Flip-Swap Languages in Binary Reflected Gray Code Order

Joe Sawada * Aaron Williams ** Dennis Wong ***

Abstract. A *flip-swap language* is a set $\mathbf{S}$ of binary strings of length n such that $\mathbf{S} \cup \{0^n\}$ is closed under two operations (when applicable): (1) Flip the leftmost 1; and (2) Swap the leftmost 1 with the bit to its right. Flip-swap languages model many combinatorial objects including necklaces, Lyndon words, prefix normal words, left factors of k-ary Dyck words, lattice paths with at most k flaws, and feasible solutions to 0-1 knapsack problems. We prove that any flip-swap language forms a cyclic 2-Gray code when listed in binary reflected Gray code (BRGC) order. Furthermore, a generic successor rule computes the next string when provided with a membership tester. The rule generates each string in the aforementioned flip-swap languages in $O(n)$-amortized per string, except for prefix normal words of length n which require $O(n^{1.864})$-amortized per string. Our work generalizes results on necklaces and Lyndon words by Vajnovszki [Inf. Process. Lett. 106(3):96−99, 2008].

1 Introduction

Combinatorial generation studies the efficient generation of each instance of a combinatorial object, such as the $n!$ permutations of $\{1, 2, \ldots, n\}$ or the $\frac{1}{n+1}\binom{2n}{n}$ binary trees with n nodes. The research area is fundamental to computer science and it has been covered by textbooks such as *Combinatorial Algorithms for Computers and Calculators* by Nijenhuis and Wilf [36], *Concrete Mathematics: A Foundation for Computer Science* by Graham, Knuth, and Patashnik [10], and *The Art of Computer Programming, Volume 4A, Combinatorial Algorithms* by Knuth [13]. The subject is important to every day programmers, and Arndt's *Matters Computational: Ideas, Algorithms, Source Code* is an excellent practical resource [1]. A primary consideration is listing the instances of a combinatorial object so that consecutive instances differ by a specified *closeness condition*. Lists of this type are called *Gray codes*. This terminology is due to the eponymous *binary reflected Gray code* (*BRGC*) by Frank Gray, which orders the 2^n binary strings of length n so that consecutive strings differ in one bit. The BRGC was patented for a pulse code communication system in 1953 [11]. For example, the order for $n = 4$ is

$$\begin{aligned} &0000, 1000, 1100, 0100, 0110, 1110, 1010, 0010, \\ &0011, 1011, 1111, 0111, 0101, 1101, 1001, 0001. \end{aligned} \tag{1}$$

* Computing and Information Science, University of Guelph, Canada. Research supported by NSERC. email:jsawada@uoguelph.ca

** Computer Science, Williams College, Williamstown, USA. email:aaron@cs.williams.edu

*** School of Applied Science, Macao Polytechnic University, China. Research supported by NRF Korea. email:cwong@uoguelph.ca

Variations that reverse the entire order of the individual strings are also commonly used in practice and in the literature. We note that the order in (1) is *cyclic* because the last and first strings also differ by the closeness condition, and this property holds for all n.

One challenge facing combinatorial generation is its relative surplus of breadth and lack of depth[1]. For example, [1], [13], [16] and [36] have separate subsections for different combinatorial objects, and the majority of the Gray codes are developed from first principles. Thus, it is important to encourage simple frameworks that can be applied to a variety of combinatorial objects. Previous work in this direction includes the following:

1. the ECO framework developed by Bacchelli, Barcucci, Grazzini, and Pergola [2] that generates Gray codes for a variety of combinatorial objects such as Dyck words in constant amortized time per instance;
2. the twisted lexico computation tree by Takaoka [29] that generates Gray codes for multiple combinatorial objects in constant amortized time per instance;
3. loopless algorithms developed by Walsh [34] to generate Gray codes for multiple combinatorial objects, which extend algorithms initially given by Ehrlich in [9];
4. greedy algorithms observed by Williams [38] that provide a uniform understanding for many previous published results;
5. the reflectable language framework by Li and Sawada [14] for generating Gray codes of k-ary strings, restricted growth strings, and k-ary trees with n nodes;
6. the permutation language framework developed by Hartung, Hoang, Mütze and Williams [12] that provides algorithms to generate Gray codes for a variety of combinatorial objects based on encoding them as permutations.

We focus on an approach that is arguably simpler than all of the above: Start with a known Gray code and then *filter* or *induce* the list based on a subset of interest. In other words, the subset is listed in the relative order given by a larger Gray code, and the resulting order is a *sublist (Gray code)* with respect to it. Historically, the first sublist Gray code appears to be the *revolving door* Gray code for combinations [35]. A *combination* is a length n binary string with *weight* (i.e. number of ones) k. The Gray code is created by filtering the BRGC, as shown below for $n = 4$ and $k = 2$ (cf. (1))

$$\begin{aligned} &\cancel{0000}, \cancel{1000}, \mathbf{1100}, \cancel{0100}, \mathbf{0110}, \cancel{1110}, \mathbf{1010}, \cancel{0010}, \\ &\mathbf{0011}, \cancel{1011}, \cancel{1111}, \cancel{0111}, \mathbf{0101}, \cancel{1101}, \mathbf{1001}, \cancel{0001}. \end{aligned} \tag{2}$$

This order is a *transposition Gray code* as consecutive strings differ by transposing at most two bits[2]. It can be generated *directly* (i.e. without filtering) by an efficient algorithm [35].

The sublist or filtering approach to constructing Gray codes has also been achieved by using cool-lex order [21] and Steinhaus-Johnson-Trotter order [22,32]. In particular, the bubble language framework developed by Ruskey, Sawada and Williams [19] provides

[1] This is not to say that combinatorial generation is always easy. For example, the 'middle levels' conjecture was confirmed by Mütze [15] after 30 years and effort by hundreds of researchers.

[2] When each string is viewed as the incidence vector of a k-subset of $\{1, 2, \ldots, n\}$, then consecutive k-subsets change via a "revolving door" (i.e. one value enters and one value exits).

algorithms to generate shift Gray codes for fixed-weight necklaces and Lyndon words, k-ary Dyck words, and representations of interval graphs, and in each case the relative orders from [21] (or more broadly [28]) provide the Gray code in cool-lex order. More recently, a Gray code of fixed-content necklaces in cool-lex order [25] was obtained by filtering the Gray code in [37].

A *t-Gray code* is a generalization of Gray code, which is a listing of a combinatorial object so that consecutive instances differ by at most t bits. As an example, the revolving door Gray code in (2) is a 2-Gray code, where each consecutive strings differ by at most two bits. Vajnovszki [30] first proved that necklaces and Lyndon words form a cyclic 3-Gray code in dual reflected order and provided a recursive algorithm to efficiently generate these sublist Gray codes. Later, Vajnovszki [31] provided a more restricted Gray code by proving that necklaces and Lyndon words form a cyclic 2-Gray code in BRGC order, and efficient algorithms can generate these sublist Gray codes directly [26]. Our goal is to expand upon the known languages that are 2-Gray codes in BRGC order, and which can be efficiently generated. To do this, we introduce a new class of languages.

A *flip-swap language* (with respect to 1) is a set $\mathbf{S}$ of length n binary strings such that $\mathbf{S} \cup \{0^n\}$ is closed under two operations (when applicable):

1. Flip the leftmost 1 (flip-first); and
2. Swap the leftmost 1 with the bit to its right (swap-first).

A flip-swap language with respect to 0 is defined similarly. Flip-swap languages encode a wide variety of combinatorial objects. The formal definitions of these languages are given in Section 3.

