LOOPLESS GENERATION OF MULTISET PERMUTATIONS BY PREFIX SHIFTS

AARON WILLIAMS

UNIVERSITY OF VICTORIA, CANADA

ABSTRACT. This paper answers the following mathematical question: Can multiset permutations be ordered so that each permutation is a prefix shift of the previous permutation? Previously, the answer was known for the permutations of any set, and the permutations of any multiset whose corresponding set contains only two elements. This paper also answers the following algorithmic question: Can multiset permutations be generated by a loopless algorithm that uses only a constant number of additional variables? Previously, the best loopless algorithm used a linear number of additional variables. The answers to these questions are both yes.

1. INTRODUCTION

The research conducted in this paper falls under the category of combinatorial generation. The area is so important to computer science that Knuth has dedicated over 400 pages to the subject in his upcoming volume of *The Art of Computer Programming* [25, 26]. The research area is applicable whenever it is necessary to efficiently consider every possible object of a particular type, such as binary strings of length n, permutations of $\{1, 2, \ldots, n\}$, binary trees with n nodes, linear extensions of a partially-ordered set, spanning trees of a directed graph, or perfect elimination orders of a chordal graph.

The most useful results in combinatorial generation tend to have a mathematical aspect and an algorithmic aspect. For example, two of the most well-known results in combinatorial generation are the binary reflected Gray code [22] and the de Bruijn cycle [20]. Both results provide a clever order for the binary strings of length n. The binary reflected Gray code provides an order in which each successive string can be obtained from the previous by changing the value of a single bit, while de Bruijn cycles provide an order in which each successive string can be obtained from the previous by removing the rightmost bit and inserting a new leftmost bit. In general, the mathematical aspect of combinatorial generation involves the discovery of a *minimal-change order*. A minimal-change order is an order in which each successive object can be obtained from the previous by making one small modification of a certain type. The existence or non-existence of minimal-change orders depend upon the type of object and the type of modification. New results in this area are often quite difficult to find, but the results that are found tend to be elegant and simple. The mathematical question answered in this paper is the following.

Question 1. Can multiset permutations be ordered so that each permutation is a prefix shift of the previous permutation?

Given a string $\boldsymbol{s} = s_1 s_2 \ldots s_n$, a *prefix shift of length k* is denoted by $\sigma_k(\boldsymbol{s})$ and is the result of moving s_k into the leftmost position. That is,

$$\sigma_k(\boldsymbol{s}) = s_k s_1 \ldots s_{k-1} s_{k+1} \ldots s_n.$$

For example, the order listed below affirmatively answers Question 1 for the multiset $\{1, 1, 2, 4\}$

$$421\underline{1}, \ 1\underline{4}21, \ 412\underline{1}, \ 141\underline{2}, \ 114\underline{2}, \ 411\underline{2}, \ 241\underline{1}, \ 1\underline{2}41, \ 214\underline{1}, \ 121\underline{4}, \ 112\underline{4}, \ 211\underline{4}.$$

In particular, each successive permutation is obtained from the previous permutation by moving the underlined symbol into the leftmost position. (In this case a prefix shift also changes the last permutation into the first permutation, and so the listed order is considered a *circular* minimal-change order with respect to prefix shifts.)

The algorithmic aspect of combinatorial generation involves the creation of efficient algorithms for generating all possible objects of a particular type. Algorithms of this type can be analyzed using a producer-consumer model. For example, a particular program, or *consumer*, may need to consider all binary trees with n nodes. To do this, the consumer begins by creating a single binary tree with n nodes. When the consumer is finished considering this binary tree, it asks the combinatorial generation algorithm, or *producer*, to modify the binary tree into the next binary tree. This process continues until the producer notifies the consumer that every binary tree with n nodes has been considered. Using this model the combinatorial generation algorithm runs in *constant amortized time* (CAT) if, on average, it can perform its modifications in $O(1)$-time. Furthermore, if every modification can be done in $O(1)$-time, then the combinatorial generation algorithm is said to be *loopless*. (In both cases the hidden constant must be independent of the size of the object being modified.) The term loopless was first introduced by Ehrlich [21]. In terms of memory consumption, *additional variables* refer to variables that are used only by the producer. In particular, the additional variables do not include those used by the consumer to store the current object. Using this terminology the algorithmic question answered in this paper is the following.

Question 2. Can the permutations of any multiset be generated by a loopless algorithm that uses only a constant number of additional variables?

This paper shows that the answers to both questions are related, and are both yes.

1.1. **Applications.** Efficient algorithms for generating multiset permutations have a number of applications. If the multiset is simply a set, then applications include communication in point-to-point multiprocessor networks [19]. If the multiset's corresponding set contains only two elements, then applications include cryptography (where orders have been implemented in hardware at NSA), genetic algorithms, software and hardware testing, statistical computation (e.g., for the bootstrap, and Diaconis and Holmes [16]).

Minimal-change orders also tend to have diverse applications. For example, the binary reflected Gray code was designed at Bell Labs for telephone systems, but has since found applications in information and communication technology, analog-to-digital conversion, error correction, and decreased power consumption in hand-held devices. It has also been used in the CODACON spectrometer, and appears in research titles ranging from measurement and instrumentation [1] to quantum chemistry [18]. The minimal-change order discovered in this paper has potential applications in genetics since prefix shifts are akin to splicing segments of genetic material.

1.2. **Previous Results.** The history of combinatorial generation is rich and fascinating. The reader is directed towards [26, 25, 27] and [29] for excellent treatments of the subject. In terms of minimal-change orders for multiset permutations, the most relevant previous results are found in [19, 8, 4] and [3] (with earlier version [2]). The first three provide minimal-change orders for set permutations by prefix shifts, while the fourth provides a

minimal-change order for the permutations of multisets whose corresponding set contains two elements. However, generalizing these orders to the permutations of multisets has so far proved impossible. For example, the order found in [4] uses only prefix shifts of length n and $n-1$, and thereby creates the first explicit *shorthand universal cycle for set permutations*, which is essentially a de Bruijn cycle for set permutations (c.f. [12, 23]). Unfortunately, the existence of shorthand universal cycles for multiset permutations is still open.

Besides prefix shifts, another well-studied modification is an adjacent transposition. Given a string $\boldsymbol{s} = s_1 s_2 \ldots s_n$, an *adjacent transposition* results in a string of the form

$$s_1 \ldots s_{i-1} s_{i+1} s_i s_{i+2} \ldots s_n.$$

The beautiful (and oft-rediscovered) Steinhaus-Johnson-Trotter order [24] proves that a minimal-change order using adjacent transpositions exists for set permutations. On the other hand, the same is not true for multiset permutations, with [28] providing exact conditions for their existence. There are also minimal-change orders for multiset permutations using (non-adjacent) transpositions [6].

Many efficient algorithms for generating multiset permutations can be found in the literature including [31, 32]. However, no previous algorithm is loopless while using a constant number of additional variables. Loopless algorithms using a linear number of additional variables (with respect to the size of the multiset) do exist for implementations that store the current permutation in an array [14] (which answered a conjecture in [31]) or a linked list [17], however, both of these algorithms are decidedly more complicated than the algorithm presented in this paper. (In fact, Algorithm 1 (on page 8) is one of the simplest ever created for generating multiset permutations, and can be implemented without reading the remainder of this document.) Loopless algorithms also exist for generating linear extensions of partially ordered sets, which include multiset permutations as a special case [7, 15, 13].

1.3. **Outline.** Section 2 introduces notation and defines a simple lexicographic order for $\mathbb{M}_{\mathsf{E}}$. This lexicographic order is modified in Section 2.3 to create the new minimal-change order that answers Question 1. This new minimal-change order is then used in Section 3 to create the algorithm that answers Question 2. Besides answering the two main questions, Section 3.1 also provides an interesting analysis of the algorithm when applied to set permutations. Specifically, it is shown that the average length of the prefix shifts performed in the minimal-change order is less than 3. Section 4 concludes with open problems.

2. MULTISET PERMUTATIONS

2.1. **Notation and Conventions.** There are two main ways to describe the elements of a multiset. Every element in the multiset can be stated, or every element in its corresponding set can be stated along with its frequency. For example, the multiset $\{1, 1, 2, 4, 4, 4\}$ can be described by its elements $1, 1, 2, 4, 4, 4$ or by noting that it contains two copies of 1, one copy of 2, and three copies of 4. Every multiset in this paper is assumed to contain integer values, and so it is also possible to specify the cumulative frequencies of the elements less than or equal to a certain value. For example, $\{1, 1, 2, 4, 4, 4\}$ can also be specified by noting that it contains two elements ≤ 1, three elements ≤ 2, and six elements ≤ 4.

