An Enumeration Problem for Sequences of n-ary Trees Arising from Algebraic Operads

A Thesis Submitted to the
College of Graduate and Postdoctoral Studies
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Mathematics and Statistics
University of Saskatchewan
Saskatoon

By

Daniel W. Stasiuk

Permission to Use

Abstract

This thesis solves an enumeration problem for sequences of complete n-ary trees. Given the sequence of all complete n-ary plane trees with a given number of internal nodes (weight), in lexicographical order, we perform graftings with the basic n-ary tree to construct sets of sequences of trees of higher weight. Determining the number of elements of these sets solves a problem originating from the theory of free nonsymmetric operads, as the sets of sequences of trees are equivalent to spanning sets of homogeneous subspaces of a principal operad ideal. Two different solutions will be presented: one using recurrence relations and properties of forests, the other using occupancy problems.

Acknowledgements

I would like to extend my sincere gratitude to my supervisors, Chris Soteros and Murray Bremner, for providing their expertise and continuous support over the course of the project. I would also like to thank the other members of my advisory committee, Ebrahim Samei and Mik Bickis.

To my grandfather, Greg Barnsley.

Contents

List of Tables

List of Figures

List of Notation

ψ	Denotes the basic operation.
$\circ_i$	Given two operations α and β, $\alpha \circ_i \beta$ denotes replacing the ith argument of α with the output of β.
T_α	Denotes the tree representation of the operation α.
$N(w)$	The number of leaves in a tree with exactly w internal nodes for a given arity.
n	Denotes the arity of a tree or operation.
w	Denotes the (starting) weight, that is the number of internal nodes.
$\mathfrak{T}_\psi(K)$	Denotes the set of all operations arising from ψ whose tree diagrams are complete n-ary plane trees with exactly K leaves.
l_w	Denotes the number of n-ary trees of weight w.
R_0	The 1-set containing the sequence $(X_1, X_2, X_3, \ldots, X_{l_w})$, the sequence of n-ary trees of weight w in lexicographical order for a fixed n and w.
R_i	Denotes the set of sequences generated by performing i sequence compositions of the starting sequence with T_ψ.
R	$\bigcup_{i\geq 0} R_i$
$\ell(i)$	For a given element of R_i, this denotes the number of leaves each tree in the sequence has.
$F(i, t; n)$	Denotes the number of forests of t n-ary trees having a total of i internal nodes.
a	Denotes an internal node in the word representation of a tree or forest.
b	Denotes a leaf node in the word representation of a tree or forest.
σ_j	Denotes an application of Procedure 1.2.1 at the jth leaf in the word representation of an element of R_i.
τ_k	Denotes an application of Procedure 1.2.2 at the kth leaf in the word representation of an element of R_i.
$R_i^{(1)}$	Denotes the elements of R_i that can be generated by applying Procedure 1.2.1 to an element of R_{i-1}.
$R_i^{(2)}$	Denotes the elements of R_i that can be generated by applying Procedure 1.2.2 to an element of R_{i-1}.
$R_i^{(1o)}$	The set of all elements of R_i that can be generated by a sequence of i applications of Procedure 1.2.1.
L_τ	Denotes the length of the τ sequence (number of τ's) in the word representation of an element of R_i.

$\overline{R_i}^{(1o)}$	the set of all elements of R_i that cannot be generated by applying Procedure 1.2.1 i times. Procedure 1.2.2 must have been applied at least once.
$R_i^{(\text{unique})}$	The set of all elements of R_i that are j-unique sequences for some j $(1 \leq j \leq n)$.
$R_i^{(\text{unique},\,k)}$	The set of all elements of $R_i^{(\text{unique})}$ in which the common level-1 subtrees have a total of k internal nodes.

Chapter 1

Introduction and motivation

Enumerative combinatorics is the area of mathematics dealing with counting problems, one of the oldest classes of mathematical problem. Its purpose is to construct and identify a counting function, that is a map from a collection of finite sets to the whole numbers, that outputs the number of elements in a given set. The counting function is ideally an explicit formula or a polynomial-time algorithm to compute the function [19, 20]. Some well-known counting problems include calculating the number of linear arrangements of a given set of objects (permutations), finding the number of subsets of a given finite set (the binomial coefficients), enumerating the spanning trees on a complete graph with labelled vertices (Cayley's formula), and counting the Latin squares of a certain order. For a more extensive overview, see [20].

The focus of this thesis is an enumerative problem on trees. Of the literature on the combinatorics of trees, perhaps the most important result for our purposes is the Catalan numbers that enumerate the complete binary trees with a given number of internal nodes [5, 20]. This solution can be generalized to complete n-ary trees for arbitrary n [16], and its proof will be reviewed later in the thesis. The enumeration problem we are interested in solving originates from the theory of algebraic operads. Operads are a general way of describing the composition of functions, and they can naturally be represented in terms of trees [2, 7, 11]. This connection will allow us to formulate the enumerative problem in a combinatorial way.

Concepts from operad theory date back to at least 1898 [12], but the term "operad" was first introduced and formally defined in 1972 by topologist J. Peter May in his work on homotopy theory [13]. Operad theory was further developed throughout the 1990s and early 2000s, finding applications to algebraic geometry and mathematical physics; for more information, see [12] and [2]. The enumeration problem we will consider in this thesis is based on the concept of principal ideals in the free nonsymmetric operad. However, a formal treatment of operads is outside the scope of this thesis because a complete explanation would require a discussion of operads over general polynomial rings as well as module theory. We will instead construct a composition system that is isomorphic to the free nonsymmeric operad, which will allow us to present the enumeration problem strictly in terms of sequences of trees. For detailed information on the connection between trees and nonsymmetric operads, see [3] and [2, Chapter 3].

We will begin by defining a system of compositions in terms of a single operation, which we will represent with operation diagrams. Operation diagrams represent operations in terms of n-ary plane trees, where a composition is equivalent to a grafting of trees. We will then naturally extend the definition of composition to sequences of trees, which will allow us to define the enumeration problem of interest. Finally, we will use numerical data to propose a counting function that solves the enumeration problem. Proving this conjecture will be the main purpose of this thesis.

Please note that in definitions, in order to make clear which terms are being defined, symbols will be made boldface as a convention. Boldface symbols are not used for any other purpose in this thesis.

1.1 Constructing the composition system

For this section, our main point of reference will be [2, Chapter 3] except where otherwise stated. We will emphasize and expand on concepts needed for the remainder of this thesis. We begin with an arbitrary (but fixed) n-ary operation, that is a function (denoted by ψ) mapping a sequence of n inputs (from a given set) to a single output (from the same set) for some natural number $n \geq 2$. We are not concerned about the internal workings of the function. Instead, it will be considered a “black box”, expressed only in terms of its input and output.

1.1.1 Operation diagrams

We will use a black square, $\blacksquare$, to represent an n-ary operation with inputs denoted $x_1, \dots, x_n$ from left to right, and one output given by $\psi(x_1, x_2, \dots, x_n)$. This results in the tree diagram, denoted by T_ψ, shown in (1.1) (trees will be defined more precisely in Section 1.1.3); note that the inputs will always be placed at the bottom and the output at the top.

$$T_\psi \left\{ \begin{tikzcd} & \psi(x_1, x_2, \dots, x_n) & \\ & \blacksquare \arrow[u] & \\ x_1 \cdots \arrow[ur] & x_i \arrow[u] & \cdots x_n \arrow[ul] \end{tikzcd} \right. \tag{1.1}$$

Using the concept of function compositions, more complicated operations arise from ψ. Given a natural number $i \le n$ we can take the output of one copy of ψ as the ith input of a second copy of ψ. This new operation will again have one output, but $2n-1$ inputs. As before the inputs are labelled $x_1, \dots, x_{2n-1}$. In the resulting tree diagram (shown in (1.2)), we start with a single copy of T_ψ and graft the second copy of T_ψ at what was originally the ith input.

$$T_{\psi \circ_i \psi} \left\{ \begin{tikzcd} & & & \psi(x_1, x_2, \dots, x_{i-1}, \psi(x_i, \dots, x_{i+n-1}), x_{i+n}, \dots, x_{2n-1}) & & & \\ & & & \blacksquare \arrow[u] & & & \\ x_1 \arrow[urrr] & \cdots & x_{i-1} \arrow[ur] & \blacksquare \arrow[u] & x_{i+n} \arrow[ul] & \cdots & x_{2n-1} \arrow[ulll] \\ & & x_i \arrow[ur] & \cdots & x_{i+n-1} \arrow[ul] & & \end{tikzcd} \right. \tag{1.2}$$

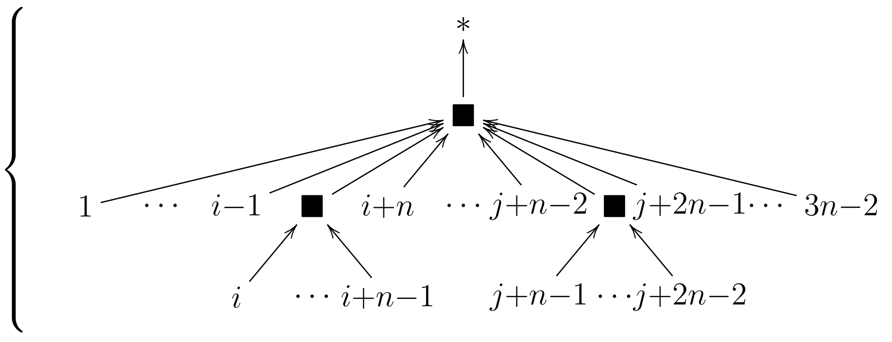
Definition 1.1.1. The notation $\boldsymbol{\psi \circ_i \psi}$ represents the result of taking the output of the second copy of ψ and using it as the ith input of the first copy of ψ. This is called the ***i*th composition of $\boldsymbol{\psi}$ with itself**. For each $i \in \{1, 2, ..., n\}$ the operation has $2n-1$ inputs.

Note that by definition two functions that have the same tree diagram will produce the same output given the same inputs, provided that both functions are compositions of two or more copies of ψ.

1.1.2 Composing more than two copies of $\boldsymbol{\psi}$: associativity

There are two different ways in which we can compose three copies of ψ: parallel composition and sequential composition [2]. For parallel composition, we choose two numbers i and j such that $1 \leq i < j \leq n$. We start with one copy of ψ and take the output of a second copy of ψ as the ith input. Then, we take the output of a third copy of ψ as the jth input of the *first* copy of ψ. The diagram is given in (1.3); note that we have simplified it by dropping the x's (writing only the subscripts) and replacing the final output with *.

$T_{(\psi \circ_i \psi) \circ_{j+n-1} \psi}$



(1.3)

Definition 1.1.2 ([2]). A composition of the form $\boldsymbol{(\psi \circ_i \psi) \circ_{j+n-1} \psi}$ (where $1 \leq i < j \leq n$) is called a **parallel composition**.

There are two equivalent ways to express a parellel composition in terms of compositions of ψ with itself:

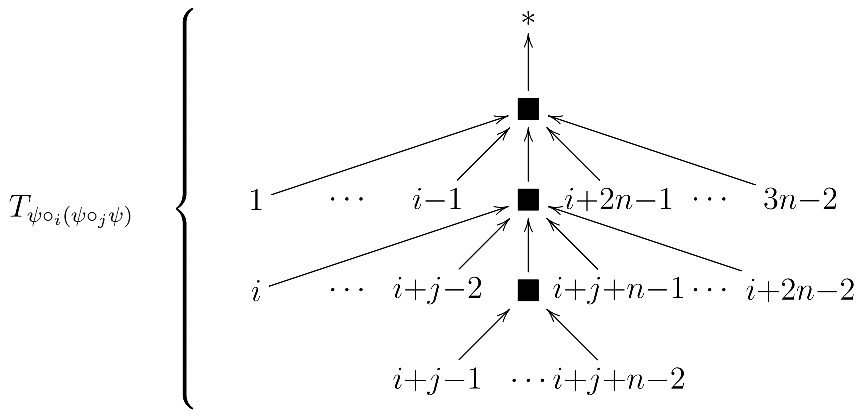
- When we replace the ith argument of the first copy of ψ with another copy of ψ, arguments $i+1$ through n become arguments $i+n$ through $2n-1$. That is, the argument number increases by $n-1$. So argument j becomes argument $j+n-1$ (see Equation 1.3). Thus, the composition can be expressed as $(\psi \circ_i \psi) \circ_{j+n-1} \psi$.

- We can instead first replace the jth argument of the first copy of ψ with another copy of ψ. The labels on the first $j-1$ arguments of the first copy of ψ do not change, so the ith argument remains the ith argument because $i < j$ (see Equation 1.3). Thus we can express this composition as $(\psi \circ_j \psi) \circ_i \psi$.

This proves the following known identity:

Proposition 1.1.1 ([2]). *If* $1 \leq i < j \leq n$ *then* $(\psi \circ_i \psi) \circ_{j+n-1} \psi = (\psi \circ_j \psi) \circ_i \psi$.

For sequential composition, we choose two numbers i and j (each between 1 and n). We start with one copy of ψ and take the output of a second copy of ψ as the ith input. Then, we take the output of a third copy of ψ as the jth input of the *second* copy of ψ as given in (1.4).



(1.4)

Definition 1.1.3 ([2]). A composition of the form $\boldsymbol{\psi \circ_i (\psi \circ_j \psi)}$ is called a **sequential composition**.

As with the parellel compositions there are two equivalent ways to express a sequential composition in terms of compositions of ψ with itself:

- We can first replace the jth input of the "second" copy of ψ by the output of a third copy of ψ, then replace the ith input of the "first" copy of ψ by the output of this operation (see Equation 1.4). So the composition as a whole is represented as $\psi \circ_i (\psi \circ_j \psi)$.

- We first replace the ith input of the "first" copy of ψ by the output of a second copy of ψ (see Equation 1.4). In this second copy of ψ, arguments 1 through n are labelled i through $i + n - 1$ (respectively). So what was originally the jth input of the second copy of ψ is now the $(i + j - 1)$th. Thus the composition as a whole is represented as $(\psi \circ_i \psi) \circ_{i+j-1} \psi$.

This proves the following known identity:

Proposition 1.1.2 ([2]). $\psi \circ_i (\psi \circ_j \psi) = (\psi \circ_i \psi) \circ_{i+j-1} \psi$

Propositions (1.1.1) and (1.1.2) together define a generalized form of associativity for the composition of operations. We can naturally extend this composition process indefinitely to obtain all possible operation diagrams, generated by composing any number of copies of the original diagram. Before we further generalize composition however, we must introduce standard terminology and review some known results related to trees.

1.1.3 Trees and general compositions

The remainder of this thesis will assume the standard graph theory textbook terminology such as graphs, edges, and cycles. Unless stated otherwise definitions are consistent with those in [9]. In this section we will first review several terms related to trees, then apply them to general compositions. These terms will be used extensively in the following chapters. We will begin with the definition of a tree.

Definition 1.1.4. [9] A **tree** is a simple connected graph with no cycles; see Figure 1.1.

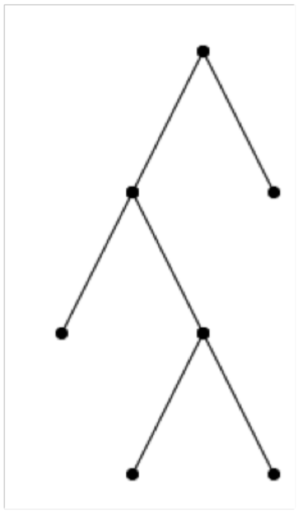


Figure 1.1: An example of a tree with 5 leaf nodes and 4 internal nodes.

Definition 1.1.5 ([9]).

- A **rooted tree** is a tree in which one vertex has been designated as the **root**.
- Given any vertex in the tree, each edge leads to either its **parent** vertex (closer to the root) or one of its **child** vertices (further from the root). By definition, the root has no parent.
- A vertex with no children is called a **leaf**, while a vertex with children is called an **internal node**. By definition the root of any tree with more than one vertex will be an internal node.

- The **level** of a vertex is the length of the path (that is, the number of edges) connecting the vertex to the root. For example the root will have a level of 0 and each of its children will have a level of 1.

- A **plane tree** is an embedding of a tree into the plane; this is equivalent to specifying a left-to-right ordering of the leaves [6].

- An ***n*-ary tree** is a rooted tree in which each vertex has at most n children. A **complete *n*-ary tree** is an n-ary tree in which each internal node has exactly n children.

Note that the operation diagrams we have been using for the composition system, such as Equations 1.3 and 1.4, are equivalent to complete n-ary plane trees, in which the numbered inputs are the leaves and the final output is the root.

Remark 1.1.1. In this thesis, the root will always be placed at the top of the tree by convention. Figure 1.1 depicts a complete binary plane tree with three internal nodes.

The following result, related to the number of leaves in an n-ary tree, will be essential in keeping track of the total number of arguments in a composition system.

Proposition 1.1.3 ([10])**.** *Every complete n-ary tree with w internal nodes has exactly $(n-1)w+1$ leaves.*

Proof. This is a standard result that can be proven by mathematical induction on w. For full details, see [10]. □

Definition 1.1.6. Given a fixed arity n, we will use $\boldsymbol{N(w)}$ to denote the number of leaves that a complete n-ary tree with w internal nodes has. Thus $N(w) = (n-1)w+1$.

We can now use this terminology to finish defining our composition system and generalize Definition 1.1.1 to all functions arising from ψ. If we compose w copies of the original diagram (1.1) in some way, we obtain a complete n-ary plane tree (where n is the arity of ψ) with exactly w internal nodes. By Proposition 1.1.3 such a tree will have exactly $N(w) = 1 + w(n-1)$ leaves, corresponding to the number of inputs that the operation has.

For the sake of simplicity, we will draw the trees without arrowheads on the edges (with the understanding that the leaves represent the inputs, the root represents the final output, and so on). We will use black circles (●) to represent the vertices (internal nodes, including the root, as well as leaves).

Definition 1.1.7. For a tree corresponding to an operation, the leaf corresponding to the jth input is called the **jth leaf** of the tree.

This follows naturally from how we have defined operation diagrams in the previous section, but it should be noted that we can equivalently number the leaves in the order they would be visited in a depth-first traversal which will be defined in Definition 2.1.2. To further simplify our operation diagrams the leaves will be unlabelled with the understanding that the leftmost leaf corresponds to the first input of the operation (and so on), consistent with previous diagrams such as Equations 1.3 and 1.4.

Definition 1.1.8. The tree form of ψ, denoted by $\boldsymbol{T_\psi}$, is called the **basic n-ary tree**.

Definition 1.1.9. The number of internal nodes (compositions) in a given tree (operation) is called the **weight**. $\boldsymbol{w}$ will often be used to denote weight.

Definition 1.1.10. We will use $\boldsymbol{\mathfrak{T}_\psi(K)}$ to denote the set of all operations arising from ψ whose tree diagrams are complete n-ary plane trees with exactly K leaves. Note that every tree in $\mathfrak{T}_\psi(K)$ will have the same weight w, where $K = N(w)$. We denote the disjoint union of $\mathfrak{T}_\psi(N(w))$ for all $w \geq 1$ by $\boldsymbol{\mathfrak{T}_\psi}$.

This next definition will generalize the composition operation, for both operations and for their tree representations.

Definition 1.1.11. Let $g \in \mathfrak{T}_\psi(K)$ and $h \in \mathfrak{T}_\psi(K')$ be operations with tree diagrams T_g and T_h respectively. Thus g has K total inputs (T_g has K leaves) and h has K' total inputs (T_h has K' leaves). Given an $i \in \{1, ..., K\}$ we form the **composition $\boldsymbol{g \circ_i h}$** by replacing the ith input of g by the output of h. Equivalently, the composition $\boldsymbol{T_g \circ_i T_h}$ denotes the tree formed by grafting the root of T_h at the ith leaf of T_g. $g \circ_i h \in \mathfrak{T}_\psi(K + K' - 1)$.

Proposition 1.1.4. *The compositions of arbitrary elements of $\mathfrak{T}_\psi$ (their tree diagrams) satisfy the associativity relations Proposition 1.1.1 and Proposition 1.1.2 if we replace ψ with arbitrary operations f, g, and h (arbitrary trees T_f, T_g, and T_h).*

Proof. This follows immediately from the preceding definition and from the proofs of Proposition 1.1.1 and Proposition 1.1.2. □

Remark 1.1.2. Note that if n-ary trees T_g and T_h have weights w_1 and w_2 respectively, $T_g \circ_i T_h$ (where $1 \leq i \leq N(w_1)$) will have weight $w_1 + w_2$. Importantly, this means that the weight of $T_g \circ_i T_\psi$ or $T_\psi \circ_j T_g$ (where $j \leq n$) will be $w_1 + 1$.

We will use our definitions of composition to construct a representation of the free nonsymmetric operad. Note that this equivalent to the partial-composition definition of an operad, which is defined precisely in [2, Section 3.2.2].

Definition 1.1.12 ([2])**.** Given an arbitrary n-ary operation ψ, the set $\mathfrak{T}_\psi$ equipped with the composition operation given in Definition 1.1.11 is called the **free nonsymmetric operad** generated by ψ; ψ is known as the **operad generator**.

We will use this definition of the free nonsymmetric operad to present and solve an open enumeration problem from operad theory using the properties of trees. This problem will be introduced in the following section.

1.2 Introducing the enumeration problem

In this section, we will mainly work in terms of trees rather than operations. The definitions and results for operations are equivalent, but our solutions to the enumeration problem will be based on n-ary trees thus it is reasonable to define the problem in terms of trees as well. We begin by naturally defining a distributive property of compositions over a sequence of trees, after introducing some convenient notation for the number of n-ary trees of a given weight.

