{b_1, b_2, \ldots, b_{n-1}, z\} - \{b_2, \ldots, b_{n-1}\} = \{b_1, z\}. \quad ◀$$

It is important to note that a window's first symbol and missing symbol can be equal. For example, if $S = \{1,1,2,3\}$ then $b_1 = z = 1$ for $123 \in \boldsymbol{Short}(S)$ and $132 \in \boldsymbol{Short}(S)$. In these situations, the two cases of Lemma 1 are identical, and there is only one choice for next symbol and window.

1.2 Characterizations using Graphs

Fixed-content universal cycles can be characterized using several different directed graphs with labeled edges. Each characterization provides its own insights into this new type of universal cycles.

1.2.1 Transition Graphs: Hamilton Cycles and Binary Representation

The graph $T(S)$ has vertex set $\boldsymbol{Short}(S)$ and edges uv for $u = b_1 \cdot b_2 \cdots b_{n-1} \in \boldsymbol{Short}(S)$ and $v = b_2 \cdot b_3 \cdots b_n$ with label b_n. In other words, the vertices are windows, and edges transition from window to window by shifting in their label. This type of graph is often known as a *transition graph* within the universal cycle literature, and it leads directly to the characterization in Remark 2, which views universal cycles and Hamilton cycles as cyclic objects (i.e., they are unchanged by rotation).

▶ Remark 2. Universal cycles with fixed-content S are in one-to-one correspondence with the concatenation of the edge labels along the Hamilton cycles in $T(S)$.

For our purposes, it is helpful to consider an *augmented transition graph* $T'(S)$. This graph has vertex set $\boldsymbol{Short}(S)$ and its edge set is defined based on the first and missing symbols from Lemma 1. That is, if $u = b_1 \cdot b_2 \cdots b_{n-1} \in \boldsymbol{Short}(S)$ is missing z, then there are two edges of the form uv where

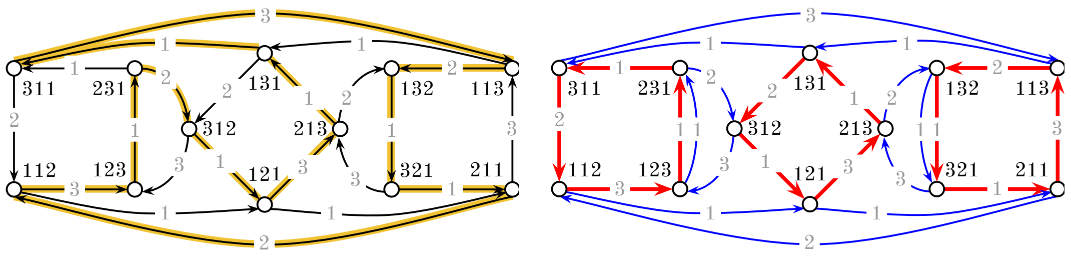

(a) The transition graph $T(S)$ with a Hamilton cycle. (b) The augmented transition graph $T'(S)$.

Figure 1 Transition graphs for $S = \{1, 1, 2, 3\}$. Our universal cycle 112312131132 follows the Hamilton cycle from vertex 132 in (a). It has binary representation 001000110001 starting from vertrex 112 in (b), where thick straight and thin curved edges are of type 0 and 1, respectively. Its weight of 4 is minimum possible for S.

Edge 0: $v = b_2 \cdots b_{n-1} \cdot z$ with label z, and
Edge 1: $v = b_2 \cdots b_{n-1} \cdot b_1$ with label b_1.

This definition ensures that each vertex has out-degree two. Furthermore, each vertex has in-degree two. This is because the edges partition into two vertex-disjoint directed cycle covers: one includes the 0 edges, and the other includes the 1 edges. Figure 1 shows $T(S)$ and $T'(S)$ for $S = \{1, 1, 2, 3\}$.

The augmented transition graph visualizes a simple consequence of Lemma 1: fixed-content universal cycles can be represented in binary. More precisely, a *binary representation* is an initial window in $\boldsymbol{Short}(S)$ and a binary string of length $|\boldsymbol{Short}(S)|$ whose bits follow the cases of Lemma 1 or the edge types of $T'(S)$. If we use the convention that the initial window is the lexicographically smallest string in $\boldsymbol{Short}(S)$ (e.g., 112 for $S = \{1, 1, 2, 3\}$), then the binary string suffices by itself. A fixed-content universal cycle has *weight* w if it has a binary representation of weight w, and it has *minimum-weight* if it has a binary representation of minimum possible weight for its content S.

1.2.2 Arc Digraphs: Eulerian Circuits and Universal Cycle Existence

To prove that universal cycles exist we can model the windows as edges rather than vertices. Let the vertices of $A(S)$ be the $(n-2)$-permutations of S (i.e., two symbols are absent from S) with an edge from $u = b_1 \cdot b_2 \cdots b_{n-2}$ to $v = b_2 b_3 \cdots b_{n-1}$ with label b_{n-1} if $b_1 \cdot b_2 \cdots b_{n-1} \in \boldsymbol{Short}(S)$. This type of graph is known as an *arc digraph* within the literature, and it leads to a second characterization.

▶ Remark 3. Universal cycles with fixed-content S are in one-to-one correspondence with the concatenation of the edge labels along the Eulerian circuits in $A(S)$.

Figure 2 shows $A(S)$ for $S = \{1, 1, 2, 3\}$. The fact that $A(S)$ is always Eulerian is a consequence of our new results; it also provides a nice exercise for active readers. Remark 3 was previously proven for permutations (i.e., $k = n$) by Jackson [27] and fixed-weight binary strings (i.e., $k = 2$) [35].

1.2.3 Rotator Graphs: Shift Gray Codes for (Multiset) Permutations

Recall that the vertex set of $T'(S)$ is $\boldsymbol{Short}(S)$. By replacing each vertex with its corresponding member of $\boldsymbol{Perm}(S)$ we obtain our final graph $R(S)$. Figure 3 illustrates $R(S)$ for $S = \{1, 1, 2, 3\}$.

The edges of $R(S)$ can be understood as applying an operation known as a *shift*. Given string $\alpha = b_1 \cdot b_2 \cdots b_n$, let $\mathsf{shift}_\alpha(i, j)$ (or simply $\mathsf{shift}(i, j)$) be the result of moving b_i into the jth position. In the special case of $i = 1$ (i.e., the first symbol is moved to the right) we let σ_j denote $\mathsf{shift}(i, j)$. If $u = b_1 \cdot b_2 \cdots b_{n-1} b_n \in \boldsymbol{Perm}(S)$, then $R(S)$ contains two edges of the form uv where

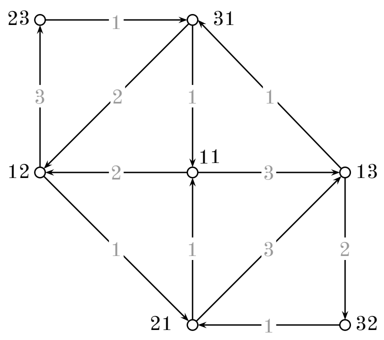


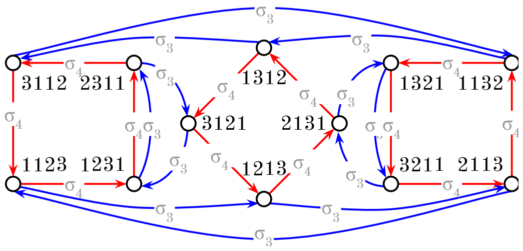
Figure 2 The arc digraph $A(S)$ for $S = \{1, 1, 2, 3\}$. Our shorthand universal cycle 112312131132 starts at vertex 32 and then travels to 21 and 11.

Edge 0: $v = \mathsf{shift}_u(1, n) = b_2 \cdots b_{n-1} \cdot b_n \cdot b_1$ with label σ_n, and

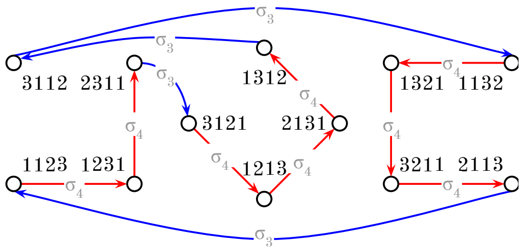
Edge 1: $v = \mathsf{shift}_u(1, n-1) = b_2 \cdots b_{n-1} \cdot b_1 \cdot b_n$ with label σ_{n-1}.

In other words, an edge shifts the first symbol into the last or second-last position. Thus, Hamilton cycles of $R(S)$ provide a *shift Gray code* of $\boldsymbol{Perm}(S)$, meaning that each succesive string is obtained by a shift. The Gray codes are also *cyclic* since a shift transforms the last string into the first string.

▶ Remark 4. Universal cycles with fixed-content S are in one-to-one correspondence with the Hamilton cycles of $R(S)$. In turn, the Hamilton cycles are in one-to-one correspondence with cyclic Gray codes of $\boldsymbol{Perm}(S)$ in which each $\alpha \in \boldsymbol{Perm}(S)$ is followed by $\mathsf{shift}_\alpha(1, n)$ or $\mathsf{shift}_\alpha(1, n-1)$.



(a) The graph $R(S)$ for $S = \{1, 1, 2, 3\}$.



(b) Our Hamilton cycle is a shift Gray code of $\boldsymbol{Perm}(S)$.

Figure 3 The graph $R(S)$ for $S = \{1, 1, 2, 3\}$ in (a). The Hamilton cycle in $R(S)$ starting from vertex 1321 corresponds to our universal cycle $\mathcal{V}(S) = \mathcal{U}(S) = 112312131132$ in (b).

Our main results contribute to the literature on shift Gray codes for (multiset) permutations due to Remark 4. Corbett provided the first such result with a Hamilton cycle in the *rotator graph* whose vertices are permutations (i.e., $\boldsymbol{Perm}(S)$ with $n = k$) and whose edges apply σ_i for all $2 \le i \le n$ [8]. The rotator graph is used as a multiprocessor network topology and Corbett's cycle can be generated by the greedy Gray code algorithm [54]. It is possible to create a Hamilton cycle using only the following three operations [49]: $\tau = \sigma_2$; σ_3; $\sigma = \sigma_n$. If edges in the opposite direction are also allowed, then σ, σ^{-1}, and τ are sufficient [7]. When n is odd, σ and τ allow for Hamilton paths [43] and cycles [44], but only Hamilton paths are possible when n is even [33, 50]. When S is a multiset rather than a set, then Gray codes do not exist using σ, σ^{-1}, and τ when $k = 2$ [5]. However, a cyclic Gray code known as *cool-lex order* had been shown to exist using all σ_i [52]. Cool-lex order was first discovered for fixed-weight binary strings (i.e., $\boldsymbol{Perm}(S)$ with $k = 2$) which are also known as *combinations*, and its name comes from its similarity to co-lexicographic order [36].

1.3 Necklaces and Necklace Cycles

A *necklace* is a lexicographically smallest string in an equivalence class of strings under rotation. Let $\boldsymbol{N}(S)$ be the set of necklaces with content S. For example, $\boldsymbol{N}(S) = \{1123, 1132, 1213\}$ for $S = \{1, 1, 2, 3\}$. A *necklace cycle* is a cyclic order of length n strings obtained by repeatedly applying $\mathsf{shift}(1, n)$. In general, the *length* of a necklace cycle divides n. For example, the necklace cycle containing 132132 has length 3. A necklace cycle contains one necklace, which is its *representative*. For example, the thick straight edges in Figure 3a create necklace cycles with representatives 1123, 1213, and 1132 from left-to-right. The *aperiodic prefix* of a string is its shortest prefix that can be repeated to create it. For example, the aperiodic prefix of 132132 is 132.

1.4 New Results

It is important to note that the graph-based models in Figure 1.2 require exponential space with respect to n, and so they do not lead to efficient algorithms for generating a single universal cycle. With this point in mind, we now summarize our main results below.

