Journal of Algebraic Combinatorics **15** (2002), 71–97

A Counting Formula for Labeled, Rooted Forests

KRISTEN A. LAMPE klampe@cc.edu
Department of Mathematics, Carroll College, Waukesha, WI 53186

Received October 13, 2000; Accepted September 4, 2001

Abstract. Given a power series, the coefficients of the formal inverse may be expressed as polynomials in the coefficients of the original series. Further, these polynomials may be parameterized by certain ordered, labeled forests. There is a known formula for the formal inverse, which indirectly counts these classes of forests, developed in a non-direct manner. Here, we provide a constructive proof for this counting formula that explains why it gives the correct count. Specifically, we develop algorithms for building the forests, enabling us to count them in a direct manner.

Keywords: trees, forests, counting formula, polynomials, reversion

1. Introduction

The Jacobian Conjecture, first stated by O. Keller in 1939, serves as a motivation for this paper. Let $F : k^n \to k^n$ be a polynomial over a field, k, of characteristic zero. We can write $F = (F_1, \ldots, F_n)$, where each F_i is a polynomial in n variables. Let $J(F) = (D_i F_j)$ be the Jacobian matrix for F, and $j(F) = \det(J(F))$ be the Jacobian determinant. The Jacobian Conjecture states that, if $j(F) = a \in k^*$, then F has a polynomial inverse.

One reduction shows, by increasing the dimension, that we may assume $F = (F_1, \ldots, F_n)$ is of the form $F_i = X_i - H_i$ where H_i is a homogeneous polynomial of degree $\delta = 3$ (see Bass et al. [1] for this and other reductions, as well as a more complete history of the problem.) Using a process called *reversion*, we may express the compositional inverse of F, called $G = (G_1, \ldots, G_n)$, as a series whose coefficients are polynomials over the coefficients of the F_i. Moreover, Wright [6] expresses the inverse in this manner as a sum over labeled, rooted trees. This this process applies when $F = X - H$ is a power series as well, so we will work in this more general setting.

A *labeled, rooted tree* is a minimally connected finite graph with one node designated the root and each node given a label. We denote the different labels by $1, \ldots, n$, and the tree by T. Two adjacent vertices have a parent-child relationship. The *parent* is the node closer to the root, while the *child* is the node farther from the root.

In their paper, "Reversion of Power Series and the Extended Raney Coefficients," the authors Cheng et al. [2] use generating functions in infinitely many variables to get another expression for the inverse in terms of trees, similar to that in [6]. This expression is the starting point for this paper, so we endeavor to state it clearly here.

Let $F_1, \ldots, F_n$ be power series in n variables of the form $F_i = X_i - H_i$, where H_i consists of terms of degree two and higher. It is known that the power series inverse,

$G = (G_1, \ldots, G_n)$, of $F = (F_1, \ldots, F_n)$ exists. Further, the coefficients of each G_i can be expressed as polynomials whose indeterminates are the coefficients of the F_i. These polynomials have coefficients in the integers and are, surprisingly, non-negative. They are called the *extended Raney coefficients*, and the process of finding them is, as mentioned above, called *reversion*. Next, we consider the definitions and notation used in the formula for the inverse.

Notation 1.1 Having defined a labeled, rooted tree, we say a *forest* is an ordered collection, sorted by root-label, of these trees. A forest gives rise to an *inventory*, $\alpha = (\alpha_1, \ldots, \alpha_n)$ as follows. For $i = 1, \ldots, n$, let $\alpha_i = (\alpha_i^{k_1,\ldots,k_n})$ be an n dimensional array over the natural numbers with finitely many non-zero entries. For each i and each n-tuple $k = (k_1, \ldots, k_n)$, let α_i^k be the number of i-labeled nodes in the forest with k_j children having label j, for $1 \leq j \leq n$. Next, we define $\sigma(\alpha_i) = \sum_k \alpha_i^k$ and $\sigma_j(\alpha_i) = \sum_k k_j \alpha_i^k$. So $\sigma(\alpha_i)$ is the number of i-labeled nodes and $\sigma_j(\alpha_i)$ is the number of j-labeled children of i-labeled nodes. We restrict ourselves to α such that

$$\sigma(\alpha_i) \geq \sum_{j=1}^{n} \sigma_i(\alpha_j). \tag{1}$$

We define $R(\alpha)$ to be the number of forests having inventory α.

We also know that

$$p_i = \sigma(\alpha_i) - \sum_{j=1}^{n} \sigma_i(\alpha_j) \tag{2}$$

is the number of i-labeled roots in the forest. Then we say $p = (p_1, \ldots, p_n)$ is the *root-type* of the forest. We say $q = (q_1, \ldots, q_n)$ is the *leaf-type* of the forest if there are q_i i-labeled leaves. Lastly, $k = (k_1, \ldots, k_n)$ is called the *child-type* of a node.

For $k = (k_1, \ldots, k_n)$, let $|k| = k_1 + \cdots + k_n$. For $F_i = X_i - H_i$, where

$$H_i = \sum_{|k| \geq 2} a_i^k X^k,$$

Theorem 3.1 of [2] shows that

$$X_1^{p_1} \cdots X_n^{p_n} = \sum_{q=(q_1,\ldots,q_n)} e_p^q F_1^{q_1} \cdots F_n^{q_n}$$

where

$$e_p^q = \sum R(\alpha) \prod_{j=1}^{n} \prod_{|k| \geq 2} \left(a_j^k\right)^{\alpha_j^k}$$

with the sum being indexed by all α having root-type p and leaf-type q, such that $q_i = \alpha_i^{(0,\ldots,0)}$ and $\alpha_i^k = 0$ whenever $|k| = 1$. Notice that if $p = e_i$, this formula becomes

$$X_i = \sum_{q=(q_1,\ldots,q_n)} e_{e_i}^q F_1^{q_1} \cdots F_n^{q_n}.$$

In other words, we have found the inverse

$$G_i = \sum_{q=(q_1,\ldots,q_n)} e_{e_i}^q F_1^{q_1} \cdots F_n^{q_n}.$$

Next, [2] uses a version of Jacobi's residue formula to find a determinantal formula for $R(\alpha)$. We define

$$M(\alpha_i) = \frac{\sigma(\alpha_i)!}{\prod\left(\alpha_i^k!\right)}$$

$$\widetilde{M}(\alpha_i) = \begin{cases} 1 & \text{if all } \alpha_i^k = 0 \\ \frac{1}{\sigma(\alpha_i)} M(\alpha_i) & \text{otherwise} \end{cases}$$

The formula found in Theorem 3.3 of [2] is then

Theorem 1.2

$$R(\alpha) = \left| \begin{pmatrix} M(\alpha_1) & 0 & \cdots & 0 \\ 0 & M(\alpha_2) & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & M(\alpha_n) \end{pmatrix} - \begin{pmatrix} \widetilde{M}(\alpha_1) & 0 & \cdots & 0 \\ 0 & \widetilde{M}(\alpha_2) & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \widetilde{M}(\alpha_n) \end{pmatrix} \times \begin{pmatrix} -\sigma_1(\alpha_1) & -\sigma_2(\alpha_1) & \ldots & -\sigma_n(\alpha_1) \\ -\sigma_1(\alpha_2) & -\sigma_2(\alpha_2) & \ldots & -\sigma_n(\alpha_2) \\ \vdots & \vdots & & \vdots \\ -\sigma_1(\alpha_n) & -\sigma_2(\alpha_n) & \ldots & -\sigma_n(\alpha_n) \end{pmatrix} \right|$$

We may assume that $\sigma(\alpha_i) > 0$ for all i. Indeed, suppose some $\sigma(\alpha_i) = 0$, meaning there are no nodes of label i and, by Eq. (1), $\sigma_i(\alpha_j) = 0$ as well. Then, in Theorem 1.2, $M(\alpha_i) = 1$, $\widetilde{M}(\alpha_i) = 1$, and the determinant will have the ith row and column with zeroes except at the (i, i) entry, which will have a 1. Hence, we may move to a lower dimension. From now on, we will assume that $\alpha = (\alpha_1, \ldots, \alpha_n)$ satisfies $\sigma(\alpha_i) > 0$ for all i. Under this condition, the

above theorem reduces to

$$R(\alpha) = \left(\prod_{i=1}^{n} \widetilde{M}(\alpha_i) \right) \begin{vmatrix} \sigma(\alpha_1) - \sigma_1(\alpha_1) & -\sigma_2(\alpha_1) & \dots & -\sigma_n(\alpha_1) \\ \vdots & & & \vdots \\ -\sigma_1(\alpha_n) & -\sigma_2(\alpha_n) & \dots & \sigma(\alpha_n) - \sigma_n(\alpha_n) \end{vmatrix} \tag{3}$$

Thus, [2] gives a formula for counting forests having inventory α *without ever referring to the forests themselves*. The way this formula comes about, then, is fairly opaque. There is no method given in [2] involving the enumeration of the forests by a direct counting process that explains why Formula (3) holds. Here, we give such a method. Aside from illuminating the formula in Eq. (3), this counting method has several other interesting consequences. In the process of enumerating forests, we develop geometric objects called *wreaths* that are new to this problem. Moreover, we develop a non-trivial method to classify wreaths that corresponds nicely with Theorem 1.2's determinantal formula. Further, using this counting procedure as the basis of the argument, we can restate the Jacobian conjecture in a simpler form. This last result will appear in a future paper, [4].

1.1. Outline of method

We first break down the nodes of the forest and write them in a linear fashion, called a *sorted permutation*. Combinatorics tells us the number of possible arrangements that can be made from a given set of nodes from a forest with inventory α, each arrangement giving one sorted permutation. We then define an algorithm for taking a sorted permutation and building a forest. Since the number of roots of each label is known, we can build this forest tree by tree. Once the forest is built, if all the nodes have been used in this process, we declare the sorted permutation a *success*. Otherwise, there will be leftover nodes and we declare the sorted permutation a *failure*. It is these failures we need to count. The remaining nodes in the sorted permutation have no roots, so they do not form trees, but instead form objects with loops in them, called *wreaths*. This first algorithm also takes a failing sorted permutation and associates to that failure a set of pairwise disjoint cycles, derived from these wreaths.

Next, we define a second algorithm. This algorithm takes a given cycle and forms a collection of sorted permutations which fail. Moreover, applying the first algorithm to these sorted permutations will always result in the given cycle being one of the associated cycles. We must take into consideration that the same sorted permutation may appear with two different given cycles. After counting the number of failing sorted permutations constructed with a given cycle, and counting how many times a sorted permutation appears when considering all possible cycles, we thus acquire a count of the total number of distinct failing sorted permutations.