Definition 1.2.1. $\boldsymbol{l_w}$ denotes the number of complete n-ary plane trees of weight w.

It is known that $l_w = \frac{\binom{nw}{w}}{w(n-1)+1}$ [5] and the proof will be reviewed in Chapter 2.1.

Definition 1.2.2. Let T be an n-ary tree with at least i leaves and let $(T_1, T_2, T_3, \ldots, T_m)$ be a sequence of m n-ary trees of weight w. Then, $\boldsymbol{T \circ_i (T_1, T_2, T_3, \ldots, T_m)}$ produces the sequence

$(T \circ_i T_1, T \circ_i T_2, T \circ_i T_3, \ldots, T \circ_i T_m)$. Likewise, if we assume each tree in the sequence has at least j inputs then $\boldsymbol{(T_1, T_2, T_3, \ldots, T_m) \circ_j T} = (T_1 \circ_j T, T_2 \circ_j T, T_3 \circ_j T, \ldots, T_m \circ_j T)$. This type of composition, a tree with a sequence of trees, will be called a **sequence composition**.

Of interest will be sets generated by sequence compositions on the lexicographically ordered sequence of tree representations of elements of $\mathfrak{T}_\psi(N(w))$ (for a fixed arity and a given weight w). Lexicographical order will be defined in Section 2.2, based on the word representation of trees introduced earlier in that chapter.

Definition 1.2.3. Let w be our starting weight and denote the sequence of complete n-ary plane trees of weight w in lexicographical order by $(X_1, X_2, X_3, \ldots, X_{l_w})$, known as the **starting sequence**. We define $\boldsymbol{R_0} = \{(X_1, X_2, X_3, \ldots, X_{l_w})\}$. Given $i \geq 1$, $\boldsymbol{R_i}$ denotes the set of sequences generated by performing i sequence compositions of the starting sequence with the basic n-ary tree T_ψ. More precisely for $i \geq 1$,

$$R_i = \{S \circ_j T_\psi \mid S \in R_{i-1}, 1 \leq j \leq N(w+i-1)\} \cup \{T_\psi \circ_k S \mid S \in R_{i-1}, 1 \leq k \leq n\}.$$

We will define $\boldsymbol{R} = \bigcup_{i \geq 0} R_i$.

The purpose of this thesis will be to find a formula for enumerating the elements of R_i given fixed arity n and starting weight w. This solves an open problem from operad theory, but we have reformulated it in a purely combinatorial way using sequences of trees and sequence composition. The connection to operad theory will be made clear in the following theorem, which we will present without proof[1]:

Theorem 1.2.1. *Let $\mathfrak{T}_\psi$ be the free nonsymmetric operad generated by the n-ary operation ψ. Let g be an arbitrary element of $\mathfrak{T}_\psi$ which is homogeneous of degree $1+w(n-1)$ for a given positive integer w. Then g is a formal linear combination (with indeterminate coefficients) of all elements of $\mathfrak{T}_\psi(N(w))$. The operad ideal generated by g is spanned as a subspace of $\mathfrak{T}_\psi$ by all possible compositions of g with any number i occurences of the original n-ary operation ψ. The set of all such compositions, for a given value of i, is isomorphic to R_i.*

[1]Email correspondence with Murray Bremner, August 22, 2018

Note that "homogenous" means that all trees in the starting sequence (the sole element of R_0) will have the same number of leaves. Since g and all compositions with copies of ψ are formal linear combinations with arbitrary coefficients, they can be represented as sequences of trees in R_i. An operad ideal is a subset of an operad that is closed under partial compositions with other elements of that operad. This is somewhat analogous to ideals in ring theory. Further details of the connection between operad ideals and this enumeration problem are beyond the scope of this thesis. Complete details would require long digressions into the theory of operads over general polynomial rings and the theory of free modules over polynomial rings. For more information on these topics, see [2].

1.2.1 Enumerating the elements of R_i

Generating R_i

We will first precisely define the two methods for generating an element of R_i given an element $(T_1, T_2, T_3, \ldots, T_{l_w}) \in R_{i-1}$. These follow automatically from the recursive definition of R_i (Definition 1.2.3):

Procedure 1.2.1 (Procedure σ). *Let* $(T_1, T_2, T_3, \ldots, T_{l_w}) \in R_{i-1}$, *and suppose* $1 \leq j \leq 1 + (w + i - 1)(n - 1)$. *The composition* $(T_1, T_2, T_3, \ldots, T_{l_w}) \circ_j T_\psi$ *can be computed. We say that we are applying* ***Procedure σ at leaf j***.

Procedure 1.2.2 (Procedure τ). *Let* $(T_1, T_2, T_3, \ldots, T_{l_w}) \in R_{i-1}$, *and suppose* $1 \leq j \leq n$. *The composition* $T_\psi \circ_j (T_1, T_2, T_3, \ldots, T_{l_w})$ *can be computed. We say that we are applying* ***Procedure τ at leaf j***.

Because we are using trees in the context of a composition system, it is only possible to graft roots to internal nodes. Thus Procedure σ and Procedure τ are the only two procedures we need to consider. To distinguish the two procedures, note that Procedure τ (Procedure 1.2.2) grafts at the root ('T'op) of each tree in the sequence $(T_1, T_2, T_3, \ldots, T_{l_w})$. Given R_{i-1}, we can then use these two procedures to generate R_i with the following algorithm:

For each element $(T_1, T_2, T_3, \ldots, T_{l_w}) \in R_{i-1}$,

1. For each j from 1 to $1 + (w + i - 1)(n - 1)$ (the number of leaves in each tree), apply Procedure σ at leaf j and include the element generated in R_i.

2. For each j from 1 to n, apply Procedure τ at leaf j and include the element generated in R_i.

Note that this algorithm can generate duplicate elements, as will be shown in Example 1.2.4, making the enumeration of the elements of R_i non-trivial. In Section 2.2, it will be shown that both Procedure σ and Procedure τ preserve lexicographical order (Theorem 2.2.3).

Example 1.2.4. Take $n = 2$, $w = 2$. Then the sole element of R_0, the starting sequence, consists of two binary trees. Each of these binary trees has two internal nodes: the root and its left child, or the root and its right child. In the top-left of Figure 1.2, the element of R_1 generated by applying Procedure σ to the starting sequence at leaf 2. When we apply Procedure σ to the first leaf of this element, we get the sequence of trees in the top-right of Figure 1.2[2].

If we instead apply Procedure σ to the starting sequence at leaf 1 (bottom-left of Figure 1.2) followed by Procedure σ at leaf 3, we get the sequence of trees in the bottom-right of Figure 1.2. This sequence is identical to the one in the top-right. Thus in generating R_2, duplicate elements are generated.

For $w = 1$, $R_0 = \{T_\psi\}$ and each sequence in R_i has only one element (a n-ary tree with $i + 1$ internal nodes). Thus $|R_i|$ is simply the number of n-ary trees with $i + 1$ internal nodes.

The algorithm in this section was run in Maple for $w > 1$, and the numerical data were searched on the Online Encyclopedia of Integer Sequences (OEIS). The results are given in Table 1.1.

Based on these numerical results, we propose the following formulas for calculating $|R_i|$, which will be proven in this thesis.

Main Theorem 1. *For $w = 1$,*

$$|R_i| = \frac{1}{1 + (n - 1)(i + 1)} \binom{n(i + 1)}{i + 1}.$$

[2]Figures created with Draw.io, `https://www.draw.io/`

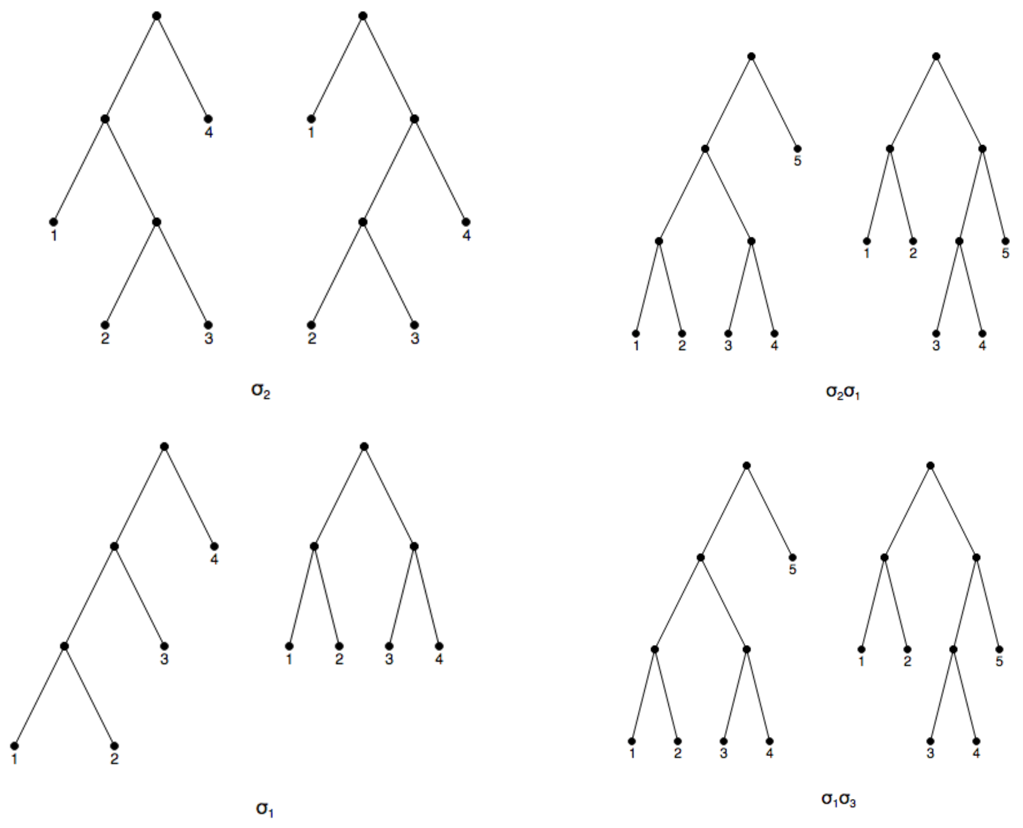


Figure 1.2: Two different sequences of procedures resulting in the same element of R_2.

Proof. As mentioned earlier, $|R_i|$ is the number of n-ary trees with $i+1$ internal nodes. The number of such trees is equivalent to the $(i+1)$th n-ary Catalan number, $\frac{1}{1+(n-1)(i+1)}\binom{n(i+1)}{i+1}$. This is a well-known result [5, 16], but its proof will be presented in the following chapter for the sake of completeness. $\square$

Main Theorem 2. *For $w > 1$,*

$$|R_i| = \binom{(n-1)(w+i)+i+1}{i}.$$

Chapter 3 and Chapter 4 will use original arguments to prove Main Theorem 2.

Summary and Overview

Through the use of operation diagrams, we have been able to construct a composition system isomorphic to the free nonsymmetic operad. This allowed us to express the principal ideal problem of interest in terms of sets of sequences of trees, which we can state as an enumerative

n	w	$\|R_i\|$	OEIS	Formula
2	2	1, 5, 21, 84, 330, ...	A002054	$\binom{2i+3}{i}$
2	3	1, 6, 28, 120, 495, ...	A002694	$\binom{2i+4}{i}$
2	4	1, 7, 36, 165, 715, ...	A003516	$\binom{2i+5}{i}$
2	5	1, 8, 45, 220, 1001, ...	A002696	$\binom{2i+6}{i}$
3	2	1, 8, 55, 364, 2380, ...	A013698	$\binom{3i+5}{i}$
3	3	1, 10, 78, 560, 3876, ...	n/a	$\binom{3i+7}{i}$
3	4	1, 12, 105, 816, 5985, ...	A004321	$\binom{3i+9}{i}$
3	5	1, 14, 136, 1140, 8855, ...	n/a	$\binom{3i+11}{i}$
4	2	1, 11, 105, 969, 8855, ...	n/a	$\binom{4i+7}{i}$
4	3	1, 14, 153, 1540, 14950, ...	n/a	$\binom{4i+10}{i}$
4	4	1, 17, 210, 2300, 23751, ...	n/a	$\binom{4i+13}{i}$
4	5	1, 20, 276, 3276, 35960, ...	A004334	$\binom{4i+16}{i}$

Table 1.1: Numerical data for $2 \leq n \leq 4$, $2 \leq w \leq 5$.

combinatorics problem. Using empirical data, we were able to formulate a counting function for the size of these sets.

We will prove Main Theorem 2 in the following chapters. In Chapter 2 we will review known results regarding Catalan numbers and the enumeration of plane trees as well as plane forests. These results will be necessary to support the proofs in the following chapter. Chapter 3 will prove Main Theorem 2 case through a recurrence relation, as well as introduce a new formula for $|R_i|$ that provides a non-recursive method of generating the elements of R_i. We will also introduce more compact notation for elements of R_i which, in Chapter 4, will lead to another proof for Main Theorem 2 involving an occupancy problem. The proofs in these two chapters will be original work. In Chapter 5 we will summarize our results and discuss future work that can build on these findings.

Chapter 2

Review of combinatorial results about trees and forests

In this chapter, it will be proven that the the number of complete n-ary plane trees with $i+1$ internal nodes is equal to $\frac{1}{1+(n-1)(i+1)}\binom{n(i+1)}{i+1}$, the $(i+1)$th n-ary Catalan number. This standard result ([5, 16]) proves Main Theorem 1 (the $w = 1$ case) as discussed in Section 1.2.1. Its proof will be reviewed here for the sake of completeness and to add clarity to the proof of the $w > 1$ case given in Chapters 3 and 4.

We will also provide the proofs for other known results. Firstly, we will prove that the number of forests of t n-ary trees and i internal nodes is $\frac{t}{ni+t}\binom{ni+t}{i}$ [5, 16]. This will be a generalization of the proof for the number of complete n-ary plane trees with a given number of internal nodes. We will also prove the Hagen-Rothe Identity $\sum_{j=0}^{L} \frac{p}{p+qj}\binom{p+qj}{j}\binom{r-qj}{L-j} = \binom{p+r}{L}$ [18], a useful convolution property. Both the formula for the number of forests and the Hagen-Rothe Identity will be necessary for the proof of the general case given in Chapter 3.

2.1 Enumerating the complete n-ary plane trees

We will enumerate the complete n-ary plane trees with a given number of internal nodes. This proof involves representing trees as words over a binary alphabet $\{a, b\}$. We can easily determine the number of words of a given length, but we need a way of identifying which words are actually valid (that is, represent trees). This can be done by converting the words to paths over the Cartesian plane, which will allow us to show that invalid words can be converted to valid words by the means of a cyclic shift. The problem of enumerating complete n-ary plane trees is closely related to the Cycle Lemma and the Ballot Theorem

[5, 16]. Unless stated otherwise we will use [5] as our main point of reference, though the basic outline of the proof comes from [1].

2.1.1 The word representation of a tree

To help with the enumeration task, we will first introduce a more compact way of writing trees [1].

Definition 2.1.1 ([8])**.** A **word** is an (ordered) sequence of **letters** from a given finite set, known as an **alphabet**.

The purpose of this section will be to establish a natural bijection between complete n-ary trees and a certain subset of words over the alphabet $A = \{a, b\}$ [5]. For our word representation (Definition 2.1.3), we will use pre-order depth-first traversal [9]. Depth-first traversal is defined recursively in Definition 2.1.2. Note that post-order depth-first traversal is used in [5], but we will instead use pre-order depth-first traversal because it is consistent with the treatment of trees as operation diagrams and leaves as arguments of a function given in the previous chapter. It will also make the definition of lexicographical order and the proof of the Hagen-Rothe identity more intuitive. The arguments, however, are parallel.

Definition 2.1.2 (**Depth-first traversal** [9])**.**

Given a plane tree T with root vertex r.,

1. Visit the root vertex r.
2. If r is a leaf node, stop.
3. Otherwise, for each child vertex of r (going from left to right in the plane), perform a Depth-first traversal on the subtree rooted at that vertex (the subtree of T consisting of that vertex, its descendants, and all connecting edges.

In a **pre-order depth-first traversal** of a tree, the vertices are labelled as they are first visited. In a **post-order depth-first traversal**, the leaves are labelled in the order they are visited but an internal node is only labelled after all of its children have been labelled.

We will use pre-order depth-first traversal throughout the remainder of this thesis.

Definition 2.1.3 ([1, 9])**.** To write the **word representation of a complete $\boldsymbol{n}$-ary tree over the alphabet $\boldsymbol{A = \{a, b\}}$**, traverse the tree through depth-first traversal. Each time a vertex is visited, write an a if the vertex is an internal node (including the root) and a b if the vertex is a leaf node.

Some examples trees and their corresponding word representations are given in Figure 2.1.

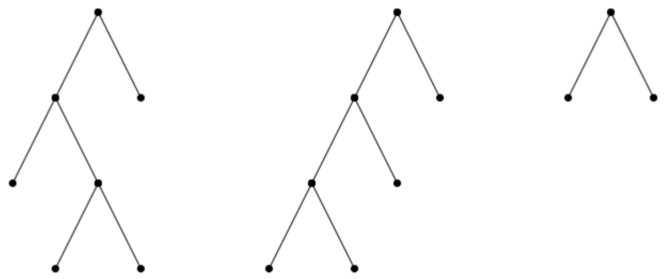


Figure 2.1: Three binary trees with respective word representations *aababbb*, *aaabbbb*, and *abb*.

Conversely given a word W of length L, we can construct the corresponding tree through the procedure defined in Definition 2.1.4. The algorithm constructs a tree recursively, while classifying vertices as leaves or internal nodes as the procedure executes (though vertices will be unclassified as they are first generated).

Definition 2.1.4 (**Constructing a tree from its word representation**).

Given a valid word W of length $L > 0$,

1. Let v_1 be a new vertex. This will be the root of the tree.

   Given a vertex v, we define $\text{next}(v)$ to be the vertex visited immediately after v in a depth-first pre-order traversal of the tree under construction.

2. For each m from 1 to L do:

   (a) If the mth letter of W is a, classify v_m as an internal node. Replace v_m with the basic n-ary tree.

   (b) If the mth letter of W is b, classify v as a leaf node.

   (c) If $m < L$, $v_{m+1} = \text{next}(v_m)$.

Note that the word b produces the tree with only one node. Definition 2.1.3 and Definition 2.1.4 together show a well-defined natural bijection between complete n-ary plane trees and a subset of words over $A = \{a, b\}$. We will define this subset more precisely in the following section, in which we will introduce the concept of word validity.

2.1.2 Identifying valid words

We can determine the number of complete n-ary plane trees that have exactly w internal nodes by identifying words that represent such trees, then enumerating these words [5].

Definition 2.1.5. A word is said to be **(n, w)-tree-valid** if it represents a complete n-ary tree with w internal nodes. For the remainder of this section, the term "**tree-valid**" will refer to an (n, w)-tree valid word.

Lemma 2.1.1 ([5])**.** *A tree-valid word must satisfy the following three conditions. Conversely, every word satisfying these conditions is tree-valid:*

1. The total number of a's is equal to w, one for each internal node.

2. The total number of b's is equal to $(n - 1)w + 1$, one for each leaf node.

3. Let the word have length L. For the first m letters of the word ($m < L$), the number of b's does not exceed $n-1$ times the number of a's.

Proof. Condition 1 follows immediately. Condition 2 follows from Proposition 1.1.3. To see that condition 3 is necessary, imagine constructing a tree from a word as described in Definition 2.1.4. Each vertex in the tree under construction can be placed into one of three categories: internal nodes, leaves (vertices that will definitely be leaves in the final tree), and unclassified vertices (vertices that are leaves in the tree under construction, but may become internal nodes as the tree is built further). After reading the first m letters of a word W (where $m < L$), the tree under construction must have at least one unclassified vertex (otherwise there is no way to extend the tree). Each b classifies one vertex, while each a classifies one vertex but generates n unclassified vertices. Since we begin with the root (an unclassified vertex that we classify with the first letter) the total number of unclassified vertices is 1 plus $n-1$ times the number of a's minus the number of b's.

Conditions 1 and 2 ensure that the tree has the correct number of leaves and internal nodes. Condition 3 ensures that when constructing the tree, we never "run out" of unclassified vertices (until the final step, when the number of b's is exactly 1 more than $(n-1)$ times the number of a's). As we construct the word through pre-order depth-first search, the same way we would traverse a tree to write its word representation, it follows that we visit and classify every vertex and that Conditions 1 through 3 are necessary and sufficient for a word to be tree-valid. $\square$

Thus to find the number of complete n-ary plane trees with w internal nodes, we must find the number of words with w a's and $1 + w(n-1)$ b's that satisfy the third condition in Lemma 2.1.1. In the following section, we will use the concept of cyclic shifts to identify the words that satisfy this condition.

2.1.3 The path representation of a word

To more easily distinguish tree-valid from tree-invalid words, we can represent them as paths in the xy plane [16]. This will make the proof more intuitive. We take a word with w a's and $[w(n-1)+1]$ b's, and begin at the origin. At the point (x, y) join a line segment to

$(x+1, y+n-1)$ if the following letter is a, or to $(x+1, y-1)$ if it is b. By definition, all paths will start at $(0,0)$ and end at $(nw+1, -1)$. If the word is tree-valid, the path will never fall below the x-axis until the final step. If the word is tree-invalid, the path will fall below the x-axis before the final step (see Figure 2.2).

Definition 2.1.6. The ***j*th cyclic shift** of a word of length L is a word formed by removing the last j letters of the word ($1 \leq j \leq L$) and rewriting them at the start of the word (in order).