1. The construction of a minimum-weight universal cycle $\mathcal{V}(S)$ using cycle-joining in Section 2. More specifically, necklace cycles are joined together according to a *first-inversion tree*.
   - A successor rule that generates successive symbols of $\mathcal{V}(S)$ in $O(n)$-time.
2. The construction of universal cycle $\mathcal{U}(S)$ using a necklace concatenation approach in Section 4. More specifically, we concatenate $\boldsymbol{N}(S)$ in reverse cool-lex order (using aperiodic prefixes).
   - A new shift Gray code for fixed-content necklaces with an $O(n)$-amortized time algorithm.
   - The reversal of $\mathcal{U}(S)$ can be generated in $O(1)$-amortized time per symbol using $O(n)$ space.
3. A proof that the two constructions are equivalent in Section 5. That is, $\mathcal{V}(S) = \mathcal{U}(S)$.

Our constructions are implemented in C and are provided in the Appendix. The output can be viewed at `debruijnsequence.org` [40]. Section 6 concludes with future work and open problems.

Prior to this article, universal cycles for shorthand permutations (where $n = k$) and shorthand fixed-weight binary strings (where $k = 2$) were constructed and efficiently generated under slightly different names (see [37, 24] and [35]). Note that these previous works use lexicographically largest representatives for necklaces. Our use of lexicographically smallest representatives is more standard, but it requires an adjustment to the original definition of cool-lex order in Section 4. A preliminary version of this paper presented our necklace concatenation construction [45].

1.5 Granddaddy and Cool-Daddy

Besides our main results, we wish to suggest to the reader that our fixed-weight universal cycle is both natural and fundamental. As a point of comparison, we consider the most famous de Bruijn sequence.

The *granddaddy de Bruijn sequence* $\mathcal{G}_k(n)$ (as coined by Knuth [29]) is the lexicographically smallest de Bruijn sequence for k-ary strings of length n. For example, $\mathcal{G}_2(4) = 0000100110101111$ is the granddaddy for $n = 4$ and $k = 2$. The granddaddy can be constructed in several elegant ways.

- The granddaddy is constructed by a simple prefer-smallest greedy algorithm (see [32]).
- The granddaddy is constructed by a simple cycle joining approach. More specifically, necklace cycles are joined together according to a *last non-k tree* (see [18, 19]); here we use the term *last-0 tree* when referring to the binary case (i.e., $k = 2$).
- The granddaddy is constructed by a simple necklace concatenation approach. More specifically, necklaces are concatenated in lexicographic order (using aperiodic prefixes) (see [15, 16]).

To our knowledge, no simple greedy algorithm constructs our fixed-content universal cycle. However, our contributions match the latter two bullets quite closely, especially in the binary case.

In terms of cycle joining, when constructing the granddaddy $\mathcal{G}_2(6)$, the necklace cycle represented by $001\underline{0}11$ is joined to the necklace cycle represented by 001111 by complementing the last zero (as underlined). Similarly, when constructing our fixed-content universal cycle with $S = \{1, 1, 2, 2, 3, 3\}$, the necklace cycle represented by $12\underline{31}23$ is joined to the necklace cycle represented by 121323 by swapping the first inversion (as underlined).

In terms of necklace concatenation, when $n = 4$ and $k = 3$, the granddaddy $\mathcal{G}_3(4)$ is equal to

$$0{\cdot}000001{\cdot}000011{\cdot}000101{\cdot}000111{\cdot}001{\cdot}001011{\cdot}001101{\cdot}001111{\cdot}01{\cdot}010111{\cdot}011{\cdot}011111{\cdot}1 \tag{1}$$

owing to the lexicographic order of binary necklaces of length 6 (and their aperiodic prefixes). Similarly, our fixed-content universal cycle $\mathcal{U}(S)$ with content $S = \{1, 1, 2, 2, 3, 3\}$ is equal to

$$112233{\cdot}122313{\cdot}123213{\cdot}122133{\cdot}121233{\cdot}112332{\cdot}123132{\cdot}132{\cdot}121332{\cdot}113322{\cdot}131322{\cdot}113232{\cdot}112323{\cdot}123{\cdot}121323{\cdot}113223 \tag{2}$$

owing to the reverse cool-lex order of necklaces in $\boldsymbol{N}(S)$ (and their aperiodic prefixes).

These comparisons with the binary granddaddy are summarized in Figure 4. Due to the similarities, we refer to our fixed-content universal cycle as the *cool-daddy*.

	Greedy	Cycle-Joining	Necklace Concatenation
Binary Granddaddy	prefer-smallest	last-0	lexicographic order
Cool-Daddy	N/A	first-inversion	reverse cool-lex order

Figure 4 Comparing the binary granddaddy de Bruijn sequence $\mathcal{G}_2(n)$ for n-bit binary strings, and our fixed-content universal cycle $\mathcal{U}(S) = \mathcal{V}(S)$ with content S. See Figure 5 and (1)–(2) for specific examples.

2 Cycle Joining feat. the First-Inversion Spanning Tree

We begin this section by outlining the classic cycle-joining approach used to construct de Bruijn sequences and universal cycles. We then apply the approach to derive a simple successor rule for a fixed-content universal cycle. The rule is based on a string's "first-inversion", where an *inversion* in a string $a_1 \cdots a_n$ is a consecutive pair a_i, a_{i+1} where $a_i > a_{i+1}$.

2.1 Cycle Joining

Call a string in $\boldsymbol{Short}(S)$ a *state*. A function $f : \boldsymbol{Short}(S) \to S$ is said to be a *feedback function*. Given such a feedback function f, let function $F : \boldsymbol{Short}(S) \to \boldsymbol{Short}(S)$ map a state $\beta = b_1 b_2 \cdots b_{n-1}$ to state $b_2 \cdots b_{n-1} f(\beta)$. A *cycle* is a sequence of distinct states $\beta_1, \beta_2, \ldots, \beta_j$ such that $F(\beta_i) = \beta_{i+1}$ for $1 \le i < j$ and $F(\beta_j) = \beta_1$. Each cycle can be represented by a single string $c_1 \cdots c_j$ where c_i corresponds to the first symbol of β_i.

Example 4 Consider content $S = \{1, 1, 2, 2, 3, 3\}$. Let $f(\beta) = z$, recalling βz has content S. That is, $\beta \in \boldsymbol{Short}(S)$ and z is its missing symbol with respect to S. Then the cycles

$$\underline{1}2132, \underline{2}1323, \underline{1}3231, \underline{3}2312, \underline{2}3121, \underline{3}1213 \quad \text{and} \quad \underline{1}2312, \underline{2}3123, \underline{3}1231$$

can be represented by the strings $C_1 = 121323$ and $C_2 = 123$, respectively.

Note that C_1 is a universal cycle for $\{12132, 21323, 13231, 32312, 23121, 31213\}$ and C_2 is a universal cycle for $\{12312, 23123, 31231\}$.

If $\beta_1 = xb_2 \cdots b_{n-1}$ and $\beta_2 = yb_2 \cdots b_{n-1}$ are both states where $x \neq y$ then we say β_1 and β_2 form a *conjugate pair*. For each state β there is at most one other state that forms a conjugate pair with β because of the content restrictions. If C_1 and C_2 are disjoint cycles where C_1 contains one state from a conjugate pair and C_2 contains the other, then the two cycles can be "joined" together to form a single cycle by swapping the successors of the conjugate states. This "cycle-joining" process is well known, and formally stated in [13, Thm. 1] and [46, Lemma 3]. The process is a special case of Hierholzer's cycle joining algorithm for producing Euler cycles [23].

Example 5 Consider the cycles C_1 and C_2 from Example 4 and the conjugate pair $3\underline{2312}$, $1\underline{2312}$; the state 32312 is in C_1 and the state 12312 is in C_2. By swapping the successors of these states we obtain a single cycle for the union of the states from C_1 and C_2:

$$12132, 21323, 13231, 32312, 23123, 31231, 12312, 23121, 31213$$

corresponding to $C = 121323123$.

This cycle-joining process has been formalized in [18, 19] to produce a number of simple successor rules for de Bruijn sequences and universal cycles. Next we apply the cycle-joining approach to construct a universal cycle for $\textbf{\textit{Short}}(S)$.

2.2 The First-Inversion Tree

Let $\beta = b_1b_2 \cdots b_{n-1} \in \textbf{\textit{Short}}(S)$, where z denotes the missing symbol, and consider the following feedback function

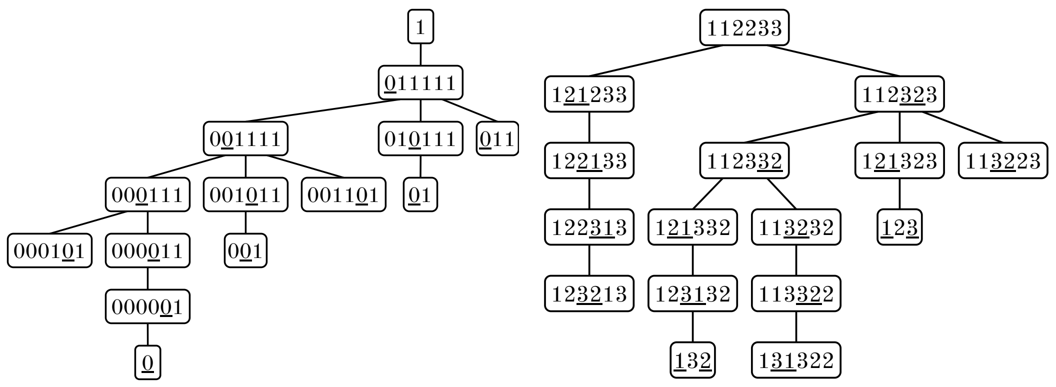
$$f(\beta) = z. \tag{3}$$

This function can be used to partition $\textbf{\textit{Short}}(S)$ into necklace cycles whose representatives are in $\textbf{\textit{N}}(S)$. More specifically, each $b_1b_2 \cdots b_{n-1} \in \textbf{\textit{Short}}(S)$ with missing symbol z is followed by $b_2 \cdots b_{n-1}z \in \textbf{\textit{Short}}(S)$ in a cycle. For example, the cycle associated with $112233 \in \textbf{\textit{N}}(S)$ includes the strings $11223, 12233, 22331, 23311, 33112, 31122 \in \textbf{\textit{Short}}(S)$.

Let $tail(S)$ denote the unique non-decreasing string composed of all the elements of S; it is a necklace. The cycles can be joined together into a spanning tree rooted at $\text{tail}(S)$ where the parent of every necklace α, not including $\text{tail}(S)$, is obtained by transposing the two symbols in the "first inversion" of α. We call the resulting tree the *first-inversion tree*[3]. For example, see the first-inversion tree in Figure 5b for content $S = \{1, 1, 2, 2, 3, 3\}$. It is easy to see that such a mapping always induces a tree since the parent of each necklace α is lexicographically smaller than α. In fact, the paths from each node to the root resembles insertion sort (or more accurately, gnome sort [39]).

[3] This generalizes the *decreasing spanning tree* presented in [24], but with the relative values of the symbols inverted.

[4] The non-binary grand-daddy sequences (i.e., with $k > 2$) are similar, but somewhat more complicated. Specifically, the necklace cycles do not group into conjugate pairs, but rather groups of size k. Thus, the actual cycle joining is more complicated, even though a similar tree-like structure is present.



(a) The last-0 tree for binary strings of length $n = 6$ rooted at the necklace 111111. Parents are obtained by complementing the last zero (underlined).

(b) The first-inversion tree for content $S = \{1,1,2,2,3,3\}$ rooted at the necklace 112233. Parents are obtained by swapping the first inversion (underlined).

Figure 5 Necklace cycles are joined into tree-like structures in both the (a) (binary) grand-daddy de Bruijn sequence $\mathcal{G}_k(n)^4$, and (b) our cool-daddy universal cycle $\mathcal{V}(S)$. In both figures, the underlines illustrate the parent rule, and each necklace cycle is denoted by the aperiodic prefix of its representative. For example, in (a) the necklace cycle containing $0101\underline{0}1$ is denoted $\underline{0}1$, and in (b) the necklace cycle $123\underline{12}3$ is denoted $\underline{12}3$.