Theorem 1. *The following sets of length n binary strings are flip-swap languages.*

Flip-Swap languages (with respect to 1)	**Flip-Swap languages (with respect to 0)**
i. all strings	*i. all strings*
ii. strings with weight $\leq k$	*ii. strings with weight $\geq k$*
iii. strings $\leq \gamma$	*iii. strings $\geq \gamma$*
iv. strings with $\leq k$ inversions re: 0^*1^*	*iv. strings with $\leq k$ inversions re: 1^*0^**
v. strings with $\leq k$ transpositions re: 0^*1^*	*v. strings with $\leq k$ transpositions re: 1^*0^**
vi. strings $<$ their reversal	*vi. strings $>$ their reversal*
vii. strings $\leq$ their reversal (neckties)	*vii. strings $\geq$ their reversal*
viii. strings $<$ their complemented reversal	*viii. strings $>$ their complemented reversal*
ix. strings $\leq$ their complemented reversal	*ix. strings $\geq$ their complemented reversal*
x. strings with forbidden 10^t	*x. strings with forbidden 01^t*
xi. strings with forbidden prefix 1γ	*xi. strings with forbidden prefix 0γ*
xii. 0-prefix normal words	*xii. 1-prefix normal words*
xiii. necklaces (smallest rotation)	*xiii. necklaces (largest rotation)*
xiv. Lyndon words	*xiv. aperiodic necklaces (largest rotation)*
xv. prenecklaces (smallest rotation)	*xv. prenecklaces (largest rotation)*
xvi. pseudo-necklaces with respect to 0^*1^*	*xvi. pseudo-necklaces with respect to 1^*0^**
xvii. left factors of k-ary Dyck words	*xvii. right factors of k-ary Dyck words*
xviii. lattice paths with $\leq k$ flaws	*xviii. lattice paths with $\geq k$ flaws*
xix. feasible solutions to 0-1 knapsack problems	

Our second result is that every flip-swap language forms a cyclic 2-Gray code when listed in BRGC order. This generalizes the previous sublist BRGC results [26,31]. Our Gray codes are also suffix partitioned Gray codes, where strings with the same suffix are contiguous.

Theorem 2. *When a flip-swap language* $\mathbf{S}$ *is listed in BRGC order the resulting listing is a 2-Gray code. If* $\mathbf{S}$ *includes* 0^n *then the listing is cyclic.*

Our third result is a generic successor rule that efficiently computes the next string in the 2-Gray code of a flip-swap language as long as a fast membership test is given.

Theorem 3. *The languages in Theorem 1 can be generated in* $O(n)$*-amortized time per string, with the exception of prefix normal words which require* $O(n^{1.864})$*-time.*

A preliminary version of this paper appeared at the WORDS 2021 conference [27]. This extended version includes a full proof of Theorem 1 which additionally showing that the following sets of length n binary strings are flip-swap languages:

- ▷ strings with weight $\leq k$;
- ▷ strings $\leq \gamma$;
- ▷ strings with $\leq k$ inversions;
- ▷ strings with $\leq k$ transpositions;
- ▷ strings $<$ their reversal;
- ▷ strings $\leq$ their reversal (neckties);
- ▷ strings $<$ their complemented reversal;
- ▷ strings $\leq$ their complemented reversal;
- ▷ strings with forbidden 10^t;
- ▷ strings with forbidden prefix 1γ;
- ▷ prenecklaces;
- ▷ pseudo-necklaces; and
- ▷ left factors of k-ary Dyck words.

We also introduce additional flip-swap languages including right factors of k-ary Dyck words and lattice paths with at most k flaws.

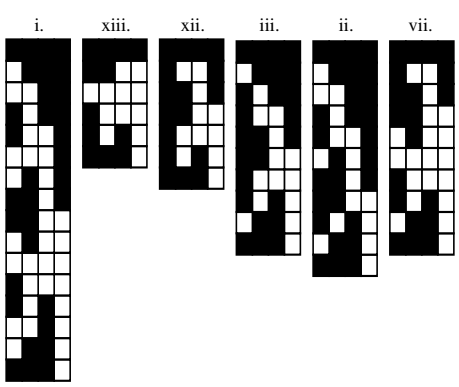
The remainder of this paper is outlined as follows. In Section 2, we formally define our version of the BRGC. In Section 3, we prove Theorem 1, and define the flip-swap partially ordered set. In Section 4, we give our generic successor rule and prove Theorem 2. In Section 5, we present a generic generation algorithm that list out each string of a flip-swap language, and we prove Theorem 3.

2 The binary reflected Gray code

Let $\mathbf{B}(n)$ denote the set of length n binary strings. Let $BRGC(n)$ denote the listing of $\mathbf{B}(n)$ in BRGC order. Let $\overline{BRGC}(n)$ denote the listing $BRGC(n)$ in reverse order. Then $BRGC(n)$ can be defined recursively as follows, where $\mathcal{L} \cdot x$ denotes the listing

$n=4$ BRGC	all i.	necklaces xiii.	0-PNW xii.	≤ 1001 iii.	$k \leq 2$ ii.	neckties vii.
0000	✓	✓	✓	✓	✓	✓
1000	✓			✓	✓	
1100	✓				✓	
0100	✓			✓	✓	
0110	✓		✓	✓	✓	✓
1110	✓					
1010	✓				✓	
0010	✓		✓	✓	✓	✓
0011	✓	✓	✓	✓	✓	✓
1011	✓					✓
1111	✓	✓				✓
0111	✓	✓	✓	✓		✓
0101	✓	✓	✓	✓	✓	✓
1101	✓					
1001	✓			✓	✓	✓
0001	✓	✓	✓	✓	✓	✓

(a) String membership in 6 flip-swap languages.



(b) Visualizing the 2-Gray codes in (a).

Table 1: Flip-swap languages ordered as sublists of the binary reflected Gray code. Theorem 1 covers each language, so the resulting orders are 2-Gray codes.

$\mathcal{L}$ with the character x appended to the end of each string:

$$BRGC(n) = \begin{cases} 0, 1 & \text{if } n = 1; \\ BRGC(n-1) \cdot 0,\ \overline{BRGC}(n-1) \cdot 1 & \text{if } n > 1. \end{cases}$$

For example, $BRGC(2) = 00, 10, 11, 01$ and $\overline{BRGC}(2) = 01, 11, 10, 00$, thus

$$BRGC(3) = 00\mathbf{0}, 10\mathbf{0}, 11\mathbf{0}, 01\mathbf{0}, 01\mathbf{1}, 11\mathbf{1}, 10\mathbf{1}, 00\mathbf{1}.$$

This definition of BRGC order is the same as the one used by Vajnovzski [31]. When the strings are read from right-to-left, we obtain the classic definition of BRGC order [11]. For flip-swap languages with respect to 0, we interchange the roles of the 0s and 1s; however, for our discussions we will focus on flip-swap languages with respect to 1. Table 1 illustrates $BRGC(4)$ and six flip-swap languages listed in Theorem 1.

3 Flip-swap languages

In this section, we formalize the flip-swap languages stated in Theorem 1. We also prove Theorem 1 of the listed languages with respect to 1.