Lemma 2.1.2 ([16]). *Every word composed of w a's and $[w(n-1)+1]$ b's can be converted to a tree-valid word through a cyclic shift.*

Proof. Consider a tree-invalid word W and its corresponding path P. At some point $(q, -m)$, where $m > 0$ and $q < nw+1$, P will reach a minimum such that $y \geq -m$ for all $x \in [0, nw+1]$ and $y > -m$ for all $x < q$. To convert W to a tree-valid word, partition it into two sub-words W_1 and W_2, where W_1 consists of the first q letters of W and W_2 consists of the remaining $nw+1-q$ letters. Let P_k be the path corresponding to the subword W_k (where $k \in \{1,2\}$). P_1 starts at $(0,0)$ and ends at $(q, -m)$. The net change in y (net change for short) is $-m$, and the y-coordinate does not reach $-m$ until the very last step. P_2 starts at $(q, -m)$ and ends at $(nw+1, -1)$. The net change is $m-1$, and the path never falls below the line $y = -m$.

Now, consider the word W', formed by interchanging W_1 and W_2, and its path P'. P'_1 starts at $(0,0)$ and ends at $(q, m-1)$, never falling below the x-axis. P'_2 starts at $(q, m-1)$ and ends at $(nw+1, -1)$, never falling below the x-axis until the final step. W' is a tree-valid word, and a cyclic shift of W. □

Example 2.1.7. Consider Figure 2.2, in which we start with the path representation for the invalid word $W = abbbbba$ ($n = 3$). We have $(q, -m) = (6, -3)$ so we take $W_1 = abbbbb$ and $W_2 = a$. Interchanging W_1 and W_2, we get $W' = aabbbbb$. The path representation of W' is given in Figure 2.2.

Lemma 2.1.3 ([16]). *Every word W (with w a's and $[w(n-1)+1]$ b's) has exactly $nw+1$ distinct cyclic shifts. Of these, only one is tree-valid.*

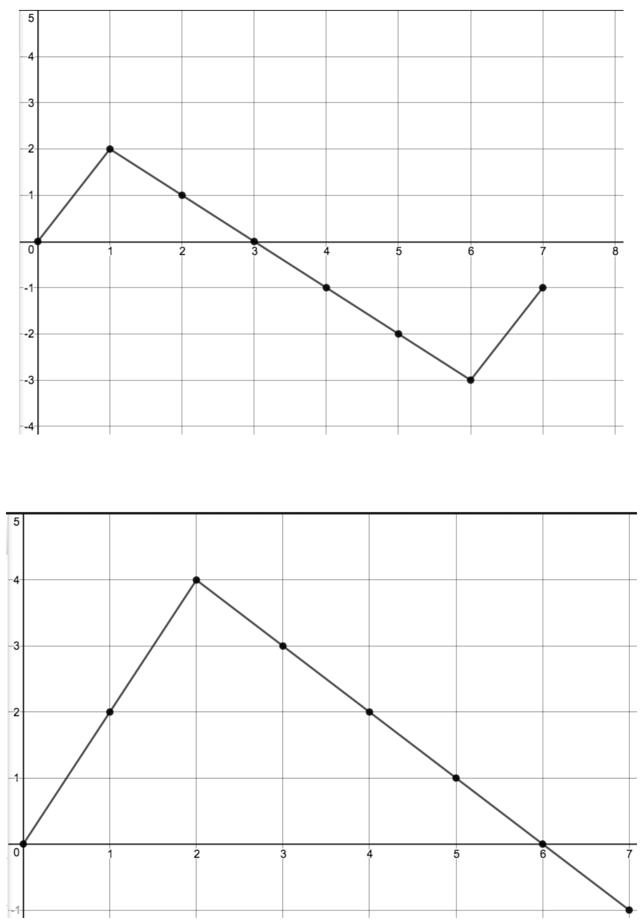


Figure 2.2: Top: the path for $W = abbbbba$ ($n = 3$). Bottom: the path for $W' = aabbbbb$, the only valid cyclic shift of W.

Proof. As shown in Lemma 2.1.2, every such word W has at least one tree-valid cyclic shift. So we will assume that W is tree-valid. The total number of unique cyclic shifts of W is at most $nw + 1$ (the length of the word, L). Let W_j be the subword of W consisting of the first j letters of W and P_j be the corresponding path. Consider the jth cyclic shift to the right (that is, where each letter of W is moved j positions to the right wrapping around as necessary) where $0 < j \leq nw + 1$. It will be shown that:

A) In the path corresponding to the jth cyclic shift, the minimum first occurs at $x = j$. If this is true for all j between 1 and $nw + 1$ (inclusive), then each cyclic shift is unique.

B) The minimum is negative. This ensures that the word is tree-invalid (unless $j = nw+1$, which is just the original word).

We will show A) by induction on j. First, consider the $j = 1$ case (first cyclic shift of W). Since W is valid its path P reaches its minimum at $(L, -1)$. Thus the last letter of W must be b and the first cyclic shift of W will be b followed by W_{L-1}). Notice that P_{L-1} never falls below the x-axis. The path corresponding to the first cyclic shift of W begins at $(0, 0)$ and falls to $(1, -1)$. The remainder of the path consists of P_{L-1} shifted one unit down and one unit to the right (never falling below the line $y = -1$). Thus the minimum first occurs at $x = 1$.

Now, consider the $j = k$ case (kth cyclic shift, where $k > 1$. Suppose that for the $(k-1)$th cyclic shift, the minimum first occurs at $(k-1, -m)$ (it will later be shown that the minimum must be negative, that is $m > 0$, though this is not necessary to prove this result). We will define the function f of x to model the y coordinate of the path for the $(k-1)$th cyclic shift of W, which consists solely of connected line segments. It immediately follows that:

i) $f(k-1) = -m$

ii) $f(x) \geq -m$

iii) If $x < k-1$, $f(x) > -m$.

The kth cyclic shift of W will be the last letter of the $(k-1)$th cyclic shift followed by the first $L-1$ letters of the $(k-1)$th cyclic shift. Thus the path for the kth cyclic shift will be the same as the path for the $(k-1)$th cyclic shift, but shifted one unit to the right and either one unit down or $n-1$ units up. If the last letter of the $(j-1)$th cyclic shift of W is a (that is, $W_{L-(k-1)}$ ends in a), the path will be shifted $n-1$ units up. If the last letter of the $(j-1)$th cyclic shift of W is b (that is, $W_{L-(k-1)}$ ends in b), the path will be shifted one unit down. We also draw a line segment from the origin to either $(1, -1)$ or $(1, n-1)$. To express this more easily, we will define g as a function of x that models the y coordinate of the path for the kth cyclic shift of w. Thus for $x \geq 1$, $g(x) = f(x-1) + h$, where $h \in \{-1, n-1\}$. It immediately follows that:

i) $g(k) = f(k-1) + h = -m + h$

ii) $g(x) \geq -m + h$

iii) If $x < k$, $g(x) > -m + h$.

In other words, the minimum of g first occurs at $x = k$. Through induction on j, we see that the minimum first occurs at $x = j$ for all j between 1 and $nw + 1$ (inclusive) as required.

Therefore, each cyclic shift of W is unique.

To show B), consider the jth cyclic shift of W. It consists of the last j letters of W followed by W_{L-j}, with the minimum occuring at $x = j$. Let W_j have α a's and β b's. We have

$$\alpha(n-1) \geq \beta.$$

At the minimum (occuring at $x = j$), we have:

$$y = (n-1)(w-\alpha) - [w(n-1)+1-\beta] = w(n-1) - \alpha(n-1) - w(n-1) - 1 + \beta = \beta - \alpha(n-1) - 1.$$

Since $\alpha(n-1) \geq \beta$,

$$\beta - \alpha(n-1) \leq 0$$
$$\beta - \alpha(n-1) - 1 < 0.$$

This means that at $x = j$, the minimum value of y occurs and is negative. Thus the jth cyclic shift of W is tree-invalid. $\square$

We can now determine the number of tree-valid words. The total number of words of length $nw + 1$ with w a's (tree-valid or not) is $\binom{nw+1}{w}$. Let $V_{n,w}$ be the number of tree-valid words. Since every word of length $nw+1$ is either tree-valid or is a cyclic shift of a tree-valid word,

$$(nw+1)V_{n,w} = \binom{nw+1}{w}.$$

Solving for $V_{n,w}$,

$$\begin{aligned} V_{n,w} &= \frac{\binom{nw+1}{w}}{nw+1} = \frac{(nw+1)!}{w!(nw+1-w)!(nw+1)} \\ &= \frac{(nw)!}{w!(nw-w)!(nw+1-w)} = \frac{\binom{nw}{w}}{nw+1-w} = \frac{\binom{nw}{w}}{w(n-1)+1}. \end{aligned}$$

Thus there are exactly $\frac{\binom{nw}{w}}{w(n-1)+1}$ complete n-ary plane trees with w internal nodes.

Remark 2.1.1 ([16])**.** Numbers of the form $\frac{\binom{nw}{w}}{w(n-1)+1}$ are known as the n-ary Catalan numbers.

We will next use the word definition to define a lexicographical order on trees and to show that Procedure σ (Procedure 1.2.1) and Procedure τ (Procedure 1.2.2) preserve this ordering. This ordering is necessary to properly define R_i as in Definition 1.2.3. In the remainder of the chapter, we will generalize the enumeration of n-ary trees to forests of n-ary trees as well as prove the Hagen-Rothe convolution identity. These results will be used in Chapter 3 to prove Main Theorem 2 (the $w > 1$ case).

2.2 Lexicographical order of trees

We can use the word representation of a tree to define a standard lexicographical order for complete n-ary plane trees with a given number of internal nodes, as required by Definition 1.2.3. We will also show that both Procedure σ and Procedure σ preserve lexicographical order.

Definition 2.2.1 (**Lexicographical order on *n*-ary trees of weight *w*** [17])**.** Let T_1 and T_2 be complete n-ary trees, each with w internal nodes. We say that $T_1 \prec T_2$ if and only if the word representation for T_1 comes before the word representation for T_2 alphabetically.

Since the trees in a given element of R_i, $i \geq 0$, all have the same number of internal nodes (thus the word representations have the same length) we need not define the comparison of two words of different lengths.

Let V and W be two words of length L over $\{a, b\}$, with $V \neq W$. Let V_i (W_i respectively) denote the ith letter of V (W respectively), we can use the following simple algorithm to determine whether $V \prec W$ [17]:

```
For each $i$ from 1 to $L$ do:

    • If $V_i = a$ and $W_i = b$, return “true”

    • Else if $V_i = b$ and $W_i = a$, return “false”.
```

We will now show that applying Procedure σ and Procedure τ to a sequence of trees preserves lexicographical order.

Proposition 2.2.1. *Let T_1 and T_2 be n-ary trees of weight w, with $T_1 \prec T_2$, and let T be any other n-ary tree. Given any i insert T at the ith leaf of T_1 and T_2, where $1 \leq i \leq 1+w(n-1)$, and denote the new trees by T_1' and T_2' respectively. Then, $T_1' \prec T_2'$.*

Proof. Let V, W, V', and W' denote the word representations of T_1, T_2, T_1', and T_2' respectively. Let V_j denote the jth letter of V and let W_j denote the jth letter of W. Suppose W_x is the ith b in W (thus corresponding to the ith leaf of T_2). There are three possibilities:

1. $V_j \neq W_j$ for some $j < x$. In this case, it immediately follows that $V' \prec W'$ since the beginning of each word is unaffected. Thus $T_1' \prec T_2'$.

2. $V_j = W_j$ for all $j \leq x$. In this case, it immediately follows that $V' \prec W'$ since the beginning and end of each word is unaffected and the "middle" is the same for both V and W (the b at position x is replaced by the word representation of T). Thus $T_1' \prec T_2'$.

3. $V_j = W_j$ for all $j < x$ but $V_x \neq W_x$. Since $V \prec W$, this must mean that $V_x = a$ and $W_x = b$. Now, in W' this b will have been replaced by the word representation of T. To generate V', on the other hand, the next b will be replaced by the word representation of T. Every letter until then will be an a. W' will reach the first b of the word representation of T before V' does, and the corresponding letter in V' must be an a. Thus $V' \prec W'$ and $T_1' \prec T_2'$. □

It immediately follows from this result that Procedure σ preserves lexicographical order.

Proposition 2.2.2. *Let T_1 and T_2 be complete n-ary plane trees of weight w, with $T_1 \prec T_2$, and let T be any other complete n-ary plane tree (with weight w'). Insert T_1 at the ith leaf of T to generate T_1'', and insert T_2 at the ith leaf of T to generate T_2'', where $1 \leq i \leq 1+w'(n-1)$. Then, $T_1'' \prec T_2''$.*

Proof. Let U, V, W, V'', and W'' denote the word representations of T, T_1, T_2, T_1'', and T_2'' respectively. Let U_j denote the jth letter of U. Suppose U_x is the ith b in U (thus corresponding to the ith leaf of T). Then U, V'', and W'' will be identical up to the first x letters. Starting at position $x+1$, V'' and W'' will concur with V and W respectively. Thus $V'' \prec W''$ and $T_1'' \prec T_2''$. □

It immediately follows from this result that Procedure τ preserves lexicographical order. Combining these two results, we obtain:

Theorem 2.2.3. *Procedure σ and Procedure τ both preserve lexicographical order.*

2.3 Enumerating forests of n-ary trees

We can generalize the argument from Section 2.1.1 to forests of complete n-ary plane trees [1, 5], which will be useful in the following chapter when we prove Main Theorem 2. The arguments that follow are largely adapted from [5] and [16]. We begin with some terminology.

Definition 2.3.1 ([9])**.** A **forest** is a graph consisting of a disjoint union of trees. A **plane forest** is an embedding of a forest in a plane. That is, the order in which the trees are placed in the plane is relevant. Conventionally, plane forests of rooted trees have the roots placed on the same horizontal axis.

Definition 2.3.2. We will use $\boldsymbol{F(i,t;n)}$ to denote the number of plane forests composed of t complete n-ary trees with i internal nodes between them.

It is well-known that $F(i,t;n) = \frac{t}{ni+t}\binom{ni+t}{i}$ [5, 1]. The proof for this result will be reviewed in this section.

Proposition 2.3.1 ([1])**.** *A forest of t n-ary trees with i internal nodes has $i(n-1)+t$ leaf nodes.*

Proof. Let the forest have t trees, and let the kth tree have i_k internal nodes. The kth tree then has $i_k(n-1)+1$ leaves, so the total number of leaf nodes in the forest is

$$\sum_{k=1}^{t}[i_k(n-1)+1] = \sum_{k=1}^{t} i_k(n-1) + \sum_{k=1}^{t} 1 = (n-1)\sum_{k=1}^{t} i_k + t = (n-1)i + t$$

□

2.3.1 The word representation of a forest

The word representation of a forest follows naturally from the word representation of n-ary trees given in Definition 2.1.3 [5].

Definition 2.3.3 ([5])**.** To write the **word representation of a forest**, write the word representation of each tree in the forest and concatenate them from left to right.

An example of a forest and its word representation is given in Figure 2.3.

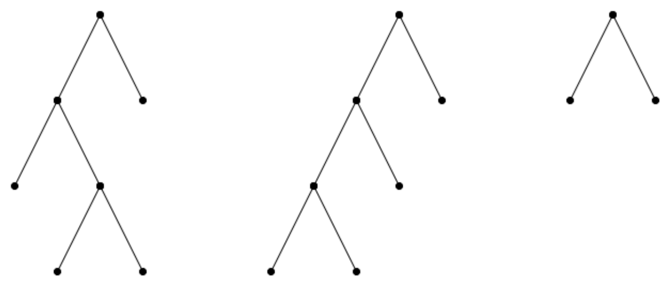


Figure 2.3: The forest with word representation *aababbbaaabbbbabb*.

Conversely, we can construct a forest from a word through the procedure in Definition 2.1.4, but starting a new tree as soon as we have classified all vertices (i.e. have nowhere else to extend the current tree).

As with trees, we will need a way of identifying which words over $A = \{a, b\}$ actually represent forests of n-ary trees.

2.3.2 Identifying valid words

Definition 2.3.4. A word is said to be **(n, i)-forest-valid** if and only if it represents a plane forest of t complete n-ary trees with a total of i internal nodes. For the remainder of this section, the term "forest-valid" will refer to a word that is (n, i)-forest-valid.

We again identify necessary and sufficient conditions for a word to be valid.

Lemma 2.3.2 ([5]). *A word W representing a plane forest of t complete n-ary plane trees with a total of i internal nodes must satisfy the following three conditions. Conversely, every word satisfying these criteria will represent such a forest:*

1. i a's, one for each internal node.

2. $i(n-1)+t$ b's, one for each leaf node.

3. It must be possible to partition W into t subwords $W_1, W_2, \ldots W_t$. Each of these subwords represents a complete n-ary tree. That is, the following is true for each $k \in \{1, 2, \ldots, t\}$:

- Let L_k be the length of W_k. The total number of b's in W_k is one more than $n-1$ times the number of $a's$. For the first m letters of W_k ($m < L_k$), the number of b's does not exceed $n-1$ times the number of a's.

Proof. Conditions 1 and 2 follow immediately and from Proposition 2.3.1, respectively. Condition 3 is necessary (and sufficient) to ensure that we get a forest with the correct number of complete n-ary trees. This follows from the forest construction algorithm. $\square$

Note that by definition the partition of W into tree-valid subwords will be unique, as W_1 would no longer be valid if letters were added or removed. The same argument can be made for W_2 through W_t.

Thus to find the number of plane forests of t complete n-ary trees with i internal nodes, we must find the number of words with i as and $i(n-1)+t$ bs that satisfy the third condition in Lemma 2.3.2. The total number of words (forest-valid or forest-invalid) with i a's and $i(n-1)+t$ b's is $\binom{ni+t}{i}$. Let r be the ratio of forest-valid words to total words. We have $F(i,t;n) = r\binom{ni+t}{i}$. So to find a formula $F(i,t;n)$, all that remains is to find a formula for r.

We will again turn to the path representation of words and cyclic shifts as in Section 2.1.3 (see also [5] and [16]). The proof will be slightly more involved this time however, because a word can have more than one valid cyclic shift. For example if we take $n = 2$, $t = 2$, $i = 1$ the valid word $abbb$ has another valid cyclic shift ($babb$). Thus, we will introduce a new system that gives us a way of distinguishing cyclic shifts of a word. Firstly, we use A' to denote the alphabet $\{a_1, a_2, a_3, ..., a_i, b_1, b_2, ..., b_{i(n-1)+t}\}$. We will consider words over A' that use each letter exactly once. A word over A_i is said to be valid if and only if removing the subscripts would produce a valid word over $\{a, b\}$.

The advantage of using A' is that, since each letter-subscript pairing is considered a distinct letter, the total number of distinct cyclic shifts (forest-valid or not) for any word is equal to the length of the word $ni+t$. For example $a_1b_1b_2a_2b_3b_4$ would be considered a different cyclic shift from $a_2b_3b_4a_1b_1b_2$, even though these two words would be the same if the subscripts were removed.

Lemma 2.3.3. *Let B be the set of all words of length $ni+t$ with exactly i a's and $i(n-1)+t$ b's (no subscripts) and let B_V be the set of all forest-valid words of length $ni+t$ with exactly*

i a's and $i(n-1)+t$ b's (no subscripts). Let B' be the set of all words using all letters of the alphabet $A' = \{a_1, a_2, a_3, ..., a_i, b_1, b_2, ..., b_{i(n-1)+t}\}$ exactly once, and let B'_V be the set of all forest-valid words using all letters of A' exactly once. Then, $|B'_V|/|B'| = |B_V|/|B| = r$.

Proof. Given an element of B, there are $i![i(n-1)+t]!$ elements of B' that would give the same element of B if the subscripts on the letters were removed (the number of ways to permute the subscripts on the a's, times the number of ways to permute the subscripts on the b's). Likewise, given an element of B_V, there are $i![i(n-1)+t]!$ elements of B'_V that would give the same element of B_V if the subscripts on the letters were removed. Thus,

$$\frac{|B'_V|}{|B'|} = \frac{|B_V| \cdot i![i(n-1)+t]!}{|B| \cdot i![i(n-1)+t]!} = \frac{|B_V|}{|B|} = r.$$

□

2.3.3 The path representation of a word

As in Section 2.1.3, every word with i a's and $[i(n-1)+t]$ b's can be represented as a path in the xy plane, starting at the origin [16]. At the point (x, y) join a line segment to $(x+1, y+n-1)$ if the following letter is a a, or to $(x+1, y-1)$ if it's a b. By definition, all paths will start at $(0,0)$ and end at $(ni+t, -t)$. If the word is forest-valid, the path will never fall below the line $y = 1-t$ until the final step. If the word is forest-invalid, the path will fall below the line $y = 1-t$ before the final step. When drawing the paths for words over A', we will ignore the subscripts (thus each path will correspond to multiple words).

Lemma 2.3.4 ([16])**.** *Every word over A' can be converted to a forest-valid word through a cyclic shift.*

Proof. Consider a forest-invalid word W and its corresponding path P. At some point $(q, -m)$, where $m > t-1$ and $q < ni+t$, P will reach a minimum such that $y \geq -m$ for all $x \in [0, ni+t]$ and $y > -m$ for all $x < q$. To convert W to a forest-valid word, partition it into two sub-words W_1 and W_2, where W_1 consists of the first q letters of W and W_2 consists of the remaining $ni+t-q$ letters. Let P_k be the path corresponding to the subword W_k. P_1 starts at $(0,0)$ and ends at $(q, -m)$. The net change is $-m$, and the y-coordinate does not reach $-m$ until the very last step. P_2 starts at $(q, -m)$ and ends at $(ni + t, -t)$. The net change is $m - t$, and the path never falls below the line $y = -m$.