Finally, we expand the given formula for $R(\alpha)$, and show that this quantity is precisely the number of total sorted permutations for a given α minus the number of distinct failing sorted permutations created by the second algorithm. In fact, interpreting the terms of the formula as counting sorted permutations with certain cycles gives an exact match, term by term, with the usual set theoretic formula for the cardinality of a union of sets.

2. The Raney coefficients

In dimension n, [2] proves that, for a given inventory, α, satisfying $\sigma(\alpha_i) > 0$ for all i, and for

$$D_n = \begin{vmatrix} \sigma(\alpha_1) - \sigma_1(\alpha_1) & -\sigma_2(\alpha_1) & \dots & -\sigma_n(\alpha_1) \\ -\sigma_1(\alpha_2) & \sigma(\alpha_2) - \sigma_2(\alpha_2) & \dots & -\sigma_n(\alpha_2) \\ \vdots & \vdots & & \vdots \\ -\sigma_1(\alpha_n) & -\sigma_2(\alpha_n) & \dots & \sigma(\alpha_n) - \sigma_n(\alpha_n) \end{vmatrix}$$

then $R(\alpha)$, the number of forests with inventory α, is shown to be

$$R(\alpha) = \left(\prod_{i=1}^{n} \widetilde{M}(\alpha_i) \right) D_n. \tag{4}$$

In order to provide a constructive proof that this formula counts the number of forests with inventory α, we need to evaluate the determinant, D_n.

Claim 2.1 For M an $n \times n$ matrix such that $M = D - A$, where $A = (a_{ij})$ is an arbitrary matrix and

$$D = \begin{vmatrix} d_1 & 0 & \dots & 0 \\ 0 & d_2 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & d_n \end{vmatrix}$$

is a diagonal matrix, then

$$\det(M) = \prod_{i=1}^{n} d_i + \sum_{r=1}^{n} (-1)^r \sum_{1 \le i_1 < i_2 < \cdots < i_r \le n} \left(\prod_{l \notin \{i_1, \dots, i_r\}} d_l \right) \begin{vmatrix} a_{i_1 i_1} & a_{i_1 i_2} & \dots & a_{i_1 i_r} \\ \vdots & \vdots & & \vdots \\ a_{i_r i_1} & a_{i_r i_2} & \dots & a_{i_r i_r} \end{vmatrix}$$

The proof follows from the bilinearity by rows of determinants. We can apply this claim to D_n and note that

$$\begin{aligned} D_n = \prod_{i=1}^{n} \sigma(\alpha_i) + \sum_{r=1}^{n} (-1)^r \sum_{1 \le i_1 < i_2 < \cdots < i_r \le n} \left(\prod_{l \notin \{i_1, \dots, i_r\}} \sigma(\alpha_l) \right) \\ \times \begin{vmatrix} \sigma_{i_1}(\alpha_{i_1}) & \sigma_{i_2}(\alpha_{i_1}) & \dots & \sigma_{i_r}(\alpha_{i_1}) \\ \vdots & \vdots & & \vdots \\ \sigma_{i_1}(\alpha_{i_r}) & \sigma_{i_2}(\alpha_{i_r}) & \dots & \sigma_{i_r}(\alpha_{i_r}) \end{vmatrix} \end{aligned} \tag{5}$$

To develop this determinant further, we need some vocabulary. In what follows, we will fix an inventory, α, such that

$$\sigma(\alpha_i) \geq \sum_{j=1}^{n} \sigma_i(\alpha_j) \quad \text{for all } 1 \leq i \leq n.$$

As in the introduction, α_i^k represents the number of i-labeled nodes with child-type $k = (k_1, \ldots, k_n)$, that is, having k_j j-labeled children.

2.2 For $i \in \{1, 2, \ldots, n\}$ and $k = (k_1, \ldots, k_n)$, let $L = L_\alpha$ be a set containing the formal symbol i^k α_i^k times. *If $\alpha_i^k > 1$, put a different shade on each element of L of that type, so all elements of L are distinct.* An *L-sorted permutation*, w, is a sorted permutation of the elements of L of the form

$$(1^{a_1}, \ldots, 1^{a_l})(2^{b_1}, \ldots, 2^{b_k}) \cdots (n^{c_1}, \ldots, n^{c_m})$$

Each entry in the sorted permutation, representing an element of L, is called a *letter*. Each letter's superscript is an n-tuple, representing the child-type of that letter's node. Thus, in dimension $n = 3$, the letter 1^{012} represents a label 1 node having child-type (0, 1, 2). We may view a letter as a node with labeled edges directed away from it. The label on the edge denotes the label of the child to be placed at edge's end.

Remark 2.3 Previously, we defined $\sigma(\alpha_i)$ and $\sigma_i(\alpha_j)$ in terms of forests. Note that, as we have defined an L-sorted permutation, we may also determine $\sigma(\alpha_i)$ and $\sigma_i(\alpha_j)$ from L, or any L-sorted permutation. Thus, $\sigma(\alpha_i)$ denotes the number of i-labeled letters and $\sigma_i(\alpha_j)$ is the sum of the ith child-type entries of all the j-labeled letters.

The number of L-sorted permutations is $\prod_{i=1}^{n}(\sigma(\alpha_i)!)$. If we were to remove the shading from 2.2 that makes each letter distinct, combinatorics tells us we would have

$$\frac{\prod_{i=1}^{n}(\sigma(\alpha_i)!)}{\prod_{i,k}\left(\alpha_i^k!\right)}$$

distinct L-sorted permutations and this equals $M(\alpha) = \prod_{i=1}^{n} M(\alpha_i)$. In what follows, we will show there is a one-to-one correspondence between forests with inventory α and "successful L-sorted permutations" without shading. Thus, to count the number of forests, we count the number of successful sorted permutations without shading. By the above argument, it suffices to count the number of shaded successful sorted permutations and then divide by the number of times a sorted permutation is repeated when the shading is removed. *Thus, using Equation* (4), *the goal is to show that the number of shaded successful sorted permutations is*

$$\left(\prod_{i=1}^{n}((\sigma(\alpha_i) - 1)!)\right) D_n. \tag{6}$$

2.1. *Cycle definitions*

A *cycle*, c, is a non-empty sequence of letters from the set L with the following three properties.

(i) Each label is represented at most once.
(ii) Each letter designates an edge to connect it to the next cycle node. This implies that if the ith letter in the cycle has label j, then the $(i-1)$st letter has $k_j \geq 1$. For $i = 1$, we use the convention that $i - 1$ is the last letter. If $k_j > 1$, the subscript of the $(i-1)$st letter denotes the edge specified.
(iii) A cycle is unique only up to cyclic rotation. That is, the cycle obtained by shifting the cycle entries x units left and moving the first x entries to be the last x entries will be considered to be the same cycle.

For example, $(1^{011}, 3_2^{200})$ is a cycle. Also, $(3_2^{200}, 1^{011})$ is the same cycle. For c a cycle, $|c|$ is the number of letters in c and $L_c \subseteq L$ is the set of letters in c.

A *cycle-type*, $\bar{c} = (x_1, \ldots, x_s)$, is a sequence, mod rotation, of distinct labels x_i where $s \leq n$. Every cycle, then, has an associated cycle-type.

The cycle $(1^{011}, 3_2^{200})$ is of type (1, 3).

Two cycles, c_1 and c_2, are said to be *disjoint* if they have no labels in common. We write $c_1 \cap c_2 = \emptyset$.

Going back to the determinant, D_n, we now associate to each term a collection of disjoint cycle-types by using the following convention: $\sigma(\alpha_j)$ is not part of a cycle, but $\sigma_i(\alpha_j)$ is read "j calls i," meaning that $(\ldots, j, i, \ldots)$ is part of a cycle-type. For example,

$$\sigma_1(\alpha_1)\sigma_2(\alpha_2)\prod_{j=3}^{n}\sigma(\alpha_j)$$

has 2 cycle-types, namely (1) and (2), whereas

$$\sigma_2(\alpha_1)\sigma_1(\alpha_2)\prod_{j=3}^{n}\sigma(\alpha_j)$$

has only one cycle-type, (1,2). We again regroup the terms in D_n, this time according to the number of cycle-types in each term. Given a set of pairwise disjoint cycle-types $\overline{c_1}, \ldots, \overline{c_j}$, let C be the set of labels appearing in $\overline{c_1} \cup \cdots \cup \overline{c_j}$. Let $\bar{c} = (x_1, x_2, \ldots, x_s)$ be a cycle-type. Denote by $\sigma(\bar{c})$ the number of cycles of cycle-type $\bar{c}$ that can be formed from the elements of L. Note that

$$\sigma(\bar{c}) = \sigma_{x_2}(\alpha_{x_1})\sigma_{x_3}(\alpha_{x_2})\cdots\sigma_{x_1}(\alpha_{x_s}).$$

Claim 2.4 With the notation above,

$$D_n = \prod_{i=1}^{n}\sigma(\alpha_i) + \sum_{j=1}^{n}(-1)^j \sum_{\star}\left(\prod_{l\notin C}\sigma(\alpha_l)\right)\sigma(\overline{c_1})\sigma(\overline{c_2})\cdots\sigma(\overline{c_j})$$

where $\star$ sums over all possible collections, $\bar{c}_1, \ldots, \bar{c}_j$ of j disjoint cycle-types and C denotes the labels represented in these j cycle-types.

Let $r = |C|$, the number of letters used in $\overline{c_1} \cup \cdots \cup \overline{c_j}$. Notice that each term in D_n, as expressed in Eq. (5) is a product of n entries, $n - r$ of which are $\sigma(\alpha_l)$'s. Let j denote the number of cycle-types in a term of D_n. That is, j is the number of cycle-types in a term from some determinant on the right hand side of Eq. (5). Then r is the number of rows in that determinant corresponding to j. Each term from that determinant is a permutation which we will call τ.