Consider a binary string $\alpha = b_1 b_2 \cdots b_n$. Let $flip_\alpha(i)$ be the string obtained by complementing b_i. Let $swap_\alpha(i, j)$ be the string obtained by swapping b_i and b_j. When the context is clear we use $flip(i)$ and $swap(i, j)$ instead of $flip_\alpha(i)$ and $swap_\alpha(i, j)$. Also, let $\ell_0(\alpha)$ denote the position of the leftmost 0 of α or $n + 1$ if no such position exists. Similarly, let $\ell_1(\alpha)$ denote the position of the leftmost 1 of α or $n + 1$ if no such position exists. To simplify the notation, we define $\ell_\alpha = \ell_1(\alpha)$.

Binary strings: Obviously the set $\mathbf{B}(n)$ satisfies the two closure properties of a flip-swap language and thus is a flip-swap language. In fact, the BRGC order induces a cyclic 1-Gray code for $\mathbf{B}(n)$ [13,17].

Binary strings with weight $\leq k$: The *weight* of a binary string is the number of 1s it contains. Let $\mathbf{S}$ be the set of binary strings of length n having weight less than or equal to some k. Observe that $\mathbf{S}$ satisfies the two closure properties of a flip-swap language as the flip-first and swap-first operations either decrease or maintain the weight. Thus, $\mathbf{S}$ is a flip-swap language.

Binary strings $\leq \gamma$: Let $\mathbf{S}$ be the set of binary strings of length n with each string lexicographically smaller or equal to some string γ. Observe that $\mathbf{S}$ satisfies the two closure properties of a flip-swap language as the flip-first and swap-first operations either make the resulting string lexicographically smaller or produce the same string. Thus, $\mathbf{S}$ is a flip-swap language.

Binary strings with $\leq k$ inversions: An *inversion* with respect to 0^*1^* in a binary string $\alpha = b_1 b_2 \cdots b_n$ is any $b_i = 1$ and $b_j = 0$ such that $i < j$. For example $\alpha = 100101$ has four inversions: $(b_1, b_2), (b_1, b_3), (b_1, b_5), (b_4, b_5)$. Let $\mathbf{S}$ be the set of binary strings of length n with less than or equal to k inversions with respect to 0^*1^*. Observe that $\mathbf{S}$ satisfies the two closure properties of a flip-swap language as the flip-first and swap-first operations either decrease or maintain the number of inversions. Thus, $\mathbf{S}$ is a flip-swap language.

Binary strings with $\leq k$ transpositions: The number of *transpositions* of a binary string $\alpha = b_1 b_1 \cdots b_n$ with respect to 0^*1^* is the minimum number of $swap(i, j)$ operations required to change α into the form 0^*1^*. For example, the number of transpositions of the string 100101 is 1. Let $\mathbf{S}$ be the set of binary strings of length n with less than or equal to k transpositions with respect to 0^*1^*. Observe that $\mathbf{S}$ satisfies the two closure properties of a flip-swap language as the flip-first and swap-first operations either decrease or maintain the number of transpositions.Thus, $\mathbf{S}$ is a flip-swap language.

Binary strings $<$ or $\leq$ their reversal: Let $\mathbf{S}$ be the set of binary strings of length n with each string lexicographically smaller than (or equal to) their reversal. Observe that $\mathbf{S}$ satisfies the swap-first property as the swap-first operation either produces the same string, or makes the resulting sting lexicographically smaller while its reversal lexicographically larger. Furthermore, $\mathbf{S} \cup \{0^n\}$ satisfies the flip-first property as the flip-first operation complements the most significant bit of α but the least significant bit of its reversal when the weight of the string is larger than one; or otherwise produces the string 0^n when the weight of the string is equal to one. Thus, $\mathbf{S}$ is a flip-swap language.

Equivalence class of strings under reversal has also been called neckties [23].

Binary strings $<$ or $\leq$ their complemented reversal: Let $\mathbf{S}$ be the set of binary strings of length n with each string lexicographically smaller than (or equal to) its complemented reversal. Observe that $\mathbf{S}$ satisfies the flip-first property as the flip-first operation makes the resulting string lexicographically smaller while its complemented reversal lexicographically larger. Furthermore, $\mathbf{S}$ satisfies the swap-first property as the swap-first operation either produces the same string, or complements the most significant bit of α and also a 1 of its complemented reversal. Thus, the resulting string must also be less than its complemented reversal. Thus, $\mathbf{S}$ is a flip-swap language.

Binary strings with forbidden 10^t: Observe that the set of binary strings of length n without the substring 10^t satisfies the two closure properties of a flip-swap language as the flip-first and swap-first operations do not create the substring 10^t. Thus, the set of binary strings of length n without the substring 10^t is a flip-swap language.

Sublists of BRGC for constructing Gray codes for strings with forbidden 10^t have also been studied in [3].

Binary strings with forbidden prefix 1γ: Observe that the set of binary strings of length n without the prefix 1γ satisfies the two closure properties of a flip-swap language as the flip-first and swap-first operations either create a string with the prefix 0 or produce the same string. Thus, the set of binary strings of length n without the prefix 1γ is a flip-swap language.

Prefix normal words: A binary string α is *prefix normal* with respect to 0 (also known as 0-prefix normal word) if no substring of α has more 0s than its prefix of the same length. For example, the string 001010010111011 is a 0-prefix normal word but the string 001010010011011 is not because it has a substring of length 5 with four 0s while the prefix of length 5 has only three 0s. Observe that the set of 0-prefix normal words of length n satisfies the two closure properties of a flip-swap language as the flip-first and swap-first operations either increase or maintain the number of 0s in its prefix. Thus, the set of 0-prefix normal words of length n is a flip-swap language.

Necklaces: A *necklace* is the lexicographically smallest (largest) string in an equivalence class under rotation. Let $\mathbf{N}(n)$ be the set of necklaces of length n and $\alpha = 0^j1b_{j+2}b_{j+3}\cdots b_n$ be a necklace in $\mathbf{N}(n)$. By the definition of necklace, it is easy to see that $flip_\alpha(\ell_\alpha) = 0^{j+1}b_{j+2}b_{j+3}\cdots b_n \in \mathbf{N}(n)$ and thus $\mathbf{N}(n)$ satisfies the flip-first property. For the swap-first operation, observe that if $\alpha \neq 0^{n-1}1$ and $b_{j+2} = 1$, then the swap-first operation produces the same necklace. Otherwise if $\alpha \neq 0^{n-1}1$ and $b_{j+2} = 0$, then the swap-first operation produces the string $0^{j+1}1b_{j+3}b_{j+4}\cdots b_n$ which is clearly a necklace. Thus, the set of necklaces is a flip-swap language.

Lyndon words: An *aperiodic necklace* is a necklace that cannot be written in the form β^j for some $j < n$. A *Lyndon word* is an aperiodic necklace when using the lexicographically smallest string as the representative. Let $\mathbf{L}(n)$ denote the set of Lyndon words of length n. Since $\mathbf{N}(n)$ is a flip-swap language and $\mathbf{L}(n) \cup \{0^n\} \subseteq \mathbf{N}(n)$, it suffices to show that applying the flip-first or the swap-first operation on a Lyndon word either yields an aperiodic string or the string 0^n.