Now, consider the word W', formed by interchanging W_1 and W_2, and its path P'. P'_1 starts at $(0, 0)$ and ends at $(q, m - t)$, never falling below the line $y = 1 - t$. P'_2 starts at $(q, m - t)$ and ends at $(ni + t, -t)$, never falling below the line $y = 1 - t$ until the final step. W' is a forest-valid word, and a cyclic shift of W. □

Lemma 2.3.5 ([16])**.** *Let W be a word over A'. W has $ni + t$ distinct cyclic shifts, of which t are forest-valid.*

Proof. Without loss of generality (see Lemma 2.3.4), assume W is forest-valid. Let

$$W = W^{(1)}W^{(2)}W^{(3)} \dots W^{(t)},$$

where each subword $W^{(j)}$ represents a tree. Clearly $W^{(t)}W^{(1)}W^{(2)} \dots W^{(t-1)}$, $W^{(t-1)}W^{(t)}W^{(1)} \dots W^{(t-2)}$, etc. are valid cyclic shifts of W, so at least t of the cyclic shifts of W are forest-valid.

Next, consider the kth cyclic shift of W, where $k < L^{(t)}$ (the length of $W^{(t)}$). Let $X = W^{(1)}W^{(2)}W^{(3)} \dots W^{(t-1)}$, and let $W_c^{(j)}$ represent the first c letters of $W^{(j)}$. The kth cyclic shift of W consists of the last k letters of $W^{(t)}$, followed by X, followed by $W_{L-k}^{(t)}$. Let $W_{L-k}^{(t)}$ have α a's and β b's. Since $W^{(t)}$ represents an n-ary tree, $\beta \leq (n-1)\alpha$. The last k letters of $W^{(t)}$ must have $\gamma - \alpha$ a's and $\gamma(n-1) + 1 - \beta$ b's. The path representation for the kth cyclic shift of W will then go through the following points:

1. The starting point $(0, 0)$
2. $(k, [\gamma - \alpha][n-1] - \gamma[n-1] - 1 + \beta) = (k, \beta - \alpha[n-1] - 1)$ (the last k letters of $W^{(t)}$)
3. $(in + t - \alpha - \beta, \beta - \alpha[n-1] - 1 - [t-1]) = (in + t - \alpha - \beta, \beta - \alpha[n-1] - t)$ (for X)
4. $(in + t, -t)$

Because $\beta \leq (n-1)\alpha$, $\beta - (n-1)\alpha \leq 0$. This means that $\beta - \alpha[n-1] - t < 1 - t$. So the path representation for the kth cyclic shift of W falls below the line $y = 1 - t$ (at step 3 above), thus the kth cyclic shift of W cannot be a forest-valid word.

This means that the only valid cyclic shifts of W are:

$$W^{(1)}W^{(2)}W^{(3)} \dots W^{(t-1)}W^{(t)}.$$

$$W^{(t)}W^{(1)}W^{(2)}\dots W^{(t-1)},$$

$$W^{(t-1)}W^{(t)}W^{(1)}\dots W^{(t-2)},$$

$$\dots$$

$$W^{(2)}W^{(3)}W^{(4)}\dots W^{(t-1)}W^{(t)}W^{(1)}.$$

So W has exactly t forest-valid cyclic shifts. □

Since every over A' is either forest-valid or a cyclic shift of a forest-valid word, $r = \frac{t}{in+t}$. Multiplying r by the total number of words with i a's and $(n-1)i+1$ b's (no subscripts) results in Proposition 2.3.6:

Proposition 2.3.6 ([5])**.**

$$F(i,t;n) = \frac{t}{ni+t}\binom{ni+t}{i}.$$

2.4 The Hagen-Rothe Identity

We will next use the word and path representations of forests from the previous section to prove an identity related to convolutions of finite sequences. This identity will be used in the next chapter. This result is fairly well-known [18, 4, 14, 15], and we will go through a proof given in [18].

Lemma 2.4.1 (**Hagen-Rothe Identity** [18])**.** *Let p, q, r, and L be non-negative integers with $p > 0$, $q > 0$, and $r \geq qL$. Then,*

$$\sum_{j=0}^{L} \frac{p}{p+qj}\binom{p+qj}{j}\binom{r-qj}{L-j} = \binom{p+r}{L} \tag{2.1}$$

Proof. Consider fixed p, q, r, and L as given in the statement of the Lemma. It does not matter what the value of q is, provided that $r \geq qL$ (to ensure that the left side of the equation is defined). The right side of the equation is simply the total number of words with L a's and $p+r-L$ b's (no other restrictions). The path representation of such a word can be defined as in Section 2.1.3, but moving from (x,y) to $(x+1, y+q-1)$ for every a.

The net change in y is $L(q-1)-(p+r-L) = Lq-p-r$. Since $r \geq qL$, $q \leq \frac{r}{L}$. So $Lq-p-r \leq L(\frac{r}{L})-p-r$. Thus the net change in y cannot exceed $-p$. This means the path representation for any word (with L a's and $p+r-L$ b's) must touch the line $y=-p$.

For the left side, consider any word W (that has L a's and $p+r-L$ b's) and its path P. Divide W into two parts W_1 and W_2 and their corresponding paths P_1 and P_2, where the path does not fall below the line $y = 1-p$ until the final step of P_1. W_1 can then represent a plane forest of p complete q-ary trees. If W_1 is to have j a's, there are $F(j,p;q)$ possibilities for W_1. W_1 is of length $p+qj$, so the length of W_2 is $p+r-(p+qj) = r-qj$. Since there are no restrictions on W_2, there are $\binom{r-qj}{L-j}$ possibilities. Applying the multiplication principle and summing over all possible values of j gives

$$\sum_{j=0}^{L} F(j,p;q)\binom{r-qj}{L-j} = \sum_{j=0}^{L} \frac{p}{p+qj}\binom{p+qj}{j}\binom{r-qj}{L-j}$$

which is the left side of the equation given in Lemma 2.4.1. □

Remark 2.4.1. Note that if $q=0$ and we have $p \geq L$ and $q \geq L$, the Hagen-Rothe identity reduces to the more familiar Vandermonde's Identity.

2.5 Summary

We have reviewed the proofs for three major known results that can be related to the proof of the Main Theorems. Firstly, we have shown that the number of complete n-ary plane trees with w internal nodes is exactly $\frac{\binom{nw}{w}}{w(n-1)+1}$, from which Main Theorem 1 immediately followed. We also generalized this result to plane forests with t complete n-ary trees and w total internal nodes, showing that there are $\frac{t}{nw+t}\binom{nw+t}{w}$ such forests. Lastly, we reviewed a proof of the Hagen-Rothe convolution identity that used the properties of forests. These latter results will be used in the following chapter to prove Main Theorem 2.

Chapter 3
Proof of Main Thoerem 2 ($w > 1$)

This chapter will provide a proof of Main Theorem 2 (the $w > 1$ case). We will develop properties of Procedure σ (Procedure 1.2.1) and Procedure τ (Procedure 1.2.2), and use them to divide R_i into $i + 1$ different subsets. We will prove that these subsets are mutually exclusive and exhaustive (a partition of R_i), and then determine the number of elements in each of them. This will ultimately lead to a recurrence relation that we can solve. We will end by providing a new, non-recursive method of generating the elements of R_i. The proofs we provide are original, though when we enumerate the elements of the subsets of R_i and solve the recurrence relations we will make use of known results reviewed in Chapter 2. The proof also involves forests of n-ary trees, and may have further applications to this topic.

We begin with notation that allows us to write elements of R_i as words based on which procedures were used to generate them. This notation can be used to prove some properties related to Procedure σ and Procedure τ. We will next characterize and enumerate the elements of R_1 and R_2, then generalize to higher values of i. Unless otherwise stated, we will assume that n and w are fixed integers greater than 1.

3.1 The word representation of elements of R_i

We will begin this section by introducing some new notation that will help us to write elements of R_i more compactly as a sequence of applications of Procedure σ and Procedure τ. We will then develop some basic identities related to this notation. These properties will be useful in our two proofs of Main Theorem 2, one of which will be presented in this chapter and one in the following chapter.

We will first introduce new notation for the number of leaves per tree for each element of R_i. This notation is simpler than the $N(w)$ notation introduced in Chapter 1 (Definition 1.1.6), and it is unambiguous because we have assumed n and w to be fixed constants.

Definition 3.1.1. Given fixed arity n and weight w and any $i \geq 0$, we define $\boldsymbol{\ell(i)}$ to be the number of leaves per tree for each element of R_i.

Note that

$$\ell(i) = N(w+i) = 1 + (w+i)(n-1), \tag{3.1}$$

by Proposition 1.1.3.

We next introduce a new notation for elements of R_i. Each element of R_i can be defined by at least one sequence of applications of Procedure σ and Procedure τ[1]. We will represent each procedure as a letter, as defined in the following notation.

Definition 3.1.2. Let $\boldsymbol{\sigma_j}$ represent applying Procedure σ at the jth leaf (that is, grafting T_ψ at the jth leaf of each tree in a given sequence), and let $\boldsymbol{\tau_k}$ denote applying Procedure τ at the kth leaf (that is, grafting each tree in a given sequence at the kth leaf of the basic tree T_ψ).

Recall that the leaves of a tree are labelled through pre-order depth-first traversal, consistent with the operation diagrams given in Chapter 1.

Recall that applying Procedure σ at the jth leaf to a given sequence of trees produces a new sequence of trees where each term is produced by grafting the basic tree T_ψ at the jth leaf of the corresponding term of the original sequence. Likewise applying Procedure τ at the kth leaf to a given sequence of trees produces a sequence of trees where each term is produced by grafting the corresponding term of the original sequence to the kth leaf of the basic tree T_ψ. Thus every element of R_i can be represented by a set of words of length i over the alphabet $A_i = \{\tau_1, \tau_2, \tau_3, \ldots, \tau_n, \sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_{\ell(i-1)}\}$ (we will discuss elements of R_i having multiple word representations, known as equivalent words, later in this section). By convention, the procedures are to be executed from left to right starting on the sole element of R_0, denoted by $(X_1, X_2, X_3, \ldots, X_{l_w})$.

[1]Adapted from unpublished paper by Nick Beaton.

Example 3.1.3. Let $n = 2$, $w = 2$. Thus $R_0 = \{(X_1, X_2)\}$. We will denote $S_0 = (X_1, X_2)$ (X_1 and X_2 are shown in Figure 3.1[2]). The word $\tau_1\tau_2\sigma_1\sigma_2$ represents the following sequence of procedures:

1. Apply Procedure τ at leaf 1. That is, insert each tree in S_0 at leaf 1 of the basic tree T_ψ to produce the sequence $T_\psi \circ_1 S_0 = (T_\psi \circ_1 X_1, T_\psi \circ_1 X_2)$ (which we will denote as $S_1 = (X_1', X_2')$). See Figure 3.2.

2. Apply Procedure τ at leaf 2. That is, insert each tree in S_1 at leaf 2 of the basic tree T_ψ to produce the sequence $T_\psi \circ_2 S_1 = (T_\psi \circ_2 X_1', T_\psi \circ_2 X_2')$ (which we will denote as $S_2 = (X_1'', X_2'')$). See Figure 3.3.

3. Apply Procedure σ at leaf 1. That is, insert the basic tree T_ψ at leaf 1 of each of the trees in the sequence S_2 to produce the sequence $S_2 \circ_1 T_\psi = (X_1'' \circ_1 T_\psi, X_2'' \circ_1 T_\psi)$ (which we will denote as $S_3 = (X_1''', X_2''')$). See Figure 3.4.

4. Apply Procedure σ at leaf 2. That is, insert the basic tree T_ψ at leaf 2 of each of the trees in the sequence S_3 to produce the sequence $S_3 \circ_2 T_\psi = (X_1''' \circ_2 T_\psi, X_2''' \circ_2 T_\psi)$. See Figure 3.5.

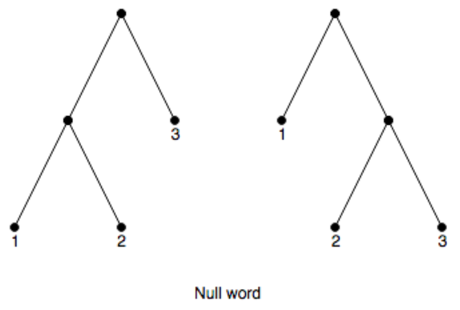


Figure 3.1: Null word for $n = 2$, $w = 2$.

[2]Figures created with Draw.io, `https://www.draw.io/`.

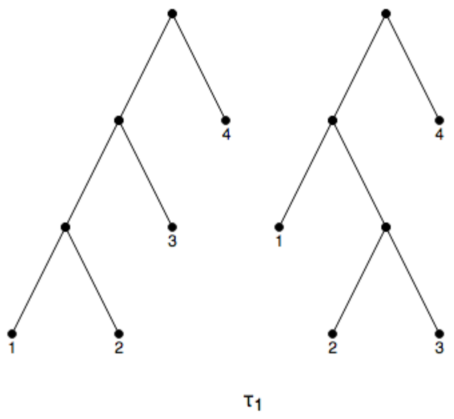


Figure 3.2: Applying Procedure τ at leaf 1, $n = 2$, $w = 2$.

3.1.1 Identifying valid words and equivalent words

We will next define valid words over A_i.

Definition 3.1.4. A word of length i over A_i is said to be $\boldsymbol{A_i}$**-valid** if it represents an element of R_i. The set of all A_i-valid words of length i over A_i is denoted by $V(A_i)$.

For the remainder of this chapter, the term "valid" will refer to A_i-valid words.

A letter σ_j or τ_j at position k in an A_i-valid word means that the corresponding operation is being applied to an element of R_{k-1}, where elements of R_{k-1} are sequences of trees each with $\ell(k-1)$ leaves (see Definition 3.1.2). Clearly, τ's can be placed anywhere as any tree can be inserted into T_ψ at any of its n leaves. However, σ_j can only be placed at a position k of a valid word if $j \leq \ell(k-1)$ so that leaf j is available for applying Procedure σ. For example if $w = 2$ and $n = 2$, the word $\sigma_4\sigma_3$ is not valid because each of the trees in the sole element of R_0 only have 3 leaves, so we cannot insert T_ψ at the 4th leaf. Note that a word will be valid if and only if the sequence of procedures could be applied to any one of the trees in the starting sequence S_0, as all trees will have the same number of leaves in any element of R_i.

Lemma 3.1.1. *If $W \in V(A_i)$, then the first j characters of W form an element of $V(A_j)$ for all $j \leq i$.*

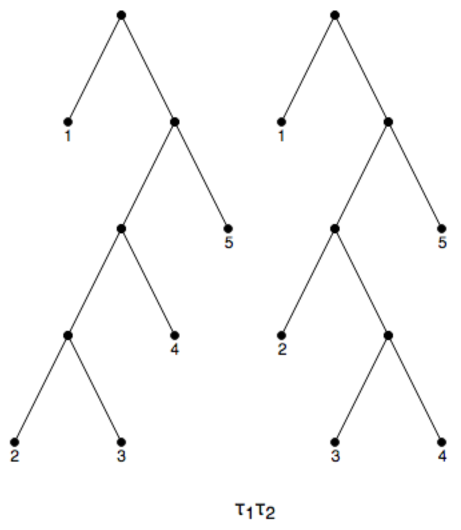


Figure 3.3: Applying Procedure τ at leaf 1 and then at leaf 2, $n = 2$, $w = 2$.

Proof. This follows immediately from the definition of validity and of a word as a sequence of procedures applied to the sole element of R_0. □

We will next define and prove two identities related to equivalent words over A_i.

Definition 3.1.5. Two valid words are said to be **equivalent** if they represent the same element of R_i. If W_1 and W_2 are equivalent words, we say that $W_1 \cong W_2$.

Lemma 3.1.2. *Given any $i \geq 1$, let W_1 and W_2 be words over A_i such that there exists j, k with $j > k$ and where $W_1\sigma_j\sigma_k W_2$ is a valid word (of length i, over A_i). Then $W_1\sigma_j\sigma_k W_2 \cong W_1\sigma_k\sigma_{j+(n-1)}W_2$.*

Likewise if $W_1\sigma_l\sigma_m W_2$ is a valid word where $m \geq l+(n-1)$ then $W_1\sigma_l\sigma_m W_2 \cong W_1\sigma_{m-(n-1)}\sigma_l W_2$.

Proof. Since $W_1\sigma_j\sigma_k W_2$ is valid so are W_1, $W_1\sigma_j$, and $W_1\sigma_j\sigma_k$ (by Lemma 3.1.1). We first note that $W_1\sigma_k\sigma_{j+(n-1)}W_2$ is guaranteed to be a valid word given the validity of $W_1\sigma_j\sigma_k W_2$. To see this, we denote the length of W_1 by L_1 (that is, $W_1 \in R_{L_1}$). Hence $W_1\sigma_j$ being valid means that $j \leq \ell(L_1)$, that is $j \leq 1 + (w + L_1)(n-1)$. Likewise $W_1\sigma_j\sigma_k$ being valid means that $k \leq \ell(L_1+1)$, that is $k \leq 1+(w+L_1+1)(n-1)$. Since $k < j$ it immediately follows that $k \leq \ell(L_1)$ and that $j + (n-1) \leq \ell(L_1 + 1)$ (since $j + (n-1) \leq 1 + (w + L_1)(n-1) + (n-1)$

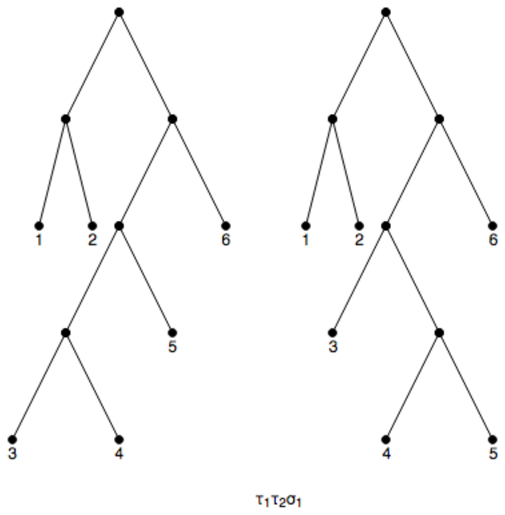


Figure 3.4: Applying Procedure τ at leaf 1, at leaf 2, and then Procedure σ at leaf 1, $n = 2$, $w = 2$.

so $j + (n - 1) \leq 1 + (w + L_1 + 1)(n - 1))$. This makes $W_1\sigma_k\sigma_{j+(n-1)}$ a valid word. Since W_2 is unchanged, $W_1\sigma_k\sigma_{j+(n-1)}W_2$ must be valid as well.

Now consider the left-hand side of the equivalence $W_1\sigma_j\sigma_kW_2 \cong W_1\sigma_k\sigma_{j+(n-1)}W_2$. In $W_1\sigma_j$, leaf j of each of the original trees in the sequence of trees represented by W_1 is replaced by the basic tree T_ψ. Leaves $j+1$ and above are relabelled, but leaf k is not because $k < j$. So the σ_k step simply inserts T_ψ at what was leaf k in each of the original trees (in the sequence of trees represented by W_1). The sequence of procedures given by W_2 is then applied.

For the right hand side, we first replace leaf k in each of the original trees (in the sequence of trees represented by W_1) by T_ψ (leaf k must exist because $k < j$ and $W_1\sigma_j$ is valid so leaf j exists). For leaves $k + 1$ and higher, the label is increased by $n - 1$ (since Procedure σ increases the total number of leaves by $n - 1$). This means leaf j becomes leaf $j + (n - 1)$. Thus $\sigma_{j+(n-1)}$ inserts T_ψ at what was leaf j in the original trees. The sequence of procedures given by W_2 is then applied.

For the converse the arguments are parallel. However, we have the additional requirement that the difference between the two subscripts exceeds $n - 1$, as otherwise the second application of Procedure σ (denoted by σ_m) would be at one of the leaves generated by the first

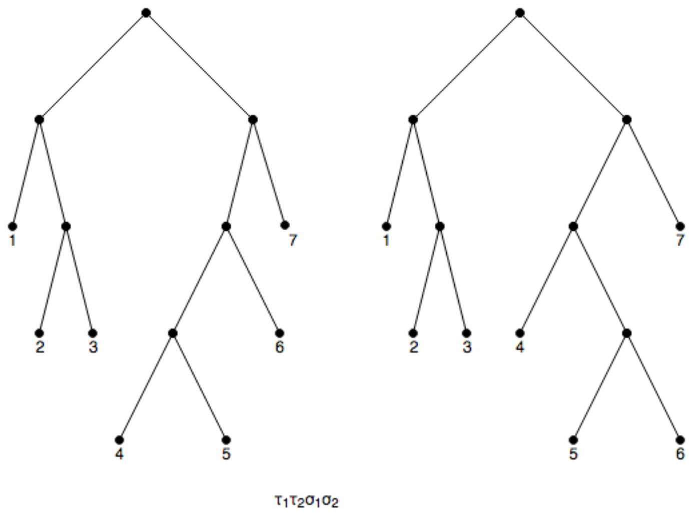


Figure 3.5: Applying Procedure τ at leaf 1, at leaf 2, Procedure σ at leaf 1, and then at leaf 2, $n = 2$, $w = 2$.

application of Procedure σ (denoted by σ_l) so it must be performed after the first application of Procedure σ. □

Example 3.1.6. Let $w = n = 2$. This example will demonstrate a special case of the preceding lemma for $A_i = A_2 = \{\tau_1, \tau_2, \sigma_1, \sigma_2, \sigma_3, \sigma_4\}$ and where both W_1 and W_2 correspond to the null word. Figure 3.6 demonstrates that $\sigma_2\sigma_1 \cong \sigma_1\sigma_3$.