To prove Claim 2.4, we need

Claim 2.5 Let j, r, and τ be as defined in the preceeding paragraph. Then,

$$(-1)^r \operatorname{sgn}(\tau) = (-1)^j$$

Proof: Viewing a cycle, c, as a permutation, sgn c makes sense. The proof, then, follows from the fact that, for any cycle, c, of length, l, $(-1)^l \operatorname{sgn} c = -1$. □

Proof of Claim 2.4: According to Eq. (5), each term of D_n comes from, for some r,

$$(-1)^r \sum_{1 \le i_1 < \cdots < i_r \le n} \left(\prod_{l \notin \{i_1, \ldots, i_r\}} \sigma(\alpha_l) \right) \begin{vmatrix} \sigma_{i_1}(\alpha_{i_1}) & \cdots & \sigma_{i_r}(\alpha_{i_1}) \\ \vdots & & \vdots \\ \sigma_{i_1}(\alpha_{i_r}) & \cdots & \sigma_{i_r}(\alpha_{i_r}) \end{vmatrix}.$$

Hence r even or odd plays a role. We are grouping the terms according to the number of cycle-types j, so j matters as well. We know

$$\begin{aligned} &(-1)^r \sum_{1 \le i_1 < \cdots < i_r \le n} \left(\prod_{l \notin \{i_1, \ldots, i_r\}} \sigma(\alpha_l) \right) \begin{vmatrix} \sigma_{i_1}(\alpha_{i_1}) & \cdots & \sigma_{i_r}(\alpha_{i_1}) \\ \vdots & & \vdots \\ \sigma_{i_1}(\alpha_{i_r}) & \cdots & \sigma_{i_r}(\alpha_{i_r}) \end{vmatrix} \\ &\quad = (-1)^r \sum_{1 \le i_1 < \cdots < i_r \le n} \left(\prod_{l \notin \{i_1, \ldots, i_r\}} \sigma(\alpha_l) \right) \left(\sum_{\tau \in S_r} \left(\operatorname{sgn}(\tau) \sigma_{i_{\tau(1)}}(\alpha_{i_1}) \cdots \sigma_{i_{\tau(r)}}(\alpha_{i_r}) \right) \right) \end{aligned}$$

We have shown $(-1)^r \operatorname{sgn}(\tau)$ to depend on j. By Claim 2.5, all terms having an even number of cycle-types are positive and all terms having an odd number of cycle-types are negative.

Notice also, in expressing D_n grouped by cycle-types, that given a collection of j disjoint cycle-types, $\overline{c_1}, \ldots, \overline{c_j}$, for $\overline{c_i} = (x_{i_1}, \ldots, x_{i_r})$ we can write $\overline{c_1}, \ldots, \overline{c_j}$ as

$$(x_{1_1} \cdots x_{1_r})(x_{2_1} \cdots x_{2_s}) \cdots (x_{j_1} \cdots x_{j_t}).$$

There is a corresponding term

$$(-1)^j \left(\prod_{l \notin C} \sigma(\alpha_l) \right) \sigma_{x_{1_2}}(\alpha_{x_{1_1}}) \cdots \sigma_{x_{1_1}}(\alpha_{x_{1_r}}) \sigma_{x_{2_2}}(\alpha_{x_{2_1}}) \cdots \sigma_{x_{j_1}}(\alpha_{x_{j_t}})$$

that can be located in the original determinant by taking

$$\{i_1, \ldots, i_r\} = C,$$

the labels in $\overline{c_1}, \ldots, \overline{c_j}$.

Since all terms in D_n are of this form and all terms of this form appear in D_n with no repeats, we have a one-to-one correspondence. Thus, in Claim 2.4, when grouping all terms with j cycle-types, we also get all possible combinations of j disjoint cycle-types. That is,

$$D_n = \prod_{i=1}^{n} \sigma(\alpha_i) + \sum_{j=1}^{n} (-1)^j \sum_{\star} \left(\prod_{l \notin C} \sigma(\alpha_l) \right) \sigma(c_1)\sigma(c_2) \cdots \sigma(c_j)$$

where $\star$ sums over all possible collections of j disjoint cycle-types and C is the set of labels represented in these j cycle-types. □

Having thus expressed D_n, the goal is to provide a constructive proof that the number of successful shaded sorted-permutations is

$$\left(\prod (\sigma(\alpha_i) - 1)! \right) D_n.$$

To do this, we will be using not only labeled rooted trees, but other geometric figures, as well, defined in the next section.

3. Algorithms for building and recording

3.1. Building arbors from sorted permutations

3.1.1. Making calls. We will say that a parent node *calls* its children. Given a shaded L-sorted permutation of inventory α, each letter has a child type, $k = (k_1, \ldots, k_n)$. The child-type, then, denotes calls to be made. Once a letter is reached, calls are made in increasing order with i-children called before j-children whenever $i < j$. In building trees, these calls go to the first unused letter of the appropriate label, where "unused" means it has neither been called nor made calls. In building wreaths (to be defined), these calls go to the first uncalled letter of the appropriate label (even if that letter has made calls).

Algorithm 3.1 f = *Building an initial forest from a sorted permutation.*

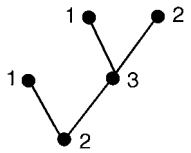
Let w be an L-sorted permutation of inventory $\alpha = (\alpha_1, \ldots, \alpha_n)$, satisfying the formulas $\sigma(\alpha_i) \geq \sum_{j=1}^{n} \sigma_i(\alpha_j)$ for $1 \leq i \leq n$. We build a forest, S, as follows: Determine the number of roots in S of each label, using Eq. (2),

$$p_i = \sigma(\alpha_i) - \sum_{j=1}^{n} \sigma_i(\alpha_j).$$

If there are no roots, then S is empty. Otherwise, start with the lowest root-label, where $1 < 2 < \cdots < n$. When drawing a node, first label the node. Then, draw its labeled edges (as in 2.2) as emanating up from the node, without drawing in the children's nodes. Once a node is drawn, it is crossed out of w and is not available to be drawn again.

Rule 1: Using the leftmost available letter in w of the appropriate label, draw the root. Call its lowest child.
Rule 2: Once a node is drawn, call its lowest child first. Draw this above its parent on the left, and label it. The edge is directed from the node to its child.
Rule 3: If a child is a leaf, go to the leaf's nearest ancestor with calls still to be made. Call its next child, drawing it to the right of previously called children.
Rule 4: Continue this process until there are no remaining calls to be made. We have built one tree.
Rule 5: If there are roots remaining, go to the lowest remaining root, select the leftmost available letter in the appropriate label-group, and repeat Rules 1 through 4, drawing the new tree to the right of previously built trees. Continue until, for each i, there are p_i trees of root-label i.

We never run out of nodes in this process because of the inequalities $\sigma(\alpha_i) \geq \sum_{j=1}^{n} \sigma_i(\alpha_j)$. As a simple example, consider the word $w = (1^{000}, 1^{000})(2^{101}, 2^{000})(3^{110})$. The root-type of w's forest is (0,1,0), and the forest built is:



Note that, at this point, we may or may not have used all the letters in the given sorted permutation. (For example, consider the forest built when we rearrange w to become $w' = (1^{000}, 1^{000})(2^{000}, 2^{101})(3^{110})$.) We call the resulting forest the *initial forest*, S, of w. If S uses all the letters in w, we define w to be a *success*. Otherwise, we declare w to be a *failure*. It is the failures we wish to study further. To do this, we need more machinery, enabling us to continue building with the letters that remain after the initial forest is built.

3.1.2. Machinery for wreaths. A *loop* consists of the nodes and edges of a directed path in a directed graph beginning at one node and returning to that same node. We identify two loops if they consist of the same set of nodes and edges in the same relative order. A *wreath*, W, is a directed, connected graph in which there is one loop and each node has precisely

one edge directed into it. The edges are directed from parent to child, as in a tree. In what follows, wreaths as well as trees are labeled.

3.2 *The wreath's loop will have a lowest label, and in the wreaths we will be considering, that label will appear in the loop only once.*

Given a wreath, W, the closed directed path is called the *direct loop*, DL_W. This can be envisioned as a vertical path, starting with the unique lowest-labeled node on the bottom. The top direct loop node calls the bottom one, and their connecting edge runs counterclockwise. The unique lowest-labeled node is called the *wreath root*, ρ. It coincides with the bottom node in the vertical representation of the direct loop.

Just as we called an ordered collection of trees a forest, a *bramble*, B, is an ordered collection of wreaths such that all i-rooted wreaths come before j-rooted wreaths whenever $i < j$. An *arbor*, A, is an ordered pair (S, B) consisting of a forest, S, and a bramble, B. A *branch* is a minimally connected subgraph of a tree or wreath in which each node has all its descendents present. It has a branch root, that node with no ancestors present. The branch, then, consists of the branch root and all descendants of the branch root.

Within an arbor, the ith root-label group of trees (or wreaths) is the ordered collection of trees (or wreaths) of root-label i.

Algorithm 3.3 $\bar{f}$ = *Building an arbor from a sorted permutation.*

Let w be a failing L-sorted permutation satisfying the formulas

$$\sigma(\alpha_i) \geq \sum_{j=1}^{n} \sigma_i(\alpha_j) \quad 1 \leq i \leq n.$$

Because it is failing, there are letters not used in the initial forest. For each label i, we start with $\sigma(\alpha_i)$ i-letters. In S, we use p_i of these letters as roots and $n_i(w)$ of them as children, for some $n_i(w) \geq 0$. This leaves $\sigma'(\alpha_i) = \sigma(\alpha_i) - p_i - n_i(w)$ i-letters. Also, in S, we start with $\sum_{j=1}^{n} \sigma_i(\alpha_j)$ calls to i-letters. We use $n_i(w)$ of these calls, leaving $\sum_{j=1}^{n} \sigma_i(\alpha_j) - n_i(w)$. Subtracting $p_i + n_i(w)$ from both sides of Eq. (2) leaves $\sigma'(\alpha_i) = \sum_{j=1}^{n} \sigma_i(\alpha_j) - n_i(w)$. Thus, the number of calls remaining to a given label group equals the number of letters of that label remaining. In other words, there are no roots left. We will form wreaths with the leftover letters, using the following rules.

Rule 1: Given w, use f (as in Algorithm 3.1) to build the initial forest, S. If all letters are used, then the bramble is empty. Otherwise, go to Rule 2.
Rule 2: We begin this step with a forest, S', initially empty. We call S' a holding forest, as it will be used to hold trees which will later be moved. Reading the sorted permutation, w, left-to-right, start with the first letter that is not yet used. Draw and label it as a node in S'. Call its lowest child, which will be the leftmost letter of the appropriate label which is not in the initial forest and has not been called (see Section 3.1.1). It is drawn above the first node, on the left, with the edge directed away from the first node.