Clearly $\mathbf{L}(n) \cup \{0^n\}$ satisfies the two closure properties when $\alpha \in \{0^n, 0^{n-1}1\}$. Thus in the remaining of the proof, $\alpha \notin \{0^n, 0^{n-1}1\}$. We first prove by contradiction that $\mathbf{L}(n) \cup \{0^n\}$ satisfies the flip-first property. Let $\alpha = 0^j1b_{j+2}b_{j+3}\cdots b_n$ be a string in $\mathbf{L}(n) \cup \{0^n\}$. Suppose that $\mathbf{L}(n) \cup \{0^n\}$ does not satisfy the flip-first property and $flip_\alpha(\ell_\alpha)$ is periodic. Thus $flip_\alpha(\ell_\alpha) = (0^{j+1}\beta)^t$ for some string β and $t \geq 2$. Observe that $\alpha = 0^j1\beta(0^{j+1}\beta)^{t-1}$ which is clearly not a Lyndon word, a contradiction. Therefore $\mathbf{L}(n) \cup \{0^n\}$ satisfies the flip-first property. Then similarly we prove by contradiction that $\mathbf{L}(n) \cup \{0^n\}$ satisfies the swap-first property. If $b_{j+2} = 1$, then applying the swap-first operation on α produces the same Lyndon word. Thus in the remaining of the proof, $b_{j+2} = 0$. Suppose that $\mathbf{L}(n) \cup \{0^n\}$ does not satisfy the swap-first property such that $\alpha \in \mathbf{L}(n) \cup \{0^n\}$ but $swap_\alpha(\ell_\alpha, \ell_\alpha + 1)$ is periodic. Thus

$swap_\alpha(\ell_\alpha, \ell_\alpha + 1) = (0^{j+1}1\beta)^t$ for some string β and $t \geq 2$. Thus α contains the prefix 0^j1 but also the substring $0^{j+1}1$ in its suffix which is clearly not a Lyndon word, a contradiction. Thus, $\mathbf{L}(n)$ is a flip-swap language.

Prenecklaces: A *prenecklace* is a prefix of a necklace. Since $\mathbf{N}(n)$ is a flip-swap language, applying the flip-first or the swap-first operation on a prenecklace also creates a string that is a prefix of a necklace. Thus, the set of prenecklaces of length n is a flip-swap language.

Pseudo-necklaces: A *block* with respect to 0^*1^* is a maximal substring of the form 0^*1^*. Each block B_i with respect to 0^*1^* can be represented by two integers (s_i, t_i) corresponding to the number of 0s and 1s respectively. For example, the string $\alpha = 000110100011001$ can be represented by $B_4B_3B_2B_1 = (3,2)(1,1)(3,2)(2,1)$. A block $B_i = (s_i, t_i)$ is said to be *lexicographically smaller* than a block $B_j = (s_j, t_j)$ (denoted by $B_i < B_j$) if $s_i < s_j$ or $s_i = s_j$ with $t_i < t_j$. A string $\alpha = b_1b_2 \cdots b_n = B_bB_{b-1} \cdots B_1$ is a *pseudo-necklace* with respect to 0^*1^* if $B_b \leq B_i$ for all $1 \leq i < b$. Observe that the set of pseudo-necklaces of length n satisfies the two closure properties of a flip-swap language as the flip-first and swap-first operations do not make the first block B_b lexicographically larger, while the remaining blocks either remain the same or become lexicographically larger. Thus, the set of pseudo-necklaces of length n is a flip-swap language.

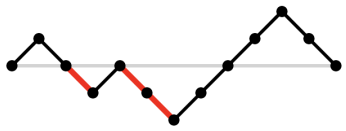
Left factors of k-ary Dyck words: A *k-ary Dyck word* is a binary string of length $n = tk$ with t copies of 1 and $t(k-1)$ copies of 0 such that every prefix has at least $k-1$ copies of 0 for every 1. A string is said to be a *left factor of a k-ary Dyck word* if it is the prefix of some k-ary Dyck word. Similarly, a string is said to be a *right factor of a k-ary Dyck word* if it is the suffix of some k-ary Dyck word. It is well-known that k-ary Dyck words are in one-to-one correspondence with k-ary trees with t internal nodes. When $k = 2$, Dyck words are counted by the Catalan numbers and are equivalent to *balanced parentheses*. As an example, 001011 is a 2-ary Dyck word while 011001 is not. k-ary Dyck words and balanced parentheses strings are well studied and have lots of applications including trees and stack-sortable permutations [5,18,20,33].

The set of k-ary Dyck words of length n is not a flip-swap language since 001011 is a 2-ary Dyck word but 000011 is not. The set of length n prefixes of k-ary Dyck words is, however, a flip-swap language. This set is also called *ballot sequences*. Observe that the set of left factors of k-ary Dyck words satisfies the two closure properties of a flip-swap language as the flip-first and swap-first operations do not increase the number 1s in the prefix. Thus, the set of left factors of k-ary Dyck words is a flip-swap language.

Efficient generation of left factors of k-ary Dyck words or ballot sequences of length n in BRGC order has been studied in [39].

Lattice paths with $\leq k$ flaws: A *lattice path* is a path in the two-dimensional integer grid $\mathbb{Z}^2$ that starts at the origin $(0,0)$ and consists of steps that change the current position by *upsteps* $(+1,+1)$ or by *downsteps* $(+1,-1)$. A *flaw* of a lattice path is a downstep below the line $y = 0$. A *Dyck path* (2-ary Dyck word) is thus a lattice path with no flaws.

A lattice path can be encoded as a binary string with an upstep represented by a 0 and a downstep represented by a 1. For example, the below lattice path can be represented by the binary string 011011000011 and has three flaws (highlighted in bold).



Let $\mathbf{S}$ be the set of binary strings of length n representing lattice paths with at most k flaws. Observe that $\mathbf{S}$ satisfies the flip-first property as the flip-first operation changes a downstep to an upstep and thus produces a lattice path with the same or less number of flaws. Furthermore, $\mathbf{S}$ satisfies the swap-first property as the swap-first operation either produces the same string, or replaces a downstep-upstep (10) with an upstep-downstep (01) and thus produces a lattice path with the same or less number of flaws. Thus, the set of binary strings of length n representing lattice paths with at most k flaws is a flip-swap language.

Feasible solutions to 0-1 knapsack problems: The input to a 0-1 *knapsack problem* is a knapsack capacity W, and a set of n items each of which has a non-negative weight $w_i \geq 0$ and a value v_i. A subset of items is *feasible* if the total weight of the items in the subset is less than or equal to the capacity W. Typically, the goal of the problem is to find a feasible subset with the maximum value, or to decide if a feasible subset exists with value $\geq c$. Given the input to a 0-1 knapsack problem, we reorder the items by non-increasing weight. That is, $w_i \geq w_{i+1}$ for $1 \leq i \leq n-1$. Notice that the incidence vectors of feasible subsets are now a flip-swap language. More specifically, flipping any 1 to 0 causes the subset sum to decrease, and so does swapping any 1 with the bit to its right. Hence, the language satisfies the flip-first and the swap-first closure properties and is a flip-swap language.

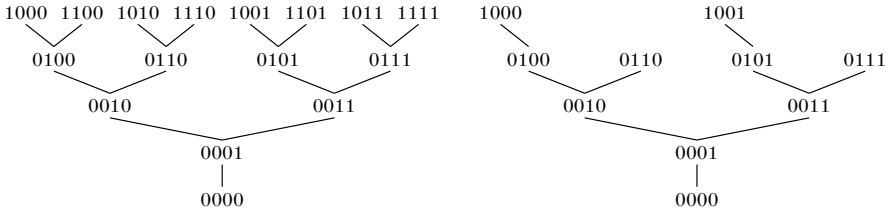
3.1 Flip-Swap poset

In this section we introduce a poset whose ideals correspond to a flip-swap language which includes the string 0^n.