Throughout this document, E is used to represent a multiset and it will be assumed that E contains n elements, and its corresponding set has m elements. Furthermore, e_i is used for the ith smallest element in the multiset (for $1 \leq i \leq n$), and d_i is used for the ith smallest element in its corresponding set (for $1 \leq i \leq m$) with f_i giving the frequency of d_i. For

cumulative frequencies, $\overrightarrow{f_t}$ represents the number of elements in E that are less than or equal to d_t. Thus, if $\mathsf{E} = \{1, 1, 2, 4, 4, 4\}$ then

$$n = 6 \qquad \{e_1, e_2, e_3, e_4, e_5, e_6\} = \{1, 1, 2, 4, 4, 4\} \qquad f_1, f_2, f_3 = 2, 1, 3$$

$$m = 3 \qquad \{d_1, d_2, d_3\} = \{1, 2, 4\} \qquad \overrightarrow{f_1}, \overrightarrow{f_2}, \overrightarrow{f_3} = 2, 3, 6.$$

Given multiset E, the set of permutations of E is denoted by $\mathbb{M}_\mathsf{E}$. Each permutation is treated as a string so that certain string-related concepts are natural. For example, the terms *prefix* and *suffix* are used along with the adjective *proper* to describe a prefix or suffix that is not equal to the entire string. *Concatenation* is used to add symbols to the end of strings, and is represented by adjacent symbols or by "$\cdot$". In particular, concatenation can be applied to the end of every string in a list. For example, given list $\mathcal{L} = 42, 24$, then

$$\mathcal{L} \cdot 11 = 42 \cdot 11, \ 24 \cdot 11 = 4211, \ 2411.$$

While the term concatenation and $\cdot$ are reserved for making strings longer, the term *append* and "," are reserved for making lists longer. For example, if $\mathcal{L}_1 = 1222, 2122$ and $\mathcal{L}_2 = 2212$ then

$$\mathcal{L}_1, \ \mathcal{L}_2, \ 2221 = 1222, \ 2122, \ 2212, \ 2221.$$

The symbol $\bigoplus$ is also used for automating the process of appending lists. If the initial value of the index variable is below the $\bigoplus$ symbol, then the index variable counts upwards (as usual). If the initial value of the index variable is above the $\bigoplus$ symbol, then index variable counts downwards. For example,

$$\bigoplus_{i=1}^{3} \mathcal{L}_i = \mathcal{L}_1, \ \mathcal{L}_2, \ \mathcal{L}_3 \qquad\qquad \bigoplus_{1}^{i=3} \mathcal{L}_i = \mathcal{L}_3, \ \mathcal{L}_2, \ \mathcal{L}_1.$$

Lists are used to in this paper for describing orders of $\mathbb{M}_\mathsf{E}$. Thus, lists will be constructed to contain each string in $\mathbb{M}_\mathsf{E}$ exactly once.

The symbol $\setminus$ is used to represent set difference in terms of multisets. Furthermore, the symbol $-$ is used as a shorthand for removing all of the symbols of a string from a multiset. For example,

$$\{1, 2, 2, 3, 3, 3\} - 223 = \{1, 2, 2, 3, 3, 3\} \setminus \{2, 2, 3\} = \{1, 3, 3\}.$$

The *tail* of a multiset is the string containing its elements in non-increasing order.

Definition 2.1 (Tail).

$$\mathsf{tail}(\mathsf{E}) = e_n e_{n-1} \cdots e_1$$

This moniker is used so that $\mathsf{tail}(\mathsf{E})$ can be compared with other strings called *scuts*, which are defined in the next section.

2.2. **Lexicographic Order.** The *ascending co-lexicographic order*, or simply *co-lex order*, for multiset permutations orders the strings of $\mathbb{M}_\mathsf{E}$ in increasing lexicographic order when the strings are read from right-to-left. $\mathcal{L}_\mathsf{E}$ is used to denote co-lex order for $\mathbb{M}_\mathsf{E}$. For example,

$$\mathcal{L}_{\{1,1,2,4\}} = 421\mathbf{1}, \ 241\mathbf{1}, \ 412\mathbf{1}, \ 142\mathbf{1}, \ 214\mathbf{1}, \ 124\mathbf{1}, \ 411\mathbf{2}, \ 141\mathbf{2}, \ 114\mathbf{2}, \ 2114, \ 1214, \ 1124.$$

Recursively, the most natural way to describe $\mathcal{L}_\mathsf{E}$ is by the value of the rightmost symbol which appear in bold above.

Definition 2.2 (Co-lex order by rightmost symbol)**.**

$$\mathcal{L}_\mathsf{E} = \bigoplus_{i=1}^{m} \mathcal{L}_{\mathsf{E}-d_i} \cdot d_i$$

where $\mathcal{L}_{\{a\}} = a$ for the single symbol a.

$\mathcal{L}_\mathsf{E}$ can also be recursively defined in a less natural, but ultimately more useful manner. In English, a *scut* is a short thick tail found on an animal such as a deer or rabbit. In the context of this paper, every multiset permutation that is not equal to $\mathsf{tail}(\mathsf{E})$ has some shortest suffix that is not a suffix of $\mathsf{tail}(\mathsf{E})$; this suffix is referred to as the *scut* of the permutation. For example, the scuts are bolded in the restatement of $\mathcal{L}_{\{1,1,2,4\}}$ below

$\mathcal{L}_{\{1,1,2,4\}} = 4211, \mathbf{2411}, 41\mathbf{21}, 14\mathbf{21}, 21\mathbf{41}, 12\mathbf{41}, 411\mathbf{2}, 141\mathbf{2}, 114\mathbf{2}, 2114, 1214, 1124.$

Notice that the scuts appear in the following order: 411, 21, 41, 2, 4.

Every scut can be written as $d_j e_k e_{k-1} \ldots e_1$ where $d_j > e_{k+1}$. This idea is formalized by the following definition.

Definition 2.3 (Scut)**.** If $d_j > e_{k+1}$ then

$$\mathsf{scut}(j,k) = d_j e_k e_{k-1} \cdots e_1.$$

(Note: The condition $d_j > e_{k+1}$ is equivalent to $2 \le j \le m$ and $0 \le k \le \overrightarrow{f_{j-1}} - 1$, and this alternate expression is more useful when providing Definition 2.4.)

$\mathcal{L}_\mathsf{E}$ can be viewed as the order that starts with $\mathsf{tail}(\mathsf{E})$, and then orders the remaining strings by decreasing values of k followed by increasing values of j, with respect to $\mathsf{scut}(j,k)$. (The base case occurs when $m = 1$ and $\mathcal{L}_\mathsf{E} = \mathsf{tail}(\mathsf{E})$.) For example, recall that in the list $\mathcal{L}_{\{1,1,2,4\}}$ given above, the scuts appeared in the following order: 411, 21, 41, 2, 4. In other words, they are ordered by decreasing length and then by increasing leftmost symbol. The minimal-change order defined in the next section is a slight perturbation of this alternate view of co-lex order. In particular, $\mathsf{tail}(\mathsf{E})$ is ordered last (instead of first), and then the remaining strings are ordered by increasing values of j followed by decreasing values of k (instead of vice-versa).

2.3. **Cool-lex Order.** This section defines a new order of $\mathbb{M}_\mathsf{E}$. The order is referred to as the *cool-lex order* for multiset permutations and is denoted by $\mathcal{C}_\mathsf{E}$. The term *cool-lex* is a modification of the term *co-lex* and is further justified at the end of this section.

Definition 2.4 (Cool-lex order by scut)**.**

$$\mathcal{C}_\mathsf{E} = \bigoplus_{j=2}^{m} \bigoplus_{0}^{k=\overrightarrow{f_{j-1}}-1} \mathcal{C}_{\mathsf{E}-\mathsf{scut}(j,k)} \cdot \mathsf{scut}(j,k), \ \mathsf{tail}(\mathsf{E})$$

The base case occurs when $m = 1$ and $\mathcal{C}_\mathsf{E} = \mathsf{tail}(\mathsf{E})$.

For example, $\mathcal{C}_{\{1,1,2,4\}}$ appears below with its scuts in bold

$\mathcal{C}_{\{1,1,2,4\}} = 14\mathbf{21}, 41\mathbf{21}, 141\mathbf{2}, 114\mathbf{2}, 411\mathbf{2}, \mathbf{2411}, 12\mathbf{41}, 21\mathbf{41}, 1214, 1124, 2114, 4211.$

Notice that the scuts appear in the following order: 21, 2, 411, 41, 4. In other words, they are ordered by increasing leftmost symbol and then by decreasing length. A more detailed illustration of the recursive structure(s) of $\mathcal{L}_\mathsf{E}$ and $\mathcal{C}_\mathsf{E}$ appears in Figure 1 for $\mathsf{E} = \{1, 1, 2, 2, 3\}$.

(i)	(ii)	(iii)	(iv)	(v)
	1 32211	32211	13221	21
	1 23211	$\mathcal{L}_{\{2\}}$3211	31221	21
	1 22311	$\mathcal{L}_{\{2,2\}}$311	23121	21 $\mathcal{C}_{\{1,2,3\}}$
	1 32121	21	12321	21
	1 23121	21	21321	21
$\mathcal{L}_{\{1,2,2,3\}}$	1 31221	21 $\mathcal{L}_{\{1,2,3\}}$	32121	21
	1 13221	21	13212	2
	1 21321	21	31212	2
	1 12321	21	13122	2
	1 22131	31	11322	2
	1 21231	31 $\mathcal{L}_{\{1,2,2\}}$	31122	2
	1 12231	31	23112	2 $\mathcal{C}_{\{1,1,2,3\}}$
	2 32112	2	12312	2
	2 23112	2	21312	2
	2 = 31212 =	2	12132 =	2
	2 13212	2	11232	2
	2 21312	2	21132	2
$\mathcal{L}_{\{1,1,2,3\}}$	2 12312	2 $\mathcal{L}_{\{1,1,2,3\}}$	32112	2
	2 31122	2	23211	3211 $\mathcal{C}_{\{2\}}$
	2 13122	2	22311	311 $\mathcal{C}_{\{2,2\}}$
	2 11322	2	12231	31
	2 21132	2	21231	31 $\mathcal{C}_{\{1,2,2\}}$
	2 12132	2	22131	31
	2 11232	2	12213	31
	3 22113	3	21213	3
	3 21213	3	12123	3
$\mathcal{L}_{\{1,1,2,2\}}$	3 12213	3 $\mathcal{L}_{\{1,1,2,2\}}$	11223	3 $\mathcal{C}_{\{1,1,2,2\}}$
	3 21123	3	21123	3
	3 12123	3	22113	3
	3 11223	3	32211	32211

co-lex order   cool-lex order

FIGURE 1. Column (ii) contains $\mathcal{L}_{\mathsf{E}}$ for $\mathsf{E} = \{1,1,2,2,3\}$ while columns (i) and (iii) illustrate its recursive structure by rightmost symbol and scut, respectively. Column (iv) contains $\mathcal{C}_{\mathsf{E}}$ for $\mathsf{E} = \{1,1,2,2,3\}$ while column (v) illustrates its recursive structure by scut.