Rule 3: Each node called calls its lowest child first, just as in trees. Continue as in Rules 2 and 3 of f, making calls as directed in Section 3.1.1 until we form a tree or we repeat the first node.
Rule 4a: If we have formed a tree with root-label i, then we move on to the next letter in w which has neither been called nor made calls, draw it to the right of the i-rooted group in S', and repeat the process, returning to Rule 2. Notice that the next call to j for $j \leq i$ goes to the leftmost j-rooted tree in S', as it has not yet been called, and places this tree as a branch above the letter making the call.
Rule 4b: If the first node has been repeated, then we have a direct loop. This is drawn with the first node (now the wreath root) at the base. Edges are directed up to children. The call back to the wreath root bends counterclockwise to form the loop, enclosing the nodes that have already been drawn but are not on the direct loop. Once the first node has been repeated, we say the wreath is *closed* and we move this component out of S' and into B, placing it rightmost in its root-label group. Go to Rule 5.
Rule 5: To complete the wreath, we need to finish making the calls outside the direct loop. First, identify the direct loop. Writing this loop vertically, we begin with the top node in the direct loop (that node which calls the wreath root) and make its remaining calls, moving left to right, finishing each branch before moving on to the next. Notice that these calls will be drawn outside the loop. Next, we move down the direct loop and make the next loop-node's remaining calls, and so on until all direct loop nodes have made their calls. (This process is consistent with the algorithm for building a tree, if we regard the call back to the wreath root as a leaf of its parent node.) We have now built one wreath.
Rule 6: If there are still letters remaining, repeat the process until all letters have been used, putting new wreaths on the right to form a bramble. Notice, by the argument preceding Rule 1, the holding forest, S', will again be empty when we finish.

3.4 (Observations about Algorithm 3.3) We don't run out of letters because of the relation between the number of letters of each label and the number of times letters of that label are called. The lowest label in the direct loop of a wreath is that of the wreath root. This is because any calls to labels lower than that of the wreath root are made to trees already formed. The root-label is never repeated in the direct loop, although it may appear in branches of the wreath other than the direct loop. Also, the bramble formed is in nondecreasing root order. These are all direct results of Algorithm 3.3.

3.2. Recording arbors as sorted permutations

To record an arbor, we proceed one connected component at a time. In what follows, a *tree-geodesic* is a directed path emanating from the root and ending in a leaf. A *wreath-geodesic* is a directed path containing precisely one direct loop node, z. Note, then, the path emanates from z. We also say the path is a *z-geodesic*.

Algorithm 3.5 $g_1 =$ *Recording a tree as a sorted permutation.*

Procedure for recording: letters are recorded from right to left in their appropriate label groups. Once a letter is recorded, its node is deleted from the graph, but the labeled edge

emanating from its parent node, called an *incomplete edge*, remains. Thus, we delete the recorded letter's node and its outgoing edges.

Rule 1: Given a tree, go up the rightmost geodesic and record the leaf.
Rule 2: Once a node, x, is recorded, go up the rightmost remaining geodesic, ignoring incomplete edges, and record the last node still present.
Rule 3: Repeat Rule 2 until all nodes in the tree have been recorded. Note that the root is the last node recorded.

This procedure for recording a tree as a sorted permutation was known previously. See Wilf [8], for a similar algorithm.

In what follows, we need to denote the label of a node, x, which we will do by writing $\ell(x)$.

Algorithm 3.6 $g_2 =$ *recording a wreath as a sorted permutation.*

We will be dealing with a pair (S', W) where S' is a holding forest and W is a wreath. We begin with S' empty. As the recording process proceeds, this empty forest will become a holding place for trees which will be removed from the wreaths. These trees are recorded in the final step.

Rule 1: Let ρ be the wreath root. Note the root-label of the wreath, $\ell(\rho)$.

Before we state Rule 2, note that the recording of a wreath will involve two procedures, "removing" and "recording." The following procedures, and their criteria, are needed before we can discuss the remaining rules.

Procedure for removal. This procedure applies to the removal of non-direct loop nodes. If the branch of a node, x, is to be removed from W, we first delete x and all its descendents from W. We then put this branch, x and its descendents, leftmost in the $\ell(x)$ root-label group of the holding forest, S'.

Procedure for recording. If a node, x, is to be recorded, all descendents, with the exception of the wreath root, will have been removed or recorded. Write its letter leftmost in the $\ell(x)$ label-group of w. Then, delete x and all its outgoing edges from W. The incoming edge remains.

Criterion for removing an outer node. A node, x, and its branch outside the direct loop of W are removed if $\ell(x) < \ell(\rho)$.

Criteria for recording an outer node. A node x, outside the direct loop of W, is recorded if all descendents of x have been removed or recorded (or never existed), and $\ell(x) \geq \ell(\rho)$.

Criterion for removing an S' node. A node, x, and its branch inside the direct loop of W are removed if $\ell(x) \leq \ell(\rho)$.

Criteria for recording an S' node. A node x inside the direct loop of W is recorded if all descendents of x have been removed or recorded (or never existed), and $\ell(x) > \ell(\rho)$.

Criterion for proceeding. We proceed to the rightmost child of x if x is neither removed nor recorded.

Rule 2: If ρ has no outer children, go to Rule 5. If ρ does have outer children, proceed up the rightmost ρ-geodesic.
Rule 3: At each node x, either remove x and its branch, record x, or proceed to the rightmost child of x, according to the above criteria. If we proceed, repeat this rule. If we record or

remove, go to x's parent, x', and redefine the rightmost ρ-geodesic. Starting with x', repeat this rule.

Continue until all outer children of ρ have been recorded. (Note: none of the outer children of ρ will be removed, since their labels are greater than or equal to $\ell(\rho)$.) Then, go to Rule 4.
Rule 4: Move up the direct loop to the next direct loop node, z. Repeat Rules 2, 3, and 4 using z instead of ρ, except when referring to $\ell(\rho)$, which remains fixed. Continue until all of W's outer nodes have been removed or recorded. Then, go to Rule 5.
Rule 5: Once all the outer children are recorded, open the loop (without recording any nodes), and move what is now a tree to the rightmost spot in S'.
Rule 6: What remains are the removed branches, now trees in S', whose roots have labels less than or equal to $\ell(\rho)$. Notice that these branches are trees, but we will not record them using the algorithm for recording trees. Instead, go to the rightmost tree. Note the root-label and proceed up the rightmost geodesic, ignoring incomplete edges. At each node, remove, record, or proceed according to the criteria above. After removing or recording, redefine the rightmost geodesic, and repeat the process. Record or remove all nodes, and then record the root. Go left to the next tree. Continue until all trees are recorded. This is a finite process since we always record the root.

3.2.1. Recording an arbor. To record an arbor $A = (S, B)$, we begin with the rightmost wreath of B, and follow Algorithm g_2 up to, but not including Rule 6. Before recording the rightmost remaining wreath in B, note its root-label, x. Go to S' and follow the procedure of Rule 6 until there are no trees having root-label greater than or equal to x remaining. If branches were removed during this process, so long as their root-label is less than x, they stay in S' for now. We then record the rightmost remaining wreath, adding to the forest, S', by putting new trees on the left. Again, we only follow Rule 6 until there are no trees remaining having root-label greater than or equal to that of the next wreath. Continue recording the wreaths and pieces of the holding forest, until all wreaths have been recorded. Then record the remaining constructed forest following Rule 6 of g_2. At this point, all the nodes of B have been recorded. We will refer to the process of recording the bramble as $\tilde{g}_2$. Next, record the rightmost tree of S following Algorithm g_1, and record each tree until all the nodes of S have been recorded.

3.3. Lemma about building and recording

The goal of this section is to show that building and recording are inverse procedures.

Define $\mathbb{W}_L$ to be the set of L-sorted permutations, and define $\mathbb{A}_\alpha$ to be the set of arbors with inventory α. Let $\bar{f} : \mathbb{W}_L \to \mathbb{A}_\alpha$ as in Algorithm 3.3. Using the notation of Section 3.2.1, let $\bar{g} = (\tilde{g}_2, g_1) : \mathbb{A}_\alpha \to \mathbb{W}_L$ by first recording the bramble according to $\tilde{g}_2$, and then recording the initial forest using g_1. Note that, at any stage of $\bar{f}$ or $\bar{g}$'s procedures, we have a triple, (w, S', A). In this triple, the sorted-permutation, w, the holding forest, S', and the arbor, $A = (S, B)$, are all in various stages of composition or decomposition. Thus, we can view $\bar{f}$ and $\bar{g}$ as acting on such a triple at each step of the process.

Claim 3.7 $\bar{f}$ and $\bar{g}$ are inverse maps.

Proof: Let (w, S', A) be any triple operated on by $\bar{f}$, resulting in $(\bar{w}, \bar{S}', \bar{A})$. To show that $\bar{g} \circ \bar{f}(w) = w$, we will show that $\bar{g}$ acting on $(\bar{w}, \bar{S}', \bar{A})$ results in (w, S', A).

We first define the operations of $\bar{f}$ and $\bar{g}$. There are six ways $\bar{f}$ can operate on (w, S', A):

(i) $\bar{f}$ draws a node, x, on a tree in A
(ii) $\bar{f}$ draws a node, x, on a tree in S' (Note: we are given the word, w, so it is possible to determine the root-type of S, and thus distinguish between (i) and (ii).)
(iii) $\bar{f}$ draws a node, x, on a closed wreath
(iv) $\bar{f}$ moves a tree of S', having root x, to an outside branch of a wreath
(v) $\bar{f}$ moves a tree of S', having root x, onto another tree in S'.
(vi) $\bar{f}$ closes a loop on a tree of S' and moves the new wreath into A.

Similarly, there are five operations for $\bar{g}$:

(i) $\bar{g}$ records a node from a tree in $\bar{A}$
(ii) $\bar{g}$ records a node from a tree in $\bar{S}'$
(iii) $\bar{g}$ records a node from an outside branch of a wreath
(iv) $\bar{g}$ removes a branch outside a wreath and places it in $\bar{S}'$
(v) $\bar{g}$ removes a branch from a tree in $\bar{S}'$
(vi) $\bar{g}$ opens a loop on a wreath in $\bar{A}$ and moves the new tree into $\bar{S}'$.

We will proceed one operation at a time. If, acting on (w, S', A), $\bar{f}$ performed operation (i), x is the rightmost node drawn in the rightmost geodesic extending from the root of the rightmost tree in what is now $\bar{A}$. Further, S', and therefore $\bar{S}'$, must be empty, as must be the bramble. Thus, we can identify the operation $\bar{g}$ performs as coming from g_1. Rule 2 of g_1 tells us to record the same x as drawn by $\bar{f}$.

Next, suppose $\bar{f}$ performed operation (ii). Again, x is drawn on the rightmost geodesic in S', now renamed $\bar{S}'$. Letting ρ be the root of the tree, we know all ancestors of x, excluding the root, have label greater than the label of ρ. We also know the rightmost wreath has root-label less than or equal to $\ell(\rho)$. Thus, by Section 3.2.1 Rule 6, $\tilde{g}_2$ travels up this rightmost geodesic to x, and records it.