Let $\alpha = b_1 b_2 \cdots b_n$ be a length n binary string. We define $\tau(\alpha)$ as follows:

$$\tau(\alpha) = \begin{cases} \alpha & \text{if } \alpha = 0^n, \\ flip_\alpha(\ell_\alpha) & \text{if } \alpha \neq 0^n \text{ and } (\ell_\alpha = n \text{ or } b_{\ell_\alpha+1} = 1) \\ swap_\alpha(\ell_\alpha, \ell_\alpha + 1) & \text{otherwise.} \end{cases}$$

Let $\tau^t(\alpha)$ denote the string that results from applying the τ operation t times to α. We define the binary relation $<_R$ on $\mathbf{B}(n)$ to be the transitive closure of the cover relation τ, that is $\beta <_R \alpha$ if $\beta \neq \alpha$ and $\beta = \tau^t(\alpha)$ for some $t > 0$. It is easy to see that the binary relation $<_R$ is irreflexive, anti-symmetric and transitive. Thus $<_R$ is a strict partial order. The relation $<_R$ on binary strings defines our *flip-swap poset*.



(a) The flip-swap poset $\mathcal{P}(4)$. (b) An ideal of $\mathcal{P}(4)$.

Fig. 1: Flip-swap languages are the ideals of the flip-swap poset. The ideal in (b) contains the 4-bit binary strings that are ≤ 1001 with respect to lexicographic order.

Definition 1. *The flip-swap poset $\mathcal{P}(n)$ is a strict poset with $\mathbf{B}(n)$ as the ground set and $<_R$ as the strict partial order.*

Figure 1 shows the Hasse diagram of $\mathcal{P}(4)$ with the ideal for binary strings of length 4 that are lexicographically smaller or equal to 1001 in (b). Observe that $\mathcal{P}(n)$ is always a tree with 0^n as the unique minimum element, and that its ideals are the subtrees that contain this minimum.

Lemma 1. *A set $\mathbf{S}$ over $\mathbf{B}(n)$ that includes 0^n is a flip-swap language if and only if $\mathbf{S}$ is an ideal of $\mathcal{P}(n)$.*

Proof. Let $\mathbf{S}$ be a flip-swap language and α be a string in $\mathbf{S}$. Since $\mathbf{S}$ is a flip-swap language, $\mathbf{S}$ satisfies the flip-first and swap-first properties and thus $\tau(\alpha)$ is a string in $\mathbf{S}$. Therefore every string $\gamma <_R \alpha$ is in $\mathbf{S}$ and hence $\mathbf{S}$ is an ideal of $\mathcal{P}(n)$. The other direction is similar. □

If $\mathbf{S}$ is a set of binary strings and γ is a binary string, then the *quotient* of $\mathbf{S}$ and γ is $\mathbf{S}/\gamma = \{\alpha \mid \alpha\gamma \in \mathbf{S}\}$.

Lemma 2. *If $\mathbf{S}_1$ and $\mathbf{S}_2$ are flip-swap languages and γ is a binary string, then $\mathbf{S}_1 \cap \mathbf{S}_2$, $\mathbf{S}_1 \cup \mathbf{S}_2$ and $\mathbf{S}_1/\gamma$ are flip-swap languages.*

Proof. Let $\mathbf{S}_1$ and $\mathbf{S}_2$ be two flip-swap languages and let γ be a binary string. The intersection and union of ideals of any poset are also ideals of that poset, so $\mathbf{S}_1 \cap \mathbf{S}_2$ and $\mathbf{S}_1 \cup \mathbf{S}_2$ are flip-swap languages. Now consider $\alpha \in \mathbf{S}_1/\gamma$.

Suppose $\alpha \in \mathbf{S}_1/\gamma$ for some non-empty γ where $j = |\alpha|$. This means that $\alpha\gamma \in \mathbf{S}_1$. Consider three cases depending $\ell_{\alpha\gamma}$. If $\ell_{\alpha\gamma} < j$, then clearly $\tau(\alpha\gamma) = \tau(\alpha)\gamma$. From Lemma 1, $\tau(\alpha)\gamma \in \mathbf{S}_1$ and thus $\tau(\alpha) \in \mathbf{S}_1/\gamma$. If $\ell_{\alpha\gamma} = j$, then $\alpha = 0^{j-1}1$ and $\tau(\alpha) = 0^j$. Since $\mathbf{S}_1$ is a flip-swap language $0^j\gamma \in \mathbf{S}_1$. Again this implies that $\tau(\alpha) \in \mathbf{S}_1/\gamma$. If $\ell_{\alpha\gamma} > j$ then $\alpha = 0^j$ and $\tau(\alpha) = \alpha$ in this case. For each case we have shown that $\tau(\alpha) \in \mathbf{S}_1/\gamma$ and thus $\mathbf{S}_1/\gamma$ is a flip-swap language by Lemma 1. □

Corollary 1. *Flip-swap languages are closed under union, intersection, and quotient.*

Proof. Let $\mathbf{S}_1$ and $\mathbf{S}_2$ be flip-swap languages and γ be a binary string. Since $\mathbf{S}_1$ and $\mathbf{S}_2$ can be represented by ideals of the flip-swap poset, possibly excluding 0^n, by Lemma 2 the sets $\mathbf{S}_1 \cap \mathbf{S}_2$, $\mathbf{S}_1 \cup \mathbf{S}_2$ and $\mathbf{S}_1/\gamma$ are flip-swap languages. □

Lemma 3. *If $\alpha\gamma$ is a binary string in a flip-swap language $\mathbf{S}$, then $0^{|\alpha|}\gamma \in \mathbf{S}$.*

Proof. This result follows from the flip-first property of flip-swap languages. □

4 A generic successor rule for flip-swap languages

Consider any flip-swap language $\mathbf{S}$ that includes the string 0^n. Let $\mathcal{BRGC}(\mathbf{S})$ denote the listing of $\mathbf{S}$ in BRGC order. Given a string $\alpha \in \mathbf{S}$, we define a generic *successor rule* that computes the string following α in the cyclic listing $\mathcal{BRGC}(\mathbf{S})$.

Let $\alpha = b_1 b_2 \cdots b_n$ be a string in $\mathbf{S}$. Let t_α be the leftmost position of α such that $flip_\alpha(t_\alpha) \in \mathbf{S}$ when $|\mathbf{S}| > 1$. Such a t_α exists since $\mathbf{S}$ satisfies the flip-first property and $|\mathbf{S}| > 1$. Recall that ℓ_α is the position of the leftmost 1 of α (or $|\alpha| + 1$ if no such position exists). Notice that $t_\alpha \leq \ell_\alpha$ when $|\mathbf{S}| > 1$ since $\mathbf{S}$ is a flip-swap language.