Now it is time to describe how $\mathcal{C}_{\mathsf{E}}$ behaves on an iterative, or string-to-string, basis. To describe its behavior, the following definition is required.

Definition 2.5 ($\searrow$). If $\boldsymbol{s} = s_1 s_2 \ldots s_n$ is a string, then $\searrow(\boldsymbol{s})$ is the maximum value such that $s_{j-1} < s_j$ for all $2 \leq j \leq \searrow(\boldsymbol{s})$.

In other words, $\searrow(\boldsymbol{s})$ is the length of the longest non-increasing prefix. For example,

$$\searrow(55432413) = 5 \qquad \searrow(4532154) = 1 \qquad \searrow(33415312) = 2.$$

Now the iterative behavior of $\mathcal{C}_{\mathsf{E}}$ can be described by the function

$$\lhd : \mathbb{M}_{\mathsf{E}} \to \mathbb{M}_{\mathsf{E}}$$

which maps any string $\boldsymbol{s}$ into the string that follows $\boldsymbol{s}$ within $\mathcal{C}_{\mathsf{E}}$. (The symbol $\lhd$ was chosen because the operation uses prefix shifts to move a single symbol into the leftmost position.)

Definition 2.6 ($\lhd$). Let $\boldsymbol{s} = s_1 \cdots s_n$ and $i = \searrow(\boldsymbol{s})$. Then,

(1a)
(1b) $$\lhd(\boldsymbol{s}) = \begin{cases} \sigma_{i+1}(\boldsymbol{s}) & \text{if } i \leq n-2 \text{ and } s_{i+2} > s_i \\ \sigma_{i+2}(\boldsymbol{s}) & \text{if } i \leq n-2 \text{ and } s_{i+2} \leq s_i \\ \sigma_n(\boldsymbol{s}) & \text{otherwise (if } i \geq n-1) \end{cases}$$
(1c)

Notice that the definition is incredibly simple: To obtain the permutation that follows $\boldsymbol{s}$, either a prefix shift of length $i+1$ or $i+2$ or n will be performed, where $i = \diagdown(\boldsymbol{s})$. Moreover, at most two comparisons are needed to determine which length of prefix shift to perform. To illustrate the definition,

$$\lhd\,(6442313134) = \lhd\,(6442\cdot 31\cdot 3134) \qquad \lhd\,(6442343131) = \lhd\,(6442\cdot 34\cdot 3131)$$
$$= \lhd\,(64423\cdot 1\cdot 3134) \qquad\qquad = \lhd\,(6442\cdot 3\cdot 43131)$$
$$= 1644233134 \qquad\qquad\qquad = 3644243131$$

where the $\cdot$ are used to visually separate the prefix of length $i = \diagdown(\boldsymbol{s})$ and the symbols s_{i+1} and s_{i+2}. The reader can also verify that Definition 2.6 properly describes the string-by-string behavior found in Figure 1.

To describe multiple applications of $\lhd$ let $\lhd^0\,(\boldsymbol{s}) = \boldsymbol{s}$ and

$$\lhd^k\,(\boldsymbol{s}) = \lhd\left(\lhd^{k-1}\,(\boldsymbol{s})\right)$$

for all $k > 0$. For example, $\lhd^3\,(\boldsymbol{s}) = \lhd\,(\lhd\,(\lhd\,(\boldsymbol{s})))$, which would result in the third string following $\boldsymbol{s}$ within $\mathcal{C}_{\mathsf{E}}$. The main result of this section is stated in Theorem 2.7.

Theorem 2.7 (Equivalence of definitions for cool-lex order).

$$\mathcal{C}_{\mathsf{E}} = \bigoplus_{k=0}^{|\mathbb{M}_{\mathsf{E}}|-1} \lhd^k\,(\mathsf{tail}(\mathsf{E}))\,.$$

In other words, Theorem 2.7 proves that $\lhd$ does in fact provide an iterative description of $\mathcal{C}_{\mathsf{E}}$. The result is proven by induction on $n - f_1$ with details contained in the appendix. (In fact, a slightly stronger version of Theorem 2.7 is proven showing that $\lhd$ circularly generates $\mathcal{C}_{\mathsf{E}}$ in the sense that applying $\lhd$ to the last string in $\mathcal{C}_{\mathsf{E}}$ also results in the first string in $\mathcal{C}_{\mathsf{E}}$. This slight extension is used in the statement of Theorem 3.2 found in Section 3.1.) Since $\lhd$ always performs a single prefix shift, a simple corollary to Theorem 2.7 is that $\mathcal{C}_{\mathsf{E}}$ provides a constructive and affirmative answer to Question 1.

To conclude this section, it is noted that $\lhd$ actually provides a subtle generalization of the iterative rule for generating multiset permutations with $m = 2$ originally described in [3, 2]. The order in that paper is described as the *cool-lex order for combinations*, and so $\mathcal{C}_{\mathsf{E}}$ can be referred to as the *cool-lex order for multiset permutations*.

3. Algorithms

This section describes an algorithm that generates every multiset permutation for any specified multiset E. The algorithm is loopless, iterative (i.e. not recursive), uses a constant number of additional variables, and generates the permutations in the order given by $\mathcal{C}_{\mathsf{E}}$. (To be more precise, the algorithm generates the permutations in the order given by $\mathcal{C}_{\mathsf{E}}$, with the one exception that $\mathsf{tail}(\mathsf{E})$ is generated first instead of last.)

There is one hurdle in translating the iterative description of $\mathcal{C}_{\mathsf{E}}$ from Definition 2.6, into such an algorithm: For each successive permutation $\boldsymbol{s}$, the algorithm must determine the value of $i = \diagdown(\boldsymbol{s})$ in constant time, and without the use of any complex data structures. Fortunately, the value of i for the current permutation is strongly related to the value of i for the previous permutation.

Lemma 3.1. Let $\boldsymbol{s} = s_1 s_2 \cdots s_n$ and $\lhd\left(\boldsymbol{s}\right) = \boldsymbol{s}' = s_1' s_2' \ldots s_n'$. Then

(2a)

(2b)

$$\searrow\left(\lhd\left(\boldsymbol{s}\right)\right) = \begin{cases} 1 & \text{if } s_i' < s_2' \\ \searrow\left(\boldsymbol{s}\right) + 1 & \text{otherwise (if } s_1' \geq s_2') \end{cases}$$

In other words, the length of the longest non-increasing prefix is either increased by one, or is reset to the value of 1, after each application of $\lhd$, and the correct value can be determined by a single comparison. The proof of this lemma follows relatively easily from Definition 2.6, and is formally proven in the appendix.


Algorithm 1 Visits every permutation of integer multiset E. The $\mathtt{init(E)}$ call creates a singly linked list storing the elements of E in non-increasing order with *head*, *min*, and *inc* pointing to its first, second-last, and last nodes, respectively. All variables are pointers and the null pointer value is given by ϕ. As an example, if $\mathsf{E} = \{1, 1, 2, 4\}$ then the first three $\mathtt{visit(E)}$ calls will produce the configurations given below, where the left and right boxes of each node refer to its *value* and *next* fields, respectively. Note: If E is empty then $\mathtt{init(E)}$ should exit, and if E contains only one element then *min* need not be initialized.

$[\mathtt{head}, \mathtt{i}, \mathtt{afteri}] \leftarrow \mathtt{init(E)}$
$\mathtt{visit(head)}$
while $\mathtt{afteri}.next \neq \phi$ or $\mathtt{afteri}.value < \mathtt{head}.value$ **do**
  if $\mathtt{afteri}.next \neq \phi$ and $\mathtt{i}.value \geq \mathtt{afteri}.next.value$
    $\mathtt{beforek} \leftarrow \mathtt{afteri}$
  else
    $\mathtt{beforek} \leftarrow \mathtt{i}$
  end
  $\mathtt{k} \leftarrow \mathtt{beforek}.next$
  $\mathtt{beforek}.next \leftarrow \mathtt{k}.next$
  $\mathtt{k}.next \leftarrow \mathtt{head}$
  if $\mathtt{k}.value < \mathtt{head}.value$
    $\mathtt{i} \leftarrow \mathtt{k}$
  end
  $\mathtt{afteri} \leftarrow \mathtt{i}.next$
  $\mathtt{head} \leftarrow \mathtt{k}$
  $\mathtt{visit(head)}$
end
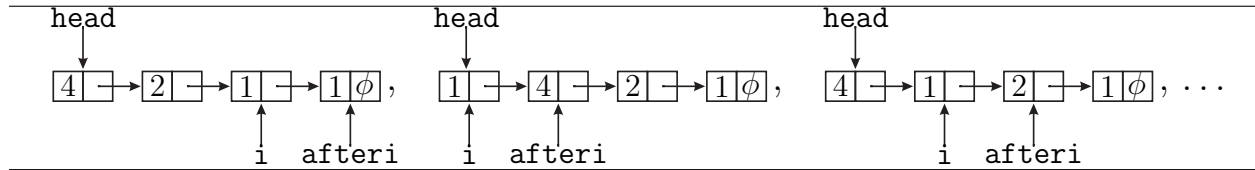
Now the specifics of the Algorithm 1 can be discussed. At the start of each iteration of the loop, there are three pointers that have significant meaning

- $\mathtt{head}$ points to the first node of the current permutation
- $\mathtt{i}$ points to the ith node of the current permutation
- $\mathtt{afteri}$ points to the $(i+1)$st node of the current permutation

where i is the value of $\searrow$ applied to the current permutation. In order to apply the prefix shift of length k, the following two pointers are then used

- k points to the kth node of the current permutation
- beforek points to the $(k-1)$st node of the current permutation.