We now turn to operation (iii). By Rule 4 of $\bar{f}$, x is placed on the rightmost geodesic emanating from the highest direct loop node having incomplete branches. Turning to g_2 at this point, since the wreath is closed, Rules 2, 3, and 4 of g_2 tell us the next operation occurs on the rightmost geodesic emanating from the lowest direct loop node having outside children remaining. Thus, we have identified the same geodesic. Since $\bar{f}$ drew x, it also drew all the ancestors of x in this geodesic, as opposed to moving trees to form branches. Let the wreath root be ρ. Therefore, all such ancestors have label greater than or equal to $\ell(\rho)$. According to g_2's procedures, then, we continue up the geodesic until reaching x. Since, at this stage, x has no descendents and $\ell(x) \geq \ell(\rho)$, g_2 records x.

If operation (iv) was performed by $\bar{f}$, again let ρ be the root. Then we know $\ell(x) < \ell(\rho)$ and x is now a node in the rightmost geodesic emanating from the lowest direct loop node having children in place. Further, all ancestors of x in this geodesic have labels greater than or equal to $\ell(\rho)$. So, g_2 proceeds up this same geodesic. Upon reaching x, g_2 removes x and its branch and places it back in S', leftmost in its root-label group.

We next consider operation (v) of $\bar{f}$. Here, $\ell(x) \leq \ell(\rho)$, and x is in the rightmost geodesic of that tree. All ancestors of x, excluding the root, must have label greater than $\ell(\rho)$. Therefore, g_2 travels up this geodesic until reaching x, at which time x and its branch are removed and placed in S' as a tree leftmost in its root-label group.

Finally, we turn to operation (vi) of $\bar{f}$. Since $\bar{f}$ moved the new wreath into $\bar{A}$, all entries in $\bar{S}'$ will have root color less than that of the new wreath. Thus $\bar{g}$ will operate on the new wreath. By Rule 5 of $\bar{g}$, since there are no outer children, the wreath is opened and placed in S'.

We have shown that $\bar{g} \circ \bar{f}(w) = w$. The proof for $\bar{f} \circ \bar{g}$ proceeds in a similar manner. □

4. The map Φ

4.1 For W, a wreath, we can identify an *associated cycle*, c_W, as follows. Write the direct loop vertically. Proceed from bottom to top until a label has been repeated. Remove from the direct loop all nodes from the first occurrence of the repeated label, up to, but not including, the second occurrence. Rewrite the direct loop without the removed nodes, so the second occurrence is now in the place formerly occupied by the first occurrence, and repeat this process until all labels are distinct in the direct loop. What remains is called the associated cycle. Note that the associated cycle contains W's wreath root, as its label is never repeated in DL_w (see 3.4).

Lemma 4.2 *Let W be a wreath with associated cycle $c_W = (b_1, \ldots, b_l)$. Then, if b' is the node following b_{i-1} in DL_W, then $\ell(b') = \ell(b_i)$.*

Proof: Since $c_W = (b_1, \ldots, b_l)$, then b' was removed in 4.1. Since b_{i-1} was not removed, then $\ell(b')$ was repeated. If $\ell(b')$ was first repeated at b_i, we are done. Otherwise, it was repeated before b_i, as b_i was not removed. Say $\ell(b')$ was first repeated at b''. Once the nodes from b' up to but not including b'' are removed, we begin again. We know b'' was removed in 4.1, since $b'' \neq b_i$. By similar arguments, $\ell(b'') = \ell(b')$ is repeated no later than b_i and we remove a sequence of nodes beginning at b''. There are a finite number of nodes between b_{i-1} and b_i to remove, therefore, at some point the first repeated label must be at b_i. Thus, $\ell(b') = \ell(b_i)$. □

4.3 Given an L-sorted permutation, w, Φ associates a set of disjoint cycles to w in such a way that w succeeds precisely when this set is empty. The steps to Φ's algorithm are as follows:

Step 1: Form $A = \bar{f}(w)$. If $A = (S, B)$ is a forest, that is, if B is empty, define $\Phi(w) = \emptyset$ and stop. Else, proceed to Step 2.
Step 2: Go to the rightmost wreath, W. Identify c_W as in 4.1. Record this cycle in the image of Φ.
Step 3: Move left in the arbor until reaching the first wreath, $\widehat{W}$, with a new, hence lower, root-label. If there is no such wreath, $\Phi(w)$ is the set of recorded cycles. If $c_{\widehat{W}}$ is disjoint from the *direct loop* of every wreath in B to the right of $\widehat{W}$, record $c_{\widehat{W}}$ in the image of Φ. Repeat this Step until there are no new wreaths to consider.

Notice that not all wreaths are eligible to contribute to the image of Φ; only the rightmost wreath of each root-label is considered. (Step 3 also makes it useless to consider the other wreaths, as their roots prevent them from being disjoint from the next direct loop).

The result of this algorithm is that, given a sorted permutation, Φ draws the arbor and associates to that arbor a set of pairwise disjoint cycles. Note that $\Phi(w) = \emptyset$ if and only if the arbor associated to w has no wreaths, i.e. w is a success.

5. The map Ψ

5.1. *Prelude to* Ψ

5.1.1. Construction definitions. Given a shaded collection of letters, L, let c be a cycle from it. Let $\bar{L} = L - L_c$, where L_c is as in Section 2.1. An $\bar{L}$*-sorted permutation*, $\bar{w}$, is a sorted permutation of $\bar{L}$. We say $\bar{w}$ is *complementary to* c in L. Note that $\bar{L}$ has root-type $\bar{p} = (\overline{p_1}, \ldots, \overline{p_n})$ with $\overline{p_i} \geq p_i$ for all i. (See Eq. (2).)

In Section 5.3 below, we begin with a cycle and a complementary $\bar{L}$-sorted permutation, and we construct a wreath with the given cycle as the direct loop. This wreath is called the *constructed wreath* and is denoted $\overline{W}$. Its wreath root is denoted $\bar{\rho}$, and is the node of lowest label in the given cycle.

We say a wreath, W, *belongs* to a cycle, c, if that cycle is associated to the wreath (see 4.1).

Notation 5.1 For two arbors, A_1, A_2, we write $A_1 \cdot A_2$ as the arbor whose ith root-label group of trees is the ith root-label group of trees from A_1 followed by the ith root-label group of trees from A_2. The ith root-label group of wreaths is similarly constructed.

Given a shaded collection of letters, L, and a cycle, c, from it, we list all $\bar{L}$-sorted permutations. There are

$$\frac{\prod_{i=1}^{n}(\sigma(\alpha_i)!)}{\prod_{l\in c}\sigma(\alpha_l)}$$

$\bar{L}$-sorted permutations.

5.2. *Splicing and unsplicing*

Notation 5.2 Recall, the cycle associated to wreath W is c_W and the direct loop is DL_W. Also, for any collections of nodes a and b, we say $a \cap b = \emptyset$ if a and b have no labels in common. If they do have a label in common, we say $a \cap b \neq \emptyset$. This is consistent with the notation from Section 2.1.

Let

$$X_L = \{(B, \overline{W}) \mid B \text{ a bramble, } \overline{W} \text{ a wreath not in } B \text{ with } DL_{\overline{W}} = c_{\overline{W}},\ L = L_B \cup L_{\overline{W}}\}$$

where L_B and $L_{\overline{W}}$ are the sets of letters in B and $\overline{W}$, respectively. Let

$$Y_L = \{(B, W) \mid B \text{ a bramble}, W \in B, c_W \cap DL_{W'} = \emptyset \text{ for } W' \text{ right of } W, L = L_B\}.$$

5.2.1. Algorithm for splicing, ζ. We define $\zeta : X_L \to Y_L$ via the following steps.

Step 1: Let $(B, \overline{W}) \in X_L$. Write $B = B_1, \ldots, B_n$, where B_j is all wreaths with root-label j. Suppose $\overline{W}$ has root-label i. Starting with B_n and finishing with B_{i+1} we proceed with a general root-label group B_j as follows.
Step 2: Suppose $B_j = W_1, \ldots, W_r$ is a bramble where all wreath roots have label $j > i$. Consider the sequence $W_1, \ldots, W_r, \overline{W}$, where $\overline{W}$ is the given wreath with root-label i.
Step 3: Start with W_r. If $DL_{W_r} \cap c_{\overline{W}} = \emptyset$, move W_r to the immediate right of $\overline{W}$ and repeat this step with W_{r-1}. Continue until some wreath, W_k, has $DL_{W_k} \cap c_{\overline{W}} \neq \emptyset$ or until all wreaths have scooted past $\overline{W}$, preserving their relative order. If $DL_{W_k} \cap c_{\overline{W}} \neq \emptyset$, go to Step 4. If all wreaths scoot, go to Step 6.
Step 4: Identify the first cycle node in $c_{\overline{W}}$, starting with $\rho_{\overline{W}}$, whose label appears in DL_{W_k}. Call it b_i. Let b_{i-1} be the previous cycle node in $\overline{W}$. We know that b_{i-1} and b_i are different labels since all cycle nodes have distinct labels. Let b' be that node following b_{i-1} in $DL_{\overline{W}}$. Now, b' may not be b_i, but note that b' and b_i have the same label, by Lemma 4.2. We call b' the *splicing node*. Next, go to the first DL_{W_k} node of this label, call it x. Detach x from its parent node, x', and splice W_k into $\overline{W}$ so that $b_{i-1}, x, \ldots, x', b'$ is now in $DL_{\overline{W}}$. Note that we no longer have $DL_{\overline{W}} = c_{\overline{W}}$, but the associated cycle of $\overline{W}$ has not changed (see Lemma 5.3 below). Go to Step 5.
Step 5: Now, we splice all other wreaths W_l in B_j with $l < k$ as follows (note this process may differ from Step 4): If $c_{\overline{W}}$ has a node at or before b_i meeting DL_{W_l}, reset $k = l$ and go to Step 4. Otherwise, splice in between b_{i-1} and b_i before ρ_k, the first node of label j between b_{i-1} and b_i. If $l > 1$, reset $k = l$ and repeat Step 5.
Step 6: Return to Step 1, replacing the original $\overline{W}$ with the altered $\overline{W}$ if necessary, and repeat the process for the next root-label group. Continue until reaching and processing B_{i+1}.

Let $\overline{W}'$ be the result of splicing wreaths into $\overline{W}$, and B'_j be the sub-bramble of B_j consisting of those wreaths that scooted past $\overline{W}$ in Step 3. Also, let $B_i \cdot \overline{W}'$ be defined as in Notation 5.1. Set

$$B' = (B_1, \ldots, B_i \cdot \overline{W}', B'_{i+1}, \ldots, B'_n).$$

Then, define $\zeta(B, \overline{W}) = (B', \overline{W}') \in Y_L$.