Lemma 3.1.3. *Given any $i \geq 1$ let W_1 and W_2 be words over A_i such that $W_1\sigma_j\tau_kW_2$ is a valid word over A_i for some positive integers j and k. Then $W_1\sigma_j\tau_kW_2 \cong W_1\tau_k\sigma_{j+(k-1)}W_2$.*

Likewise if we denote the length of W_1 by L_1 and $W_1\tau_l\sigma_mW_2$ is a valid word where $l \leq m \leq \ell(L_1 + 1) - (l - 1)$ then $W_1\tau_l\sigma_mW_2 \cong W_1\sigma_{m-(l-1)}\tau_lW_2$.

Proof. Firstly note that similar to Lemma 3.1.2, $W_1\tau_k\sigma_{j+(k-1)}W_2$ is guaranteed to be a valid word. We know that $j \leq \ell(L_1)$; that is $j \leq 1 + (w + L_1)(n - 1)$. It immediately follows that $j + (n - 1) \leq \ell(L_1 + 1)$, since $j + (n - 1) \leq 1 + (w + L_1)(n - 1) + (n - 1)$ so $j + (n - 1) \leq 1 + (w + L_1 + 1)(n - 1)$.

On the left-hand side, Procedure σ is applied at leaf j in each of the trees in the sequence represented by W_1. Then Procedure τ is applied at leaf k of the basic tree (that is, inserting each tree in the sequence at leaf k in T_ψ). The relabelling of leaves resulting from the first

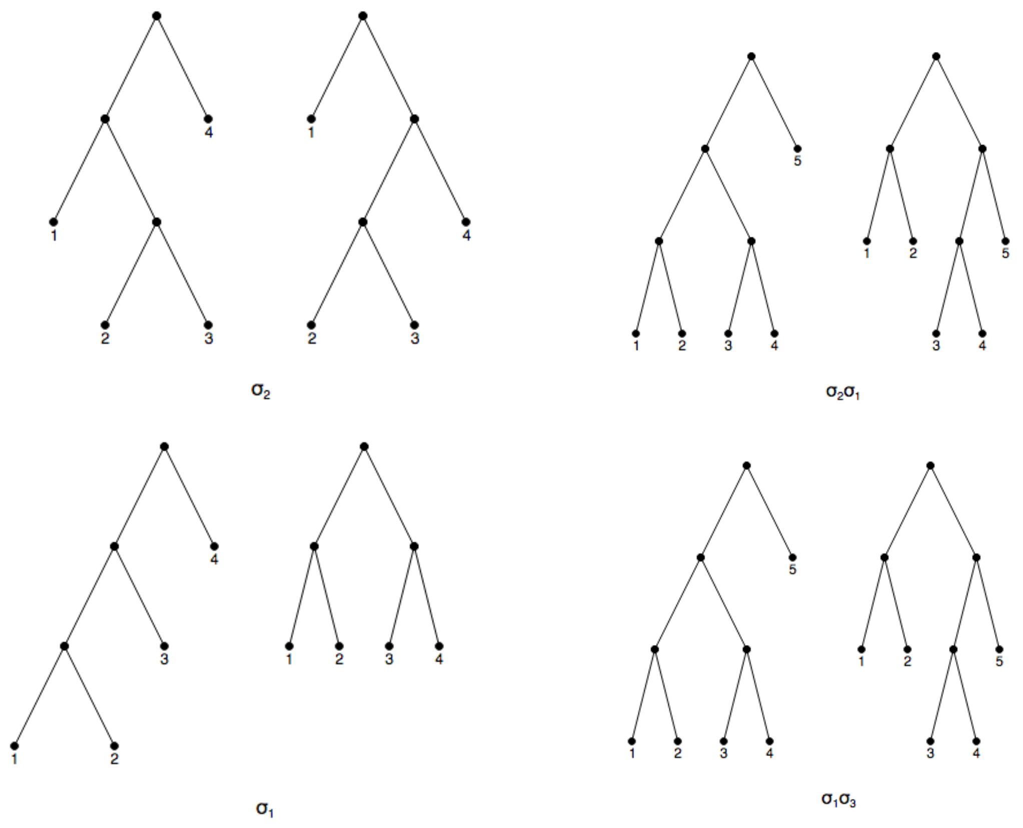


Figure 3.6: Example 3.1.6.

procedure does not affect Procedure τ. The sequence of procedures represented by W_2 is then applied.

For the right-hand side, Procedure τ is first applied at leaf k. This means leaf k in the basic tree was replaced by each tree in the sequence represented by W_1. This procedure added $k-1$ leaves to the left of the original tree and $n-k$ to the right. Thus each of the original leaves has its label increased by $k-1$. So what was originally leaf j is now leaf $j+(k-1)$. Thus we apply $\sigma_{j+(k-1)}$.

For the converse the arguments are parallel. However, we have the additional requirement that the application of Procedure σ (denoted by σ_m) is not at one of the leaves added by the application of Procedure τ (denoted by τ_l). This means we cannot have $m < l$ or $m > \ell(L_1+1)-(l-1)$. Note also that if we had $m < l$ or $m > \ell(L_1+1)-(l-1)$, $W_1\sigma_{m-(l-1)}\tau_l$ would be invalid (as we would have either $m-(l-1) < 1$ or $m-(l-1) > \ell(L_1+1)$. $\square$

Example 3.1.7. Let $w = n = 2$. This example will demonstrate a special case of the preceding lemma for $A_i = A_2 = \{\tau_1, \tau_2, \sigma_1, \sigma_2, \sigma_3, \sigma_4\}$ and where both W_1 and W_2 correspond to the null word. Figure 3.7 demonstrates that $\sigma_1\tau_1 \cong \tau_1\sigma_1$.

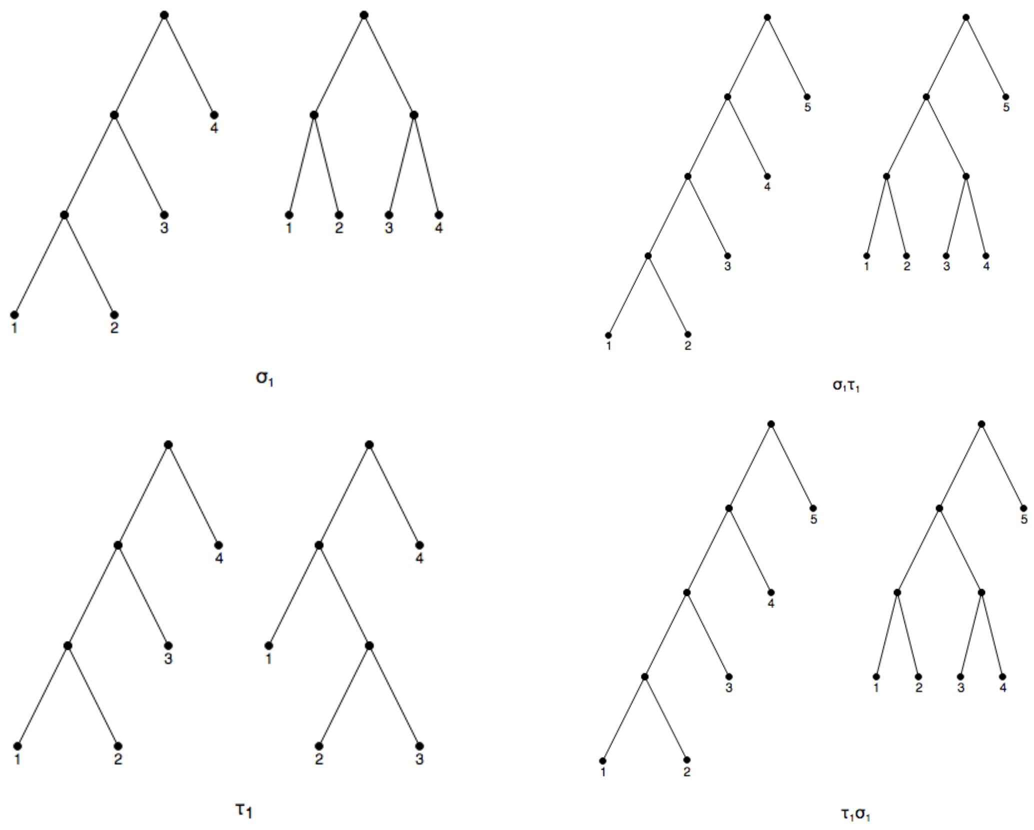


Figure 3.7: Proof that $\sigma_1\tau_1 \cong \tau_1\sigma_1$.

These identities will be useful in proving key properties of Procedure σ and Procedure τ that we will use in our partitions of R_i.

3.2 Basic properties and enumerating the elements of R_1

Our strategy in proving Main Theorem 2 will be to develop a partition of R_i. We will develop this partition based on the procedure used to generate a given element of R_i. In this section we will develop properties of Procedure σ and Procedure τ that we will use to partition R_1. Our strategy for enumerating the elements of R_2 and R_i for higher values of i will be parallel, though we will require slightly more complex properties of Procedure σ and Procedure τ.

3.2.1 Basic properties of Procedure σ and Procedure τ

This next definition will help us develop our partition of R_i.

Definition 3.2.1. Let $\boldsymbol{R_i^{(1)}}$ denote the elements of R_i that can be generated by applying Procedure σ to an element of R_{i-1} (that is, the element of R_i has a word representation that ends with a σ). Similarly, let $\boldsymbol{R_i^{(2)}}$ denote the elements of R_i that can be generated by applying Procedure τ to an element of R_{i-1} (that is, the element of R_i has a word representation that ends with a τ). Thus $R_i = R_i^{(1)} \cup R_i^{(2)}$ and

$$|R_i| = |R_i^{(1)}| + |R_i^{(2)}| - |R_i^{(1)} \cap R_i^{(2)}|.$$

The following properties will help us to identify elements of $R_i^{(2)}$, providing a necessary and sufficient condition. We begin with a definition.

Definition 3.2.2. A **singular tree** is a complete n-ary plane tree with exactly $n-1$ leaves at level 1. A ***j*-singular tree** is a singular tree in which the sole internal node at level 1 is located at position j (from the left).

Example 3.2.3. In Figure 3.7, both trees in τ_1 are 1-singular as the first (leftmost) vertex is the only internal node at level 1.

We will use Definition 3.2.2 to characterize elements of $R_i^{(2)}$ in the following properties.

Proposition 3.2.1. *If $S \in R_i^{(2)}$ for some $i \in \mathbb{N}$, then S is a sequence of j-singular trees for some $j \in \{1, 2, \dots, n\}$.*

Proof. This follows immediately from Definition 3.2.2 and the definition of Procedure τ. In particular S having a word representation ending in τ_j for some $j \in \{1, 2, \dots, n\}$ implies S is a sequence of j-singular trees. $\square$

This property allows us to simplify the enumeration of the elements of R_1.

Corollary 3.2.2. *If $w > 1$, $R_1^{(1)} \cap R_1^{(2)} = \emptyset$*

Proof. The sole element of R_0, $(X_1, X_2, X_3, \ldots, X_{l_w})$, cannot be a sequence of singular trees for $w > 1$ as this element is a sequence of all complete n-ary plane trees of weight w, which must necessarily include at least one tree with two internal nodes at level 1. Applying Procedure σ to this sequence will not result in a sequence of j-singular trees because Procedure σ cannot reduce the number of level-1 internal nodes. Thus by Proposition 3.2.1 all elements of $R_1^{(2)}$ can only be generated through Procedure 2. $\square$

These results are sufficient for enumerating the elements of R_1. However, the converse of Proposition 3.2.1 can be proven easily and will be used in the following sections so its proof will be presented now.

Proposition 3.2.3. *Given any $i \in \mathbb{N}$, suppose $S \in R_i$ and S is a sequence of j-singular trees for some value of $j \leq n$. Then, $S \in R_i^{(2)}$.*

Proof. The proof is by induction on i. For $i = 1$, the only way to generate a sequence of j-singular trees is to apply Procedure τ to the only element of R_0 (see Corollary 3.2.2). Thus the $i = 1$ case follows immediately.

Now, suppose Proposition 3.2.3 is true for $i \leq m$. Suppose $S \in R_{m+1}$ is a sequence of j-singular trees. Then, since $R_i = R_i^{(1)} \cup R_i^{(2)}$, S must be an element of $R_{m+1}^{(1)}$ if $S \notin R_{m+1}^{(2)}$. Let S' be an element of R_m that produces S if Procedure σ is applied at leaf l for some $l \leq \ell(m-1)$ (that is, $S = S' \circ_l T_\psi$. Then, since every tree in S is j-singular, the leaf l where T_ψ is inserted must not be one of the $n-1$ level-1 leaves and S' must also be a sequence of j-singular trees. From the inductive hypothesis, $S' \in R_m^{(2)}$. Thus there is an element of R_{m-1} that produces S' after Procedure τ is applied at leaf j. We will call this element S'' (that is, $S' = T_\psi \circ_j S''$).

Thus if W is the word representation of S'', the word representation of S will be $W\tau_j\sigma_l$. Since S (as well as S') must be a sequence of j-singular trees we cannot have $l < j$ or $l > \ell(m) - j$. Thus by Lemma 3.1.3, $W\sigma_{l-(j-1)}\tau_j$ will also be a word representation for S. Since a word representation of S ends in a τ, $S \in R_{m+1}^{(2)}$. Thus by induction, Proposition 3.2.3 is true for all $i \in \mathbb{N}$. $\square$

Combining Propositions 3.2.1 and 3.2.3, we get:

Proposition 3.2.4. *Given a sequence $S \in R_i$, $S \in R_i^{(2)}$ if and only if S is a sequence of j-singular trees for some value of $j \leq n$.*

We will next use these properties of Procedure σ and Procedure τ to enumerate the elements of R_i

3.2.2 Enumerating the elements of R_1

As stated in Corollary 3.2.2, $R_1^{(1)} \cap R_1^{(2)} = \emptyset$.

This means that for $i = 1$, R_i can be partitioned into $i+1 = 2$ mutually exclusive subsets:

- $R_1^{(1)}$, the set of all elements generated by applying Procedure σ to the sole element of R_0. Since the trees in this sequence have $1 + w(n-1)$ leaves, $|R_1^{(1)}| = 1 + w(n-1)$.

- $R_1^{(2)}$, the set of all elements generated by applying Procedure τ to the sole element of R_0. Since the basic n-ary tree has n leaves, $|R_1^{(2)}| = n$.

Thus $|R_1| = 1+w(n-1)+n$. The value predicted by Main Theorem 2 is $\binom{(n-1)(w+i)+i+1}{i} = \binom{(n-1)(w+1)+1+1}{1} = (n-1)(w+1)+1+1 = 1+w(n-1)+n-1+1 = 1+w(n-1)+n$. Thus Main Theorem 2 holds for $i = 1$.

3.3 Further properties and enumerating the elements of R_2

We will next extend the arguments to R_2. We will first need to develop some more complex properties of Procedure σ and Procedure τ, as the partition of R_2 will involve more subsets.

3.3.1 Further properties of Procedure σ and Procedure τ

Firstly, we have a recursive property for $|R_{i+1}^{(2)}|$.

Proposition 3.3.1. *For $i \geq 0$, $|R_{i+1}^{(2)}| = n|R_i|$.*

Proof. Suppose $S \in R_i$. Since there are n different τ's (ways to apply Procedure τ), there are at most n different ways to add a τ to the end of the word representation of S and generate an element of $R_{i+1}^{(2)}$. This means $|R_{i+1}^{(2)}| \leq n|R_i|$.

We must now show that every application of Procedure τ to an element of R_i produces a unique element of R_{i+1}. That is given elements $S_1, S_2 \in R_i$ and natural numbers j_1 and j_2 we must show that an application of Procedure τ to S_1 at leaf j_1 will produce the same sequence as an application of Procedure τ to S_2 at leaf j_2 *only if* $S_1 = S_2$ and $j_1 = j_2$. We will call the former sequence S_1' and the latter S_2'. By definition, an application of Procedure τ at leaf j_1 must produce a sequence of j_1-singular trees. Thus S_1' must be a sequence of j_1-singular trees. Likewise, S_2' must be a sequence of j_2-singular trees. Since $S_1' = S_2'$, we must have $j_1 = j_2$. If we look at the first tree in S_1', we know that it was constructed by grafting the first tree of S_1 to the j_1th leaf of the basic tree T_ψ. Likewise the first tree of S_2' was constructed by grafting the first tree of S_2 to the j_1th leaf of the basic tree T_ψ. The first tree of S_1' must be the same as the first tree of S_2', but this is only possible if the first tree of S_1 is the same as the first tree of S_2. We follow this same reasoning for each of the trees in S_1 and S_2 and conclude that $S_1 = S_2$.

It follows that $|R_{i+1}^{(2)}| = n|R_i|$.

□

This next property will relate to the case where Procedure σ is applied to an element of $R_i^{(2)}$. We will use it to partition R_i based on how many times Procedure σ was applied to the "new" leaves generated by the last application of Procedure τ. This concept will be defined more precisely in the following definitions.

Definition 3.3.1. The set of all elements of R_i that can be generated by a sequence of i applications of Procedure σ will be denoted by $\boldsymbol{R_i^{(1o)}}$, where "1o" is short for "Procedure σ only". We will denote the complement of this set by $\boldsymbol{\overline{R_i}^{(1o)}}$. That is $\overline{R_i}^{(1o)}$ is defined to be the set of all elements of R_i that cannot be generated by applying Procedure σ i times. Procedure τ must have been applied at least once (so any word representation of an element of $\overline{R_i}^{(1o)}$ will have at least one τ).

Definition 3.3.2. Given $1 \leq j \leq n$ a sequence of complete n-ary plane trees will be called

***j*-unique** if each of the trees in the sequence have $n-1$ level-1 subtrees in common but one subtree, the jth, is different for each tree in the sequence. That is the jth level-1 vertex in each tree in S is the root of a subtree which is different for each tree in the sequence but the trees are otherwise identical. In such a sequence, we will refer to the jth level-1 subtree as the **unique level-1 subtree** or simply as the **unique subtree**. The other level-1 subtrees will be referred to as **common (level-1) subtrees**. The subset of R_i consisting of all sequences which are j-unique for some j will be denoted by $\boldsymbol{R_i^{(\mathrm{unique})}}$.

We will show that $\overline{R_i}^{(1o)} = R_i^{(\mathrm{unique})}$.

Lemma 3.3.2. $\overline{R_i}^{(1o)} \subseteq R_i^{(\mathrm{unique})}$

Proof. We first show that the following two statements together are sufficient to prove the lemma:

1. Given an arbitrary $j < i$, $R_j^{(2)} \subseteq R_j^{(\mathrm{unique})}$.

2. An application of Procedure σ to an element of $R_j^{(\mathrm{unique})}$ produces an element of $R_{j+1}^{(\mathrm{unique})}$.

Since every element of $\overline{R_i}^{(1o)}$ is either an element of $R_i^{(2)}$ or is generated by a sequence of $i-j$ applications of Procedure σ to an element of $R_j^{(2)}$, conditions 1 and 2 together would automatically imply that $\overline{R_i}^{(1o)} \subseteq R_i^{(\mathrm{unique})}$.

To show that the first statement is true, we know by Proposition 3.2.4 that all trees in an element of $R_j^{(2)}$ will have $n-1$ level-1 subtrees in common, as these level-1 subtrees are simply the leaves generated by the last application of Procedure τ) (by Proposition 3.2.4). Thus $R_j^{(2)} \subseteq R_j^{(\mathrm{unique})}$ for any positive value of j.

If Procedure σ is applied to an element of $R_j^{(\mathrm{unique})}$, the basic tree will be inserted either within one of the common level-1 subtrees or within the unique level-1 subtree. Since the number of leaves within the unique level-1 subtree is the same for every tree in the sequence, either way the result will be an element of $R_{j+1}^{(\mathrm{unique})}$.

It follows that $\overline{R_i}^{(1o)} \subseteq R_i^{(\mathrm{unique})}$. □

Lemma 3.3.3. $R_i^{(\mathrm{unique})} \subseteq \overline{R_i}^{(1o)}$

Proof. We can show this by induction on i.

Base case: we will show that $R_1^{(\text{unique})} \subseteq \overline{R_1}^{(1o)}$. We will denote the sole element of R_0 (the starting sequence) by S_0. Since $w > 1$, S_0 must contain at least one tree T in which the leftmost level-1 vertex is the only one that has children. S_0 must also contain another tree U in which the second level-1 vertex from the left is the only one that has children. Clearly, T and U differ at two level-1 subtrees so $S_0 \notin R_0^{(\text{unique})}$ (thus $R_0^{(\text{unique})} = \emptyset$). Both T and U have $(n-1)w+1$ leaves. In T, leaves 1 through $(n-1)w+1-(n-1)$ inclusive (that is, all except the rightmost $n-1$ leaves) are descendants of the first (leftmost) level-1 vertex. The second level-1 vertex is leaf $(n-1)w+1-(n-1)+1$. In U, leaf 1 is the first level-1 vertex and leaves 2 through $(n-1)w+1-(n-1)+1$ inclusive (that is, all except the first and the rightmost $n-2$ leaves) are descendants of the second level-1 vertex. We will look at the four different possible cases of applying Procedure σ to S_0 and its effect on T and U.