The unwritten visit(head) is called whenever a new permutation is pointed to by head. Within a producer-consumer paradigm, the visit(head) call represents the passing of control back to the consumer. Within this paradigm the work in init(E) would also be done by the consumer. Algorithm 1 is loopless because its loop contains a constant number of elementary instructions including a single visit(head) call. It also uses a constant number of additional variables, namely i, afteri, k, and beforek (although simple modifications can reduce the number of additional variables from four to two). Therefore, the algorithm provides an affirmative answer to Question 2.

The condition on the loop ensures that the algorithm continues until it would generate tail(E). Instead of generating tail(E) as a separate case after the loop terminates, the algorithm is slightly optimized and condensed to instead initialize the linked list to tail(E) and this permutation is then visited first instead of last. Finally, this slight alteration causes the values of i and afteri to deviate slightly from their stated meaning on the first iteration.

3.1. **Analysis.** Let $\langle n \rangle$ represent the set $\{1, 2, \ldots, n\}$. In this section the average length of the prefix shift performed within $\mathcal{C}_{\langle n \rangle}$ is analyzed. Remarkably, the value is found to be less than three. To formalize the result, let $\pi(\boldsymbol{s})$ be the length of the prefix shift performed in $\mathcal{C}_E$ when making the transition from $\boldsymbol{s}$ to $\lhd(\boldsymbol{s})$. (Notice that the statement of Theorem 3.2 includes the application of $\lhd$ that maps the last permutation of $\mathcal{C}_{\langle n \rangle}$ back into the first permutation of $\mathcal{C}_{\langle n \rangle}$.)

Theorem 3.2.

$$\sum_{i=0}^{n!} i \cdot \pi\big(\lhd^i\,(\mathsf{tail}(\langle n \rangle))\big) < 3 \cdot n!$$

The proof of this theorem requires two lemmas. The first lemma is an interesting combinatorial identity, and the second counts the number of permutations that require a prefix shift of length k while generating $\mathcal{C}_{\langle n \rangle}$. Formal proofs of these lemmas, and the theorem, appear in the appendix.

Lemma 3.3 (Combinatorial identity)**.**

$$2(n^2 - n + 1) + \sum_{k=2}^{n-1} k \cdot \frac{n! \cdot 2 \cdot (k^2 - k - 1)}{(k+1)!} = 3 \cdot n!$$

when $n \geq 2$.

Lemma 3.4 (Number of strings for each value of $\pi(\boldsymbol{s})$)**.**

$$
\begin{aligned}
\text{(3a)} \\
\text{(3b)}
\end{aligned}
\qquad
|\{\boldsymbol{s} : \pi(\boldsymbol{s}) = k\}| =
\begin{cases}
\dfrac{n! \cdot 2 \cdot (k^2 - k - 1)}{(k+1)!} & \text{if } 2 \leq k < n \\[2ex]
2n - 2 & \text{otherwise (if } k = n)
\end{cases}
$$

4. OPEN PROBLEMS

Several interesting problems arise from the material in this paper.

Question 3. Theorem 3.2 provides an extremely low value for the average length of prefix shift performed within $\mathcal{C}_{\langle n \rangle}$. Is it possible that $\mathcal{C}_{\langle n \rangle}$ provides the lowest possible average length of prefix shift when considering all possible orders of $\mathbb{M}_{\langle n \rangle}$ using prefix shifts? Furthermore, can a similar result be proven when an arbitrary multiset E replaces $\langle n \rangle$?

Question 4. Experimentally, the average value of $\pi(\boldsymbol{s})$ for $\mathbb{M}_{\mathsf{E}}$ depends only upon the multiset of frequencies of E. For example, the average value of $\pi(\boldsymbol{s})$ is the same in $\mathcal{C}_{\{1,1,2,2,3,3,3,4\}}$ as it is in $\mathcal{C}_{\{1,1,1,2,3,3,4,4\}}$ since the multiset of frequencies is $\{1, 2, 2, 3\}$ in both cases. Can this result be proven?

Question 5. An important part of combinatorial generation is *ranking*, which provides a mapping between each string and its position in the order. Due to the straight-forward nature of Definition 2.4 it seems plausible that the strings in $\mathcal{C}_{\mathsf{E}}$ could be ranked efficiently. How fast can they be ranked?

Question 6. The iterative rule for combinations in cool-lex order was modified slightly to obtain a minimal-change order for balanced parentheses strings and binary trees [5]. Can the new generalized rule be modified to obtain minimal-change orders for other interesting combinatorial objects such as fixed-content necklaces? Necklaces are equivalence classes of strings under rotation, and necklaces with fixed-content E are a subset of $\mathbb{M}_{\mathsf{E}}$. Currently no loopless algorithm is known for fixed-content necklaces, although an efficient CAT algorithm does exist [30]. Efficient algorithms for generating fixed-density necklaces [9, 10] and unlabeled necklaces [11] also exist.

Question 7. It has been observed that within $\mathcal{C}_{\mathsf{E}}$ the strings with prefix d_m appear in the same relative order as they do in $\mathcal{C}_{\mathsf{E}-d_m}$. What properties of this type can be proven?

REFERENCES

[1] G. Betta and A. Pietrosanto and A. Scaglione. A Gray-code-based fiber optic liquid level transducer. *IEEE Transactions on Instrumentation and Measurement*, 47(1):174–178, February 1998.

[2] F. Ruskey and A. Williams. Generating combinations by prefix shifts. In *COCOON '05: Computing and Combinatorics, 11th Annual International Conference*, volume 3595 of *Lecture Notes in Computer Science*, Kunming, China, 2005. Springer-Verlag.

[3] F. Ruskey and A. Williams. The coolest way to generate combinations. *Discrete Mathematics*, (in press), 2008.

[4] F. Ruskey and A. Williams. An explicit universal cycle for the (n-1)-permutations of an n-set. *ACM Transactions on Algorithms*, (submitted), 2008.

[5] F. Ruskey and A. Williams. Generating balanced parentheses and binary trees by prefix shifts. In *CATS '08: Fourteenth Computing: The Australasian Theory Symposium*, volume 77 of *CRPIT*, Wollongong, Australia, 2008. ACS.

[6] C. W. Ko and F. Ruskey. Generating permutations of a bag by interchanges. *Information Processing Letters*, 41:263–269, 1992.

[7] G. Pruesse and F. Ruskey. Generating linear extensions fast. *SIAM Journal on Computing*, 23(2):373–386, April 1994.

[8] M. Jiang and F. Ruskey. Determining the Hamilton-connectedness of certain vertex-transitive graphs. *Discrete Mathematics*, 133:159–170, 1994.

[9] F. Ruskey and J. Sawada. An efficient algorithm for generating necklaces with fixed density. In *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on discrete algorithms*, pages 729–758, Baltimore, Maryland, United States, 1999. Society for Indusstrial and Applied Mathematics.

[10] F. Ruskey and J. Sawada. An efficient algorithm for generating necklaces with fixed density. *SIAM Journal of Computing*, 29(2):671–684, 1999.

[11] F. Ruskey and J. Sawada. A fast algorithm to generate unlabeled necklaces. In *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on discrete algorithms*, pages 256–262, San Francisco, California, United States, 2000. Society for Indusstrial and Applied Mathematics.

[12] F. Chung and P. Diaconis and R. Graham. Universal cycles for combinatorial structures. *Discrete Mathematics*, 110, 1992.

[13] J. F. Korsh and P. S. LaFollette. Loopless generation of linear extensions of a poset. *Order*, 18(2):115–126, 2002.

[14] J. F. Korsh and P. S. LaFollette. Loopless array generation of multiset permutations. *The Computer Journal*, 47(5):612–621, 2004.

[15] E. R. Canfield and S. G. Williamson. A loop-free algorithm for generating the linear extensions of a poset. *Order*, 12(1):57–75, 1995.

[16] P. Diaconis and S. Holmes. Gray codes for randomization procedures. *Statistical Computing*, 4:207–302, 1994.

[17] J. F. Korsh and S. Lipschutz. Generating multiset permutations in constant time. *Journal of Algorithms*, 25:321–335, 1997.

[18] R. Sawae and T. Sakata and M. Tei and K. Takarabe and Y. Manmoto. Gray code and the initialization problem of NMR quantum computers. *International Journal of Quantum Chemistry*, 95:558–560, 2003.

[19] P. F. Corbett. Rotator graphs: An efficient topology for point-to-point multiprocessor networks. *IEEE Transactions on Parallel and Distributed Systems*, 3:622–626, 1992.

[20] N.G. de Bruijn. A combinatorial problem. *Koninkl. Nederl. Acad. Wetensch. Proc. Ser A*, 49:758–764, 1946.

[21] G. Ehrlich. Loopless algorithms for generating permutations, combinations and other combinatorial configurations. *Journal of the ACM*, 20:500–513, 1973.

[22] F. Gray. Pulse code communication. *U.S. Patent 2,632,058*, 1947.

[23] J. R. Johnson. Universal cycles for permutations. *Discrete Mathematics*, (in press), 2008.

[24] S. M. Johnson. Generation of permutations by adjacent transpositions. *Mathematics of Computation*, 17:282–285, 1963.

[25] D. E. Knuth. *The Art of Computer Programming*, volume 4 fasicle 3 - Generating All Combinations and Partitions. Updated 10/02/2008.

[26] D. E. Knuth. *The Art of Computer Programming*, volume 4 fasicle 2 - Generating All Tuples and Permutations. Updated 10/02/2008.