Let $flip2_\alpha(i, j)$ be the string obtained by complementing both b_i and b_j. When the context is clear we use $flip2(i, j)$ instead of $flip2_\alpha(i, j)$. Also, let $w(\alpha)$ denote the number of 1s of α. We claim that the following function f computes the next string in the cyclic ordering $\mathcal{BRGC}(\mathbf{S})$:

$$f(\alpha) = \begin{cases} 0^n & \text{if } \alpha = 0^{n-1}1; \quad (4a) \\ flip_\alpha(t_\alpha) & \text{if } w(\alpha) \text{ is even and } (t_\alpha = 1 \text{ or } flip2_\alpha(t_\alpha - 1, t_\alpha) \notin \mathbf{S}); \quad (4b) \\ flip2_\alpha(t_\alpha - 1, t_\alpha) & \text{if } w(\alpha) \text{ is even and } flip2_\alpha(t_\alpha - 1, t_\alpha) \in \mathbf{S}; \quad (4c) \\ flip2_\alpha(\ell_\alpha, \ell_\alpha + 1) & \text{if } w(\alpha) \text{ is odd and } flip_\alpha(\ell_\alpha + 1) \notin \mathbf{S}; \quad (4d) \\ flip_\alpha(\ell_\alpha + 1) & \text{if } w(\alpha) \text{ is odd and } flip_\alpha(\ell_\alpha + 1) \in \mathbf{S}. \quad (4e) \end{cases}$$

Thus, successive applications of the function f on a flip-swap language $\mathbf{S}$, starting with the string 0^n, list out each string in $\mathbf{S}$ in BRGC order. As an illustration of the function f, successive applications of this rule for the set of necklaces of length 6 starting with the necklace 000000 produce the listing in Table 2.

Theorem 4. *If $\mathbf{S}$ is a flip-swap language including the string 0^n and $|\mathbf{S}| > 1$, then $f(\alpha)$ is the string immediately following the string α in $\mathbf{S}$ in the cyclic ordering $\mathcal{BRGC}(\mathbf{S})$.*

We will provide a detailed proof of this theorem in the next subsection. Observe that each rule in f complements at most two bits and thus successive strings in $\mathbf{S}$ differ by at most two bit positions. Furthermore, $\mathcal{BRGC}(\mathbf{S})$ is still a 2-Gray code (although not necessarily cyclic) when 0^n is excluded from $\mathbf{S}$. This proves Theorem 2.

4.1 Proof of Theorem 4

This section proves Theorem 4. We begin with a lemma by Vajnovszki [31], and a remark that is due to the fact that $0^{n-1}1$ is in a flip-swap language $\mathbf{S}$ when $|\mathbf{S}| > 1$.

Necklaces	Parity of $w(\alpha)$	t_α	ℓ_α	Successor	Case
000000	even	6		$flip2(5,6)$	(4c)
000011	even	3		$flip2(2,3)$	(4c)
011011	even	2		$flip(2)$	(4b)
001011	odd		3	$flip(4)$	(4e)
001111	even	2		$flip2(1,2)$	(4c)
111111	even	1		$flip(1)$	(4b)
011111	odd		2	$flip(3)$	(4e)
010111	even	3		$flip(2)$	(4b)
000111	odd		4	$flip(5)$	(4e)
000101	even	2		$flip(2)$	(4b)
010101	odd		2	$flip2(2,3)$	(4d)
001101	odd		3	$flip(4)$	(4e)
001001	even	3		$flip(3)$	(4b)
000001	odd			$flip(6)$	(4a)

Table 2: The necklaces of length 6 induced by successive applications the function f starting from 000000. The sixth column of the table lists out the corresponding rules in f that apply to each necklace to obtain the next necklace.

Lemma 4. *Let $\alpha = b_1 b_2 \cdots b_n$ and β be length n binary strings such that $\alpha \neq \beta$. Let r be the rightmost position in which α and β differ. Then α comes before β in BRGC order (denoted by $\alpha \prec \beta$) if and only if $w(b_r b_{r+1} \cdots b_n)$ is even.*

Remark 1. A flip-swap language $\mathbf{S}$ in BRGC order ends with $0^{n-1}1$ when $|\mathbf{S}| > 1$.

Let $succ(\mathbf{S}, \alpha)$ be the *successor* of α in $\mathbf{S}$ in BRGC order (i.e. the string after α in the cyclic ordering $\mathcal{BRGC}(\mathbf{S})$). Next we provide two lemmas, and then prove Theorem 4.

Lemma 5. *Let $\mathbf{S}$ be a flip-swap language with $|\mathbf{S}| > 1$ and α be a string in $\mathbf{S}$. Let t_α be the leftmost position such that $flip_\alpha(t_\alpha) \in \mathbf{S}$. If $w(\alpha)$ is even, then t_α is the rightmost position in which α and $succ(\mathbf{S}, \alpha)$ differ.*

Proof. By contradiction. Let $\alpha = b_1 b_2 \cdots b_n$ and $\beta = succ(\mathbf{S}, \alpha)$. Let r be the rightmost position in which α and β differ with $r \neq t_\alpha$. If $t_\alpha > r$, then β has the suffix $1 b_{r+1} b_{r+2} \cdots b_n$ since $b_r = 0$ because $r < t_\alpha \leq \ell_\alpha$. Thus by the flip-first property, $0^{r-1}1 b_{r+1} b_{r+2} = flip_\alpha(r) \in \mathbf{S}$ and $r < t_\alpha$, a contradiction.

Otherwise if $t_\alpha < r$, then let $\gamma = flip_\alpha(t_\alpha)$. Clearly $\gamma \neq \alpha$. Now observe that $w(b_t b_{t+1} \cdots b_n)$ is even because $t_\alpha \leq \ell_\alpha$ and $w(\alpha)$ is even, and thus by Lemma 4, $\alpha \prec \gamma$. Also, γ has the suffix $b_r b_{r+1} \cdots b_n$ and $w(b_r b_{r+1} \cdots b_n)$ is even because $\alpha \prec \beta$ and r is the rightmost position α and β differ, and thus also by Lemma 4, $\gamma \prec \beta$. Thus $\alpha \prec \gamma \prec \beta$, a contradiction. Therefore $r = t_\alpha$. □

Lemma 6. *Let $\mathbf{S}$ be a flip-swap language with $|\mathbf{S}| > 1$ and $\alpha \neq 0^{n-1}1$ be a string in $\mathbf{S}$. If $w(\alpha)$ is odd, then $\ell_\alpha + 1$ is the rightmost position in which α and $succ(\mathbf{S}, \alpha)$ differ.*

Proof. Since $\alpha \neq 0^{n-1}1$ and $w(\alpha)$ is odd, $\ell_\alpha < n-1$. We now prove the lemma by contradiction. Let $\alpha = b_1 b_2 \cdots b_n$ and $\beta = succ(\mathbf{S}, \alpha)$. Let $r \neq \ell_\alpha + 1$ be the rightmost position in which α and β differ. If $r < \ell_\alpha + 1$, then $w(b_r b_{r+1} \cdots b_n)$ is odd but $\alpha \prec \beta$, a contradiction by Lemma 4. Otherwise if $r > \ell_\alpha + 1$, then let $\gamma = flip2_\alpha(\ell_\alpha, \ell_\alpha + 1)$. Clearly $\gamma \neq \alpha$, and by the flip-first and swap-first properties, $\gamma \in \mathbf{S}$. Also, observe that $w(b_{\ell_\alpha+1} b_{\ell_\alpha+2} \cdots b_n)$ is even because $w(\alpha)$ is odd, and thus by Lemma 4, $\alpha \prec \gamma$. Further, γ has the suffix $b_r b_{r+1} \cdots b_n$ and $w(b_r b_{r+1} \cdots b_n)$ is even because $\alpha \prec \beta$ and r is the rightmost position α and β differ, and thus also by Lemma 4, $\gamma \prec \beta$. Thus $\alpha \prec \gamma \prec \beta$, a contradiction. Therefore $r = \ell_\alpha + 1$. □

Proof of Theorem 4. Let $\alpha = a_1 a_2 \cdots a_n$ and $\beta = succ(\mathbf{S}, \alpha) = b_1 b_2 \cdots b_n$. Let t_α be the leftmost position such that $flip_\alpha(t_\alpha) \in \mathbf{S}$. First we consider the case when $\alpha = 0^{n-1}1$. Recall that the first string in $\mathbf{B}(n)$ in BRGC order is 0^n [17] and 0^n is a string in $\mathbf{S}$ by Lemma 3. Also, the last string in $\mathbf{S}$ in BRGC order is $0^{n-1}1$ by Remark 1 when $|\mathbf{S}| > 1$. Thus the string that appears immediately after α in the cyclic ordering $\mathcal{BRGC}(\mathbf{S})$ is $f(\alpha)$ when $\alpha = 0^{n-1}1$. In the remainder of the proof, $\alpha \neq 0^{n-1}1$ and we consider the following two cases.