- If we apply Procedure σ to S_0 at leaf 1, the number of leaves within the first level-1 subtree increases by $n-1$ in both T and in U. We will call these new trees T' and U' respectively. Thus the number of leaves in the first level-1 subtree of the resulting tree T' will still exceed the number of leaves in the first level-1 subtree of U'. Since the second level-1 subtree has not been changed in either tree, the second level-1 subtree in T' still will not have as many leaves as the second level-1 subtree in U'. Thus T' and U' still differ at two level-1 subtrees and we do not get an element of $R_i^{(\text{unique})}$.

- If we apply Procedure σ to S_0 at leaf j, where $2 \leq j \leq (n-1)w+1-(n-1)$, the number of leaves within the first level-1 subtree increases by $n-1$ in T. In U, the number of leaves on the second level-1 subtree will increase by $n-1$. We will (again) call these new trees T' and U' respectively. Thus the number of leaves in the first level-1 subtree of the resulting tree T' will still exceed the number of leaves in the first level-1 subtree of U'. Since the second level-1 subtree has not been changed in T, the second level-1 subtree in T' still will not have as many leaves as the second level-1 subtree in U'. Thus T' and U' still differ at two level-1 subtrees and we do not get an element of $R_i^{(\text{unique})}$.

- If we apply Procedure σ to S_0 at leaf $(n-1)w+1-(n-1)+1$ (the second level-1 vertex of T; a descendant of the second level-1 vertex of U), the number of leaves in the second level-1 subtree increases by $n-1$ in both T and U, but the first level-1 subtree will be unchanged. Thus the two trees still differ at two level-1 subtrees and the sequence produced is not an element of $R_1^{(\text{unique})}$.

- If we apply Procedure σ to S_0 at leaf j, where $(n-1)w+1-(n-1)+2 \leq j \leq (n-1)w+1$, the first two level-1 subtrees are unchanged in both T and U (thus they are different for both trees). The sequence produced is not an element of $R_1^{(\text{unique})}$.

This covers all possibilities for applying Procedure σ to S_0, so we must conclude that the only way to generate an element of $R_1^{(\text{unique})}$ is to apply Procedure τ to S_0. Thus $R_1^{(2)} = \overline{R_1}^{(1o)}$ and $R_i^{(\text{unique})} \subseteq \overline{R_1}^{(1o)}$.

Inductive step: Firstly notice in the base case that no matter where we apply Procedure σ to, the difference between the number of leaves within the first subtree of T' and the first level-1 subtree of U' will always be equal to the difference between the number of leaves within the second level-1 subtree of U' and the second subtree of T'. In S_0 this difference is equal to $(n-1)w+1-(n-1)-1$. Depending on where we apply Procedure σ the four possibilities are (respectively): increase the number of leaves by $n-1$ within the first level-1 subtree in both T and U, increase the number of leaves by $n-1$ within the first level-1 subtree of T and the second level-1 subtree of U, increase the number of leaves by $n-1$ within the second level-1 subtree in both T and U, or increase the number of leaves in a different level-1 subtree and leave the first two unchanged in both T and U. Either both differences remain unchanged, or both increase by $n-1$. For the inductive step then, we will take $S \in R_m$ and $S \notin R_m^{(\text{unique})}$ for some $m > 0$. We will also assume that S has two trees T and U satisfying the following properties (where x, y, and a are arbitrary natural numbers):

	T		U	
	1st level-1 subtree	**2nd level-1 subtree**	**1st level-1 subtree**	**2nd level-1 subtree**
Number of leaves	x	$y-a$	$x-a$	y
Leaf range	1 to x	$x+1$ to $x+y-a$	1 to $x-a$	$x-a+1$ to $x+y-a$

We have defined a to be the difference between the number of leaves within the first subtree of T and the first level-1 subtree of U. We will use T' and U' to denote T and U after Procedure σ has been applied. Note that, as explained above, every element of R_1 that is not an element of $R_1^{(\text{unique})}$ will have two trees that have these properties. Thus if we can prove that no application of Procedure σ to S produces an element of $R_{m+1}^{(\text{unique})}$, and that the difference between the number of leaves within the first subtree of T' and the first level-1 subtree of U' will always be equal to the difference between the number of leaves within the second level-1 subtree of U' and the second subtree of T', it will follow by induction that for all values of i Procedure τ must be applied at least once to generate an element of $R_i^{(\text{unique})}$.

Notice that in both T and U, the total number of leaves in the first two level-1 subtrees is $x+y-a$. If we apply Procedure σ to S at one of the first $x-a$ leaves, we get the following table:

	T'		U'	
	1st level-1 subtree	**2nd level-1 subtree**	**1st level-1 subtree**	**2nd level-1 subtree**
Number of leaves	$x+n-1$	$y-a$	$x-a+n-1$	y
Leaf range	1 to $x+n-1$	$x+n$ to $x+n+y-a-1$	1 to $x-a+n-1$	$x-a+n$ to $x-a+y+n-1$

Thus the number of leaves in the first level-1 subtree of the resulting tree T' will still exceed the number of leaves in the first level-1 subtree of U'. Since the second level-1 subtree has not been changed in either tree, the second level-1 subtree in T' still will not have as many leaves as the second level-1 subtree in U'. Thus the two trees again differ at two level-1 subtrees — we simply replace x with $x+n-1$. Furthermore, the difference between the number of leaves within the first subtree of T' and the first level-1 subtree of U' is equal to the difference between the number of leaves within the second level-1 subtree of U' and the second subtree of T' (both are equal to a), as required.

If we instead apply Procedure σ to S at leaf j, where $x-a+1 \leq j \leq x$, we get the following:

	T'		U'	
	1st level-1 subtree	**2nd level-1 subtree**	**1st level-1 subtree**	**2nd level-1 subtree**
Number of leaves	$x+n-1$	$y-a$	$x-a$	$y+n-1$
Leaf range	1 to $x+n-1$	$x+n$ to $x+y-a+n-1$	1 to $x-a$	$x-a+1$ to $x+y-a+n-1$

Thus the number of leaves in the first level-1 subtree of the resulting tree T' will still exceed the number of leaves in the first level-1 subtree of U'. Likewise the number of leaves in the second level-1 subtree of the resulting tree U' will still exceed the number of leaves in the second level-1 subtree of T'. The two trees again differ at two level-1 subtrees — we simply replace a with $a + n - 1$. Furthermore, the difference between the number of leaves within the first subtree of T' and the first level-1 subtree of U' is equal to the difference between the number of leaves within the second level-1 subtree of U' and the second subtree of T' (both are equal to $a + n - 1$), as required.

If we instead apply Procedure σ to S at leaf j, where $x + 1 \leq j \leq x + y - a$, we get the following:

	T'		U'	
	1st level-1 subtree	**2nd level-1 subtree**	**1st level-1 subtree**	**2nd level-1 subtree**
Number of leaves	x	$y-a+n-1$	$x-a$	$y+n-1$
Leaf range	1 to x	$x+1$ to $x+y-a+n-1$	1 to $x-a$	$x-a+1$ to $x+y-a+n-1$

Thus the number of leaves in the second level-1 subtree of the resulting tree U' will still exceed the number of leaves in the second level-1 subtree of T'. Since the first level-1 subtree has not been changed in either tree, the first level-1 subtree in U' still will not have as many leaves as the first level-1 subtree in T'. Thus the two trees again differ at two level-1 subtrees — we simply replace y with $y + n - 1$. Furthermore, the difference between the number of leaves within the first subtree of T' and the first level-1 subtree of U' is equal to the difference between the number of leaves within the second level-1 subtree of U' and the second subtree of T' (both are equal to a), as required.

Applying Procedure σ to S at leaf j, where $j > x + y - a$, does not add leaves to either the first or the second level-1 subtree. Thus T' and U' still differ at two different subtrees, and the difference between the number of leaves within the first level-1 subtree of T' and the first level-1 subtree of U' is the same as the difference between the number of leaves within the second level-1 subtree of U' and the second level-1 subtree of T'.

This covers all possibilities for applying Procedure σ to S. Combined with the base case, it follows that repeated applications of Procedure σ to S_0 cannot produce an element of $R_i^{(\text{unique})}$. □

Combining these two lemmas, we obtain:

Proposition 3.3.4. $\overline{R_i}^{(1o)} = R_i^{(\text{unique})}$.

Two major consequences immediately follow from this result. The first will be used to develop our partition of R_i and the recurrence relation that will be introduced in the next section.

Corollary 3.3.5. *For each element $S \in \overline{R_i}^{(1o)}$ there exists a unique ordered triple (j, F_k, S_k), where:*

- $1 \leq j \leq n$.
- $0 \leq k \leq i - 1$.
- τ_j *is the rightmost* τ *in any word representation of* S.
- F_k *is a plane forest with* $n - 1$ *complete* n*-ary trees and a total of* k *internal nodes.*
- $S_k \in R_{i-k-1}$.

Because each element of $\overline{R_i}^{(1o)}$ has a specified value of k, we can partition $\overline{R_i}^{(1o)}$ based on this value. There are i possible values k can take, so such a partition will divide $\overline{R_i}^{(1o)}$ into i mutually exclusive subsets.

Proof. By Proposition 3.3.4, an element of R_i will be an element of $\overline{R_i}^{(1o)}$ if and only if it's an element of $R_i^{(\text{unique})}$ — that is, a j-unique sequence for some value of j. This j-value will be the last application of Procedure τ and therefore τ_j must be the rightmost τ in any word representation of S. We construct the plane forest F_k by considering (without loss of generality) the first tree in S, which we will call T_1. We remove the unique level-1 subtree from T_1 (i.e. the jth) as well as the root and all edges joining the root to a level-1 vertex. This results in the plane forest F_k. Note that because S is j-unique, the same forest results regardless of which tree in S is used to generate it. The subscript k denotes the number of internal nodes in the forest. Equivalently, k is the total number of internal nodes in the common level-1 subtrees of T_1 or the number of σ's to the right of τ_j in a word representation of S where τ_j is as far to the right as possible. We will denote this word representation by W. S_k is defined to be the sequence of unique level-1 subtrees, one from each tree in S. More precisely, given $1 \leq m \leq l_w$, the mth tree in S_k will be the unique level-1 subtree of the mth tree in S. Because W is of length i and there are k σ's to the right of τ_j, the first $i-k-1$ letters of W are a word representation of S_k. Therefore $S_k \in R_{i-k-1}$. $\square$

Corollary 3.3.5 can be understood more easily through an example.

Example 3.3.3. Suppose $w = n = 2$ and $i = 4$. Consider the sequence $S \in R_4$ given in Figure 3.8. We can see that the first (leftmost) level-1 subtree is the same for both trees in the sequence, but the second (rightmost) differs. Thus S is a 2-unique sequence, $S \in R_4^{(\text{unique})}$, and $S \in \overline{R_4}^{(1o)}$. We can conclude that the last application of Procedure τ was at the second leaf (so the rightmost τ in any word representation of S would be τ_2). The common level-1 subtree (the first) has a total of two internal nodes, so a word representation of S with τ_2 as far to the right as possible will end in $\tau_2\sigma\sigma$ (in this case, it would end with $\tau_2\sigma_1\sigma_2$). Since there is only one common level-1 subtree, the forest F_2 consists of only one tree (the leftmost level-1 subtree in either tree in S).

We will next define a partition of R_i based on Corollary 3.3.5.

Definition 3.3.4. We have already defined $R_i^{(1o)}$ and $\overline{R_i}^{(1o)}$, the latter being equivalent to $R_i^{(\text{unique})}$. We will use the notation $\boldsymbol{R_i^{(\text{unique},\,k)}}$ (where $0 \leq k \leq i-1$) to denote the subset of

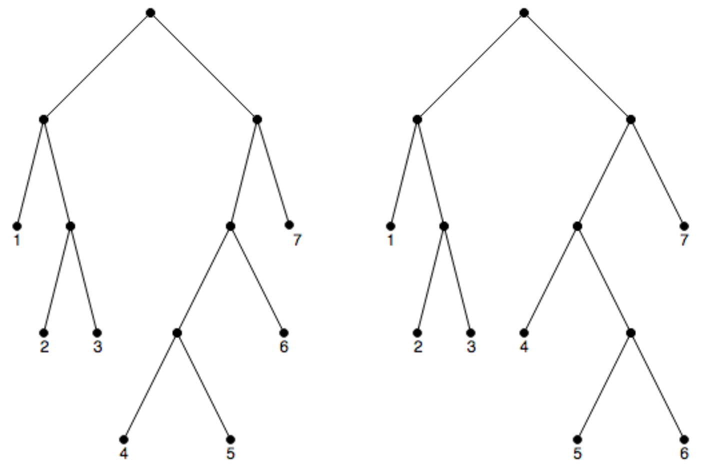


Figure 3.8: Example 3.3.3.

$R_i^{(\text{unique})}$ in which the common level-1 subtrees have a total of k internal nodes. Thus the sets $R_i^{(1o)}$, $R_i^{(\text{unique},0)}$, $R_i^{(\text{unique},1)}$, $R_i^{(\text{unique},2)}$, $\ldots$, $R_i^{(\text{unique},i-1)}$ will be a partition of R_i.

Our next corollary will ultimately help us develop an algorithm for directly generating R_i (that is, without generating duplicate elements).

Corollary 3.3.6. *For each element $S \in R_i$ there exists a unique ordered pair (s, F), where s is a sequence of τ's of length L_τ (where $0 \le L_\tau \le i$) and F is a plane forest of $1 + w(n-1) + L_\tau(n-1)$ complete n-ary trees with a total of $i - L_\tau$ internal nodes.*

Proof. As in the proof for Corollary 3.3.5 we can tell whether or not Procedure τ has ever been used to generate S. If indeed $S \in \overline{R_i}^{(1o)}$, then there exists j_1 (as in the statement of Corollary 3.3.5) such that S is a j_1-unique sequence. τ_{j_1} will be the rightmost τ in any word representation of S.

From S we will form a new sequence of trees, denoted by S_2, that consists of the unique level-1 subtrees of each tree in S. S_2 is necessarily an element of R_k for some $k \ge 0$. If $S_2 \in \overline{R_k}^{(1o)}$, then Corollary 3.3.5 applies again and there exists j_2 such that the j_2th level-1 subtree is the unique one and τ_{j_2} is the rightmost τ in any word representation of S_2. We can form a new sequence of trees denoted by S_3, that consists of the unique level-1 subtrees of each tree in S_2, and repeat the process until we get a sequence of trees S_{x+1} such that

$S_{x+1} \in R_m^{(1o)}$ for some $m \geq 0$. Thus any word representation of S will have x τ's, with their subscripts determined by where Procedure τ was applied. In the case where $S \in R_m^{(1o)}$, we have $x = 0$ so the sequence of $\tau's$ is empty. Thus $L_\tau = x$.

From Lemma 3.1.3, there exists a word representation of S in which all of the τ's are at the beginning. Given this sequence of τ's (which will be empty if $L_\tau = 0$ and $S \in R_i^{(1o)}$), we obtain the forest as follows. We first apply the τ sequence to the sole element of R_0 to obtain a sequence of trees, which we will call $\widetilde{S}$ (if $L_\tau = 0$ then $\widetilde{S} = S$). We will denote these sequences by $S = (T_1, T_2, \ldots, T_{l_w})$ and $\widetilde{S} = (\widetilde{T}_1, \widetilde{T}_2, \ldots, \widetilde{T}_{l_w})$. For a tree T_p in S (where $1 \leq p \leq l_w$), the corresponding tree $\widetilde{T_p}$ in $\widetilde{S}$ is a plane subtree of T_p with the same root as T_p. Hence corresponding to each leaf vertex v of $\widetilde{T_p}$ there will be a subtree of T_p whose root is v - call this subtree $T_p(v)$. Labelling the leaves of $\widetilde{T_p}$ lexicographically and according to their position in the plane (the way we would with a pre-order depth-first labelling), we obtain a corresponding sequence of trees $T_p(v_1), T_p(v_2), \ldots, T_p(v_{\ell(L_\tau)})$. This is the required plane forest. Note that the subtrees $T_p(v)$ must have resulted from applications of Procedure σ (since they are not part of $\widetilde{T_p}$) and hence every tree in S yields the same forest.

□

This will again be made more clear with an example.

Example 3.3.5. Suppose $w = n = 2$ and $i = 4$. Consider the sequence $S \in R_4$ given in Figure 3.9. We will determine the sequence of τ's that any given word representation of S will have.

1. We first note that the first (leftmost) level-1 subtree is the same for both trees in S. Thus we conclude that the last (rightmost) τ is τ_2, since the unique level-1 subtree is the second.

2. We next take the second level-1 subtree from each tree in S and form the new sequence S_2 (Figure 3.10). We see that the second (rightmost) level-1 subtree is the same for both trees in S_2, so the second last τ is τ_1 as the unique level-1 subtree is the first.

3. We take the first level-1 subtree from each tree in S_2 and form the new sequence S_3 (Figure 3.11). We see that the first level-1 subtree in the first tree of S_3 is not the same

as the first level-1 subtree in the second tree of S_3. Likewise the second level-1 subtree in the first tree of S_3 is not the same as the second level-1 subtree in the second tree of S_3. Thus $S_3 \notin \overline{R_k}^{(1o)}$ for any $k \geq 0$ (in this case $S_3 \in R_0$ but $S_3 \notin \overline{R_i}^{(1o)}$, which is empty).

Thus every word representation of S will contain a τ_1 somewhere in the word, with a τ_2 somewhere to the right of the τ_1.

For the forest F, we compare the original sequence of trees S with the sequence $\widetilde{S}$, which we obtain by applying the τ sequence to the sole element of R_0 (Figure 3.12). Without loss of generality we compare the first tree in S (denoted by T_1) with the first tree in $\widetilde{S}$ (denoted by $\widetilde{T_1}$). Each leaf of $\widetilde{T_1}$ corresponds to the root of a subtree of T_1 (Figure 3.9). F consists of these subtrees. Since $\widetilde{T_1}$ has five leaves, F will consist of the following five trees:

1. A binary tree with two internal nodes (the root and its right child). This corresponds to leaves 1, 2, and 3 in T_1 (and the two internal nodes along the shortest path connecting these three leaf vertices).

2. A single vertex. This corresponds to leaf 4 in T_1.

3. A single vertex. This corresponds to leaf 5 in T_1.

4. A single vertex. This corresponds to leaf 6 in T_1.

5. A single vertex. This corresponds to leaf 7 in T_1.

Note that F will be the same if we use T_2 and $\widetilde{T_2}$ instead of T_1.

To provide an example of the partitioning strategy, we will use it to enumerate the elements of R_2.

3.3.2 Enumerating the elements of R_2

By Definition 3.3.4, we can partition R_2 into $2 + 1 = 3$ different classes:

1. Elements of $R_2^{(1o)}$. Every word representation consists of two σ's. This can be broken up into two sub-cases:

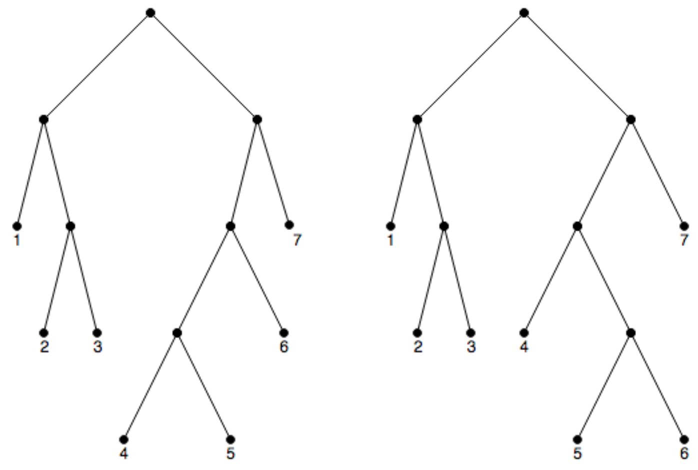


Figure 3.9: Example 3.3.5, step 1.

(a) Starting from the sole element R_0, choose a pair of leaves on each tree (ie the first and third leaf on every tree in the original sequence) and apply Procedure σ at both of these leaves. There are $\binom{1+w(n-1)}{2}$ elements of this type.

(b) Starting from the sole element of R_0, apply Procedure σ at leaf j ($1 \leq j \leq 1 + w(n-1)$). Then apply Procedure σ to one of the new leaves generated in the first step (leaves j through $j+n-1$). There are $[1+w(n-1)]n$ elements of this type.

2. Elements of $R_2^{(\text{unique},0)}$ (equivalent to $R_2^{(2)}$). By Proposition 3.3.1, $|R_2^{(2)}| = n|R_1| = n[1+w(n-1)] + n^2$. These elements of R_2 have a word representation that ends with a τ.

3. Elements of $R_2^{(\text{unique},1)}$. That is, the word representation has exactly one τ and the common level-1 subtrees have one internal node between them (so one of them has an internal node, while the other $n-2$ common level-1 subtrees are just leaf nodes). The word representation will be of the form $\tau_j\sigma_k$, where $1 \leq j \leq n$ and $k < j$ or $k > j + w(n-1)$. There are $n(n-1)$ possible ways to perform this sequence of procedures.

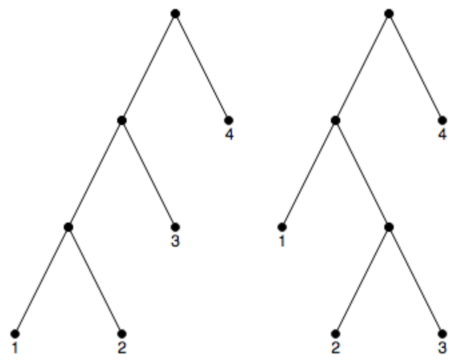


Figure 3.10: Example 3.3.5, step 2.