[27] D. E. Knuth. *The Art of Computer Programming*, volume 4 fasicle 4 - Generating All Trees. Updated 10/02/2008.

[28] F. Ruskey. Generating linear extensions of posets by transpositions. *Journal of Combinatorial Theory (B)*, 54:77–101, 1992.

[29] C. Savage. A survey of combinatorial gray codes. *SIAM Review*, 39(4):605–629, 1997.

[30] J. Sawada. A fast algorithm to generate necklaces with fixed-content. *Theoretical Computer Science*, 1-3(301):477–489, 2003.

[31] T. Takaoka. An O(1) time algorithm for generating multiset permutations. In *ISAAC '99: Algorithms and Computation, 10th International Symposium*, volume 1741 of *Lecture Notes in Computer Science*, pages 237–246, Chennai, India, 1999. Springer.

[32] V. Vajnovszki. A loopless algorithm for generating the permutations of a multiset. *Theoretical Computer Science*, 2(307):415–431, 2003.

APPENDIX A. PROOFS

This appendix contains the proofs that were omitted in the body of the text. The proofs are presented in the same relative order, and under an appropriate section header.

A.1. **Cool-lex Order.** When proving the results in this section it is often necessary to refer to a second multiset. When this is necessary E' denotes the second multiset, and $'$ is placed above every other symbol to denote that it refers to E' and not E. For example, if $\mathsf{E} = \{1, 1, 2, 4\}$ and $\mathsf{E}' = \mathsf{E} - \{1, 2\}$ then the following values are obtained.

$\mathsf{E} = \{1,1,2,4\}$					$\mathsf{E}' = \{1,4\}$			
$n = 4$	$m = 3$				$n' = 2$	$m' = 2$		
$e_1 = 1$	$d_1 = 1$	$f_1 = 2$	$\overrightarrow{f_1} = 2$		$e_1' = 1$	$d_1' = 1$	$f_1' = 1$	$\overrightarrow{f_1}' = 1$
$e_2 = 1$	$d_2 = 2$	$f_2 = 1$	$\overrightarrow{f_2} = 3$		$e_2' = 2$	$d_2' = 4$	$f_2' = 1$	$\overrightarrow{f_2}' = 2$
$e_3 = 2$	$d_3 = 4$	$f_3 = 1$	$\overrightarrow{f_3} = 4$					
$e_4 = 4$								

Instead of directly proving that $\lhd$ iteratively generates $\mathcal{C}_\mathsf{E}$, it is more convenient to equivalently prove that the operation that is inverse to $\lhd$ iteratively generates the $\mathcal{C}_\mathsf{E}$ in reverse. To define the the reverse of $\mathcal{C}_\mathsf{E}$, one need only move $\mathsf{tail}(\mathsf{E})$ from the last string to the first string, and reverse the indices of each $\bigoplus$. The symbol $\mathcal{R}_\mathsf{E}$ will be used to represent this reversed list. It is also useful to explicitly add the base cases into the definition.

Definition A.1 (Reverse cool-lex order by scut).

$$
\mathcal{R}_\mathsf{E} = \begin{cases}
\varepsilon & \text{if } m = 0 \\
d_1^{f_1} & \text{if } m = 1 \\
\mathsf{tail}(\mathsf{E}), \displaystyle\bigoplus_{2}^{j=m} \bigoplus_{k=0}^{\overrightarrow{f_{j-1}}-1} \mathcal{R}_{\mathsf{E}-\mathsf{scut}(j,k)} \cdot \mathsf{scut}(j, k) & \text{otherwise.}
\end{cases}
$$

(4a) if $m = 0$
(4b) if $m = 1$
(4c) otherwise.

The inverse of $\lhd (\boldsymbol{s})$ is represented by $\rhd (\boldsymbol{s})$ and its statement appears below.

Definition A.2 ($\rhd$). Let $\boldsymbol{s} = s_0 s_1 \cdots s_{n-1}$ and $i = \searrow (s_1 s_2 \cdots s_{n-1})$. Then,

$$
\rhd (\boldsymbol{s}) = \begin{cases}
s_1 \ldots s_i \cdot s_0 \cdot s_{i+1} \ldots s_{n-1} & \text{if } s_0 > s_i \\
s_1 \ldots s_{i+1} \cdot s_0 \cdot s_{i+2} \ldots s_{n-1} & \text{if } s_0 \le s_i \\
s_1 \ldots s_{n-1} \cdot s_0 & \text{if } i = \infty
\end{cases}
$$

(5a)
(5b)
(5c)

It is easy to verify that these are in fact the inverse operations of the σ_{i+1}, σ_{i+2}, and σ_n found in the definition of $\lhd$.

Now the equivalent version of Theorem 2.7 can be proven over the span of several lemmas. The first lemma formally proves the values of the first and last strings within $\mathcal{R}_\mathsf{E}$.

Lemma A.3 (Boundary strings for multiset permutations in cool-lex order). When $n > 0$,

$$
(6) \qquad first(\mathcal{R}_\mathsf{E}) = \mathsf{tail}(\mathsf{E})
$$
$$
(7) \qquad last (\mathcal{R}_\mathsf{E}) = d_1 \cdot \mathsf{tail}(\mathsf{E} - \{d_1\})
$$
$$
(8) \qquad = e_1 \cdot \mathsf{tail}(\mathsf{E} - \{e_1\}).
$$

Proof. The first string equation in (6) is an immediate consequence of Definition A.1, and it holds even when $n = 0$. For the last string, notice that (7) is correct when $m = 1$ by the following derivation that uses (4b),

$$last\,(\mathcal{R}_{\mathsf{E}}) = last\left(d_1^{f_1}\right) = d_1^{f_1} = d_1 \cdot d_1^{f_1-1} = d_1 \cdot \mathsf{tail}(\mathsf{E} - \{d_1\})$$

as desired. Now we prove that (7) is correct by induction on n. The base case, when $n = 1$, is already proven because $n = 1$ implies $m = 1$. So assume that (7) is correct when $n \leq x$ for some $x \geq 1$, and then let us prove that (7) is correct when $n = x + 1$. Again, if $m = 1$ then (7) is correct without using induction. Otherwise, if $m \geq 2$ then by (4c),

$$last\,(\mathcal{R}_{\mathsf{E}}) = last\left(\mathsf{tail}(\mathsf{E}), \bigoplus_{2}^{j=m} \bigoplus_{k=0}^{\overrightarrow{f_{j-1}}-1} \mathcal{R}_{\mathsf{E}-\mathsf{scut}(j,k)} \cdot \mathsf{scut}(j,k)\right)$$

$$= last\left(\bigoplus_{2}^{j=m} \bigoplus_{k=0}^{\overrightarrow{f_{j-1}}-1} \mathcal{R}_{\mathsf{E}-\mathsf{scut}(j,k)} \cdot \mathsf{scut}(j,k)\right)$$

$$= last\left(\bigoplus_{k=0}^{\overrightarrow{f_1}-1} \mathcal{R}_{\mathsf{E}-\mathsf{scut}(2,k)} \cdot \mathsf{scut}(2,k)\right)$$

$$= last\left(\bigoplus_{k=0}^{f_1-1} \mathcal{R}_{\mathsf{E}-\mathsf{scut}(2,k)} \cdot \mathsf{scut}(2,k)\right)$$

$$= last\left(\mathcal{R}_{\mathsf{E}-\mathsf{scut}(2,f_1-1)} \cdot \mathsf{scut}(2,f_1-1)\right)$$

$$= last\left(\mathcal{R}_{\mathsf{E}-\mathsf{scut}(2,f_1-1)}\right) \cdot \mathsf{scut}(2,f_1-1)$$

$$= last\left(\mathcal{R}_{\mathsf{E}-\mathsf{scut}(2,f_1-1)}\right) \cdot d_2 \cdot e_{f_1}e_{f_1-1}\cdots e_2.$$

Now let $\mathsf{E}' = \mathsf{E} - \mathsf{scut}(2, f_1 - 1)$ and recall that, by convention, $'$ is used to refer to quantities relating to E' instead of E. Now let us derive the value of $last\,(\mathcal{R}_{\mathsf{E}'})$

$$last\,(\mathcal{R}_{\mathsf{E}'}) = d_1' \cdot \mathsf{tail}(\mathsf{E}' - \{d_1'\})$$
$$= d_1 \cdot e_n e_{n-1} \cdots e_{f_1+2}$$

In the above derivation, the first equality follows by our inductive assumption since $0 < n' \leq x$, while the second equality follows from the fact that $d_1' = d_1$ (since one copy of the smallest element in E is not contained in $\mathsf{scut}(2, f_1 - 1)$) and the fact that

$$\mathsf{tail}(\mathsf{E}' - \{d_1'\}) = \mathsf{tail}(\mathsf{E} - \mathsf{scut}(2, f_1 - 1) - \{d_1'\})$$
$$= \mathsf{tail}(\mathsf{E} - \mathsf{scut}(2, f_1 - 1) - \{d_1\})$$
$$= \mathsf{tail}(\mathsf{E} - \{d_2\} - \{\underbrace{d_1, \ldots, d_1}_{f_1 \text{ copies}}\})$$
$$= \mathsf{tail}(\mathsf{E} - \{e_{f_1+1}, e_{f_1}, \ldots, e_1\})$$
$$= e_n e_{n-1} \cdots e_{f_1+2}.$$

Therefore, we can continue the unfinished derivation as follows,

$$
\begin{aligned}
&= last\left(\mathcal{R}_{\mathsf{E-scut}(2, f_1-1)}\right) \cdot d_2 \cdot e_{f_1} e_{f_1-1} \cdots e_2 \\
&= last\left(\mathcal{R}_{\mathsf{E}'}\right) \cdot d_2 \cdot e_{f_1} e_{f_1-1} \cdots e_2 \\
&= d_1 \cdot e_n e_{n-1} \cdots e_{f_1+2} \cdot d_2 \cdot e_{f_1} e_{f_1-1} \cdots e_2 \\
&= d_1 \cdot e_n e_{n-1} \cdots e_{f_1+2} \cdot e_{f_1+1} \cdot e_{f_1} e_{f_1-1} \cdots e_2 \\
&= d_1 \cdot e_n e_{n-1} \cdots e_2 \\
&= d_1 \cdot \mathsf{tail}(\mathsf{E} - \{d_1\})
\end{aligned}
$$

as claimed by (7). Therefore, by induction (7) is correct for all $n \geq 1$. The restatement in (8) follows from the fact that $e_1 = d_1$. $\qquad\square$

The second lemma provides an invariant for the iterative operation $\triangleright$.