2.3 A Simple Successor Rule

When the symbols involved in the first inversion of a necklace are rotated to the front of both the necklace and its parent, then the length $n-1$ suffixes correspond to conjugate pairs whose successors are swapped during the cycle joining process. For example, given the necklace $113\underline{32}2$ and its parent $113\underline{23}2$, the conjugate pairs are 22113 and 32113. Let $\boldsymbol{X}(S)$ denote the set of all $2|\mathbf{N}(S) - 1|$ states from such conjugate pairs. Repeated application of the cycle-joining approach yields a universal cycle for $\boldsymbol{Short}(S)$ with the following successor-rule:

$$g(\beta) = \begin{cases} b_1 & \text{if } \beta \in \boldsymbol{X}(S) \\ z & \text{otherwise.} \end{cases}$$

Let $\mathcal{V}(S)$ denote the universal cycle for $\boldsymbol{Short}(S)$ obtained by starting with $\mathrm{tail}(S)$ and repeatedly applying g on the last $n-1$ symbols. In other words, $g(\beta)$ is the symbol following β in the universal cycle $\mathcal{V}(S)$. For example, $\mathcal{V}(\{1,1,2,2,3,3\})$ =

112233122313123213122133121233112332123132132121332113322131322113232112323123121323113223.

Testing whether or not a state β is in $\boldsymbol{X}(S)$ can be done in $O(n)$ time as follows. Let $h(\beta)$ be the rotation of β that takes the longest non-decreasing suffix of $b_3 \cdots b_n$ and rotates it to the front of β. For example $h(123321233) = 123312332$.

▶ **Lemma 5.** *Let $\beta = b_1 b_2 \cdots b_{n-1}$ be a state with missing symbol z. Then $\beta \in \boldsymbol{X}(S)$ if and only if*

- *$b_1 > z$ and $h(b_1 z b_2 \cdots b_{n-1})$ is a necklace and $b_1 \geq b_{n-1}$, or*
- *$z > b_1$ and $h(z b_1 b_2 \cdots b_{n-1})$ is a necklace and $z \geq b_{n-1}$.*

Proof. Let x, y denote the elements z and b_1 listed in decreasing order. For β to be in $\boldsymbol{X}(S)$, the conjugate pair $z b_1 b_2 \cdots b_{n-1}$ and $b_1 z b_2 \cdots b_{n-1}$ must satisfy the properties that $x \neq y$ and xy is the first inversion in the necklace representative for $x y b_2 \cdots b_{n-1}$. Since xy is the first inversion, $h(x y b_2 \cdots b_{n-1})$ must be a necklace and $x \geq b_{n-1}$. ◀

Together, Lemma 5 and Lemma 9 (in Section 4.2) imply the following result.

▶ **Corollary 6.** *If $\beta = b_1 \cdots b_{n-1}$ is in $\boldsymbol{X}(S)$ then both $h(z\beta)$ and $h(b_1 z b_2 \cdots b_{n-1})$ are necklaces.*

Example 6 If $\beta = b_1 \cdots b_5 = 32113$ for content $S = \{1,1,2,2,3,3\}$, then $z = 2$ and $b_1 = 3$. Since $b_1 > z$, $h(322113) = 113322$ is a necklace, and $b_1 \geq b_5$, the state belongs to $\boldsymbol{X}(S)$.

Based on Lemma 5, the following is a successor rule to construct $\mathcal{V}(S)$. An illustration is in Figure 6.

Successor rule for fixed-content universal cycle $\mathcal{V}(S)$

Let $\beta = b_1 b_2 \cdots b_{n-1}$ be a state where z is the missing symbol. Then

$$g(\beta) = \begin{cases} b_1 & \text{if } z > b_1 \text{ and } h(zb_1b_2\cdots b_{n-1}) \text{ is a necklace and } z \geq b_{n-1} & \text{(4a)} \\ b_1 & \text{if } b_1 > z \text{ and } h(b_1zb_2\cdots b_{n-1}) \text{ is a necklace and } b_1 \geq b_{n-1} & \text{(4b)} \\ z & \text{otherwise.} & \text{(4c)} \end{cases}$$

Since testing if a string is a necklace can be done in $O(n)$ time [2] we obtain the following theorem. The minimum-weight property is a direct consequence of using the feedback function in (3) since the spanning tree joins the necklace cycles (which use weight 0 edges) using as few weight 1 edges as possible.

▶ **Theorem 7.** *$\mathcal{V}(S)$ is a minimum-weight universal cycle for $\boldsymbol{Short}(S)$ that can be constructed in $O(n)$ time per symbol.*

3 A Shift Gray Code for Multiset Permutations

An immediate consequence of the successor rule described in the previous section is a shift Gray code for $\boldsymbol{Perm}(S)$ whose successor-rule, defined as follows, shifts the first symbol to either the nth or $(n{-}1)$st position in the string.

Successor rule for $\boldsymbol{Perm}(S)$ using two operations

Let $\alpha = a_1 a_2 \cdots a_n \in \boldsymbol{Perm}(S)$. Then

$$\mathsf{nextMultiPerm}(\alpha) = \begin{cases} \mathsf{shift}_\alpha(1, n{-}1) & \text{if } g(a_1 \cdots a_{n-1}) = a_1, & \text{(5a)} \\ \mathsf{shift}_\alpha(1, n) & \text{if } g(a_1 \cdots a_{n-1}) = a_n. & \text{(5b)} \end{cases}$$

▶ **Theorem 8.** *Starting with any initial string α in $\boldsymbol{Perm}(S)$ and repeatedly applying the function* $\mathsf{nextMultiPerm}(\alpha)$ *a total of $|\boldsymbol{Perm}(S) - 1|$ times produces a cyclic shift Gray code for $\boldsymbol{Perm}(S)$. Moreover, the Gray code can be generated in $O(n)$-time per string.*

Example 7 Let $S = \{1,1,2,3\}$ and consider $\mathcal{V}(S) = 112312131132$. It corresponds to the following shift Gray code for $\boldsymbol{Perm}(S)$, with the index where the first symbol is shifted to obtain the next string in the listing given in the final column:

$\beta = b_1b_2b_3b_4b_5$		case	$\boldsymbol{N}(S)$
11223	🟦🟦🟩🟩🟧	(4c)	
12233	🟦🟩🟩🟧🟧	(4c)	
22331	🟩🟩🟧🟧🟦	(4b)	121233
23312	🟩🟧🟧🟦🟩	(4b)	122133
33122	🟧🟧🟦🟩🟩	(4b)	122313
31223	🟧🟦🟩🟩🟧	(4c)	
12231	🟦🟩🟩🟧🟦	(4c)	
22313	🟩🟩🟧🟦🟧	(4c)	
23131	🟩🟧🟦🟧🟦	(4c)	
31312	🟧🟦🟧🟦🟩	(4b)	123213
13123	🟦🟧🟦🟩🟧	(4c)	
31232	🟧🟦🟩🟧🟩	(4c)	
12321	🟦🟩🟧🟩🟦	(4c)	
23213	🟩🟧🟩🟦🟧	(4c)	
32131	🟧🟩🟦🟧🟦	(4c)	
21312	🟩🟦🟧🟦🟩	(4a)	123213
13122	🟦🟧🟦🟩🟩	(4a)	122313
31221	🟧🟦🟩🟩🟦	(4c)	
12213	🟦🟩🟩🟦🟧	(4c)	
22133	🟩🟩🟦🟧🟧	(4c)	
21331	🟩🟦🟧🟧🟦	(4c)	
13312	🟦🟧🟧🟦🟩	(4a)	122133
33121	🟧🟧🟦🟩🟦	(4c)	
31212	🟧🟦🟩🟦🟩	(4c)	
12123	🟦🟩🟦🟩🟧	(4c)	
21233	🟩🟦🟩🟧🟧	(4c)	
12331	🟦🟩🟧🟧🟦	(4a)	121233
23311	🟩🟧🟧🟦🟦	(4c)	
33112	🟧🟧🟦🟦🟩	(4b)	112323
31123	🟧🟦🟦🟩🟧	(4b)	112332
11233	🟦🟦🟩🟧🟧	(4c)	
12332	🟦🟩🟧🟧🟩	(4c)	
23321	🟩🟧🟧🟩🟦	(4b)	121332
33212	🟧🟧🟩🟦🟩	(4b)	123132
32123	🟧🟩🟦🟩🟧	(4c)	
21231	🟩🟦🟩🟧🟦	(4c)	
12313	🟦🟩🟧🟦🟧	(4c)	
23132	🟩🟧🟦🟧🟩	(4c)	
31321	🟧🟦🟧🟩🟦	(4b)	132132
13213	🟦🟧🟩🟦🟧	(4c)	
32132	🟧🟩🟦🟧🟩	(4c)	
21321	🟩🟦🟧🟩🟦	(4a)	132132
13212	🟦🟧🟩🟦🟩	(4a)	123132
32121	🟧🟩🟦🟩🟦	(4c)	
21213	🟩🟦🟩🟦🟧	(4c)	
12133	🟦🟩🟦🟧🟧	(4c)	
21332	🟩🟦🟧🟧🟩	(4c)	
13321	🟦🟧🟧🟩🟦	(4a)	121332
33211	🟧🟧🟩🟦🟦	(4b)	113232
32113	🟧🟩🟦🟦🟧	(4b)	113322
21133	🟩🟦🟦🟧🟧	(4c)	
11332	🟦🟦🟧🟧🟩	(4c)	
13322	🟦🟧🟧🟩🟩	(4c)	
33221	🟧🟧🟩🟩🟦	(4b)	131322
32213	🟧🟩🟩🟦🟧	(4c)	
22131	🟩🟩🟦🟧🟦	(4c)	
21313	🟩🟦🟧🟦🟧	(4c)	
13132	🟦🟧🟦🟧🟩	(4c)	
31322	🟧🟦🟧🟩🟩	(4c)	
13221	🟦🟧🟩🟩🟦	(4a)	131322
32211	🟧🟩🟩🟦🟦	(4c)	
22113	🟩🟩🟦🟦🟧	(4a)	113322
21132	🟩🟦🟦🟧🟩	(4c)	
11323	🟦🟦🟧🟩🟧	(4c)	
13232	🟦🟧🟩🟧🟩	(4c)	
32321	🟧🟩🟧🟩🟦	(4c)	
23211	🟩🟧🟩🟦🟦	(4a)	113232
32112	🟧🟩🟦🟦🟩	(4c)	
21123	🟩🟦🟦🟩🟧	(4a)	112332
11232	🟦🟦🟩🟧🟩	(4c)	
12323	🟦🟩🟧🟩🟧	(4c)	
23231	🟩🟧🟩🟧🟦	(4b)	121323
32312	🟧🟩🟧🟦🟩	(4b)	123123
23123	🟩🟧🟦🟩🟧	(4c)	
31231	🟧🟦🟩🟧🟦	(4c)	
12312	🟦🟩🟧🟦🟩	(4a)	123123
23121	🟩🟧🟦🟩🟦	(4c)	
31213	🟧🟦🟩🟦🟧	(4c)	
12132	🟦🟩🟦🟧🟩	(4c)	
21323	🟩🟦🟧🟩🟧	(4c)	
13231	🟦🟧🟩🟧🟦	(4a)	121323
32311	🟧🟩🟧🟦🟦	(4b)	113223
23113	🟩🟧🟦🟦🟧	(4c)	
31132	🟧🟦🟦🟧🟩	(4c)	
11322	🟦🟦🟧🟩🟩	(4c)	
13223	🟦🟧🟩🟩🟧	(4c)	
32231	🟧🟩🟩🟧🟦	(4c)	
22311	🟩🟩🟧🟦🟦	(4a)	113223
23112	🟩🟧🟦🟦🟩	(4a)	112323
31122	🟧🟦🟦🟩🟩	(4c)	