Case 1: $w(\alpha)$ is even: If $t_\alpha = 1$, then clearly $\beta = flip_\alpha(t_\alpha) = f(\alpha)$. For the remainder of the proof, $t_\alpha > 1$.
Since $t_\alpha \leq \ell_\alpha$, $flip2_\alpha(t_\alpha - 1, t_\alpha)$ has the prefix $0^{t_\alpha - 2}1$. We now consider the following two cases. If $flip2_\alpha(t_\alpha - 1, t_\alpha) \notin \mathbf{S}$, then $flip_\alpha(t_\alpha)$ is the only string in $\mathbf{S}$ that has t_α as the rightmost position that differ with α and has the prefix 0^{t-2}. Therefore, $\beta = flip_\alpha(t_\alpha) = f(\alpha)$. Otherwise, $flip2_\alpha(t_\alpha - 1, t_\alpha)$ and $flip_\alpha(t_\alpha)$ are the only strings in $\mathbf{S}$ that have t_α as the rightmost position that differ with α and have the prefix $0^{t_\alpha - 2}$. By Lemma 4, $flip2_\alpha(t_\alpha - 1, t_\alpha) \prec flip_\alpha(t_\alpha)$ since $w(1\overline{a}_{t_\alpha} a_{t_\alpha+1} a_{t_\alpha+2} \cdots a_n)$ is even. Thus, $\beta = flip2_\alpha(t_\alpha - 1, t_\alpha) = f(\alpha)$.

Case 2: $w(\alpha)$ is odd: By Lemma 6, β has the suffix $\overline{a}_{\ell_\alpha+1} a_{\ell_\alpha+2} a_{\ell_\alpha+3} \cdots a_n$. Observe that if $flip_\alpha(\ell_\alpha + 1) \notin \mathbf{S}$, then by the flip-first and swap-first properties, $flip2_\alpha(\ell_\alpha, \ell_\alpha + 1)$ is the only string in $\mathbf{S}$ that has $\ell_\alpha + 1$ as the rightmost position that differ with β. Thus, $\beta = flip2_\alpha(\ell_\alpha, \ell_\alpha + 1) = f(\alpha)$. Otherwise by Lemma 4, any string $\gamma \in \mathbf{S}$ with the suffix $\overline{a}_{\ell_\alpha+1} a_{\ell_\alpha+2} a_{\ell_\alpha+3} \cdots a_n$ and $\gamma \neq flip_\alpha(\ell_\alpha + 1)$ has $flip_\alpha(\ell_\alpha + 1) \prec \gamma$ because $w(1\overline{a}_{\ell_\alpha+1} a_{\ell_\alpha+2} a_{\ell_\alpha+3} \cdots a_n)$ is even. Thus, $\beta = flip_\alpha(\ell_\alpha + 1) = f(\alpha)$.

Therefore, the string immediately after α in the cyclic ordering $\mathcal{BRGC}(\mathbf{S})$ is $f(\alpha)$. □

5 Generation algorithm for flip-swap languages

In this section we present a generic algorithm to generate 2-Gray codes for flip-swap languages via the function f.

A naïve approach to implement f is to find t_α by test flipping each bit in α to see if the result is also in the set when $w(\alpha)$ is even; or test flipping the $(\ell_\alpha + 1)$-th bit of α to see if the result is also in the set when $w(\alpha)$ is odd. Since $t_\alpha \leq \ell_\alpha$, we only need to examine the length $\ell_\alpha - 1$ prefix of α to find t_α. Such a test can be done in $O(nm)$

Algorithm 1 Pseudocode of the implementation of the function f.

```
1:  function f(α)
2:      if α = 0^{n-1}1 then return flip_α(n)
3:      else if w(α) is even then
4:          t_α ← ℓ_α
5:          while t_α > 1 and flip_α(t_α − 1) ∈ S do t_α ← t_α − 1
6:          if t_α ≠ 1 and flip2_α(t_α − 1, t_α) ∈ S then return flip2_α(t_α − 1, t_α)
7:          else return flip_α(t_α)
8:      else
9:          if flip_α(ℓ_α + 1) ∉ S then return flip2_α(ℓ_α, ℓ_α + 1)
10:         else return flip_α(ℓ_α + 1)
```

Algorithm 2 Algorithm to list out each string of a flip-swap language $\mathbf{S}$ in BRGC order.

```
1:  procedure BRGC
2:      α ← 0^n
3:      do
4:          if 0^n ∈ S or w(α) > 0 then PRINT(α)
5:          α ← f(α)
6:      while α ≠ 0^n
```

time, where $O(m)$ is the time required to complete the membership test of the set under consideration. Pseudocode of the function f is given in Algorithm 1.

To list out each string of a flip-swap language $\mathbf{S}$ in BRGC order, we can repeatedly apply the function f until it reaches the starting string. We maintain the variable ℓ_α which can be easily maintained in $O(n)$ time for each string generated. We also add a condition such that the algorithm prints the string 0^n only if 0^n is a string in $\mathbf{S}$. Pseudocode for this algorithm, starting with the string 0^n, is given in Algorithm 2. The algorithm can easily be modified to generate the corresponding counterpart of $\mathbf{S}$ with respect to 0.

A simple analysis shows that the algorithm generates $\mathbf{S}$ in $O(nm)$-time per string. A more thorough analysis improves this to $O(n+m)$-amortized time per string.

Theorem 5. *If $\mathbf{S}$ is a flip-swap language, then the algorithm BRGC produces $\mathcal{BRGC}(S)$ in $O(n+m)$-amortized time per string, where $O(m)$ is the time required to complete the membership tester for $\mathbf{S}$.*

Proof. Let $\alpha = a_1 a_2 \cdots a_n$ be a string in $\mathbf{S}$. Clearly f can be computed in $O(n)$ time when $w(\alpha)$ is odd. Otherwise when $w(\alpha)$ is even, the **while** loop in line 5 of Algorithm 1 performs a membership tester on each string $\beta = b_1 b_2 \cdots b_n$ in $\mathbf{S}$ with $b_{\ell_\alpha} b_{\ell_\alpha+1} \cdots b_n = a_{\ell_\alpha} a_{\ell_\alpha+1} \cdots a_n$ and $w(b_1 b_2 \cdots b_{\ell_\alpha-1}) = 1$. Observe that each of these strings can only be examined by the membership tester once, or otherwise the **while** loop in line 5 of Algorithm 1 produces the same t_α which results in a duplicated string, a contradiction. Thus, the total number of membership testers performed by the algorithm is bound by $|\mathbf{S}|$, and therefore f runs in $O(m)$-amortized time per string. Fi-

nally, since the other part of the algorithm runs in $O(n)$ time per string, the algorithm *BRGC* runs in $O(n+m)$-amortized time per string. □

The membership tests in this paper can be implemented in $O(n)$ time and $O(n)$ space; see [4,8,24] for necklaces, Lyndon words, prenecklaces and pseudo-necklaces of length n. One exception is the test for prefix normal words of length n requires $O(n^{1.864})$ time and $O(n)$ space [6]. Together with the above theorem, this proves Theorem 3.