Lemma A.4 (Invariant for the iterative definition of multiset permutations in cool-lex order). Suppose $\boldsymbol{p} \in \mathbb{M}_\mathsf{E}$, $g > 1$, and $\boldsymbol{z}$ is any string. Then,

$$
\triangleright(\boldsymbol{p} \cdot d_g \cdot \boldsymbol{z}) = \triangleright(\boldsymbol{p}) \cdot d_g \cdot \boldsymbol{z}
$$

whenever $\boldsymbol{p} \neq d_1 \cdot \mathsf{tail}(\mathsf{E} - \{d_1\})$.

Proof. If $m = 1$ then the result is vacuously true since $\boldsymbol{p}$ must be of the form $d_1^{f_1} = d_1 \cdot \mathsf{tail}(\mathsf{E} - \{d_1\})$. Therefore, to prove the result we may assume that $m > 1$. The proof splits into three cases, depending on the substrings in $\boldsymbol{p}$ that are of the form vw with $v < w$. If $\boldsymbol{p}$ does not contain such a substring then it is of the form

$$
\boldsymbol{p} = \mathsf{tail}(\mathsf{E}) = d_m \cdot \mathsf{tail}(\mathsf{E} - \{d_1, d_m\}) \cdot d_1
$$

where $d_m > d_1$. In this case, we compare the two expressions as follows

$$
\begin{aligned}
&\triangleright(\boldsymbol{p} \cdot d_g \cdot \boldsymbol{z}) &\qquad& \triangleright(\boldsymbol{p}) \cdot d_g \cdot \boldsymbol{z} \\
&= \triangleright\left(d_m \cdot \mathsf{tail}(\mathsf{E} - \{d_1, d_m\}) \cdot \underline{d_1 d_g} \cdot \boldsymbol{z}\right) && = \triangleright(d_m \cdot \mathsf{tail}(\mathsf{E} - \{d_m\})) \cdot d_g \boldsymbol{z} \\
&= \mathsf{tail}(\mathsf{E} - \{d_1, d_m\})\underline{d_1} \cdot d_m \cdot \underline{d_g}\boldsymbol{z} && = (\mathsf{tail}(\mathsf{E} - \{d_m\}) \cdot d_m) \cdot d_g \boldsymbol{z} \\
&= \mathsf{tail}(\mathsf{E} - \{d_m\})d_m d_g \boldsymbol{z} && = \mathsf{tail}(\mathsf{E} - \{d_m\})d_m d_g \boldsymbol{z}
\end{aligned}
$$

and so the result is true. In the second case, if $\boldsymbol{p}$ contains an increase after its leftmost symbol is removed, then $\boldsymbol{p}$ can be written as

$$
\boldsymbol{p} = a \cdot \boldsymbol{p}' \cdot \underline{vw} \cdot \boldsymbol{p}''
$$

where $a \in \mathsf{E}$, and vw is the leftmost pair of adjacent symbols with $v < w$ within $\boldsymbol{p}' \cdot \underline{vw} \cdot \boldsymbol{p}''$. If $a > v$ then the result of both expressions will be

$$
\triangleright(\boldsymbol{p} \cdot d_g \cdot \boldsymbol{z}) = \boldsymbol{p}' vaw\boldsymbol{p}'' d_g \boldsymbol{z} \qquad\qquad \triangleright(\boldsymbol{p}) \cdot d_g \cdot \boldsymbol{z} = \boldsymbol{p}' vaw\boldsymbol{p}'' d_g \boldsymbol{z}
$$

and if $a \leq v$ then the result of both expressions will be

$$
\triangleright(\boldsymbol{p} \cdot d_g \cdot \boldsymbol{z}) = \boldsymbol{p}' vwa\boldsymbol{p}'' d_g \boldsymbol{z} \qquad\qquad \triangleright(\boldsymbol{p}) \cdot d_g \cdot \boldsymbol{z} = \boldsymbol{p}' vwa\boldsymbol{p}'' d_g \boldsymbol{z}
$$

and so the result is true in this case. In the third case, if $\boldsymbol{p}$ contains a vw substring with $v < w$ but does not contain such a substring after its leftmost symbol is removed, and if $\boldsymbol{p}$ is not of the form precluded by the statement of the lemma, then $\boldsymbol{p}$ is of the form

$$
\boldsymbol{p} = d_h \cdot \mathsf{tail}(\mathsf{E} - \{d_h\}) = d_h \cdot \mathsf{tail}(\mathsf{E} - \{d_h, d_1\}) \cdot d_1
$$

where d_h is some symbol in E with $d_1 < d_h < d_m$. In this case, we compare the two expressions as follows

$$\rhd (\boldsymbol{p} \cdot d_g \cdot \boldsymbol{z}) \qquad\qquad\qquad \rhd (\boldsymbol{p}) \cdot d_g \cdot \boldsymbol{z}$$

$$= \rhd \left(d_h \cdot \mathsf{tail}(\mathsf{E} - \{d_h, d_1\}) \cdot \underline{d_1 d_g} \cdot \boldsymbol{z}\right) \qquad = \rhd (d_h \cdot \mathsf{tail}(\mathsf{E} - \{d_h\})) \cdot d_g \boldsymbol{z}$$

$$= \mathsf{tail}(\mathsf{E} - \{d_h, d_1\})\underline{d_1} d_h \cdot \underline{d_g} \boldsymbol{z} \qquad\quad = (\mathsf{tail}(\mathsf{E} - \{d_h\}) \cdot d_h) \cdot d_g \boldsymbol{z}$$

$$= \mathsf{tail}(\mathsf{E} - \{d_h, d_1\}) d_1 d_h d_g \boldsymbol{z} \qquad\quad = \mathsf{tail}(\mathsf{E} - \{d_h\}) d_h d_g \boldsymbol{z}$$

$\square$

The remaining lemmas show that $\rhd$ provides the correct string when making the transition from the last string in one sublist of $\mathcal{R}_\mathsf{E}$ to the first string in the next sublist of $\mathcal{R}_\mathsf{E}$.

Lemma A.5 (First iteration using the iterative definition of multiset permutations in cool-lex order). If $f_2 > 0$ and $m = m$ then

$$\rhd (\mathsf{tail}(\mathsf{E})) = first(\mathcal{R}_{\mathsf{E}-d_m}) \cdot d_m.$$

Proof. From Lemma A.3, Definition A.2, and the observation that $\diagdown (\mathsf{tail}(\mathsf{E}) - \{d_m\}) = \infty$,

$$\rhd (\mathsf{tail}(\mathsf{E})) = \rhd (d_m \cdot \mathsf{tail}(\mathsf{E} - \{d_m\}))$$
$$= \mathsf{tail}(\mathsf{E} - \{d_m\}) \cdot d_m$$
$$= first(\mathbb{M}_{\mathsf{E}-\{d_m\}}) \cdot d_m.$$

$\square$

Lemma A.6 (Circularity for the iterative definition of multiset permutations in cool-lex order).

$$\rhd (last\,(\mathcal{R}_\mathsf{E})) = first(\mathcal{R}_\mathsf{E}).$$

Proof. From Lemma A.3, (A.2), and the observation that $\diagdown (\mathsf{tail}(\mathsf{E} - \{d_1\})) = \infty$,

$$\rhd (last\,(\mathcal{R}_\mathsf{E})) = \rhd (d_1 \cdot \mathsf{tail}(\mathsf{E} - \{d_1\}))$$
$$= \mathsf{tail}(\mathsf{E} - \{d_1\}) \cdot d_1$$
$$= \mathsf{tail}(\mathsf{E})$$
$$= first(\mathcal{R}_\mathsf{E}).$$

$\square$

Lemma A.7 (Interface 1 for the iterative definition of multiset permutations in cool-lex order). When $k < \overrightarrow{f_{j-1}} - 1$

$$\rhd \left(last\,\left(\mathcal{R}_{\mathsf{E}-\mathsf{scut}(j,k)} \cdot \mathsf{scut}(j,k)\right)\right) = first(\mathcal{R}_{\mathsf{E}-\mathsf{scut}(j,k+1)} \cdot \mathsf{scut}(j,k+1))$$