Figure 6 The fixed-content universal cycle $\mathcal{V}(S)$ for $S = \{1,1,2,2,3,3\}$ in column-major order, as generated by the successor rule in (4). More specifically, the columns labeled β show successive states of the universal cycle, and any single column (e.g., b_1 or b_5) provides the universal cycle. Each state is a member of $\boldsymbol{Short}(S)$ and hence is shorthand for a member of $\boldsymbol{Perm}(S)$. The columns labeled *case* provide the symbol following β in the universal cycle; the next symbol is β's first symbol b_1 when (4a) or (4b) applies, and is β's missing symbol $z = b_6$ when (4c) applies. For example, the first state $\beta = 11223$ has case (4c) applied. In other words, $\beta z = 112233$ appears in the universal cycle (as opposed to $\beta b_1 = 112231$). As a check, note that this next symbol 3 is the final symbol in the next state 12233. When (4b) applies, the $\boldsymbol{N}(S)$ column provides the necklace $h(zb_1b_2\cdots b_{n-1})$, and these cases correspond to going downward in the first-inversion tree (see Figure 5b). Similarly, when (4a) applies, the $\boldsymbol{N}(S)$ column provides the necklace $h(b_1zb_2\cdots b_{n-1})$, and these cases correspond to going upward in the first-inversion tree. In both of these cases, the necklace comes from the child of the parent-child conjugate pair. For example, 121233 appears twice in the $\boldsymbol{N}(S)$ column, first going downward from the root 112233 via state 22331, and then going upward to the root 112233 via state 12331 (where 22331 and 12331 are conjugates of each other). The Gray code for $\boldsymbol{Perm}(S)$ using two operations, as generated by (5), can be obtained from this figure as follows: Add the missing symbol to each β, and map the cases (4a)–(4b) to (5a), and (4c) to (5b). For example, $112233 \in \boldsymbol{Perm}(S)$ is followed by $122331 \in \boldsymbol{Perm}(S)$ using $\mathsf{shift}(1,n) = \mathsf{shift}(1,6)$ in this order because (4c) maps to (5b). Similarly, a minimum-weight binary representation is obtained by using 1 for each (4a) or (4b), and 0 for each (4c).

Shorthand	Shift Gray code	Shift index
112	1123	(4)
123	1231	(4)
231	2311	(3)
312	3121	(4)
121	1213	(4)
213	2131	(4)
131	1312	(3)
311	3112	(3)
113	1132	(4)
132	1321	(4)
321	3211	(4)
211	2113	(3)

Note that the first symbol is shifted in to the $n{-}1$st position exactly $2|\boldsymbol{N}(S)-1|$ times.

4 Necklace Concatenations feat. Cool-lex Order

In this section we start by providing a brief background of fixed-content necklaces. We introduce cool-lex order for $\boldsymbol{Perm}(S)$ and provide a successor rule to produce the corresponding listing; a special focus is given to $\mathbf{N}(S)$. We then describe a known necklace concatenation approach that is applied to construct a universal cycle for $\boldsymbol{Short}(S)$. We conclude by providing a recursive description of cool-lex and use it to efficiently list $\boldsymbol{N}(S)$ in cool-lex order.

4.1 Necklaces with Fixed-Content

Let $\alpha = a_1a_2\cdots a_n$ be a string. Let α^t denote the string composed of t copies of α. The *period* of α is the smallest value p such that $\alpha = (a_1\cdots a_p)^t$ for some integer t. Let $\text{ap}(\alpha) = a_1\cdots a_p$ where p is the period of α; we say $\text{ap}(\alpha)$ is the *aperiodic prefix* of α. If α has period n we say it is *aperiodic*; otherwise we say it is *periodic*.

The number of fixed-content necklaces in $\boldsymbol{N}(S)$ can be deduced using Pólya theory as discussed in [21]. In the following formula, it is assumed that the content S is composed of $n_i \geq 1$ occurrences of each symbol i, $|S| = n$, and $k \geq 1$:

$$|\boldsymbol{N}(S)| \quad = \quad \frac{1}{n}\sum_{j|gcd(n_1,n_2,\ldots,n_k)} \phi(j)\frac{(n/j)!}{(n_1/j)!\cdots(n_k/j)!} \tag{6}$$

where Euler's totient function $\phi(j)$ denotes the number of positive integers less than or equal to j that are relatively prime to j.

There exists a $O(1)$-amortized time algorithm to list $\boldsymbol{N}(S)$ in reverse lexicographic order [41].

4.2 Cool-lex Order

Cool-lex order for fixed-content strings was introduced in [52]. In the binary case, when $k = 2$, cool-lex order has been well-studied under two natural equivalences [34, 35, 42]. When extending cool-lex to fixed-content strings where $k > 2$, there are a number of equivalent ways to present the ordering. The presentation we give here differs from the original presentation in [52]. In particular, we consider the longest non-decreasing prefix instead of the longest non-increasing prefix of the strings in question. Cool-lex order for $\boldsymbol{Perm}(S)$ is a *shift Gray code*, where successive strings differ by the shift of a single symbol. If $\alpha = a_1a_2\cdots a_n \in \boldsymbol{Perm}(S)$, then recall that

$$\mathsf{shift}_\alpha(t,s) = a_1\cdots a_{s-1}a_ta_sa_{s+1}\cdots a_{t-1}a_{t+1}\cdots a_n,$$

denotes the operation that shifts a_t into position s. This operation can be implemented in constant time by using a doubly-linked list data structure, so long as pointers to the symbol and position are provided. Cool-lex order provides a prefix-shift Gray code for $\boldsymbol{Perm}(S)$ meaning that successive strings differ by a single prefix-shift corresponding to the operation $\mathsf{shift}_\alpha(t,1)$. Moreover, the next value of t can be updated in constant time after each shift. The ordering is also *cyclic* because a prefix-shift transforms the last string in the order into the first. As an example, the set $\boldsymbol{Perm}(\{1,1,2,2,3,3\})$ is listed in cool-lex order on the left side of Figure 7.

One of the most notable features of cool-lex order is that it has a simple *successor rule*; the prefix-shift that creates the next fixed-content string in the order can be specified by the following rule.

Cool-lex Successor Rule for Fixed-Content Strings

Let $\alpha = a_1a_2\cdots a_n \in \boldsymbol{Perm}(S)$ and let j denote the length of α's longest non-decreasing prefix. The string following α in cool-lex order, denoted $\mathsf{next}(\alpha)$, is obtained from α by the prefix-shift in the following cumulative cases

$$\mathsf{next}(\alpha) = \begin{cases} \mathsf{shift}_\alpha(j,1) & \text{if } j = n & \text{(7a)}\\ \mathsf{shift}_\alpha(j{+}1,1) & \text{if } j = n-1 \text{ or } a_j > a_{j+2} & \text{(7b)}\\ \mathsf{shift}_\alpha(j{+}2,1) & \text{otherwise.} & \text{(7c)} \end{cases}$$

Another benefit of cool-lex order is that its relative order provides shift Gray codes for numerous interesting subsets of $\boldsymbol{Perm}(S)$. This phenomenon was discussed in the binary case in [34], and more generally in [53]. In particular, this occurs for necklaces, as illustrated in Figure 7 for $S = \{1,1,2,2,3,3\}$. By adapting the techniques from [34, 53], we obtain the following successor rule for fixed-content necklaces[5].

Cool-lex Successor Rule for Fixed-Content Necklaces

Let $\alpha = a_1a_2\cdots a_n \in \boldsymbol{N}(S)$ and let j denote the length of α's longest non-decreasing prefix. When $j < n-1$, let $\alpha' = a_1\cdots a_ja_{j+2}a_{j+1}a_{j+3}\cdots a_n$; it is α with a_{j+1} and a_{j+2} transposed. The necklace following α in cool-lex order, denoted $\mathsf{next}(\alpha)$, is obtained from α by the shift in the following cumulative cases

$$\mathsf{next}(\alpha) = \begin{cases} \mathsf{shift}_\alpha(j,s) & \text{if } j = n & \text{(8a)}\\ \mathsf{shift}_\alpha(j{+}1,s) & \text{if } j = n{-}1 \text{ or } a_j > a_{j+2} \text{ or } \alpha' \notin \boldsymbol{N}(S) & \text{(8b)}\\ \mathsf{shift}_\alpha(j{+}2,s) & \text{otherwise,} & \text{(8c)} \end{cases}$$

where s is the smallest index such that the result of shifting the specified element yields a necklace.

[5] In this presentation, we use the lexicographically smallest representative for necklaces rather than the lexicographically largest.

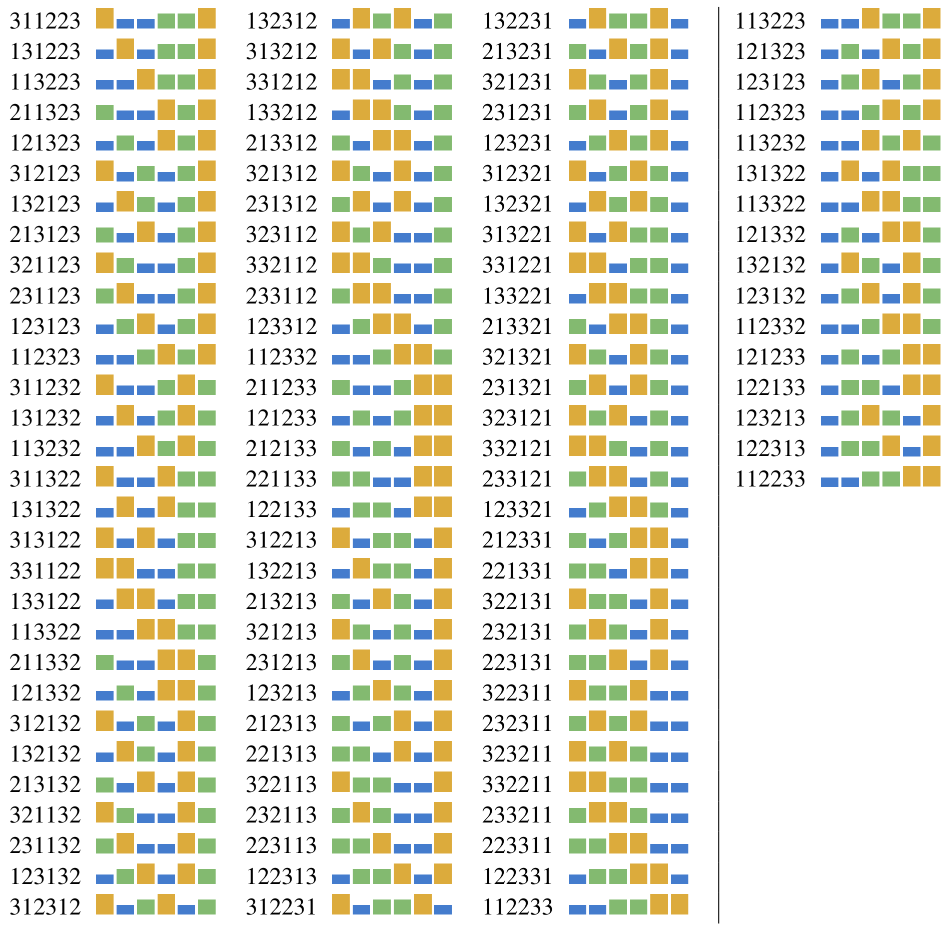


Figure 7 Cool-lex order for the strings with content $S = \{1, 1, 2, 2, 3, 3\}$ (i.e., $\boldsymbol{Perm}(S)$) appear to the left of the vertical line in column-major order, as generated by the successor rule in (7). Observe that each of these strings is obtained from the previous by a prefix-shift (i.e., $\mathsf{shift}(i, 1)$ for some $i > 1$). For example, the third string $113\underline{2}23$ is transformed into the fourth string $\underline{2}11323$ by moving the underlined symbol to the left into the first position (or equivalently by rotating the prefix $113\underline{2}$ one position to the right to obtain $\underline{2}113$). The order is also cyclic in this regard, since the last string is transformed into the first by a prefix-shift. The column to the right of the vertical line illustrates the necklaces with content S (i.e., $\boldsymbol{N}(S)$) as they appear in cool-lex order. Observe that each necklace is obtained from the previous by a shift given by (8). For example, the first necklace $113\underline{2}23$ is transformed into the second necklace $1\underline{2}1323$ by moving the underlined symbol two positions to the left (or equivalently by rotating the substring $13\underline{2}$ one position to the right to obtain $\underline{2}13$). The order is again cyclic in this regard, since the last string is transformed into the first by a shift. The fixed-content universal cycle $\mathcal{U}(S)$ is obtained by reversing the order of these necklaces, and concatenating their aperiodic prefixes, as shown in Figure 9. In particular, note that 112233 is last here, and first in Figure 9.