Visit the Combinatorial Object Server [7] for a C implementation of our algorithms.

6 Acknowledgements

The research of Joe Sawada is supported by the *Natural Sciences and Engineering Research Council of Canada* (NSERC) grant RGPIN-2018-04211. The research of Dennis Wong is supported by the National Research Foundation of Korea (NRF) grant funded by the Ministry of Science and ICT (MSIT), Korea (No. 2020R1F1A1A01070666 and No.2021K2A9A2A1110161711).

A part of this work was done while the third author was visiting IBS discrete mathematics group in Korea. The authors would also like to thank Torsten Mütze and Vincent Vajnovszki for their helpful advice that greatly improved this paper.

References

1. J. Arndt. *Matters Computational: Ideas, Algorithms, Source Code*. Springer, 2011.
2. S. Bacchelli, E. Barcucci, E. Grazzini, and E. Pergola. Exhaustive generation of combinatorial objects by ECO. *Acta Inform.*, 40(8):585–602, 2004.
3. A. Bernini, S. Bilotta, R. Pinzani, A. Sabri, and V. Vajnovszki. Gray code orders for q-ary words avoiding a given factor. *Acta Inform.*, 52(7-8):573–592, 2015.
4. K. S. Booth. Lexicographically least circular substrings. *Inf. Process. Lett.*, 10(4/5):240–242, 1980.
5. B. Bultena and F. Ruskey. An Eades-McKay algorithm for well-formed parenthesis strings. *Inf. Process. Lett.*, 68(5):255–259, 1998.
6. T. M. Chan and M. Lewenstein. Clustered integer 3SUM via additive combinatorics. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing, STOC 15*, pages 31–40, New York, NY, USA, 2015.
7. COS++. The Combinatorial Object Server. `http://combos.org/brgc`.
8. J. P. Duval. Factorizing words over an ordered alphabet. *J. Algorithms*, 4(4):363–381, 1983.
9. G. Ehrlich. Loopless algorithms for generating permutations, combinations, and other combinatorial configurations. *J. ACM*, 20(3):500–513, 1973.
10. R. Graham, D. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1994.
11. F. Gray. Pulse code communication, 1953. US Patent 2,632,058.
12. E. Hartung, H. P. Hoang, T. Mütze, and A. Williams. Combinatorial generation via permutation languages. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '20, page 1214–1225, USA, 2020.
13. D. Knuth. *The Art of Computer Programming. Volume 4, fascicule 2., Generating all tuples and permutations*. The Art of Computer Programming. Addison-Wesley, Upper Saddle River (N.J.), 2005. Autre tirage : 2010.
14. Y. Li and J. Sawada. Gray codes for reflectable languages. *Inf. Process. Lett.*, 109(5):296 – 300, 2009.
15. T. Mütze. Proof of the middle levels conjecture. *Proceedings of the London Mathematical Society*, 112:677—-713, 2016.
16. T. Mütze. Combinatorial Gray codes - an updated survey. arXiv Preprint, Feb. 2022. arXiv:2202.01280
17. F. Ruskey. *Combinatorial Generation*. Working version (1j-CSC 425/520) edition, 2003.
18. F. Ruskey and A. Proskurowski. Generating binary trees by transpositions. *J. Algorithms*, 11(1):68 – 84, 1990.
19. F. Ruskey, J. Sawada, and A. Williams. Binary bubble languages and cool-lex order. *J. Comb. Theory Ser. A*, 119(1):155 – 169, 2012.
20. F. Ruskey and A. Williams. Generating balanced parentheses and binary trees by prefix shifts. In *Proceedings of the Fourteenth Symposium on Computing: The Australasian Theory - Volume 77*, CATS '08, pages 107–115, Darlinghurst, Australia, 2008.
21. F. Ruskey and A. Williams. The coolest way to generate combinations. *Discrete Math.*, 309:5305—-5320, 2009.
22. A. Sabri. *Gray codes and efficient exhaustive generation for several classes of restricted words. (Codes de Gray et génération exhaustive pour certaines classes de mots sous contrainte)*. PhD thesis, University of Burgundy, Dijon, France, 2015.
23. C. Savage. A survey of combinatorial Gray codes. *SIAM Rev.*, 39(4):605–629, 1997.
24. J. Sawada and A. Williams. A Gray code for fixed-density necklaces and Lyndon words in constant amortized time. *Theor. Comput. Sci.*, 502:46 – 54, 2013.

25. 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.
26. J. Sawada, A. Williams, and D. Wong. Necklaces and Lyndon words in colexicographic and reflected Gray code order. *J. Discrete Algorithms*, 46-47:25–35, 2017.
27. J. Sawada, A. Williams, and D. Wong. Inside the binary reflected Gray code: Flip-Swap languages in 2-Gray code order. In *International Conference on Combinatorics on Words*, pages 172–184. Springer, 2021.
28. B. Stevens and A. Williams. The coolest way to generate binary strings. *Theor. Comput. Syst.*, 54:551—-577, 2014.
29. T. Takaoka. An $O(1)$ time algorithm for generating multiset permutations. In *Algorithms and Computation, 10th International Symposium, ISAAC '99, Chennai, India, December 16-18, 1999, Proceedings*, volume 1741 of *Lecture Notes in Computer Science*, pages 237–246. Springer, 1999.
30. V. Vajnovszki. Gray code order for Lyndon words. *Discret. Math. Theor. Comput. Sci.*, 9(2), 2007.
31. V. Vajnovszki. More restrictive Gray codes for necklaces and Lyndon words. *Inf. Process. Lett.*, 106(3):96–99, 2008.
32. V. Vajnovszki and R. Vernay. Restricted compositions and permutations: From old to new Gray codes. *Inform. Process. Letters*, 111(13):650–655, 2011.
33. V. Vajnovszki and T. Walsh. A loop-free two-close Gray-code algorithm for listing k-ary Dyck words. *J. Discrete Algorithms*, 4(4):633–648, 2006.
34. T. Walsh. Generating Gray codes in $O(1)$ worst-case time per word. In *Discrete Mathematics and Theoretical Computer Science, 4th International Conference, DMTCS 2003, Dijon, France, July 7-12, 2003. Proceedings*, volume 2731 of *LNCS*, pages 73–88. Springer, 2003.
35. H. S. Wilf. A unified setting for sequencing, ranking, and selection algorithms for combinatorial objects. *Adv. Math.*, 24:281–291, 1977.
36. H. S. Wilf and A. Nijenhuis. *Combinatorial Algorithms: For Computers and Calculators*. Academic Press, 2nd edition, 1978.
37. 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*, pages 987–996. SIAM, 2009.
38. A. Williams. The greedy Gray code algorithm. In *13th International Symposium, WADS 2013, London, ON, Canada, August 12-14, 2013. Proceedings*, pages 525–536, 2013.
39. D. Wong, F. Calero, and K. Sedhai. Generating 2-Gray codes for ballot sequences in constant amortized time. *(submitted)*, 2022.