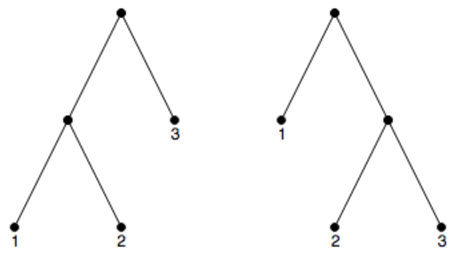


Figure 3.11: Example 3.3.5, step 3.

Summing these terms, we get $n[1+w(n-1)]+n^2+n(n-1)+\binom{1+w(n-1)}{2}+n[1+w(n-1)]$, which can be shown to be equal to $\binom{(n-1)(w+2)+2+1}{2}$ through expansions. This is the value predicted by Main Theorem 2.

3.4 Enumerating the elements of R_i

Definition 3.3.4 allows us to partition R_i into $i+1$ mutually exclusive classes. We will use this partition to set up a recurrence relation for enumerating the elements of R_i. For the enumeration, we will need to set up two bijections: one for $\overline{R_i}^{(1o)}$, and one for $R_i^{(1o)}$.

Proposition 3.4.1. *Let k be a fixed integer (where $0 \leq k \leq i-1$) and let $\mathcal{T}_i^{(k)}$ be the set of*

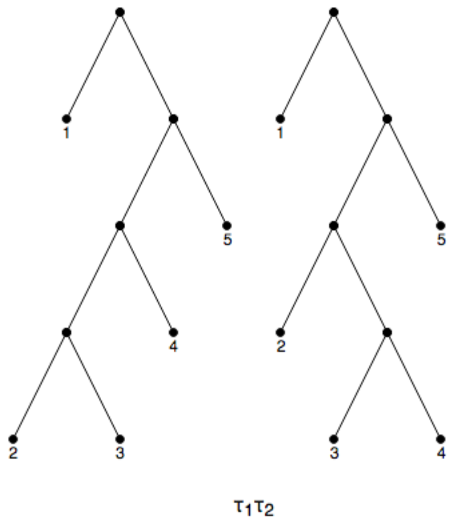


Figure 3.12: Example 3.3.5, $\widetilde{S}$.

ordered triples (j, F_k, S_k)*, where:*

- $1 \leq j \leq n$.
- τ_j *is the rightmost* τ *in any word representation of* S.
- F_k *is a plane forest with* $n-1$ *complete* n*-ary trees and a total of* k *internal nodes.*
- $S_k \in R_{i-k-1}$.

Then there is a bijection between $\mathcal{T}_i^{(k)}$ *and* $R_i^{(\text{unique},\,k)}$.

Proof. By Corollary 3.3.5, for every element of $R_i^{(\text{unique},\,k)}$ there exists a unique element of $\mathcal{T}_i^{(k)}$. Thus to show a bijection, we must show that for each element of $\mathcal{T}_i^{(k)}$ there exists a unique element of $R_i^{(\text{unique},\,k)}$. Given (j, F_k, S_k) we begin by applying Procedure τ to S_k at leaf j. This gives us an element of $R_{i-k}^{(2)}$, which we will denote by S_k'. We can write a word representation of this sequence by writing τ_j at the end of any word representation of S_k. We will use F_k to extend this word so that we get an element of $R_i^{(\text{unique},\,k)}$. First, we draw a forest of $n-1$ single-node trees which we will denote by F_k'. We then label the first $j-1$ leaf vertices in F_k' sequentially from left to right. For leaf vertices j through n, we assign the labels $\ell(i-k)-(n-j)$ through $\ell(i-k)$ (this matches the level-1 leaves in the sequence of trees S_k'). We then apply the following recursive algorithm:

1. Simultaneously traverse both F'_k and F_k through pre-order depth-first traversal, starting from the leftmost tree and moving rightward, until we reach a vertex which is a leaf node in F' but an internal node in F_k. Suppose this is the mth leaf node in F'_k. We replace this leaf node with the basic n-ary tree. We write σ_m at the end of the word (since this represents an application of Procedure σ, then add $n-1$ to the labels of all leaves in F' with labels greater than m. The leaves of the basic tree that were just added are labelled m through $m+n-1$.

2. Repeat step 1 until $F'_k = F_k$. The final word will represent an element of $R_i^{(\text{unique}, k)}$.

□

This result allows us to use the multiplication rule to enumerate the elements of $R_i^{(\text{unique}, k)}$, proving the following:

Proposition 3.4.2. $|R_i^{(\text{unique}, k)}| = n\, F(k, n-1; n)\, |R_{i-k-1}|$.

The following bijection will build on Corollary 3.3.6. It will allow us to enumerate the elements of $R_i^{(1o)}$.

Proposition 3.4.3. *Let $\mathcal{P}_i$ be the set of ordered pairs (s, F), where s is a sequence of τ's of length L_τ (where $0 \le L_\tau \le i$) and F is a plane forest of $1 + (w + L_\tau)(n-1)$ complete n-ary trees with a total of $i - L_\tau$ internal nodes. Then there is a bijection between $\mathcal{P}_i$ and R_i.*

Proof. By Corollary 3.3.6, for every element of R_i there exists a unique element of $\mathcal{P}_i$. Thus to show a bijection, we must show that for each element of $\mathcal{P}_i$ there exists a unique element of R_i. Given (s, F) we first apply the sequence of iterations of Procedure τ given in s to the sole element of R_0. We will call this sequence of trees $\widetilde{S}$. Each tree in $\widetilde{S}$ has $1 + (w + L_\tau)(n-1)$ leaves. We will use F to extend s so we get a word representation of an element of R_i. First, we draw a forest of $1 + (w + L_\tau)(n-1)$ single-node trees which we will denote by F'. Then, we follow this recursive algorithm:

1. Label the leaf nodes of F' through pre-order depth-first traversal starting from the leftmost tree and moving rightward.

2. Simultaneously traverse both F and F' through pre-order depth-first traversal, starting from the leftmost tree and moving rightward, until we reach a vertex which is a leaf node in F' but an internal node in F. Suppose this is the mth leaf node in F'. We replace this leaf node with the basic n-ary tree. We write σ_m at the end of the word, then go back to the previous step.

Once $F = F'$, the algorithm ends and we have a word representation of an element of R_i corresponding to F. Thus there is a bijection between $\mathcal{P}_i$ and R_i. $\square$

This result proves the following:

Proposition 3.4.4. $|R_i^{(1o)}| = F(i, 1 + w(n-1); n)$.

Combining Proposition 3.4.2 with Proposition 3.4.4 we get the following recurrence relation:

Proposition 3.4.5. $|R_0| = 1$. *For $i > 0$, the following equation holds:*

$$\begin{aligned}|R_i| &= |R_i^{(1o)}| + \sum_{k=0}^{i-1} |R_i^{(\text{unique}, k)}| \\ &= F(i, 1 + w(n-1); n) + \sum_{k=0}^{i-1} n|R_{i-k-1}|F(k, n-1; n) \\ &= \frac{1 + w(n-1)}{ni + 1 + w(n-1)} \binom{ni + 1 + w(n-1)}{i} + \sum_{k=0}^{i-1} n|R_{i-k-1}| \frac{n-1}{nk + n - 1} \binom{nk + n - 1}{k}.\end{aligned} \tag{3.2}$$

Note that the initial condition ($|R_0| = 1$) implies that any solution to this recurrence relation must be unique. This immediately follows by induction on i. So if the conjectured value of $|R_i|$ satisfies the recurrence relation, Main Theorem 2 is proven automatically. We will show this using the Hagen-Rothe identity (Lemma 2.4.1).

Proposition 3.4.6. *The conjectured value of $|R_i|$ satisfies the recurrence relation in Proposition 3.4.5.*

Proof. From Main Theorem 2, $|R_i| = \binom{i+(w+i)(n-1)+1}{i} = \binom{wn+in-w+1}{i}$. Substituting this into the right side of 3.2:

$$\frac{1+w(n-1)}{ni+1+w(n-1)} \binom{ni+1+w(n-1)}{i} + \sum_{k=0}^{i-1} n|R_{i-k-1}| \frac{n-1}{nk+n-1} \binom{nk+n-1}{k}$$

$$
\begin{aligned}
&= \frac{1+w(n-1)}{ni+1+w(n-1)}\binom{ni+1+w(n-1)}{i}+n\sum_{k=0}^{i-1}\binom{wn+in-nk-n-w+1}{i-k-1}\frac{n-1}{nk+n-1}\binom{nk+n-1}{k}\\
&= \frac{1+w(n-1)}{ni+1+w(n-1)}\binom{ni+1+w(n-1)}{i}+n\sum_{k=0}^{i-1}\frac{n-1}{nk+n-1}\binom{nk+n-1}{k}\binom{wn+in-nk-n-w+1}{i-k-1}\\
&= \frac{1+w(n-1)}{ni+1+w(n-1)}\binom{ni+1+w(n-1)}{i}+n\binom{(n-1)+(wn+in-n-w+1)}{i-1}
\end{aligned}
$$

(by Lemma 2.4.1 with $L = i-1$, $p = n-1$, $q = n$, and $r = wn+in-n-w+1$)

$$
\begin{aligned}
&= \frac{1+w(n-1)}{ni+1+w(n-1)}\binom{ni+1+w(n-1)}{i}+n\binom{wn+in-w)}{i-1}\\
&= \frac{1+w(n-1)}{ni+1+w(n-1)}\binom{ni+1+w(n-1)}{i}+\frac{n(wn+in-w)!}{(i-1)!(wn+in-w-i+1)!}\\
&= \frac{1+w(n-1)}{ni+1+w(n-1)}\binom{ni+1+w(n-1)}{i}+\frac{n(wn+in-w)!}{(i-1)!(wn+in-w-i+1)!}\cdot\frac{i}{i}\cdot\frac{(wn+in-w+1)}{(wn+in-w+1)}\\
&= \frac{1+w(n-1)}{ni+1+w(n-1)}\binom{ni+1+w(n-1)}{i}+\frac{ni(wn+in-w+1)!}{i!(wn+in-w-i+1)!(wn+in-w+1)}\\
&= \frac{1+w(n-1)}{ni+1+w(n-1)}\binom{ni+1+w(n-1)}{i}+\frac{ni}{wn+in-w+1}\binom{ni+1+w(n-1)}{i}\\
&= \frac{1+w(n-1)}{ni+1+w(n-1)}\binom{ni+1+w(n-1)}{i}+\frac{ni}{ni+1+w(n-1)}\binom{ni+1+w(n-1)}{i}\\
&= \binom{ni+1+w(n-1)}{i}\frac{1+w(n-1)+ni}{ni+1+w(n-1)} = \binom{ni+1+w(n-1)}{i}.
\end{aligned}
$$

Thus Main Theorem 2 holds for all $w > 1$. □

3.5 An algorithm for generating R_i

We will end this chapter by using Proposition 3.4.3 to derive an alternate fomula for $|R_i|$. This formula will provide a straightforward algorithm for generating R_i for a given value of i. We will show that the formula satisfies the recurrence relation in Proposition 3.4.5, which proves that it's equivalent to the conjectured value of $|R_i|$.

Firstly we know from Corollary 3.3.6 that given an element $S \in R_i$, the sequence of τ's will be the same for every word representation for that element. This sequence can be moved to the beginning of a word representation of S. Secondly we know from Proposition 3.4.3 that there is a bijection between R_i and the set of ordered pairs of the form (s, F), where s is a sequence of τ's of length L (where $0 \leq L \leq i$) and F is a plane forest of $1 + (w + L_\tau)(n - 1)$ complete n-ary trees with a total of $i - L_\tau$ internal nodes.

Thus each element $S \in R_i$ consists of two components, each of which can easily be generated:

1. A sequence of τ's of length L, where $0 \leq L \leq i$. This corresponds to L applications of Procedure τ.

2. A plane forest of $1 + (w + L)(n - 1)$ complete n-ary trees with a total of $i - L$ internal nodes (for some $0 \leq L \leq i$).

Thus we have the following result:

Theorem 3.5.1.

$$|R_i| = \sum_{L=0}^{i} F(i - L, 1 + (w + L)(n - 1); n)n^L.$$

We will lastly verify that this formula for $|R_i|$ satisfies the recurrence relation given in Proposition 3.4.5.

Proposition 3.5.2. *The formula for $|R_i|$ given in Theorem 3.5.1 satisfies the recurrence recurrence relation given in Proposition 3.4.5.*

Proof. For $i = 0$, we have:

$$|R_i| = \sum_{L=0}^{0} F(0 - L, 1 + (w + L)(n - 1); n)n^L = F(0, 1 + w(n - 1); n)n^0 = 1.$$

Thus the initial condition holds. For $i > 0$,

$$\begin{aligned} |R_i| &= F(i, 1{+}w(n{-}1); n) + \sum_{k=0}^{i-1} n|R_{i-k-1}|F(k, n{-}1; n) \\ &= F(i, 1{+}w(n{-}1); n) + \sum_{k=0}^{i-1} n \cdot \sum_{L=0}^{i-k-1} F(i{-}k{-}1{-}L, 1{+}(w{+}L)(n{-}1); n)n^L F(k, n{-}1; n) \end{aligned}$$

$$
\begin{aligned}
&= F(i,1{+}w(n{-}1);n){+}\sum_{k=0}^{i-1}\sum_{L=0}^{i-k-1} F(i{-}k{-}1{-}L,1{+}(w{+}L)(n{-}1);n)n^{L+1}F(k,n{-}1;n)\\
&= F(i,1{+}w(n{-}1);n){+}\sum_{k=0}^{i-1}\sum_{L=0}^{i-k-1} F(i{-}k{-}1{-}L,1{+}(w{+}L)(n{-}1);n)n^{L+1}F(k,n{-}1;n)\\
&= F(i,1{+}w(n{-}1);n){+}\sum_{L=0}^{i-1}\sum_{k=0}^{i-L-1} F(i{-}k{-}1{-}L,1{+}(w{+}L)(n{-}1);n)n^{L+1}F(k,n{-}1;n)\\
&= F(i,1{+}w(n{-}1);n){+}\sum_{L=0}^{i-1}n^{L+1}\sum_{k=0}^{i-L-1} F(i{-}L{-}1{-}k,1{+}(w{+}L)(n{-}1);n)F(k,n{-}1;n)\\
&= F(i,1{+}w(n{-}1);n){+}\sum_{L=0}^{i-1}n^{L+1}F(i{-}(L{+}1),1{+}(w{+}L{+}1)(n{-}1);n)
\end{aligned}
$$

(combining the two forests with a fixed number of total internal nodes into a single forest)

$$
\begin{aligned}
&= F(i,1{+}w(n{-}1);n){+}\sum_{L=1}^{i}n^{L}F(i{-}L,1{+}(w{+}L)(n{-}1);n)\\
&= \sum_{L=0}^{i}n^{L}F(i{-}L,1{+}(w{+}L)(n{-}1);n)
\end{aligned}
$$

This is the formula for $|R_i|$ given in Theorem 3.5.1. □

3.6 Summary

We developed properties of Procedure σ and Procedure τ that allowed us to partition R_i into $i+1$ mutually exclusive subsets. We used the properties of forests to enumerate these subsets and develop a recurrence relation for $|R_i|$. We were then able to show that the conjectured value of $|R_i|$ satisfies the recurrence relation, thus proving Main Theorem 2 for the general case. These properties also helped us to develop an alternative formula for $|R_i|$ and an algorithm for generating elements of R_i. We also introduced new notation for Procedure σ and Procedure τ that will allow us to write elements of R_i in a more compact way. This notation will be particularly useful in the following chapter, in which we will provide an additional proof for Main Theorem 2 using the occupancy problem.

Chapter 4

Rephrasing elements of R_i as words: an additional proof of Main Theorem 2

In this chapter, we will demonstrate an additional proof of Main Theorem 2. It will again be assumed throughout that, unless otherwise stated, n and w are fixed constants (with $n > 1$) so we have fixed $R_0 = \{(X_1, X_2, X_3, \ldots, X_{l_w})\}$. As such, the definitions that follow will be dependent on the values of n and w. Recall from Section 3.1, each element of R_i can be defined as a sequence of procedures, which can then be denoted as a word over the alphabet $A_i = \{\tau_1, \tau_2, \tau_3, \ldots, \tau_n, \sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_{\ell(i-1)}\}$, where $\ell(i-1)$ is the number of leaves that every tree in an element of R_i has (see Definition 3.1.2)[1]. Since n and w are fixed, we have fixed $R_0 = \{(X_1, X_2, X_3, \ldots, X_{l_w})\}$. Enumerating the elements of R_i, then, becomes a matter of finding the number of (non-redundant) words that represent elements of R_i. The method we present will be original. However some standard results, as well as material from the previous chapter, will be applied where necessary. To eliminate redundant words, we will define a standard form for valid words over A_i. We will then develop a bijection from the set of standard-form valid words to another set, one that can easily be enumerated through the occupancy problem. The proof in this chapter will also lead to a straightforward algorithm for generating R_i.

4.1 Equivalent words and standard form

Recall from Definition 3.1.4 in Section 3.1.1 that a word is said to be A_i-valid (hereafter valid) if it represents an element of R_i. We used the set $V(A_i)$ to represent all valid words.

[1]Adapted from unpublished paper by Nick Beaton.

Recall that a τ can be placed anywhere in a valid word, but we can place σ_j at position k if and only if $j \leq \ell(k-1)$. In Lemmas 3.1.2 and 3.1.3, we proved some properties related to equivalent words over A_i - that is, words that represent the same element of R_i.

To enumerate R_i, then, we must enumerate the valid words of length i (over A_i) while eliminating words that are redundant (that is, equivalent to one another). We will do this by defining a standard form of words over A_i. This will naturally lead to an enumeration problem on valid words in standard form, which will be of equal cardinality to R_i. We will solve this problem by constructing a bijection to a set of words that is easier to enumerate.

Definition 4.1.1. A word of length i over A_i is said to be in **standard form** if the τ's are written before the σ's, and the indices on the σ's are written in non-decreasing order. We denote the set of all such words by $\boldsymbol{S(A_i)}$.

Remark 4.1.1. Standard form does not require the τ's to be in any particular order, and Figure 4.1[2] illustrates that $\tau_1\tau_2 \neq \tau_2\tau_1$. Furthermore, a word in standard form is not necessarily valid. $\tau_1\sigma_5\sigma_5\sigma_6$ is in standard form, but is not valid for $n = w = 2$.

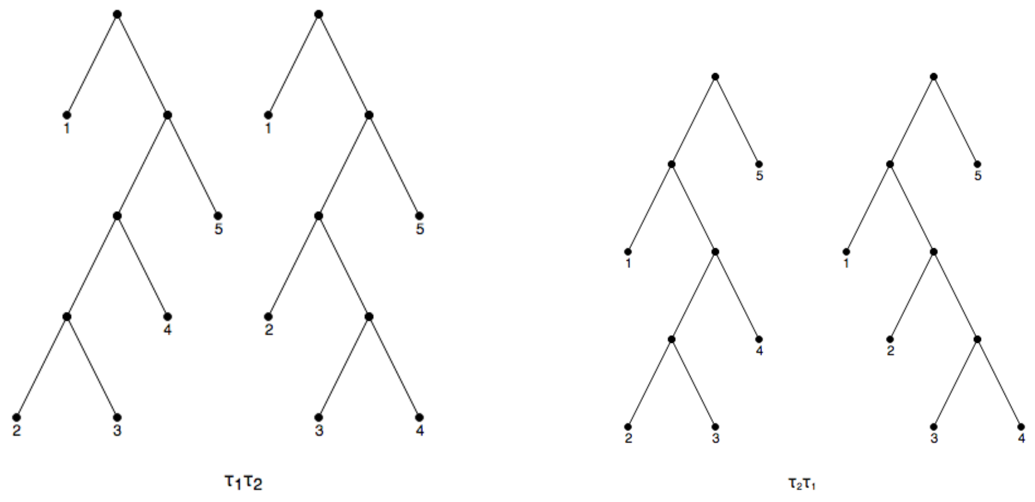


Figure 4.1: $\tau_1\tau_2 \neq \tau_2\tau_1$ ($n = 2$, $w = 2$).

Definition 4.1.2. $\boldsymbol{SV(A_i)} = S(A_i) \cap V(A_i)$.

We will now show that there is a bijection between $SV(A_i)$ and R_i.

[2]Figures created with Draw.io, `https://www.draw.io/`.

Lemma 4.1.1 (Surjection)**.** *Let $W \in V(A_i)$. Then, there exists a word W' such that $W' \in S(A_i)$ and $W \cong W'$.*

Proof. We can apply Lemma 3.1.3 repeatedly so that all of the τ's are before the σ's. Lemma 3.1.2 can then be applied to rearrange the σ's so that their subscripts are in non-decreasing order. □

Thus every element of R_i has at least one word representation that is in standard form.

Lemma 4.1.2 (Injection)**.** *Let $W \in SV(A_i)$. Then, no other word over A_i can be both equivalent to W and in standard form.*

Proof. Corollary 3.3.6 implies that any standard-form word equivalent to W must begin with the same sequence of τ's. To construct the sequence of trees represented by W, Procedure τ (Procedure 1.2.2) is first performed as indicated by the τ's.