Example 8 Consider the necklace $\alpha = a_1a_2a_3a_4a_5a_6 = 122133$ with longest non-decreasing prefix of length $j = 3$. Since $a_3 < a_5$ and 122313 is a necklace, $\mathsf{next}(\alpha) = \mathsf{lshift}_\alpha(j+2)$. We can determine the result of $\mathsf{lshift}_\alpha(5)$ by bubbling the symbol $a_5 = 3$ to the left, starting from α, as follows:

$$\begin{aligned}
a_1a_2a_3a_4a_5a_6 &= 122133 \text{ is a necklace;}\\
a_1a_2a_3a_5a_4a_6 &= 122313 \text{ is a necklace;}\\
a_1a_2a_5a_3a_4a_6 &= 123213 \text{ is a necklace;}\\
a_1a_5a_2a_3a_4a_6 &= 132213 \text{ is not a necklace;}\\
a_5a_1a_2a_3a_4a_6 &= 312213 \text{ is not a necklace.}
\end{aligned}$$

The result is the last necklace in this list. Hence, $\mathsf{lshift}_\alpha(5) = a_1a_2a_5a_3a_4a_6 = 123213$.

Observe from the previous example, that the necklaces that result from shifting the specified symbol to the left are all contiguous. This property is formalized in the upcoming Lemma 10. Its proof requires the following technical result.

▶ **Lemma 9.** *If $a_1a_2\cdots a_n$ is a necklace that contains a smallest index t such that $a_t > a_{t+1}$, then $a_1\cdots a_{t-1}a_{t+1}a_ta_{t+2}\cdots a_n$ is a necklace.*

Proof. Let $\beta = b_1b_2\cdots b_n = a_1\cdots a_{t-1}a_{t+1}a_ta_{t+2}\cdots a_n$. Let β_j denote the rotation of β starting at b_j and let α_j denote the rotation of $\alpha = a_1a_2\cdots a_n$ starting at a_j. If $\beta \le \beta_j$ for each $2 \le j \le n$, then β is a necklace. Since α is a necklace, each $a_i \ge a_1$ and thus each $b_i \ge b_1$. Since $b_1\cdots b_{t-1}$ is non-decreasing it is straightforward to observe that $\beta_j > \beta$ for $2 \le j \le t+1$. Now consider the prefix of length t for β_j where $t+2 \le j \le n$. This prefix is the same as the length t prefix of α_j. If this prefix is less than or equal to $b_1\cdots b_t$, then it must be strictly less than $a_1\cdots a_t$ since $a_t > b_t$. But this contradicts the fact that α is a necklace. Thus this prefix must be strictly greater than $b_1\cdots b_t$. Thus $\beta_j \ge \beta$ for each $2 \le j \le n$ and hence β is a necklace. ◀

Given $\alpha = a_1a_2\cdots a_n \in \boldsymbol{N}(S)$, then from (8), let t correspond to the index of the element shifted to position s so

$$\mathsf{next}(\alpha) = a_1\cdots a_{s-1}a_ta_s\cdots a_{t-1}a_{t+1}\cdots a_n.$$

Using this notation, the following result illustrates the "bubble property" that necklaces have with respect to cool-lex order.

▶ **Lemma 10.** *For $s \le i < t$, $a_1\cdots a_{i-1}a_ta_i\cdots a_{t-1}a_{t+1}\cdots a_n$ is a necklace.*

Proof. When $i = s$, $\mathsf{next}(\alpha) = a_1\cdots a_{s-1}a_ta_s\cdots a_{t-1}a_{t+1}\cdots a_n$ is a necklace by definition. For $i > s$, we step through the cases of (8) and the possible values of t, recalling that j is the length of the longest non-decreasing prefix of α. If $|\boldsymbol{N}(S)| = 1$, then the necklace is either 1^n or $1^{n-1}2$, and the result clearly holds as part of case (8a). Otherwise there are at least two symbols that are not 1. Case (8a) implies that α is non-decreasing and $t = n$. From the definition of a necklace, it is easy to observe that $s = \lceil n_1/2 \rceil + 1$, where n_1 is the number of occurrences of the symbol 1. Moreover, $\mathsf{shift}_\alpha(n,i)$ is a necklace for all $s \le i < n$. Otherwise, if $t = j+1$ (from case (8b)) then by Lemma 9, $\mathsf{shift}_\alpha(t, t-1)$ is a necklace; if $t = j+2$ (from case (8c)) then the successor-rule itself requires $\mathsf{shift}_\alpha(t, t-1)$ is a necklace. Now, suppose there exists some r, where $s < r < t-1$, such that $\sigma = d_1\cdots d_n = \mathsf{shift}_\alpha(t, r) = a_1\cdots a_{r-1}a_ta_r\cdots a_{t-1}a_{t+1}\cdots a_n$ is not a necklace. Since $a_1 \cdots a_{t-2}$ is non-decreasing, Lemma 9 implies that $d_1 \cdots d_r$ is non-decreasing. Thus, by the definition of a necklace, there is some suffix σ' of $d_{r+1} \cdots d_n$ that is less than or equal to $d_1 \cdots d_r$ (namely, a prefix of a rotation that is a necklace). If $a_t > d_s$, then since $\mathsf{next}(\alpha)$ also has suffix σ', $a_1 \cdots a_{s-1} a_t$ is greater than σ', contradicting the fact that $\mathsf{next}(\alpha)$ is a necklace. Otherwise it must be that each element in $a_s \cdots a_{r-1}$ is a_t and hence $\sigma = \mathsf{next}(\alpha)$ which we already stated is a necklace, a contradiction. ◀

Since $O(n)$ time is sufficient for testing whether or not a string is a necklace [2], the necklace successor rule runs in $O(n^2)$ time. In Section 4.4, a recursive description of cool-lex order is provided. When focusing on necklaces, an optimization allows $\boldsymbol{N}(S)$ to be listed in cool-lex order in $O(n)$-amortized time per necklace.

4.3 Necklace Concatenation Construction

As mentioned in Section 1.5, the most well-known de Bruijn sequence is the so-called grand-daddy de Bruijn sequence; it is the lexicographically smallest k-ary de Bruijn sequence of order n. It can be generated very elegantly using an approach that is often referred to as the *FKM construction* or *FKM algorithm*, due to its discoverers [15, 16]. As discussed in [35], the authors of this article prefer to describe the construction using a slightly different approach called the *necklace-prefix algorithm*. The difference between the algorithms is that the former uses an ordering of *Lyndon words* (i.e., aperiodic necklaces) whose length divides n, while the latter uses an order of necklaces that is then reduced to the same set of Lyndon words[6]. For example, the FKM algorithm is based on column (b) of Figure 8, while the necklace-prefix algorithm uses column (a). When using lexicographic order, the resulting constructions are identical, but this is not true for other orders. For example, the two concatenation schemes give different results when using co-lexicographic order (which orders strings from right-to-left), with the necklace-prefix algorithm creating the grandmama de Bruijn sequence [12] and the FKM algorithm not working. Similarly, we use the necklace-prefix algorithm when working with cool-lex order, since cool-lex order is only defined for strings of the same length.

Formally, the *necklace-prefix algorithm* takes an order of strings, filters out the non-necklaces, reduces the remaining necklaces to their aperiodic prefix, and concatenates the prefixes. Amazingly, the granddaddy de Bruijn sequence is created by applying the necklace-prefix algorithm to the k-ary strings of length n in lexicographic order. This is illustrated in Figure 8 for $n = 6$ and $k = 2$. The approach has been generalized to other sets in [47].

Unfortunately, the magic runs out when we consider fixed-content strings, even in their shorthand representatives. As an illustration, note that the lexicographic order of necklaces with content $S = \{1, 1, 2, 2, 3, 3\}$ places the following necklaces consecutively,

$$\ldots 113322, 121233, \ldots,$$

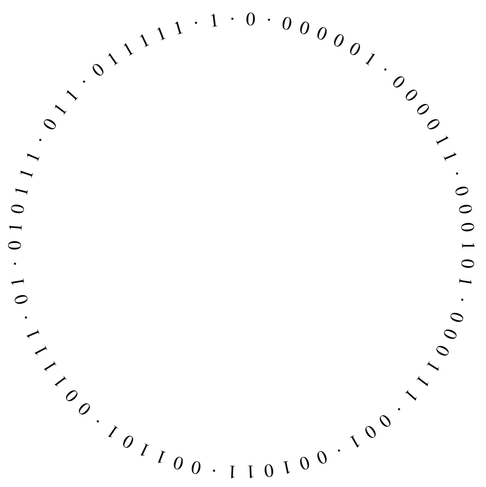
and so, the necklace-prefix algorithm generates $\cdots 1133\mathbf{22121}233 \cdots$. The bold substring of length $n-1 = 5$ is not shorthand for a string with the content S because it has too many 2's. The cause of the issue is also clear: The leftmost 2 moves several positions to the left from $1133\underline{2}2$ to $1\underline{2}1233$. This issue leads us to instead use *reverse* cool-lex order, since this will ensure that individual symbols move at most one position to the left between successive necklaces.

[6] Note that the aperiodic prefix of a necklace of length n is a Lyndon word whose length divides n.

lexicographic order

necklaces	aperiodic prefixes
000000	0
000001	000001
000011	000011
000101	000101
000111	000111
001001	001
001011	001011
001101	001101
001111	001111
010101	01
010111	010111
011011	011
011111	011111
111111	1
(a)	(b)

grand-daddy de Bruijn sequence $\mathcal{G}_2(6)$



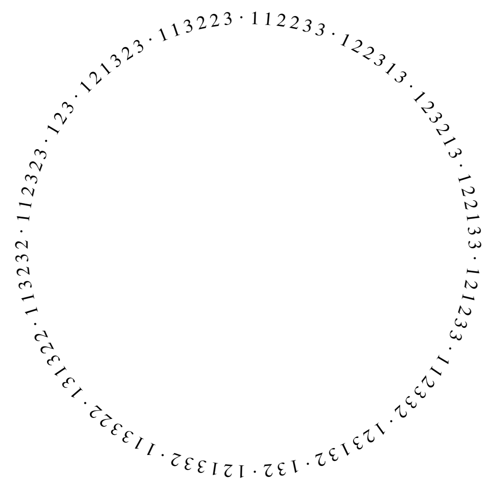
(c)

Figure 8 The grand-daddy de Bruijn sequence $\mathcal{G}_k(n)$ for $n = 6$ and $k = 2$ is constructed by the necklace-prefix algorithm applied to the binary strings of length 6. The algorithm starts with the lexicographic order of binary strings of length 6 (which are not shown), then reduces the order to the necklaces in column (a), and their aperiodic prefixes in column (b), and concatenates these prefixes to get the grand-daddy de Bruijn sequence in (c). The FKM algorithm yields the same concatenation directly from column (b), since it contains the Lyndon words of length 1, 2, and 3 (i.e., the divisors of $n = 6$) in lexicographic order.

cool-lex order

necklaces $\boldsymbol{N}(S)$	aperiodic prefixes
112233	112233
122313	122313
123213	123213
122133	122133
121233	121233
112332	112332
123132	123132
132132	132
121332	121332
113322	113322
131322	131322
113232	113232
112323	112323
123123	123
121323	121323
113223	113223
(a)	(b)

cool-daddy fixed-content universal cycle $\mathcal{U}(S)$



(c)

Figure 9 Our cool-daddy fixed-content universal cycle $\mathcal{U}(S)$ for content $S = \{1, 1, 2, 2, 3, 3\}$. The cycle uses the shorthand representation, and is constructed using the necklace-prefix algorithm on reverse cool-lex order. The fixed-content necklaces over S are given in reverse cool-lex order in column (a), they are reduced to their aperiodic prefix in column (b), and their concatenation gives the universal cycle in column (c).