Lemma 5.3 *Let $\overline{W}$ and W be wreaths such that $DL_W \cap c_{\overline{W}} \neq \emptyset$. Then, splicing W into $\overline{W}$ does not change the associated cycle of $\overline{W}$.*

Proof: By Step 4 of ζ, after splicing W into $\overline{W}$ to form $\overline{W}'$, the new direct loop looks like

$$\rho_{\overline{W}'} = b_1, \ldots, b_2, \ldots, b_{i-1}, x, \ldots, x', b', \ldots, b_i, \ldots, b_l$$

where, before splicing, $c_{\overline{W}} = (b_1, \ldots, b_l)$. The b_j are necessarily distinct labels. By 4.1, to determine $\overline{W}'$'s associated cycle, we start with $\rho_{\overline{W}'}$ and continue along the direct loop until a label is repeated.

Suppose first that x is the first node whose label is repeated. By construction, this happens no later than b'. If it occurs at b', we remove all of W's direct loop nodes and the proof is complete. If it occurs at x_2 before b', we remove from x up to, but not including x_2. Then, x_2's label is repeated, again no later than at b'. Thus, as in Lemma 4.2 the nodes from x_2 up to but not including b' will be removed after a finite number of steps.

If x is not the first node whose label is repeated, then by virtue of $(B, \overline{W})$ being in X_L, we are dealing with an earlier wreath (or wreaths) that was spliced in before b_{i-1}. The above paragraph says that this earlier wreath will be removed without interfering with W's cycle nodes, and then W will also be removed. Hence $c_{\overline{W}} = c_{\overline{W}'}$. □

5.2.2. Algorithm for unsplicing, μ. For a set L, $\mu : Y_L \mapsto X_L$ is defined as follows.

Step 1: Let $(B, \overline{W}') \in Y_L$. Identify the associated cycle of the given wreath, $\overline{W}'$, as $c_{\overline{W}'} = (b_1, \ldots, b_l)$. Say $\overline{W}'$ has root-label i. That is, $\ell(b_1) = i$.
Step 2: Starting with label $i + 1$ and ending with label n, for a general label $j > i$, proceed as follows.
Step 3: Call the block of direct loop nodes in between, but not including, cycle nodes b_k and b_{k+1} as Bl_k. Scan the blocks in increasing order, looking for j-labeled nodes. If none are found, declare $\widetilde{B_j} = \emptyset$ and go to Step 6. Otherwise, at the first occurrence, say in block Bl_k, name as s_1 the first node of Bl_k in $\overline{W}'$. Note that s_1, of label s, has the same label as b_{k+1} (according to Lemma 4.2). Moving up $DL_{\overline{W}'}$, name as $s_2, \ldots, s_n$ all nodes with label s in Bl_k. Name b_{k+1} as s_{n+1}. Next, name the first appearance of a j-labeled node in Bl_k, ρ_1. Name the other nodes with this label in Bl_k as $\rho_2, \ldots, \rho_m$, moving up $DL_{\overline{W}'}$. Notice that, if $j = s$, then $s_x = \rho_x$ for all $1 \leq x \leq n$.
Step 4: Consider the sequence of nodes $s_1, \ldots, s_2, \ldots, \widehat{s_r}$ where s_r is the first s-node following ρ_1 and the $\widehat{\ }$ means we do not include this node in the sequence. If ρ_1 is the only j-labeled node in this sequence, remove the nodes $s_1, \ldots, \widehat{s_r}$, rotate them clockwise until ρ_1 is the root, and place this new wreath, called γ, immediately to the left of $\overline{W}'$. On the other hand, if ρ_1 is not the only j-labeled node in this sequence, remove only the nodes in the consecutive subsequence $\rho_1, \ldots, \widehat{\rho_2}$ and place this wreath, γ, immediately to the left of $\overline{W}'$. Go to Step 5.
Step 5: Redraw $\overline{W}'$ without γ's nodes. Repeat Steps 3 and 4 until all j-labeled nodes are removed from the blocks of $\overline{W}'$. Note that we do not remove any nodes of the associated cycle. Thus, we form a sub-bramble, $\widetilde{B_j}$. Go to Step 6.
Step 6: Now, look to the right of $\overline{W}'$ in B, to the sub-bramble B_j. Move these wreaths, preserving their order, left of $\overline{W}'$ and form $B'_j = \widetilde{B_j} \cdot B_j$ (See Notation 5.1). Go to Step 7.
Step 7: If $j \neq n$, replace j with $j + 1$ and return to Step 3. If $j = n$, we are done. Here, we observe that the blocks of $\overline{W}'$ now contain no nodes having label greater than i. Since i is the lowest label appearing, and the root is the only node having that label, all blocks are now empty. This results in $\overline{W}'$ having been altered to a wreath whose direct loop equals its associated cycle. We call this new wreath $\overline{W}$.

Let $\overline{B_i}$ equal B_i without $\overline{W}'$. Then, letting $B' = B_1, \ldots, B_{i-1}, \overline{B_i}, B'_{i+1}, \ldots, B'_n$, we have

$$\mu : (B, \overline{W}') = \mu(B_1, \ldots, B_n, \overline{W}') = (B', \overline{W}) \in X_L.$$

Claim 5.4 $\mu \circ \zeta = Id_{X_L}$ and $\zeta \circ \mu = Id_{Y_L}$.

Proof: First $\mu \circ \zeta$: Let $(B, \overline{W}) \in X_L$. Let $B = W_1, \ldots, W_s$, and let $\overline{W}$ have root $\bar{\rho}$ with label i. Suppose at some stage of ζ we have

$$W_1, \ldots, W_r, \overline{W}', W_{h_1}, \ldots, W_{h_t},$$

and the root, ρ_r of W_r has label $j > i$. The proof will consist of showing that, after ζ operates on W_r, if we turn to μ, the first thing μ does is undo that operation. There are three possible operations ζ may perform on W_r.

Case 1 The wreath W_r is scooted by ζ to the right of $\overline{W}'$. By Step 3 of ζ, this means that $DL_{W_r} \cap c_{\overline{W}'} = \emptyset$. Further, by Step 5 of ζ, no other wreaths of root-label j have spliced in. Therefore, all blocks of $DL_{\overline{W}'}$ contain only nodes of higher label than j (recall, the root of $\overline{W}'$ has label i, and this is the lowest label in the direct loop and appears only at the root). Thus, Step 6 of μ moves W_r back to the left of $\overline{W}'$.

Case 2 ζ splices W_r into $\overline{W}'$ at a cycle node. Then, with the notation of Step 3 of μ, $s_1 \in W_r$ and there is an s-node after the wreath root ρ_r no later than the next j-labeled node. By Step 4 of μ, W_r is removed intact and placed where it started, immediately to the left of $\overline{W}'$.

Case 3 ζ splices W_r into $\overline{W}'$ at the root of a previously spliced wreath. By Step 5 of ζ, this means ρ_r is in the sequence $s_1, \ldots, s_2, \ldots, \widehat{s_r}$. Further, $s_1 \notin W_r$, and therefore another j-node appears in the sequence $s_1, \ldots, s_2, \ldots, \widehat{s_r}$. Step 4 of μ splices out the nodes from ρ_r up to but not including the next j-node, which are precisely the nodes of W_r.

Now, to $\zeta \circ \mu$: Let $(B, \overline{W}') \in Y_L$. Again, we suppose that at some stage of μ we reach the sequence

$$W_1, \ldots, W_r, \overline{W}', W_{h_1}, \ldots, W_{h_t}.$$

We will show that the next operation performed by μ is undone if we turn at that time to ζ. Again, we have three cases.

Case 1 By Step 6 of μ, the wreath W_{h_1} is unscooted by μ. This is done only after all other wreaths of that root-label have been unspliced from $\overline{W}'$. Because $(B, \overline{W}') \in Y_L$, we know that $DL_{W_{h_1}} \cap c_{\overline{W}'} = \emptyset$. Thus, at this point, ζ would scoot W_{h_1} past $\overline{W}'$, as desired.

Case 2 The sequence of nodes $s_1, \ldots, \widehat{s_r}$ is unspliced by μ from Bl_k. By Step 4 of μ, this means we have removed one wreath, W, whose direct loop has an s-node. Thus, ζ

will splice that wreath back into $\overline{W}'$ because $s \in DL_W \cap c_{\overline{W}'}$. Further, since we identified $c_{\overline{W}'} = (b_1, \ldots, b_l)$ in Step 1 of μ, before μ did any operations, then $DL_W \cap \{b_1, \ldots, b_k\} = \emptyset$. Indeed, if $DL_W \cap \{b_h\} \neq \emptyset$ for some $1 \leq h \leq k$, then $b_h \notin c_{\overline{W}'}$ by definition of associated cycle (see 4.1). Thus, ζ will splice W back into the same block, Bl_k, from which it came. Lastly, by the way x is defined in Step 4 of ζ, W will splice back in the same order.

Case 3 The sequence of nodes $\rho_1, \ldots, \widehat{\rho_2}$ is unspliced by μ from Bl_k and $\ell(\rho_1) \neq s$. Therefore, $\rho_2 \in BL_k$ (instead of being a cycle node). Thus, by Step 5 of ζ, the removed wreath, W must splice back into $\overline{W}'$. For reasoning similar to Case 2, it will splice into the same block. Because s is not a label in DL_W, W will splice back in as $\rho_1, \ldots, \widehat{\rho_2}$. □

5.3. The map Ψ defined

For c a cycle from L and $\bar{w}$ an $\bar{L}$-sorted permutation (see Section 5.1.1), Ψ acts on an ordered pair $(\bar{w}, c)$ to give an arbor A.

Step 1: Draw c as an incomplete wreath—a direct loop with all other branches empty. Assign the title of wreath root, $\bar{\rho}$, to the unique letter of lowest label appearing in c. This direct loop may have calls remaining to be made. Referring to 2.2 for notation, let

$$\delta_j = \begin{cases} 1 & \text{if some } i^k \in c \text{ has } \ell(i^k) = j \\ 0 & \text{if no } i^k \in c \text{ has } \ell(i^k) = j \end{cases}$$

Let $r = (r_1, r_2, \ldots, r_n)$ satisfy $r_j = \sum_{i^k \in c} k_j - \delta_j$. Thus, r_j is the number of j-labeled children of loop nodes that need to be filled in.
Step 2: Form an arbor from $\bar{w}$, $A' = (S', B')$. Notice that the root-type of $\bar{w}$ is $p + r$, where p is the root-type of L (see Eq. (2)).
Step 3: Complete the direct loop into a wreath in the following order. When a call is made to a label i, it goes to the leftmost i-rooted tree in A', and the entire tree becomes a branch in the constructed wreath. First, complete any calls by $\bar{\rho}$ to branches that are inside the direct loop, preceding the direct loop branch of $\bar{\rho}$. This is done left to right. Then, move up the loop, making all inside calls. Once this is done, proceed to the top node in c's direct loop, and make it's outside calls left to right. Moving down the direct loop, continue until all outside calls are made. Notice that, graphically, we are moving bottom to top on the inside calls and then top to bottom on the outside calls, just as in the algorithm for building a wreath. Thus, we have constructed our wreath, $\overline{W}$.
Step 4: An arbor, $A'' = (S'', B'')$ remains from A', with $B'' = B'$, and we have a wreath $\overline{W}$. Next, put the wreath roots in order by splicing. We define $\Psi(\bar{w}, c) = (S'', \zeta(B'', \overline{W}))$.