Proof. Within the following derivation we use Lemma A.3 and Definition A.2. Then for the $\rhd$ transition we observe that

$$e_{k+1} \le e_{k+2} \text{ and } e_{k+2} < d_j$$

where the inequality on the right comes from the fact that $k < \overrightarrow{f_{j-1}} - 1$. The derivation is as follows,

$$\triangleright \left(last \left(\mathcal{R}_{\mathsf{E-scut}(j,k)} \cdot \mathsf{scut}(j,k) \right) \right)$$
$$= \triangleright \left(e_{k+1} \cdot \mathsf{tail}(\mathsf{E} - \{e_{k+1}\} - \mathsf{scut}(j,k)) \cdot \mathsf{scut}(j,k) \right)$$
$$= \triangleright \left(e_{k+1} \cdot \mathsf{tail}(\mathsf{E} - \{e_{k+1}, e_{k+2}\} - \mathsf{scut}(j,k)) \cdot \underline{e_{k+2}d_j} \cdot e_k e_{k-1} \cdots e_1 \right)$$
$$= \mathsf{tail}(\mathsf{E} - \{e_{k+1}, e_{k+2}\} - \mathsf{scut}(j,k)) \underline{e_{k+2}d_j} \cdot e_{k+1} \cdot e_k e_{k-1} \cdots e_1$$
$$= \mathsf{tail}(\mathsf{E} - \{e_{k+1}\} - \mathsf{scut}(j,k)) \cdot d_j \cdot e_{k+1} e_k \cdots e_1$$
$$= \mathsf{tail}(\mathsf{E} - \mathsf{scut}(j, k+1)) \cdot d_j \cdot e_{k+1} e_k \cdots e_1$$
$$= \mathsf{tail}(\mathsf{E} - \mathsf{scut}(j, k+1)) \cdot \mathsf{scut}(j, k+1)$$
$$= first(\mathcal{R}_{\mathsf{E-scut}(j,k+1)}) \cdot \mathsf{scut}(j, k+1)$$

as claimed.   □

Lemma A.8 (Interface 2 for the iterative definition of multiset permutations in cool-lex order). When $k = \overrightarrow{f_{j-1}} - 1$ and $j > 2$

$$\triangleright \left(last \left(\mathcal{R}_{\mathsf{E-scut}(j,k)} \cdot \mathsf{scut}(j,k) \right) \right) = first(\mathcal{R}_{\mathsf{E-scut}(j-1,0)} \cdot \mathsf{scut}(j-1,0))$$

Proof. Within the following derivation we use Lemma A.3 and Definition A.2. Then for the $\triangleright$ transition we observe that

$$e_{k+1} = d_{j-1} \text{ and } e_{k+2} = d_j$$

since $k + 1 = \overrightarrow{f_{j-1}}$. The derivation is as follows,

$$\triangleright \left(last \left(\mathcal{R}_{\mathsf{E-scut}(j,k)} \cdot \mathsf{scut}(j,k) \right) \right)$$
$$= \triangleright \left(e_{k+1} \cdot \mathsf{tail}(\mathsf{E} - \{e_{k+1}\} - \mathsf{scut}(j,k)) \cdot \mathsf{scut}(j,k) \right)$$
$$= \triangleright \left(e_{k+1} \cdot \mathsf{tail}(\mathsf{E} - \mathsf{scut}(j, k+1)) \cdot \mathsf{scut}(j,k) \right)$$
$$= \triangleright \left(e_{k+1} \cdot \mathsf{tail}(\mathsf{E} - \{d_j\} - \{e_{k+1}, e_k, \ldots, e_1\}) \cdot \mathsf{scut}(j,k) \right)$$
$$= \triangleright \left(e_{k+1} \cdot \mathsf{tail}(\mathsf{E} - \{e_{k+2}\} - \{e_{k+1}, e_k, \ldots, e_1\}) \cdot \mathsf{scut}(j,k) \right)$$
$$= \triangleright \left(e_{k+1} \cdot \mathsf{tail}(\mathsf{E} - \{e_{k+2}, e_{k+1}, \ldots, e_1\}) \cdot \mathsf{scut}(j,k) \right)$$
$$= \triangleright \left(e_{k+1} \cdot e_n e_{n-1} \cdots e_{k+3} \cdot \mathsf{scut}(j,k) \right)$$
$$= \triangleright \left(e_{k+1} \cdot e_n e_{n-1} \cdots e_{k+3} \cdot d_j e_k e_{k-1} \cdots e_1 \right)$$
$$= \triangleright \left(e_{k+1} \cdot e_n e_{n-1} \cdots e_{k+3} \cdot e_{k+2} e_k e_{k-1} \cdots e_1 \right)$$
$$= \triangleright \left(e_{k+1} \cdot e_n e_{n-1} \cdots e_{k+2} e_k e_{k-1} \cdots e_1 \right)$$
$$= e_n e_{n-1} \cdots e_{k+2} e_k e_{k-1} \cdots e_1 \cdot e_{k+1}$$
$$= \mathsf{tail}(\mathsf{E} - \{e_{k+1}\}) \cdot e_{k+1}$$
$$= \mathsf{tail}(\mathsf{E} - \{d_{j-1}\}) \cdot d_{j-1}$$
$$= first(\mathcal{R}_{\mathsf{E-scut}(j-1,0)} \cdot \mathsf{scut}(j-1,0)).$$

                                                  □

Now the corresponding version of Theorem 2.7 can be proven. Before proceeding it is mentioned that $\mathcal{R}_{\mathsf{E}}$ contains each string in $\mathbb{M}_{\mathsf{E}}$ since it is simply a reordering of $\mathcal{L}_{\mathsf{E}}$.

The $\stackrel{\circ}{=}$ symbol represents that $\triangleright$ generates $\mathcal{R}_\mathsf{E}$ circularly with the additional property that $\triangleright^{|\mathbb{M}_\mathsf{E}|} (\mathsf{tail}(\mathsf{E})) = \mathsf{tail}(\mathsf{E})$; thus, the theorem below is a slight strengthening of Theorem 2.7.

Theorem A.9 (Equivalence of definitions for cool-lex order).

$$(9) \qquad \mathcal{R}_\mathsf{E} \stackrel{\circ}{=} \bigoplus_{k=0}^{|\mathbb{M}_\mathsf{E}|} \triangleright^k (\mathsf{tail}(\mathsf{E})).$$

Proof. This proof will be by induction on the number of elements in E that are not equal to d_1. In other words, the induction is on $n - f_1$, or equivalently on $\sum_{i=2}^m f_1$. The result holds when $n = f_1$ because $\mathcal{R}_\mathsf{E} = d_1^{f_1}$ (by (4b) in Definition A.1) and $\triangleright \left(0^{f_0}\right) = 0^{f_0}$ (by Lemma A.5). Therefore, to prove the result by induction on $n - f_1$ we first assume that the result holds when $n - f_1 = x \geq 0$. Next, consider the recursive structure of $\mathcal{R}_\mathsf{E}$ given by Definition 4

$$\mathsf{tail}(\mathsf{E}), \bigoplus_2^{j=m} \bigoplus_{k=0}^{\overrightarrow{f_{j-1}}-1} \mathcal{R}_{\mathsf{E}-\mathsf{scut}(j,k)} \cdot \mathsf{scut}(j, k)$$

By Lemma A.5,

$$\triangleright (\mathsf{tail}(\mathsf{E})) = first(\mathcal{R}_{\mathsf{E}-d_m}) \cdot d_m$$
$$= first(\mathcal{R}_{\mathsf{E}-\mathsf{scut}(m,0)} \cdot \mathsf{scut}(m, 0))$$
$$= first(\bigoplus_2^{j=m} \bigoplus_{k=0}^{\overrightarrow{f_{j-1}}-1} \mathcal{R}_{\mathsf{E}-\mathsf{scut}(j,k)} \cdot \mathsf{scut}(j, k))$$

and so $\triangleright$ makes the correct transition on the first string in $\mathcal{R}_\mathsf{E}$. For the other transitions between sublists, Lemma A.7 states that

$$\triangleright \left(last \left(\mathcal{R}_{\mathsf{E}-\mathsf{scut}(j,k)} \cdot \mathsf{scut}(j, k)\right)\right) = first(\mathcal{R}_{\mathsf{E}-\mathsf{scut}(j,k+1)} \cdot \mathsf{scut}(j, k + 1))$$

for all $0 < k < \overrightarrow{f_{j-1}} - 1$ and Lemma A.8 states that

$$\triangleright \left(last \left(\mathcal{R}_{\mathsf{E}-\mathsf{scut}(j,k)} \cdot \mathsf{scut}(j, k)\right)\right) = first(\mathcal{R}_{\mathsf{E}-\mathsf{scut}(j-1,0)} \cdot \mathsf{scut}(j - 1, 0))$$

when $k = \overrightarrow{f_{j-1}} - 1$ and $j > 2$. Therefore, $\triangleright$ correctly transitions between the sublists of $\mathcal{R}_\mathsf{E}$.

For transitions within each individual sublist of the form $\mathcal{R}_{\mathsf{E}-\mathsf{scut}(j,k)}$, we will use our inductive assumption. Notice that if

$$\mathsf{E}' = \mathsf{E} - \mathsf{scut}(j, k)$$

then $n' - f_1' \leq x$ whenever $j > 2$. Therefore, we can apply our inductive assumption to any list of the form $\mathcal{R}_{\mathsf{E}-\mathsf{scut}(j,k)}$ with $j > 2$. Lemma A.4 ensures that

$$\triangleright (\boldsymbol{s} \cdot \mathsf{scut}(j, k)) = \triangleright (\boldsymbol{s}) \cdot \mathsf{scut}(j, k)$$

whenever $\boldsymbol{s}$ is not the last string in the sublist. Therefore, by the inductive assumption, $\triangleright$ correctly gives the next string within each $\mathcal{R}_{\mathsf{E}-\mathsf{scut}(j,k)}$ sublist. Finally, by Lemma A.6,

$$\triangleright (last (\mathcal{R}_\mathsf{E})) = first(\mathcal{R}_\mathsf{E}).$$

and so $\triangleright$ is also circular with respect to iteratively defining $\mathcal{R}_\mathsf{E}$. $\qquad \square$

A.2. Algorithms.