Let $\alpha_1, \alpha_2, \ldots, \alpha_m$ denote the necklaces $\boldsymbol{N}(S)$ listed in *reverse* cool-lex order. Amazingly, by applying the necklace-prefix algorithm outlined with respect to cool-lex order, we obtain a universal cycle for $\boldsymbol{Short}(S)$. Let

$$\mathcal{U}(S) = \text{ap}(\alpha_1)\,\text{ap}(\alpha_2)\cdots\text{ap}(\alpha_m).$$

An example of $\mathcal{U}(S)$ is provided in Figure 9 for $S = \{1,1,2,2,3,3\}$.

▶ **Theorem 11.** *$\mathcal{U}(S)$ is universal cycle for $\boldsymbol{Short}(S)$. Moreover, $\mathcal{U}(S) = \mathcal{V}(S)$.*

This concatenation construction does not attain the sufficient conditions provided in [17] for when concatenating smaller cycles yields a larger universal cycle. Instead, this theorem is proved in Section 5 by demonstrating that the symbol following a given length $n-1$ substring β in $\mathcal{U}(S)$ is given by the successor-rule $g(\beta)$ used to generate $\mathcal{V}(S)$. By applying an efficient algorithm to list $\boldsymbol{N}(S)$ in cool-lex order, as presented in the next section, we obtain the following result.

▶ **Theorem 12.** *The reversal of $\mathcal{U}(S)$ can be generated in $O(1)$-amortized time per symbol using $O(n)$ space.*

4.4 Recursive Generation of Fixed-Content Necklaces in Cool-lex

In [52], a recursive description is given to list all strings with fixed-content S in cool-lex order. In that description, the focus is on strings in reverse lexicographic order, whereas, we will focus on lexicographic order. In this section, we restate this recurrence using the original terminology and then apply it to generate the necklaces $\boldsymbol{N}(S)$ in cool-lex order.

Recall $\text{tail}(S)$ denotes the unique non-decreasing string (and necklace) with content S. A *scut*[7] of S is any non-decreasing string α composed of *some* of the elements of S such that α is not a suffix of $tail(S)$, but every proper suffix of α is a suffix of $tail(S)$. Let $\alpha_i(S)$ (or simply, α_i) denote the i-th scut of S when the scuts are listed in decreasing order of the first symbol, then by decreasing length. Let R_i denote the multiset S with the content of $\alpha_i(S)$ removed.

Example 9 Consider $S = \{1,1,2,2,3,3\}$. Then $tail(S) = 112233$ and the scuts of S in decreasing order of the first symbol, then decreasing length, are:

$23,\ 2,\ 1233,\ 133,\ 13,\ 1.$

Note $\alpha_4(S) = 133$ and $R_4 = \{1,1,2,2,3,3\} \setminus \{1,3,3\} = \{1,2,2\}$.

If S is a multiset with j scuts, then the following recurrence $\boldsymbol{\mathcal{C}}(S,\gamma)$ (simplified from Definition 2.4 in [52]) produces a listing for all strings of the form $\beta\gamma$ where β has content S as they appear in cool-lex order:

$$\boldsymbol{\mathcal{C}}(S,\gamma) = \boldsymbol{\mathcal{C}}(R_1,\alpha_1\gamma), \boldsymbol{\mathcal{C}}(R_2,\alpha_2\gamma), \ldots, \boldsymbol{\mathcal{C}}(R_j,\alpha_j\gamma), tail(S)\gamma.$$

Note that $\boldsymbol{\mathcal{C}}(S,\epsilon)$ will produce a listing of all strings with fixed-content S. Recall Figure 7 illustrating the cool-lex order for $\boldsymbol{Perm}(\{1,1,2,2,3,3\})$. This is the same listing generated by

7 In nature, a scut is a short tail. Here, it is a suffix of $tail(S)$ with a small symbol prepended.

Algorithm 1 Recursive algorithm to list the necklaces $\mathbf{N}(S)$ as they appear in cool-lex order. The string $a_1a_2\cdots a_n$ is intialized to $tail(S)$, and the initial call is COOL(n).

```
1:  procedure COOL( $t$ )
2:      $i \leftarrow t$
3:      while $a_i \neq a_1$ do
4:          while $a_i = a_{i-1}$ do  $i \leftarrow i-1$
5:          for $j$ from $i$ to $t$ do
6:              SWAP($j-1, j$)
7:              if $a_1a_2\cdots a_n$ is a necklace then COOL($j-1$)
8:          for $j$ from $t$ down to $i$ do  SWAP($j-1, j$)
9:          $i \leftarrow i-1$
10:     VISIT( )
```

$\mathcal{C}(\{1,1,2,2,3,3\},\epsilon)$. In particular observe that the strings are ordered by suffixes corresponding to the scuts: 23, 2, 1233, 133, 13, 1.

We now focus on how to modify this recurrence to list $\mathbf{N}(S)$ as they appear in cool-lex order.

▶ **Lemma 13.** *$\mathcal{C}(S,\gamma)$ contains a necklace if and only if $tail(S)\gamma$ is a necklace.*

Proof. ($\Leftarrow$) $tail(S)\gamma$ is in $\mathcal{C}(S,\gamma)$ by definition. Thus if $tail(S)\gamma$ is a necklace then $\mathcal{C}(S,\gamma)$ contains a necklace. ($\Rightarrow$) If $\mathcal{C}(S,\gamma)$ contains necklace then it must be of the form $\lambda\gamma$ where λ has content S. If $\lambda = tail(S)$, then we are done. Otherwise, repeated application of Lemma 9 implies that $tail(S)\gamma$ is a necklace. ◀

Based on Lemma 13, the recurrence $\mathcal{C}(S,\gamma)$ can be updated to list only the necklaces as follows (where $\langle\,\rangle$ denotes an empty list).

$$\mathcal{N}(S,\gamma) = \begin{cases} \langle\,\rangle & \text{if } tail(S)\gamma \text{ is not a necklace;} \\ \mathcal{N}(R_1,\alpha_1\gamma),\ldots,\mathcal{N}(R_j,\alpha_j\gamma), tail(S)\gamma & \text{otherwise,} \end{cases}$$

Note that $\mathcal{N}(S,\epsilon)$ will produce a listing of all necklaces with fixed-content S as they appear in $\mathcal{C}(S,\epsilon)$. Recall from (8), the corresponding successor-rule which implies that successive necklaces in this ordering differ by a shift.

The function COOL(t) in Algorithm 1 implements the recurrence for $\mathcal{N}(S,\gamma)$. Given content S, by initializing the global string $a_1a_2\cdots a_n$ to $tail(S)$, the initial call COOL(n) generates the necklaces $\mathbf{N}(S)$ in cool-lex order. The parameter t passed in the function COOL(t) indicates how the string $a_1a_2\cdots a_n$ is partitioned into the two pieces based on $\mathcal{N}(S',\gamma)$: $a_1a_2\cdots a_t = tail(S')$ and $a_{t+1}\cdots a_n = \gamma$. Each call COOL($t$) corresponding to $\mathcal{N}(S',\gamma)$ iterates through the scuts of S' in the proper order. This is done by scanning $tail(S') = a_1\cdots a_t$ from right to left until we reach an index i where $a_i \neq a_{i-1}$ (Line 4). To produce all scuts starting with a_{i-1}, and their corresponding recursive calls if a necklace can be produced, we iteratively shift this symbol through positions $i, i+1, \ldots, t$ obtaining a new scut for each swap (Lines 5-7). Once all scuts starting with a_{i-1} have been processed we restore $a_1\cdots a_t$ to $tail(S')$ (Line 8). We repeat this approach by continuing to traverse $tail(S)$ from right to left until we reach a symbol that is the same as a_1 (Line 3). The function VISIT() outputs the string $a_1a_2\cdots a_n$, and the function SWAP(i,j) swaps the symbols at index i and j in $a_1a_2\cdots a_n$.

When analyzing this algorithm, if every string tested in Line 7 was a necklace, then the work done by each necklace test can be assigned to the following recursive call. Since each recursive call generates at least one necklace, and since the necklace testing can be done in $O(n)$-time [2], the overall algorithm runs in $O(n)$-amortized time per necklace. However, within each recursive call, there can be a number of negative necklace tests. For instance, consider the string $\alpha = 112233112233$ and the call to COOL(6). This results in necklace tests for the following 6 strings, none of which are necklaces since the rotation starting with the suffix 112233 is smaller than string in question:

$$112323112233, 112332112233, 121233112233,$$
$$122133112233, 122313112233, 122331112233.$$

Fortunately there exists a simple optimization: once a string tested on Line 7 is **not** a necklace, then by Lemma 10 none of the following strings tested will be either. This optimization can be applied to COOL(t) by replacing Line 7 with the following fragment:

```
if $a_1 a_2 \cdots a_n$ is a necklace then COOL($j-1$)
else
    for $s$ from $j$ down to $i$ do   SWAP($s-1, s$)
    VISIT( )
    return
```

This optimization ensures that at most one necklace test is negative per recursive call.

▶ **Theorem 14.** *If $a_1 a_2 \cdots a_n$ is initialized to tail(S), then a call to the optimized* COOL(n) *lists the necklaces $\boldsymbol{N}(S)$ in cool-lex order in $O(n)$-amortized time per string.*

In the binary case when $k = 2$, $\boldsymbol{N}(S)$ can be generated in $O(1)$-amortized time [42].

4.4.1 Application: Efficient Construction of $\mathcal{U}(S)$ in Reverse

To construct the *reverse* of the universal cycle $\mathcal{U}(S)$, which itself is a fixed-content universal cycle over S, we can directly apply the optimized Algorithm 1 to list $\boldsymbol{N}(S)$ in cool-lex order with a simple modification. Instead of outputting the current necklace $\alpha = a_1 a_2 \cdots a_n$, the function VISIT( )

- ▷ determines the period p of α and then
- ▷ outputs $a_p a_{p-1} \cdots a_1$.

Since the aperiodic prefix of α can be determined in $O(n)$ (see [2]), the modified algorithm still runs in $O(n)$-amortized time per necklace. Since the total length of $\mathcal{U}(S)$ is proportional to $n|\boldsymbol{N}(S)|$ (see Section 5 in [41] which implies $|\mathcal{U}(S)| \geq n|\boldsymbol{N}(S)|/2$) we obtain the result previously stated in Theorem 12.

5 Proof of Theorem 11

If $|\boldsymbol{N}(S)| = 1$, then the content of S is either all 1s, or all 1s and a single 2; the necklaces are 1^n and $1^{n-1}2$, respectively. In these cases we clearly have $\mathcal{U}(S) = \mathcal{V}(S)$. Otherwise, let $\alpha_1, \alpha_2, \ldots, \alpha_m$ denote the necklaces of $\boldsymbol{N}(S)$ listed in *reverse* cool-lex order, allowing $\alpha_{m+1} = \alpha_1$. Recall that

$$\mathcal{U}(S) = \text{ap}(\alpha_1)\,\text{ap}(\alpha_2)\cdots\text{ap}(\alpha_m).$$

We focus on consecutive necklaces α_i and α_{i+1}, where $1 \leq i \leq m$. Let $\alpha_i = c_1 c_2 \cdots c_n$ and let $\alpha_{i+1} = a_1 a_2 \cdots a_n$. From (8), there exists some s and t such that

$$\alpha_i = a_1 \cdots a_{s-1} a_t a_s \cdots a_{t-1} a_{t+1} \cdots a_n.$$

Applying this notation, we obtain the following result.