Note Step 4 of Ψ assures us that the wreath constructed with the given cycle c is the rightmost wreath in its root-label group, and that c does not intersect the direct loops of those wreaths to the right.

6. Lemmas about Φ and Ψ

In this section, we show an inverse relationship between Φ and Ψ. In some lemmas that follow, we will need to extend the definitions of μ and ζ. We will call these extensions $\bar{\mu}$ and $\bar{\zeta}$. Again, the shaded set, L, is fixed. Using the vocabulary of Section 5.1.1, let

$$\overline{X_L} = \{(A', c) \mid c \text{ a cycle, } A' \text{ an arbor complementary to } c\}$$

and, for $\bar{g}$ as defined for Claim 3.7,

$$\overline{Y_L} = \{(A, W) \mid A \text{ an } L\text{–arbor, } W \in A,\ c_W \in \Phi(\bar{g}(A))\}.$$

For A' complementary to c in L, let $\bar{A} = \Psi(\bar{g}(A'), c)$ and W be the wreath belonging to c in $\bar{A}$ (again, see Section 5.1.1). Define

$$\bar{\zeta} : \overline{X_L} \to \overline{Y_L} \text{ by } \bar{\zeta}(A', c) = (\bar{A}, W).$$

In order to define $\bar{\mu} : \overline{Y_L} \to \overline{X_L}$, we begin with $(A, W) \in \overline{Y_L}$. Suppose $A = (S, B)$. Let c be the associated cycle of W and A' be the arbor obtained by the following method: First, find $\mu(B, W)$. Call the resulting bramble B'. Then, disassemble the branches of W in the reverse order described in Step 3 of Ψ. As each branch is removed, it is placed as a tree on the left of its root-label group in S, forming a new forest, S'. Let $A' = (S', B')$. Then define

$$\bar{\mu}(A,\ W) = (A', c).$$

Notice that $\bar{\zeta}$ and $\bar{\mu}$ are again inverse procedures.

Lemma 6.1 *$c \in \Phi(w)$ if and only if there is a sorted permutation, $\bar{w}$, complementary to c, such that $w = \Psi(\bar{w}, c)$*

Proof: $\Leftarrow$: This is purely by design.

$\Rightarrow$: Let $c \in \Phi(w)$. Let W be the wreath in $A = \bar{f}(w)$ belonging to c (see Section 5.1.1). Then $(A,\ W) \in \overline{Y_L}$ and

$$(A, W) = \bar{\zeta} \circ \bar{\mu}(A, W) = \bar{\zeta}(A', c) = (\Psi(\bar{g}(A'), c), W).$$

Thus, $\bar{g}(A')$ is the complementary sorted permutation sought. □

This establishes a relationship between Ψ and Φ that will allow us to count the failing L-sorted permutations. To do this, we need two more lemmas.

Lemma 6.2 *Given c_1, c_2 disjoint L-cycles, with $\ell(\rho_{c_1}) < \ell(\rho_{c_2})$, let $A = \Psi(\bar{w}_2, c_2)$ where $\bar{w}_2$ is complementary to c_2 in L. Then, for $w = \bar{g}(A)$,*

$$c_1 \in \Phi(w) \Leftrightarrow c_1 \in \Phi(\bar{w}_2)$$

Proof: $\Rightarrow$: Suppose $c_1 \in \Phi(w)$. We know that c_1 is associated to the rightmost wreath, W_1, of its root-label group in $A = \bar{f}(w)$ and therefore in $\bar{f}(\bar{w}_2)$. Also, c_1 is disjoint from all direct loop nodes to the right of W_1 in A. Since $\ell(\rho_{c_1}) < \ell(\rho_{c_2})$, in applying Ψ to $(\bar{w}_2, c_2)$, W_1 is not disturbed. While the wreaths in $\bar{f}(\bar{w}_2)$ may differ from those in A (since A is formed using splicing), the direct loop nodes of those wreaths to the right of W_1 in $\bar{w}_2$ are a subset of those nodes in A, and therefore c_1 is disjoint from them as well. Hence, $c_1 \in \Phi(\bar{w}_2)$.

$\Leftarrow$: If $c_1 \in \Phi(\bar{w}_2)$, and $\ell(\rho_{c_1}) < \ell(\rho_{c_2})$, then in $\Psi(\bar{w}_2, c_2)$, W_1 plays no role in the splicing step, ζ, and splicing introduces no direct loop nodes that intersect c_1. Hence $c_1 \in \Phi(\bar{w})$. □

Lemma 6.3 *For fixed c the map $\bar{w} \mapsto \Psi(\bar{w}, c)$ from the set of $\bar{L}$-sorted permutations to the set of L-arbors, A, such that $c \in \Phi(\bar{g}(A))$ is bijective.*

Proof: Surjectivity: This is by Lemma 6.1.

Injectivity: $\bar{w}$ can be recovered from $w = \bar{g}(\Psi(\bar{w}, c))$ by $(\bar{w}, c) = \bar{\mu}(\bar{f}(w), W_c)$, where W_c is the wreath in $\bar{f}(w)$ belonging to c. □

7. The Raney coefficients revisited

We are given a collection of letters, L, having inventory α, such that $\sigma(\alpha_i) > 0$ for all i and, as defined in Eq. (2), $p_i \geq 0$ for all labels i.

Theorem 7.1 *The number of forests having inventory α is*

$$R(\alpha) = \widetilde{M}(\alpha) D_n$$

Since the forests are in one-to-one correspondence with successful sorted permutations, this is equivalent to showing that the number of sorted permutations that fail is

$$(\widetilde{M}(\alpha)) \prod_{i=1}^{n} \sigma(\alpha_i) - \widetilde{M}(\alpha) D_n,$$

since $(\widetilde{M}(\alpha)) \prod_{i=1}^{n} \sigma(\alpha_i)$ is the total number of sorted permutations.

Recall that, in Eq. (6), we reduced this to proving that the number of failing shaded L-sorted permutations is

$$\left(\prod_{i=1}^{n} (\sigma(\alpha_i)!) \right) - \left(\prod_{i=1}^{n} ((\sigma(\alpha_i) - 1)!) \right) D_n.$$

Thus, we will prove Theorem 7.1 by showing that Ψ constructs the failing sorted permutations in a way that we can count. Proof of this theorem follows Claim 7.4.

Given a cycle and a list of all complementary sorted permutations, Ψ gives pairwise distinct sorted permutations, according to Lemma 6.3. However, this fails to be true if we let the cycle vary. In fact, there will be repeats.

Notation 7.2 Let $\Psi(*, c) = \{w \mid w = \Psi(\bar{w}, c) \text{ for some complementary } \bar{w}\}$.

Then, for the total count, we need to count the number of sorted permutations in $\Psi(*, c)$ for each c, and then account for those counted more than once.

Recall the notation from Section 2: Let $\bar{c}$ be a cycle-type. Say $\bar{c} = (x_1, x_2, \ldots, x_s)$. Denote by $\sigma(\bar{c}) = \sigma_{x_2}(\alpha_{x_1})\sigma_{x_3}(\alpha_{x_2}) \cdots \sigma_{x_1}(\alpha_{x_s})$, the number of cycles of type $\bar{c}$.

Claim 7.3 Given $\bar{c}$, the cardinality of the set

$$\{(\bar{w}, c) \mid c \text{ is of type } \bar{c} \text{ and } \bar{w} \text{ complementary to } c\}$$

is

$$\prod_{l=1}^{n}((\sigma(\alpha_l) - 1)!) \prod_{i \notin c} \sigma(\alpha_i)\sigma(\bar{c})$$

Proof: Given a cycle c of type $\bar{c}$, we get

$$\frac{\prod_{l=1}^{n}(\sigma(\alpha_l)!)}{\prod_{i \in c} \sigma(\alpha_i)}$$

$\bar{L}$-sorted permutations, each of which leads to a $(\bar{w}, c)$ pair. There are $\sigma(\bar{c})$ cycles of cycle-type $\bar{c}$. This yields a total of

$$\frac{\prod_{l=1}^{n}(\sigma(\alpha_l)!)}{\prod_{i \in c} \sigma(\alpha_i)}\sigma(\bar{c}) = \prod_{l=1}^{n}((\sigma(\alpha_l) - 1)!) \prod_{i \notin c} \sigma(\alpha_i)\sigma(\bar{c})$$

$(\bar{w}, c)$ pairs. □

Recall again some notation from Section 2: for a set of pairwise disjoint cycle-types $\overline{c_1}, \ldots, \overline{c_s}$, we let C be the set of labels appearing in $\overline{c_1} \cup \cdots \cup \overline{c_s}$. Also, let

$$V_{\overline{c_j}} = \{w \mid w = \Psi(\bar{w}, c_j) \text{ for } c_j \text{ of type } \overline{c_j} \text{ and } \bar{w} \text{ complementary to } c_j\}.$$

Claim 7.4 (i) For cycles $c_1, \ldots, c_s$ of disjoint cycle-types $\overline{c_1}, \ldots, \overline{c_s}$ respectively, and for $\bar{L} = L - \cup_{i=1}^{s} L_{c_i}$, there are

$$\prod_{l=1}^{n}((\sigma(\alpha_i) - 1)!) \prod_{i \notin C} \sigma(\alpha_i)$$

$\bar{L}$-sorted permutations.

(ii) For disjoint cycle-types $\overline{c_1}, \ldots, \overline{c_s}$ there are

$$\prod_{l=1}^{n}((\sigma(\alpha_i) - 1)!) \prod_{i \notin C} \sigma(\alpha_i) \prod_{i=1}^{s} \sigma(\overline{c_i})$$

distinct L-sorted permutations in $\cap_{j=1}^{s} V_{\overline{c_j}}$.

Proof: Part (i) of the lemma is obvious.