Lemma A.10. Let $\boldsymbol{s} = s_1 s_2 \cdots s_n$ and $\lhd\,(\boldsymbol{s}) = \boldsymbol{s}' = s_1' s_2' \ldots s_n'$. Then

$$
(10\text{a}) \qquad \diagdown\,(\lhd\,(\boldsymbol{s})) = \begin{cases} 1 & \text{if } s_i' < s_2' \ (2\text{a}) \\ \diagdown\,(\boldsymbol{s}) + 1 & \text{otherwise (if } s_1' \geq s_2') \ (2\text{b}) \end{cases}
$$

(10b)

Proof. If $s_1' < s_2'$ then $\diagdown\,(\lhd\,(\boldsymbol{s})) = 2$, which proves (2a). Otherwise, let $i = \diagdown\,(\boldsymbol{s})$ and assume $s_1' \geq s_2'$. In the first case also assume that $i < n - 1$ and $s_{i+2} \leq s_i$. In the second case also assume that $i = n - 1$ or $s_{i+2} > s_i$. An expansion for $\lhd\,(\boldsymbol{s})$ appears below, with the first case on the left and the second case on the right.

$$
\lhd\,(\boldsymbol{s}) = s_{i+2} s_1 s_2 \cdots s_i s_{i+1} s_{i+3} s_{i+4} \cdots s_n \qquad \lhd\,(\boldsymbol{s}) = s_{i+1} s_1 s_2 \cdots s_i s_{i+2} s_{i+3} s_{i+4} \cdots s_n
$$
$$
= s_1' s_2' s_3' \cdots s_{i+1}' s_{i+2}' s_{i+3}' s_{i+4}' \cdots s_n' \qquad = s_1' s_2' s_3' \cdots s_{i+1}' s_{i+2}' s_{i+3}' s_{i+4}' \cdots s_n'.
$$

To prove (2b) first notice that $s_1' s_2' s_3' \cdots s_{i+1}'$ is a non-increasing string and. (This is due to the fact that $s_1' \geq s_2'$ and $i = \diagdown\,(\boldsymbol{s})$.) Therefore, $\diagdown\,(\boldsymbol{s}') \geq i + 1$. Next, notice that $i = n - 1$ or $s_{i+1}' < s_{i+2}'$. (In the first case this is due to the fact that $s_i < s_{i+1}$, while in the second case $s_i < s_{i+2}$ or $i = n - 1$.) Therefore, $\diagdown\,(\boldsymbol{s}') \leq i + 1$. Hence, $\diagdown\,(\lhd\,(\boldsymbol{s})) = i + 1$ as claimed. $\qquad\square$

A.3. Analysis.

Lemma A.11 (Number of strings for each value of $\pi(\boldsymbol{s})$)**.**

$$
(11\text{a}) \qquad |\{\boldsymbol{s} : \pi(\boldsymbol{s}) = k\}| = \begin{cases} \dfrac{n! \cdot 2 \cdot (k^2 - k - 1)}{(k+1)!} & \text{if } 2 \leq k < n \\[3mm] 2n - 2 & \text{otherwise (if } k = n) \end{cases}
$$

(11b)

Proof. When $\pi(\boldsymbol{s}) = n$ then there are three possibilities to consider. First, it might be that $\diagdown\,(\boldsymbol{s}) = n$, and then $\pi(\boldsymbol{s}) = n$ is guaranteed. This can happen only if $\boldsymbol{s} = \mathsf{tail}(\mathsf{E})$. Second, it might be that $\diagdown\,(\boldsymbol{s}) = n - 1$, and then again $\pi(\boldsymbol{s}) = n$ is guaranteed. This can happen in $n - 1$ ways since there are $n - 1$ choices for the value of s_n (it cannot be that $s_n = d_1$) and then the rest of the string is determined by this choice. Third, it might be that $\diagdown\,(\boldsymbol{s}) = n - 2$ and $\pi(\boldsymbol{s}) = n$. In order for this to occur it must be that $s_n = d_1$. Then there are $n - 2$ choices for the value of s_{n-1} (it cannot be d_1 or d_2) and then the rest of the string is determined by this choice. In total there are $1 + (n - 1) + (n - 2) = 2n - 2$ possibilities for $\boldsymbol{s}$, thereby proving (11b).

When $\pi(\boldsymbol{s}) = k$ and $2 \leq k < n$ then there are two possibilities. First, it might be that $\diagdown\,(\boldsymbol{s}) = k - 1$ and $\pi(\boldsymbol{s}) = k$. In order for this to occur there are $\binom{n}{k+1}$ ways of choosing the first $k + 1$ symbols. Of these symbols, s_{k-1} must be the smallest, and then there are $k(k - 1)$ ways of choosing $s_k s_{k+1}$. Then $s_1 s_2 \ldots s_{k-1}$ is uniquely determined since $\diagdown\,(\boldsymbol{s}) = k - 1$. Finally, there are $n - k - 1$ symbols that are not within the first $k + 1$ symbols, and these can be placed in any order. Therefore, in total there are

$$
\binom{n}{k+1} k(k-1)(n-k-1)! = \frac{n!k(k-1)(n-k-1)!}{(k+1)!(n-k-1)!}
$$
$$
= \frac{n!k(k-1)}{(k+1)!}
$$

choices for $\boldsymbol{s}$ in the first possibility. Second, it might be that $\diagdown\,(\boldsymbol{s}) = k - 2$ and $\pi(\boldsymbol{s}) = k$. (This possibility cannot occur when $k = 2$, however, the expression obtained below equals zero when $k = 2$.) In order for this to occur there are $\binom{n}{k}$ ways of choosing the first k symbols.

Of these symbols, s_k must be the smallest, and then there are $(k-2)$ ways of choosing s_{k-1} since s_{k-1} can be any of the first k symbols except for the smallest and second-smallest. Then $s_1 s_2 \ldots s_{k-2}$ is uniquely determined since $\searrow(\boldsymbol{s}) = k - 2$. Finally, there are $n - k$ symbols that are not within the first k symbols, and these can be placed in any order. Therefore, in total there are

$$\binom{n}{k}(k-2)(n-k)! = \frac{n!(k-2)(n-k)!}{k!(n-k)!}$$
$$= \frac{n!(k-2)}{k!}$$
$$= \frac{n!(k-2)(k+1)}{(k+1)!}$$

choices for $\boldsymbol{s}$ in the second possibility. Therefore, in total there are

$$\frac{n!k(k-1) + n!(k-2)(k+1)}{(k+1)!} = \frac{n! \cdot 2 \cdot (k^2 - k - 1)}{(k+1)!}$$

choices for $\boldsymbol{s}$ as claimed in (11a).   □

Lemma A.12 (Combinatorial Identity).

$$(12) \qquad 2(n^2 - n + 1) + \sum_{k=2}^{n-1} k \cdot \frac{n! \cdot 2 \cdot (k^2 - k - 1)}{(k+1)!} = 3 \cdot n!$$

when $n \geq 2$.

Proof. The proof is by induction on n. When $n = 2$,

$$2(n^2 - n + 1) + \sum_{k=2}^{n-1} k \cdot \frac{n! \cdot 2 \cdot (k^2 - k - 1)}{(k+1)!} = 2(4 - 2 + 1) = 3 \cdot 2!.$$

So assume the identity is true when $n = x$, and let us use that assumption to prove that it is true when $n = x + 1$. The important steps in the following derivation are separating the $k = x$ term within the sum, factoring $(x + 1)$ out of each term in the remaining sum, and

adding and subtracting the value of $2(x^2 - x + 1)$ in order to apply the inductive hypothesis.

$$2((x+1)^2 - (x+1) + 1) + \sum_{k=2}^{x} k \cdot \frac{(x+1)! \cdot 2 \cdot (k^2 - k - 1)}{(k+1)!}$$

$$= 2(x^2 + x + 1) + x \cdot \frac{(x+1)! \cdot 2 \cdot (x^2 - x - 1)}{(x+1)!} + \sum_{k=2}^{x-1} k \cdot \frac{(x+1)! \cdot 2 \cdot (k^2 - k - 1)}{(k+1)!}$$

$$= 2(x^3 + 1) + \sum_{k=2}^{x-1} k \cdot \frac{(x+1)! \cdot 2 \cdot (k^2 - k - 1)}{(k+1)!}$$

$$= 2(x^3 + 1) + (x+1) \cdot \sum_{k=2}^{x-1} k \cdot \frac{x! \cdot 2 \cdot (k^2 - k - 1)}{(k+1)!}$$

$$= 2(x^3 + 1) + (x+1) \cdot \left(2(x^2 - x + 1) - 2(x^2 - x + 1) + \sum_{k=2}^{x-1} k \cdot \frac{x! \cdot 2 \cdot (k^2 - k - 1)}{(k+1)!} \right)$$

$$= 2(x^3 + 1) + (x+1) \cdot \left(2(x^2 - x + 1) + \sum_{k=2}^{x-1} k \cdot \frac{x! \cdot 2 \cdot (k^2 - k - 1)}{(k+1)!} - 2(x^2 - x + 1) \right)$$

$$= 2(x^3 + 1) + (x+1)(3 \cdot x! - 2(x^2 - x + 1))$$

$$= 3 \cdot (x+1)!$$

□

The following derivation shows how Theorem 3.2 follows from Lemmas 3.3 and 3.4

$$\sum_{i=0}^{n!} \pi(\lhd^i (\mathsf{tail}(\mathsf{E}))) = \left(\sum_{k=2}^{n-1} k \cdot \frac{n! \cdot 2 \cdot (k^2 - k - 1)}{(k+1)!} \right) + n \cdot (2n - 2)$$

$$= 2(n^2 - n) + \sum_{k=2}^{n-1} k \cdot \frac{n! \cdot 2 \cdot (k^2 - k - 1)}{(k+1)!}$$

$$< 2(n^2 - n + 1) + \sum_{k=2}^{n-1} k \cdot \frac{n! \cdot 2 \cdot (k^2 - k - 1)}{(k+1)!}$$

$$= 3 \cdot n!.$$