▶ **Lemma 15.** *The length $n-1$ suffix of* $\mathrm{ap}(\alpha_i)\,\mathrm{ap}(\alpha_{i+1})$ *is the same as the length $n-1$ suffix of* α_{i+1}.

Proof. Let p be the period of α_{i+1}. If $p = n$, the result is trivial. Otherwise α_{i+1} is periodic, and from the definition of a necklace and the assumptions on the content, $a_1 = a_{p+1} = 1$ and $a_p > 1$. Since α_{i+1} is periodic it is a simple exercise to demonstrate that α_i is aperiodic since it is the cool-lex successor of α_{i+1}. Let $a_j a_{j+1}$ be the leftmost inversion in α_{i+1}. Clearly $j \leq p$. If $j < p$, then $t \leq p+1$ and the result follows. If $j = p$ then $a_1 \cdots a_p$ is non-decreasing. We consider two cases based on (8b). If $a_p > a_{p+2} = a_2$, then $t = p+1$ and again the result follows. Otherwise, it must be that $a_2 \cdots a_p$ are all the same symbol, namely 2. Swapping $a_{p+1} = 1$ and $a_{p+2} = 2$ in α_{i+1} to obtain γ does not yield a necklace because the rotation starting from position $p+2$ in γ will be lexicographically smaller than γ. Thus based on (8b), $t = p+1$, and again the result follows. ◀

We now prove that $\mathcal{U}(S) = \mathcal{V}(S)$. Let $\beta = b_1 \cdots b_{n-1}$ be a substring of $\mathcal{U}(S)$. We show (i) β is an element of $\boldsymbol{Short}(S)$ with missing symbol z and (ii) the symbol following β in $\mathcal{U}(S)$ is $g(\beta)$, and hence one of z or b_1.

Suppose β is completely contained in some $\mathrm{ap}(\alpha_i)$. Then clearly β is an element of $\boldsymbol{Short}(S)$ and $\mathrm{ap}(\alpha_i) = \alpha_i$, which means α_i is an aperiodic necklace. Thus $\beta z = \alpha_i$ or $z\beta = \alpha_i$. Since each necklace in $\boldsymbol{N}(S)$ begins with 1, the symbol following β in each case is the missing symbol z. In both cases $h(z\beta)$ is not a necklace since it is a proper rotation of the aperiodic necklace α_i. Thus by Corollary 6, β is not in $\boldsymbol{X}(S)$ and hence $g(\beta) = z$. For the remaining cases, $\beta = \sigma_1\sigma_2$ where σ_2 is a non-empty prefix of some $\mathrm{ap}(\alpha_{i+1})$, and by applying Lemma 15, σ_1 is a non-empty suffix of α_i. Let $|\sigma_2| = x$ and thus $\sigma_2 = a_1 a_1 \cdots a_x$ and $\sigma_1 = c_{x+2} \cdots c_n$. Note $x < n-1$ and the symbol following β in $\mathcal{U}(S)$ is a_{x+1}. Let p denote the period of α_{i+1}.

Case 1: $\sigma_2 = \mathrm{ap}(\alpha_{i+1})$. This means α_{i+1} is periodic, i.e., $\alpha_{i+1} = (a_1 \cdots a_p)^j$, for some $j > 1$. By Lemma 15, β is a suffix of α_{i+1} which means $z\beta = \alpha_{i+1}$. Both z and the symbol following β are 1 as they are each the first symbol of some necklace in $\boldsymbol{N}(S)$. If $a_1 \cdots a_p$ contains an inversion, then $h(z\beta)$ is rotation of α_{i+1} that must be lexicographically larger than α_{i+1} by the definition of p. Thus by Corollary 6, $g(\beta) = z$. Otherwise, suppose $a_1 \cdots a_p$ is non-decreasing. If $z = b_1$, $g(\beta) = z$ from (4). Otherwise $b_1 > z$. By the content assumptions, it must be that $b_1 = 2$. If $b_1 < b_{n-1}$, then $g(x) = z$ from (4). Otherwise, $a_1 \cdots a_p = 12^{p-1}$, and $h(b_1 z b_2 \cdots b_{n-1}) = 12^{p-1}212^{p-2}(a_1 \cdots a_p)^{j-2}$ is not a necklace. Again $g(x) = z$ from (4).

Case 2: σ_2 is a proper prefix of $\mathrm{ap}(\alpha_{i+1})$. Consider three cases for x, noting that $x < p$:

- (A): $t-1 < x$. In this case $\sigma_1 = a_{x+2} \cdots a_n$ and $\sigma_2 a_{x+1} \sigma_1 = \alpha_{i+1}$. Clearly, β is in $\boldsymbol{Short}(S)$ with missing symbol $z = a_{x+1}$. Since $t-1 < x < n-1$, by the definition of t and (8), σ_2 must contain an inversion. Since α_{i+1} is a necklace with period p, its rotation $h(z\beta)$ must be strictly larger than α_{i+1}, and hence is not a necklace. Thus, by Corollary 6, $g(\beta) = z$.
- (B): $s-1 \leq x < t-1$. In this case $\sigma_1 = a_{x+1} \cdots a_{t-1} a_{t+1} \cdots a_n$ and thus β is in $\boldsymbol{Short}(S)$ with missing symbol $z = a_t$. Note $a_{x+1} = b_1$. If $z = b_1$, then clearly $g(\beta) = b_1 = z$. Otherwise, by the definitions of s and t and Lemma 10, both $h(zb_1 \cdots b_{n-1})$ and $h(b_1 z b_2 \cdots b_{n-1})$ are necklaces. Also, by the definition of t, $a_{x+1} (= b_1)$ is greater than or equal to $a_x (= b_{n-1})$. Thus, the larger of z and b_1 is greater than or equal to b_{n-1}. Thus $g(\beta) = b_1$, from Lemma 5.
- (C): $x < s-1$. In this case $\sigma_2 = c_1 \cdots c_x$ and $\sigma_2 c_{x+1} \sigma_1 = \alpha_i$. Clearly, β is in $\boldsymbol{Short}(S)$ and $z = c_{x+1} = a_{x+1}$. By the definition of s, $\gamma = c_1 \cdots c_{s-2} c_s c_{s-1} c_{s+1} \cdots c_n$ is not a necklace. If

$x = s - 2$, then $\gamma = h(b_1 z b_2 \cdots b_{n-1})$ and by Corollary 6 β is not in $\boldsymbol{X}(S)$. Thus $g(\beta) = z$. Otherwise $x < s - 2$ and by the definition of t, $c_{x+1} \leq c_{x+2}$. If $c_{x+1} = c_{x+2} = b_1$, then $g(\beta) = z = b_1$. Otherwise it is a simple exercise to demonstrate that $h(b_1 z b_2 \cdots b_{n-1})$ is not a necklace since γ is not a necklace. Again, by Corollary 6, β is not in $\boldsymbol{X}(S)$ and thus $g(\beta) = z$.

For each case, we have demonstrated that β is in $\boldsymbol{Short}(S)$ and the symbol following β in $\mathcal{U}(S)$ is $g(\beta)$. Thus $\mathcal{U}(S) = \mathcal{V}(S)$, completing the proof of Theorem 11.

6 Final Remarks

In this paper we presented two algorithms that construct the first fixed-content universal cycle.

1. We developed a successor-rule based on the first-inversion tree of necklaces that runs in $O(n)$ time per symbol using $O(n)$ space.
2. We developed concatenation construction based on cool-lex order of fixed-content necklaces that runs in $O(1)$-amortized time per symbol using $O(n)$ space.

The first result provides a cyclic shift Gray code for multiset permutations (i.e., string with a given Parikh vector) in which the next multiset permutation is obtained by the shifting the first symbol into the last or second last position. The Gray code can be generated in $O(n)$-time per string, starting from any string in the set. The second result involved the creation of an $O(n)$-time per string shift Gray code algorithm for listing necklaces of fixed content in cool-lex order.

6.1 Additional Observations

We conclude with additional observations and avenues for future research, some of which are expanded upon online [40].

Cycle-Joining. The first-inversion tree swaps the leftmost inversion in a necklace, and paths to the root node (i.e., $\mathrm{tail}(S)$) resemble insertion sort or gnome sort [39]. Different fixed-content universal cycles can be created by using other trees. For example, one could instead swap rightmost inversions according to the *last-inversion tree*. Alternatively, one could focus on the smallest value that is not in sorted order. This change results in paths to the root that follow selection sort, and generalizes the *decrementing spanning tree* from [24] and its associated *bell ringer* universal cycle for permutations.

Feedback Functions. Our cycle-joining construction is based on the underlying feedback function $f(b_1 b_2 \cdots b_{n-1}) = z$, which returns the missing symbol z and creates necklace cycles. Instead, one could start with the feedback function $f(b_1 b_2 \cdots b_{n-1}) = b_1$, which creates initial cycles that are not necklace cycles. Universal cycles resulting from this feedback function would have maximum-weight rather than minimum-weight cycles.

With regard to the choice of feedback function, it is worth noting some connections between this article and foundational work in the area. In the special case when $k = 2$, recall that multiset permutations correspond to fixed-weight binary strings. In this case, the feedback function $f(\beta) = z$ corresponds to two of the "simple" feedback functions presented in [22, Ch. 7], depending on the weight w of the strings. The pure summing register (PSR) and the complementing summing register (CSR) apply feedback functions defined as follows, where the operator $\oplus$ denotes addition modulo 2:

$$\mathrm{PSR}(\beta) = b_1 \oplus b_2 \oplus \cdots \oplus b_{n-1} \text{ and } \mathrm{CSR}(\beta) = b_1 \oplus b_2 \oplus \cdots \oplus b_{n-1} \oplus 1.$$

If w is even, then $f(\beta) = \mathrm{PSR}(\beta)$; if w is odd, then $f(\beta) = \mathrm{CSR}(\beta)$. Cycle-joining using these feedback functions has been previously studied in [14]. As we proved in Theorem 11, they also are the underlying feedback function used in the constructions from [35] which were later applied to construct weight-range universal cycles in [46].

Shift Gray Codes and Applications. Our Gray codes using $\mathsf{shift}(1, n)$ and $\mathsf{shift}(1, n{-}1)$ can be used to optimize exhaustive computations for objective function focused on the (ordered) pairs of symbols in a string. For example, consider a directed traveling salesman problem, where each permutation of $\{1, 2, \ldots, n\}$ represents a Hamilton path in the graph, and each ordered pair of symbols represents a directed edge. Notice that the operation $\mathsf{shift}(1, n)$ changes only one edge in the associated path, while $\mathsf{shift}(1, n{-}1)$ changes two[8]. Thus, the cost of each successive paths can be updated in $O(1)$-time. These Gray codes are also helpful when ordering events with repetition (e.g., multiple deliveries along the same route in a stacker crane problem [1]).

Other interesting questions can be asked about the existence of Gray codes using various sets of strings and shifts. For example, there is no Gray code for $\boldsymbol{Perm}(S)$ using $\mathsf{shift}(n, 1)$, $\mathsf{shift}(1, n)$, and $\mathsf{shift}(1, 2)$, even for fixed-weight binary strings (i.e., $k = 2$) [5]. However, the latter two operations are sufficient for permutations (i.e., $n = k$) [44] (also see [38]). A specific open question is whether $\boldsymbol{Perm}(S)$ has a Gray code using $\mathsf{shift}(1, 2)$, $\mathsf{shift}(1, 3)$, and $\mathsf{shift}(1, n)$ (see [49] when $n = k$).

Encodings. Shorthand representation is not the only encoding of $\boldsymbol{Perm}(S)$ that could be considered for use in universal cycles. In particular, order isomorphism [28, 20], relaxed shorthand [55], and graphical representations [4] have all been considered for permutations.