For part (ii), we extend our definition of Ψ. Let $c_1, \ldots, c_s$ be disjoint cycles arranged in increasing root order. For $\bar{w}$ complementary to $c_1 \cup \cdots \cup c_s$, let $\Psi(\bar{w}, c_1, \ldots, c_s) = \Psi(\Psi(\bar{w}, c_1), c_2, \ldots, c_s)$, noting that $\bar{w}' = \Psi(\bar{w}, c_1)$ is complementary to $c_2 \cup \cdots \cup c_s$. Notice that Ψ is still injective for fixed $c_1, \ldots, c_s$. Indeed, by induction on s, the base case is Lemma 6.3. Assuming it has been proven up to $s-1$, then the induction step is also an immediate result of Lemma 6.3.

Subclaim Using Notation 7.2, $w \in \cap_{i=1}^{s} \Psi(*, c_i) \Leftrightarrow$ there is an $\bar{L}$-sorted permutation, $\bar{w}$, with $\bar{L} = L - \cup_{i=1}^{s} L_{c_i}$, such that $w = \Psi(\bar{w}, c_1, \ldots, c_s)$.

Proof of Subclaim: $\Rightarrow$: We proceed by induction on s. The base case $s = 1$ has been done in Lemma 6.1. Arrange $c_1, \ldots, c_s$ in increasing root order. Suppose we have the result for $c_2, \ldots, c_s$. Then c_1 is a cycle disjoint from c_i, $i > 1$, such that $\ell(\rho_{c_1}) < \ell(\rho_{c_i})$. Let

$$w \in \cap_{i=1}^{s} \Psi(*, c_i) \subset \cap_{i=2}^{s} \Psi(*, c_i).$$

Then, by the induction assumption, there is an $\tilde{L}$-sorted permutation, $\tilde{w}$, with $\tilde{L} = L - \cup_{i=2}^{s} L_{c_i}$, such that $w = \Psi(\tilde{w}, c_2, \ldots, c_s)$. Also, we know that there is a sorted permutation $\bar{w}_1$ complementary to c_1 such that $w = \Psi(\bar{w}_1, c_1)$. Therefore, $c_1 \in \Phi(w)$, and by a proof identical to that in Lemma 6.2, $c_1 \in \Phi(\tilde{w})$. Then, by Lemma 6.1, there is a $\tilde{w}'$ complementary to c_1 in $\tilde{L}$ such that $\tilde{w} = \Psi(\tilde{w}', c_1)$ and therefore

$$w = \Psi(\Psi(\tilde{w}', c_1), c_2, \ldots, c_s) = \Psi(\tilde{w}', c_1, \ldots, c_2).$$

$\Leftarrow$: $w = \Psi(\tilde{w}, c_1, \ldots, c_s)$. Given c_i, we need to recover a sorted permutation, $\bar{w}_i$, complementary to c_i such that $w = \Psi(\bar{w}_i, c_i)$. However, $c_i \in \Phi(w)$ and therefore, $w = \Psi(\bar{w}_i, c_i)$, for some $\bar{w}_i$, by Lemma 6.1. □

This establishes a one-to-one correspondence between L-sorted permutations in $\cap V_{\overline{c_j}}$ and tuples $(\bar{w}, c_1, \ldots, c_s)$ with $c_1, \ldots, c_s$ arranged in increasing root order.

Now to Claim 7.4 Part (ii) : For a given set of cycle-types $\overline{c_1}, \ldots, \overline{c_s}$, there are $\prod_{i=1}^{s} \sigma(\overline{c_i})$ distinct sets of cycles having these cycle-types. By the first part of this claim, for each distinct set of cycles, there are

$$\prod_{l=1}^{n} ((\sigma(\alpha_i) - 1)!) \prod_{i \notin C} \sigma(\alpha_i)$$

complementary sorted permutations, and therefore, we get a total of

$$\prod_{l=1}^{n} ((\sigma(\alpha_i) - 1)!) \prod_{i \notin C} \sigma(\alpha_i) \prod_{i=1}^{s} \sigma(\overline{c_i})$$

pairs of complentary sorted permutations and cycles, which are in one-to-one correspondence with sorted permutations in $\cap V_{\overline{c_j}}$. By the subclaim, then, we are done. □

Proof of Theorem 7.1: We are showing that Ψ constructs

$$\left(\prod_{i=1}^{n}(\sigma(\alpha_i)-1)!\sigma(\alpha_i)\right)-\left(\prod_{i=1}^{n}((\sigma(\alpha_i)-1)!)\right)D_n$$

failing arbors. Using the notation from Claim 2.4, we see that this is equal to

$$\left(\prod_{i=1}^{n}((\sigma(\alpha_i)-1)!)\right)\left(\sum_{j=1}^{n}(-1)^{j+1}\sum_{\star}\left(\prod_{l\notin C}\sigma(\alpha_l)\right)\sigma(\overline{c_1})\sigma(\overline{c_2})\cdots\sigma(\overline{c_j})\right) \tag{7}$$

where $\star$ sums over all possible collections of j disjoint cycle-types and C denotes the labels represented in the union of these j cycle-types.

We use an adaptation of a formula known in set-theory (pg 39–40 [3]) For sets $A_1,\ldots,A_N$ we have

$$\left|\bigcup_{i=1}^{N}A_i\right|=\sum_{j=1}^{N}(-1)^{j+1}\left(\sum_{1\le i_1<\cdots<i_j\le N}\left|A_{i_1}\cap A_{i_2}\cap\cdots\cap A_{i_j}\right|\right) \tag{8}$$

For our purposes, we enumerate all possible cycle-types, $\bar{c}_1,\ \bar{c}_2,\ldots,\bar{c}_N$. Let

$$A_i=V_{\overline{c_i}}. \tag{9}$$

Then, $|\bigcup_{i=1}^{N}A_i|=|\bigcup_{i=1}^{N}V_{\overline{c_i}}|$ counts the number of distinct failing sorted permutations. Now, we must show that the right hand side of Eq. (8) equals the formula from Eq. (7). Note first that if $\overline{c_i}\cap\overline{c_j}\neq\emptyset$ for $\overline{c_i},\ \overline{c_j}$ cycle-types, then $|V_{\overline{c_i}}\cap V_{\overline{c_j}}|=0$ by definition of Φ having disjoint cycles in its image. So,

$$\sum_{1\le i_1<\cdots<i_j\le N}\left|V_{\overline{c_{i_1}}}\cap V_{\overline{c_{i_2}}}\cap\cdots\cap V_{\overline{c_{i_j}}}\right|=\sum_{\substack{1\le i_1<\cdots<i_j\le N\\ c_{i_j}\cap c_{i_l}=\emptyset,\ j\neq l}}\left|V_{\overline{c_{i_1}}}\cap V_{\overline{c_{i_2}}}\cap\cdots\cap V_{\overline{c_{i_j}}}\right|$$

We can condense the index of summation,

$$\sum_{\substack{1\le i_1<\cdots<i_j\le N\\ c_{i_j}\cap c_{i_l}=\emptyset,\ j\neq l}}\left|V_{\overline{c_{i_1}}}\cap V_{\overline{c_{i_2}}}\cap\cdots\cap V_{\overline{c_{i_j}}}\right|=\sum_{*}\left|V_{\overline{c_{i_1}}}\cap V_{\overline{c_{i_2}}}\cap\cdots\cap V_{\overline{c_{i_j}}}\right|$$

where $*$ sums over all collections of j disjoint cycle-types $c_{i_1},\ldots,c_{i_j}$. Since all terms are zero unless the cycle-types are disjoint, we may assume $j\le n$ by the pigeonhole principle. Thus, we may rewrite Eq. (8),

$$\sum_{j=1}^{N}(-1)^{j+1}\left(\sum_{1\le i_1<\cdots<i_j\le N}\left|V_{\overline{c_1}}\cap V_{\overline{c_2}}\cap\cdots\cap V_{\overline{c_j}}\right|\right)$$
$$=\sum_{j=1}^{n}(-1)^{j+1}\sum_{*}\left|V_{\overline{c_1}}\cap V_{\overline{c_2}}\cap\cdots\cap V_{\overline{c_j}}\right|$$

Lastly, by Claim 7.4,

$$\left|V_{\overline{c_1}} \cap V_{\overline{c_2}} \cap \cdots \cap V_{\overline{c_j}}\right| = \left(\prod_{i=1}^{n}((\sigma(\alpha_i)-1)!)\right)\left(\prod_{k\notin C}\sigma(\alpha_k)\right)\sigma(\overline{c_1})\sigma(\overline{c_2})\cdots\sigma(\overline{c_j})$$

Putting these last two equations together proves the theorem. □

8. Conclusion

Thus, we have provided a procedure that counts the successful forests of a given inventory. By classifying the failures, we have matched, in a term-by-term manner, the failing sorted-permutations with terms from the counting formula.

Using this procedure in conjunction with a certain class of polynomials discussed by Wright in [6], we can restate the Jacobian conjecture. Namely, let $F = X - H$ with $H_i = \sum_{|k|\geq 2} a_i^k X^k$. Then, if $j(F) = 1$, we have

$$X_1^{p_1}\cdots X_n^{p_n} = \sum_{q=(q_1,\ldots,q_n)} \bar{e}_p^q F_1^{q_1}\cdots F_n^{q_n}$$

where

$$\bar{e}_p^q = \sum M(\alpha)\prod_{j=1}^{n}\prod_{|k|\geq 2}\left(a_j^k\right)^{\alpha_j^k}$$

with the sum being indexed by all α having root-type p and leaf-type q, such that $q_i = \alpha_i^{(0,\ldots,0)}$ and $\alpha_i^k = 0$ whenever $|k| = 1$.

Notice how this is a much simpler formula than we originally had in Notation 1.1. These results will be discussed in more detail in a future paper, [4].

References

1. H. Bass, E. Connell, and D. Wright, "The Jacobian conjecture: reduction of degree and formal expansion of the inverse," *Bulletin of the American Mathematical Society* **7**(2) (1982), 287–330.
2. C. Ching-An Cheng, J. McKay, J. Towber, S. Sui-Sheng Wang, and D. Wright, "Reversion of power series and the extended Raney coefficients," *Transactions of the American Mathematical Society* **349** (1997), 1769–1782.
3. P. Hoel, S. Port, and C. Stone, *Introduction to Probability Theory*, Houghton Mifflin Company, Boston, 1971.
4. K. Lampe, Relating a Counting Algorithm to the Jacobian Conjecture, in preparation.
5. G. Raney, "Functional composition patterns and power series reversion," *Transactions of the American Mathematical Society* **94** (1960), 441–451.
6. D. Wright, "The tree formulas for reversion of power series," *Journal of Pure and Applied Algebra* **57** (1989), 191–211.
7. D. Wright, "Formal inverse expansion and the Jacobian conjecture," *Journal of Pure and Applied Algebra* **48** (1987), 199–219.
8. H. Wilf, *Combinatorial Algorithms: An Update*, Society for Industrial and Applied Mathematics, Philadelphia, 1989.