Acknowledgements. We'd like to thank the anonymous referees for helpful comments and corrections.

Ethical Statement

The authors have no known conflicts of interest to declare.

The research of Joe Sawada is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) grant RGPIN-2018-04211.

References

1 T. Avila, A. Corberán, I. Plana, and J. M. Sanchis. The stacker crane problem and the directed general routing problem. *Networks*, 65(1):43–55, 2015.

2 K. S. Booth. Lexicographically least circular substrings. *Information Processing Letters*, 10(4/5):240–242, 1980.

3 G. Brockman, B. Kay, and E. E. Snively. On universal cycles of labeled graphs. *the electronic journal of combinatorics*, 17(R4):1, 2010.

4 A. Cantwell, J. Geraci, A. Godbole, and C. Padilla. Graph universal cycles of combinatorial objects. *Advances in Applied Mathematics*, 127:102166, 2021.

5 Y. Cheng. Generating combinations by three basic operations. *Journal of Computer Science and Technology*, 22(6):909–913, 2007.

6 F. Chung, P. Diaconis, and R. Graham. Universal cycles for combinatorial structures. *Discrete Mathematics*, 110(1):43–59, 1992.

7 R. C. Compton and S. Gill Williamson. Doubly adjacent gray codes for the symmetric group. *Linear and Multilinear Algebra*, 35(3-4):237–293, 1993.

[8] In contrast, the swaps used in the Steinhaus-Johnson-Trotter order [48] cause up to three edges to be changed.

8 P. F. Corbett. Rotator graphs: An efficient topology for point-to-point multiprocessor networks. *IEEE Transactions on Parallel & Distributed Systems*, 3(05):622–626, 1992.

9 N. G. de Bruijn. A combinatorial problem. In *Proc. Koninklijke Nederlandse Academie van Wetenschappen*, volume 49, pages 758–764, 1946.

10 N. G. de Bruijn. Acknowledgement of priority to c. flye sainte-marie on the counting of circular arrangements of 2^n zeros and ones that show each n-letter word exactly once. 1975.

11 P. Diaconis and R. L. Graham. Products of universal cycles, in *A Lifetime of Puzzles. E. Demaine, M. Demaine and T. Rodgers, eds., A K Peters/CRC Press*, pages 35–55, 2008.

12 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 Journal of Combinatorics*, 72:1–11, 2018.

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

14 T. Etzion and A. Lempel. Algorithms for the generation of full-length shift-register sequences. *IEEE Trans. Inform. Theory*, 30(3):480–484, 1984.

15 H. Fredricksen and I. Kessler. An algorithm for generating necklaces of beads in two colors. *Discrete Math.*, 61(2):181 – 188, 1986.

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

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

18 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.

19 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.

20 A. L. Gao, S. Kitaev, W. Steiner, and P. B. Zhang. On a greedy algorithm to construct universal cycles for permutations. *International Journal of Foundations of Computer Science*, 30(01):61–72, 2019.

21 E. N. Gilbert and J. Riordan. Symmetry types of periodic sequences. *Illinois J. Math.*, 5(4):657–665, 12 1961.

22 S. W. Golomb. *Shift register sequences: secure and limited-access code generators, efficiency code generators, prescribed property generators, mathematical models*. World Scientific, 3rd edition, 2017.

23 C. Hierholzer. Ueber die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen*, 6:30–32, 1873.

24 A. E. Holroyd, F. Ruskey, and A. Williams. Shorthand universal cycles for permutations. *Algorithmica*, 64(2):215–245, 2012.

25 V. Horan and G. Hurlbert. Universal cycles for weak orders. *SIAM J. Discrete Math.*, 27(3):1360–1371, 2013.

26 B. Jackson, B. Stevens, and G. Hurlbert. Research problems on Gray codes and universal cycles. *Discrete Mathematics*, 309(17):5341 – 5348, 2009.

27 B. W. Jackson. Universal cycles of k-subsets and k-permutations. *Discrete mathematics*, 117(1-3):141–150, 1993.

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

29 D. Knuth. *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*. Number pt. 1. Pearson Education, 2014.

30 A. Leitner and A. Godbole. Universal cycles of classes of restricted words. *Discrete Mathematics*, 310(23):3303 – 3309, 2010.

31 A. Leitner and A. Godbole. Universal cycles of classes of restricted words. *Discrete Mathematics*, 310(23):3303–3309, 2010.

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

33 R. A. Rankin. A campanological problem in group theory. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 44, pages 17–25. Cambridge University Press, 1948.

34 F. Ruskey, J. Sawada, and A. Williams. Binary bubble languages and cool-lex Gray codes. *Journal of Combinatorial Theory, Series A*, 119(1):155–169, 2012.

35 F. Ruskey, J. Sawada, and A. Williams. De Bruijn sequences for fixed-weight binary strings. *SIAM Journal on Discrete Mathematics*, 26(2):605–617, 2012.

36 F. Ruskey and A. Williams. The coolest way to generate combinations. *Discrete Mathematics*, 309(17):5305–5320, 2009.

37 F. Ruskey and A. Williams. An explicit universal cycle for the $(n-1)$-permutations of an n-set. *ACM Transactions on Algorithms (TALG)*, 6(3):45, 2010.

38 W. Rytter and W. Zuba. Syntactic view of sigma-tau generation of permutations. *Theoretical Computer Science*, 882:49–62, 2021.

39 H. Sarbazi-Azad. Stupid sort: A new sorting algorithm. *Newsletter of Computing Science Department, Univ. of Glasgow*, 599:4, 10 2000.

40 J. Sawada. De Bruijn Sequence and Universal Cycle Constructions. `http://debruijnsequence.org/`. Accessed: 2021-12-31.

41 J. Sawada. A fast algorithm to generate necklaces with fixed content. *Theoretical Computer Science*, 301(1):477–489, 2003.

42 J. Sawada and A. Williams. A Gray code for fixed-density necklaces and Lyndon words in constant amortized time. *Theoretical Computer Science*, 502:46–54, 2013. Generation of Combinatorial Structures.

43 J. Sawada and A. Williams. A Hamilton path for the sigma-tau problem. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 568–575. SIAM, 2018.

44 J. Sawada and A. Williams. Solving the sigma-tau problem. *ACM Transactions on Algorithms (TALG)*, 16(1):1–17, 2019.

45 J. Sawada and A. Williams. A universal cycle for strings with fixed-content (which are also known as multiset permutations). In *Workshop on Algorithms and Data Structures*, pages 599–612. Springer, 2021.

46 J. Sawada, A. Williams, and D. Wong. Universal cycles for weight-range binary strings. In *In: Lecroq T., Mouchard L. (eds) Combinatorial Algorithms (IWOCA 2013)*, pages 388–401. Springer, 2013.

47 J. Sawada, A. Williams, and D. Wong. Generalizing the classic greedy and necklace constructions of de Bruijn sequences and universal cycles. *Electron. J. Comb.*, 23(1):P1.24, 2016.

48 H. Steinhaus. *One hundred problems in elementary mathematics*. Courier Corporation, 1979.

49 B. Stevens and A. Williams. Hamilton cycles in restricted and incomplete rotator graphs. *J. Graph Algorithms Appl.*, 16(4):785–810, 2012.

50 R. G. Swan. A simple proof of rankin's campanological theorem. *The American mathematical monthly*, 106(2):159–161, 1999.

51 T. van Aardenne-Ehrenfest. Circuits and trees in oriented linear graphs. *Simon Stevin*, 28:203–217, 1951.

52 A. Williams. Loopless generation of multiset permutations using a constant number of variables by prefix shifts. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 987–996, USA, 2009. SIAM.

53 A. Williams. *Shift Gray codes*. PhD thesis, University of Victoria, 2009.

54 A. Williams. The greedy gray code algorithm. In *Workshop on Algorithms and Data Structures*, pages 525–536. Springer, 2013.

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

Appendix A - Successor-rule approach to construct $\mathcal{V}(S)$

```c
//-----------------------------------------------------------------------------
// SHORTHAND UNIVERSAL CYCLES FOR FIXED-CONTENT STRINGS IN O(n)-TIME PER SYMBOL
//-----------------------------------------------------------------------------
#include <stdio.h>
int N,K,a[100],b[100],total=0;

//-------------------------------------------------------------------------------------------
// Returns length of longest aperiodic prefix if b[1..n] is a necklace; return 0 otherwise
//-------------------------------------------------------------------------------------------
int IsNecklace(int b[], int n) {
    int i, p=1;

    for (i=2; i<=n; i++) {
        if (b[i-p] > b[i]) return 0;
        if (b[i-p] < b[i]) p = i;
    }
    if (n % p != 0) return 0;
    return p;
}
//---------------------------------
int Coollex(int z) {
    int j,t,count,b[100],c[100];

    // Set b[1..N] = z a[1..N-1], swapping the first two elements if not an inversion
    b[1] = z;
    for (j=1; j<N; j++) b[j+1] = a[j];
    if (b[1] < b[2]) { b[1] = b[2]; b[2] = z; }

    // Rotate the longest increasing suffix b[j..N] of b[3..N] to the front of b[1..N]
    j = N;
    while (b[j-1] <= b[j] && j > 3) j--;
    count = 1;
    for (t=j; t<=N; t++) c[count++] = b[t];
    for (t=1; t<=j; t++) c[count++] = b[t];

    if (b[1] >= b[N] && b[1] != b[2] && IsNecklace(c, N) ) return a[1];
    return z;
}
//---------------------------------
void GenUC(int z) {
    int i,x;

    while (1) {
        printf("%d", a[1]);  total++;
        x = Coollex(z);
        if (z == x) z = a[1];  // Update missing symbol

        for (i=1; i<N-1; i++) a[i] = a[i+1];
        a[N-1] = x;

        // Break when at initial increasing
        i = 1;
        while (a[i] <= a[i+1] && i <N-1) i++;
        if (i == N-1 && a[N-1] <= z) break;
    }
}
//---------------------------------
int main() {
    int i,j,tmp;

    printf("Enter K: "); scanf("%d", &K);
    N = 0;
    for (i=1; i<=K; i++) {
        printf("N_%d: ", i);  scanf("%d", &tmp);
        for (j=1; j<=tmp; j++) a[N+j] = i;
        N += tmp;
    }
    GenUC( a[N] );
    printf("\nTotal = %d\n", total);
}
```

Appendix B Appendix - Concatenation approach to construct the reverse of $\mathcal{U}(S)$

```c
#include <stdio.h>
int N, K, a[100];
//-------------------------------------------------------------------
// If a[1..n] is a necklace return its period p; otherwise return 0
//-------------------------------------------------------------------
int Necklace() {
    int i, p=1;

    for (i=2; i<=N; i++) {
        if (a[i-p] > a[i]) return 0;
        if (a[i-p] < a[i]) p = i;
    }
    if (N % p != 0) return 0;
    return p;
}
//------------------------------
void Visit() {
    int i;
    for (i=Necklace(); i>=1; i--) printf("%d ", a[i]);
}
//------------------------------
void Swap(int i, int j) {
    int temp;
    temp = a[i];  a[i] = a[j];   a[j] = temp;
}
//---------------------------------
void Gen(int t) {
    int i,j,s;

    i = t;
    while (a[i] != a[1]) {
        while (a[i] == a[i-1]) i--;
        for (j=i; j<=t; j++) {
            Swap(j-1,j);
            if (Necklace()) Gen(j-1);
            else {
                for (s=j; s>=i; s--) Swap(s-1,s);
                Visit();
                return;
            }
        }
        for (j=t; j>=i; j--) Swap(j-1,j);
        i--;
    }
    Visit();
}
//---------------------------------
int main() {
    int i,j,tmp;

    printf("Enter K: "); scanf("%d", &K);
    N = 0;
    for (i=1; i<=K; i++) {
        printf("N_%d: ", i);  scanf("%d", &tmp);
        for (j=1; j<=tmp; j++) a[N+j] = i;
        N += tmp;
    }
    Gen(N);
}
```