Note that the leaves of the trees in any given element of R_i are labelled through pre-order depth-first traversal (Definition 2.1.2). When Procedure σ (Procedure 1.2.1) is applied at a given leaf, that leaf becomes an internal node and the labels on subsequent leaves are increased by $n - 1$. When the subscripts on the σ's are in non-decreasing order (as in a standard-form word), this therefore implies that the trees are constructed and traversed in the same way (pre-order depth-first traversal) as we apply Procedure σ according to the σ's in W. Since a tree has only one pre-order depth-first traversal, there must be only one sequence of σ's in increasing order that can correspond to that element and therefore no word can be both equivalent to W and in standard form. □

Thus every element of R_i has exactly one standard form word representation. This one-to-one correspondence gives the following result.

Proposition 4.1.3. $|SV(A_i)| = |R_i|$.

We have now reframed the problem of enumerating elements of R_i (sequences of trees) to a problem of enumerating words, now that we have a well-defined set $SV(A_i)$ which is of equal cardinality to R_i. We will next describe a strategy for enumerating the elements of $SV(A_i)$.

4.2 A strategy for enumerating the elements of $SV(A_i)$

A method for direct enumeration of the elements of $SV(A_i)$ is not immediately obvious. However, it is possible to set up a bijection between $SV(A_i)$ and a set of words that is much easier to enumerate.

Definition 4.2.1. Let $\boldsymbol{U(A_i)}$ be the set of elements of $S(A_i)$, with the additional condition that the indices on the τ's are in non-decreasing order.

$U(A_i)$ excludes some elements of $SV(A_i)$ ($\tau_2\tau_1$ where $i = 2$, for example). However there are elements of $U(A_i)$ that are not in $SV(A_i)$, as there is no requirement for the words in $U(A_i)$ to be valid. If we set $n = i = w = 2$ for instance, we get $\sigma_4\sigma_4 \in U(A_i)$ but $\sigma_4\sigma_4 \notin SV(A_i)$. Equivalently, the following four conditions define $SV(A_i)$ and $U(A_i)$:

1. σ_j can only be placed at the kth position if $j \leq \ell(k-1)$.

2. All of the τ's occur before the σ's.

3. The indices on the σ's are in non-decreasing order.

4. The indices on the τ's are in non-decreasing order.

Conditions 1-3 define $SV(A_i)$, while conditions 2-4 define $U(A_i)$.

We will first show that $|U(A_i)|$ is the conjectured value of $|R_i|$. We will then establish a bijection between $U(A_i)$ and $SV(A_i)$.

4.2.1 Enumerating the elements of $U(A_i)$: the occupancy problem

Notice that conditions 2-4 imply that calculating $|U(A_i)|$ is equivalent to finding the number of words of length i over an alphabet of size $|A_i|$ with the only restriction that the letters must be arranged in alphabetical order. Thus the only information we need to construct a word is the number of each type of letter. This can be modelled as an **occupancy problem**, the combinatorial problem of determining the number of ways in which a given number of balls can be sorted into a given number of bins. There are many variants of the occupancy problem - the balls and the bins may be identical or distinguishable, and empty bins may or may not be allowed [17]. Calculating $|U(A_i)|$ is equivalent to the version in which bins are distinguishable but balls are not. Empty bins are allowed, and there are no other constraints. The length of the word (i) is the number of balls, and each letter type corresponds to a bin. Thus there are $|A_i|$ total bins. The solution to this occupancy problem is a standard result [17], though its proof will be reviewed here for the sake of completeness.

Proposition 4.2.1 ([17]). *There are $\binom{i+|A_i|-1}{i}$ ways to sort i indistinguishable balls into $|A_i|$ distinguishable bins.*

Proof. We can represent each possibility by using a dot to represent each ball and a vertical line to represent each boundary between two bins [17]. For example if we have $i = 2$, $|A_i| = 3$, the possible sortings are as given in Table 4.1[3].

Representation	**Description**
$\bullet\bullet\,|\,|$	Both balls in first bin.
$\bullet|\bullet|$	One ball in first bin, one ball in second bin.
$\bullet||\bullet$	One ball in first bin, one ball in third bin.
$|\bullet\bullet|$	Both balls in second bin.
$|\bullet|\bullet$	One ball in second bin, one ball in third bin.
$||\bullet\bullet$	Both balls in third bin.

Table 4.1: Occupancy problem for $i = 2$, $|A_i| = 3$

If there are $|A_i|$ bins in total, there must be $|A_i| - 1$ vertical lines between bins. So the number of possible sortings is the number of arrangements of i dots and $|A_i|-1$ vertical lines. Thus we have a word of length $i + |A_i| - 1$, and we must choose i positions to be occupied by dots. There are $\binom{i+|A_i|-1}{i}$ ways in which to do this. □

Since $|A_i| = n + 1 + (w + i - 1)(n - 1)$,

$$|S| = \binom{i + |A_i| - 1}{i} = \binom{i + n + 1 + (w + i - 1)(n - 1) - 1}{i}$$

[3]This example isn't actually possible in the context of words over A_i, as it would imply that $n < 2$ or $w < 2$. However, it still works to illustrate this case of the occupancy problem.

$$= \binom{i+n+1+(w+i)(n-1)-n+1-1}{i} = \binom{(w+i)(n-1)+i+1}{i}.$$

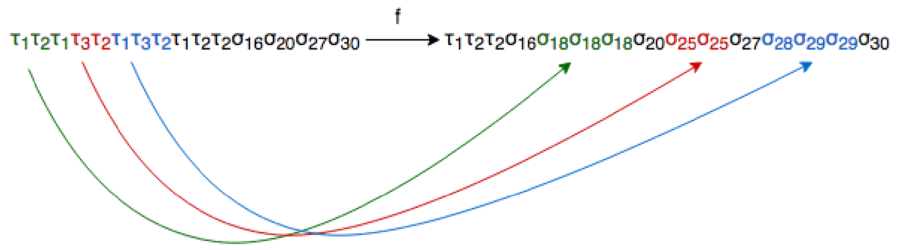
This is exactly the value conjectured for $|R_i|$. Thus showing that $|SV(A_i)| = |U(A_i)|$ is enough to prove Main Theorem 2. We will prove this by constructing a bijection between the two sets.

4.3 Proving that $|SV(A_i)| = |U(A_i)|$

We will construct a bijection $f : SV(A_i) \to U(A_i)$. We can partition $SV(A_i) \cup U(A_i)$ into three subsets: $SV(A_i) \cap U(A_i)$, $SV(A_i) \backslash U(A_i)$, and $U(A_i) \backslash SV(A_i)$. To simplify the notation we introduce the following definitions.

Definition 4.3.1. We will define $\boldsymbol{SV'(A_i)} = SV(A_i) \backslash U(A_i)$ and $\boldsymbol{U'(A_i)} = U(A_i) \backslash SV(A_i)$. That is: $SV'(A_i) \subset SV(A_i)$, $SV'(A_i) \cap U(A_i) = \emptyset$, $U'(A_i) \subset U(A_i)$, $U'(A_i) \cap SV(A_i) = \emptyset$, and $SV'(A_i) \cup U'(A_i) \cup (SV(A_i) \cap U(A_i)) = SV(A_i) \cup U(A_i)$.

We will naturally define f so that any element of $SV(A_i) \cap U(A_i)$ is mapped to itself. Next, we must find a way to map each element of $SV'(A_i)$ to a unique element of $U'(A_i)$. This will involve the τ sequence at the beginning of the word, which we will define more formally in the following section along with other key terms related to $SV'(A_i)$ and $U'(A_i)$. Roughly speaking, sequences of τ's that are not in non-decreasing order are sent to blocks of σ's that result in an invalid word. This is illustrated in Figure 4.2 and full details are given in the following two sections.



$\tau_1\tau_2\tau_1\tau_3\tau_2\tau_1\tau_3\tau_2\tau_1\tau_2\tau_2\sigma_{16}\sigma_{20}\sigma_{27}\sigma_{30}$
f
$\tau_1\tau_2\tau_2\sigma_{16}\sigma_{18}\sigma_{18}\sigma_{18}\sigma_{20}\sigma_{25}\sigma_{25}\sigma_{27}\sigma_{28}\sigma_{29}\sigma_{29}\sigma_{30}$

Figure 4.2: Mapping $SV'(A_i) \to U'(A_i)$

4.3.1 Prefixes and the tier of a letter

Definition 4.3.2. A **simple prefix** is defined to be a sequence of τ's of any length $L > 1$ satisfying the condition that the indices on the first $L-1$ $\tau's$ are in non-decreasing order, but the index on the Lth τ is less than the index on the $(L-1)$th τ. A **compound prefix** is a sequence of two or more simple prefixes.

A standard-form word over A_i must either begin with a compound prefix, begin with a simple prefix, or have no prefix. To avoid confusion we will say that a word begins with a simple prefix only if the remainder of the word does not have any other prefixes - that is the original word did not begin with a compound prefix (even though a word beginning with a compound prefix technically does begin with a simple prefix, as a compound prefix consists of two or more simple prefixes in sequence).

Note that a word in $SV(A_i)$ without a prefix must have either no τ's or its τ's will be in non-decreasing order, so it will be an element of $SV(A_i) \cap U(A_i)$. Conversely every word in $SV'(A_i)$ must begin with either a simple prefix (followed either by no τ's at all or τ's with indices in non-decreasing order) or a compound prefix. A word in $U(A_i)$ will have no prefix by definition, so our objective will be to map each valid word beginning with a simple prefix to an invalid word without a prefix. Some examples are provided for clarity.

Example 4.3.3. Suppose $n = 4$, $w = 2$. The word $\tau_1\tau_2\tau_1\sigma_1$ begins with the simple prefix $\tau_1\tau_2\tau_1$. The word $\tau_1\tau_2\tau_3\tau_4\tau_4\tau_3\tau_2\tau_1\tau_1\tau_2\tau_3\sigma_2$ begins with the compound prefix $\tau_1\tau_2\tau_3\tau_4\tau_4\tau_3\tau_2\tau_1$ (which consists of the simple prefixes $\tau_1\tau_2\tau_3\tau_4\tau_4\tau_3$ and $\tau_2\tau_1$).

Example 4.3.4. For $n = w = 2$ and $i = 4$, a list of words in $SV(A_i)$ and their prefixes is given in Table 4.2.

Definition 4.3.5. The **tier** of a letter is the earliest possible position at which that letter can appear in a word. If $l \in A_i$, we denote the tier of l by $t(l)$.

For any $j \in \{1, 2, 3, \ldots, n\}$, $t(\tau_j) = 1$. As noted in 3.1.1, σ_j can be placed at the kth position in a word if and only if $j \leq 1 + (w + k - 1)(n - 1)$. Solving this inequality for k gives $k \geq \frac{j-1}{n-1} - w + 1$. By definition, the tier of any letter must be a positive integer. Thus $t(\sigma_j) = \max(1, \lceil \frac{j-1}{n-1} - w + 1 \rceil)$.

Word	**Prefix**	**Type of Prefix**
$\sigma\sigma\sigma\sigma$	None	N/A
$\tau\sigma\sigma\sigma$	None	N/A
$\tau_1\tau_1\sigma\sigma$	None	N/A
$\tau_1\tau_2\sigma\sigma$	None	N/A
$\tau_2\tau_1\sigma\sigma$	$\tau_2\tau_1$	Simple
$\tau_2\tau_2\sigma\sigma$	None	N/A
$\tau_1\tau_1\tau_1\sigma$	None	N/A
$\tau_1\tau_1\tau_2\sigma$	None	N/A
$\tau_1\tau_2\tau_1\sigma$	$\tau_1\tau_2\tau_1$	Simple
$\tau_1\tau_2\tau_2\sigma$	None	N/A
$\tau_2\tau_1\tau_1\sigma$	$\tau_2\tau_1$	Simple
$\tau_2\tau_1\tau_2\sigma$	$\tau_2\tau_1$	Simple
$\tau_2\tau_2\tau_1\sigma$	$\tau_2\tau_2\tau_1$	Simple
$\tau_2\tau_2\tau_2\sigma$	None	N/A
$\tau_1\tau_1\tau_1\tau_1$	None	N/A
$\tau_1\tau_1\tau_1\tau_2$	None	N/A
$\tau_1\tau_1\tau_2\tau_1$	$\tau_1\tau_1\tau_2\tau_1$	Simple
$\tau_1\tau_1\tau_2\tau_2$	None	N/A
$\tau_1\tau_2\tau_1\tau_1$	$\tau_1\tau_2\tau_1$	Simple
$\tau_1\tau_2\tau_1\tau_2$	$\tau_1\tau_2\tau_1$	Simple
$\tau_1\tau_2\tau_2\tau_1$	$\tau_1\tau_2\tau_2\tau_1$	Simple
$\tau_1\tau_2\tau_2\tau_2$	None	N/A
$\tau_2\tau_1\tau_1\tau_1$	$\tau_2\tau_1$	Simple
$\tau_2\tau_1\tau_1\tau_2$	$\tau_2\tau_1$	Simple
$\tau_2\tau_1\tau_2\tau_1$	$\tau_2\tau_1\tau_2\tau_1$	Compound
$\tau_2\tau_1\tau_2\tau_2$	$\tau_2\tau_1$	Simple
$\tau_2\tau_2\tau_1\tau_1$	$\tau_2\tau_2\tau_1$	Simple
$\tau_2\tau_2\tau_1\tau_2$	$\tau_2\tau_2\tau_1$	Simple
$\tau_2\tau_2\tau_2\tau_1$	$\tau_2\tau_2\tau_2\tau_1$	Simple
$\tau_2\tau_2\tau_2\tau_2$	None	N/A

Table 4.2: List of words in $SV(A_i)$ for $n = 2$, $w = 2$ and $i = 4$. Note that for letters with missing subscripts, the prefix and type of prefix are the same regardless of the subscripts on these letters.

A word will be valid if and only if for every position p occupied by letter l in the word, $t(l) \leq p$. For any given word in $U'(A_i)$, there is a letter l at some position p such that $t(l) > p$. This follows from the definition of $U'(A_i)$, as every word in $U'(A_i)$ must be invalid.

The rule we will define for mapping an element of $SV'(A_i)$ to $U'(A_i)$ will depend on the prefix the word begins with. Roughly, the τ sequence that makes up the prefix will be mapped to sequences of σ's with high enough tiers to cause the output word to be invalid. The other τ's and σ's in the original word are then added to the output word in between these σ's. This process is illustrated in Figure 4.2.

We begin with words that have simple prefixes.

4.3.2 Words beginning with simple prefixes

We must develop a rule for mapping each word in $SV'(A_i)$ that begins with a simple prefix (but not a compound prefix) to an element of $U'(A_i)$. It becomes more apparent how to do this after we notice that there is a one-to-one correspondence between simple prefixes of length L and words of length L in which all of the letters are of the same tier t, where t ranges from 2 to L. Note that all of the letters in these words will be σ's, as the tier of any τ is 1, and all must be invalid (as the first letter will be of tier 2 or higher). Our strategy will be to first map each (valid) simple prefix to one of these (invalid) words. This is given more precisely in the following definition.

Definition 4.3.6. Let L be a fixed integer, where $L \geq 2$. We will denote the set of all simple prefixes of length L by $\boldsymbol{X}$. The set of all standard-form words of length L consisting only of tier t letters (where $2 \leq t \leq L+1$) will be denoted by $\boldsymbol{Y_t}$.

Proposition 4.3.1. $|X| = |\bigcup_{t=2}^{L} Y_t| = (L-1)\binom{L+n-2}{L}$

Proof. $|\bigcup_{t=2}^{L} Y_t|$ is relatively simple to calculate. With the exception of the first tier (which includes the n τ's as well as σ_1 through $\sigma_{1+w(n-1)}$), each tier has exactly $n-1$ letters (as each procedure adds exactly $n-1$ new leaves). For example tier 2 consists of $\sigma_{2+w(n-1)}$ through $\sigma_{1+(w+1)(n-1)}$ and in general, tier t (where $t > 1$) consists of $\sigma_{2+(w+t-2)(n-1)}$ through $1+(w+t-1)(n-1)$. The only restrictions on a word in Y_t are that all letters need to be of tier t and that the indices on the σ's must be in non-decreasing order. So given a tier $t \geq 2$, the number of words is equal to the number of ways in which L indistinguishable balls can be sorted into $n-1$ distinguishable bins (empty bins are allowed). Since $2 \leq t \leq L$, there are $L-1$ possible values for t. Thus the total number of such words is $(L-1)\binom{L+n-1-1}{L} = (L-1)\binom{L+n-2}{L}$.

We can calculate $|X|$, the number of simple prefixes of length L, by considering each possible case for the second-last letter in the prefix. Suppose the second-last letter is τ_j. Then the first $L-2$ letters of the prefix are elements of the set $\{\tau_1, \tau_2, \tau_3, \ldots, \tau_j\}$, and the indices on the first $L-2$ $\tau's$ are in non-decreasing order. The number of possibilities for the first $L-2$ letters of the prefix is equal to the number of ways in which $L-2$ indistinguishable balls can be sorted into j distinguishable bins (empty bins are allowed). There are also $j-1$ possibilities for the last letter of the prefix as it must be a τ with an index strictly less than j. Thus if the second-last letter is τ_j, there are $\binom{L-2+j-1}{L-2}(j-1) = \binom{L+j-3}{L-2}(j-1)$ prefixes of length L. Summing over all possible values of j, there are

$$\sum_{j=2}^{n} \binom{L+j-3}{L-2}(j-1)$$

total prefixes of length L.

Thus we must now prove that the equality

$$\sum_{j=2}^{n} \binom{L+j-3}{L-2}(j-1) = (L-1)\binom{L+n-2}{L}$$

holds for all possible combinations of L, n. This is done through induction on n. For $n=2$, the left side becomes

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

The right side becomes

$$(L-1)\binom{L+n-2}{L} = (L-1)\binom{L+2-2}{L} = (L-1)\binom{L}{L} = L-1$$

So the equality holds for $n=2$.

Next, suppose that the equality holds for all $n \leq m$ for some $m \geq 2$. For $n = m+1$, the left-side becomes

$$\sum_{j=2}^{n} \binom{L+j-3}{L-2}(j-1) = \sum_{j=2}^{m+1} \binom{L+j-3}{L-2}(j-1)$$
$$= \binom{L+m+1-3}{L-2}(m+1-1) + \sum_{j=2}^{m} \binom{L+j-3}{L-2}(j-1)$$
$$= m\binom{L+m-2}{L-2} + (L-1)\binom{L+m-2}{L}$$

$$
\begin{aligned}
&= m\frac{(L+m-2)!}{(L-2)!m!} + (L-1)\frac{(L+m-2)!}{L!(m-2)!} \\
&= \frac{(L+m-2)!}{(L-2)!(m-2)!}(\frac{m}{m(m-1)} + \frac{L-1}{L(L-1)}) \\
&= \frac{(L+m-2)!}{(L-2)!(m-2)!}(\frac{1}{m-1} + \frac{1}{L}) \\
&= \frac{(L+m-2)!}{(L-2)!(m-2)!}(\frac{L+m-1}{L(m-1)}) = \frac{(L+m-1)!}{(m-1)!(L)(L-2)!} \\
&= \frac{(L+m-1)!}{(m-1)!L(L-2)!} \cdot \frac{L-1}{L-1} = \frac{(L+m-1)!(L-1)}{(m-1)!L!} = (L-1)\binom{L+m-1}{L} \\
&= (L-1)\binom{L+(m+1)-2}{L} = (L-1)\binom{L+n-2}{L}
\end{aligned}
$$

Thus the equality holds for $n = m+1$. This proves the proposition for all L, n combinations. $\square$

So we can map each simple prefix of length L to a word of length L composed only of tier t letters, where $2 \le t \le L$ (with indices on the σ's in non-decreasing order). It does not really matter which word we map each prefix to, but the most natural way would be to list both sets in lexiographical order and simply make a correspondence according to this ordering. We denote this function by $\boldsymbol{g}$.

Example 4.3.7. For $w = 2$, $n = 3$, the simple prefixes of length 2, 3, and 4 and their corresponding images are as given in Table 4.3.

For any prefix P, $g(P)$ is not a valid word (as the tier of the first letter is at least 2). We will next define our function mapping $SV'(A_i) \to U'(A_i)$ for all elements of $SV'(A_i)$ (that is, valid words in standard form) beginning with a simple prefix.

Definition 4.3.8. Let $W \in SV'(A_i)$ be a word beginning with a simple prefix. We use the following procedure to define $f(W)$:

1. If W is a simple prefix, then $f(W) = g(W)$.

2. Otherwise,

   (a) Identify the (simple) prefix of W, hereafter P.

P	$g(P)$	P	$g(P)$
$\tau_2\tau_1$	$\sigma_6\sigma_6$	$\tau_1\tau_1\tau_3\tau_2$	$\sigma_6\sigma_6\sigma_7\sigma_7$
$\tau_3\tau_1$	$\sigma_6\sigma_7$	$\tau_1\tau_2\tau_2\tau_1$	$\sigma_6\sigma_7\sigma_7\sigma_7$
$\tau_3\tau_2$	$\sigma_7\sigma_7$	$\tau_1\tau_2\tau_3\tau_1$	$\sigma_7\sigma_7\sigma_7\sigma_7$
$\tau_1\tau_2\tau_1$	$\sigma_6\sigma_6\sigma_6$	$\tau_1\tau_2\tau_3\tau_2$	$\sigma_8\sigma_8\sigma_8\sigma_8$
$\tau_1\tau_3\tau_1$	$\sigma_6\sigma_6\sigma_7$	$\tau_1\tau_3\tau_3\tau_1$	$\sigma_8\sigma_8\sigma_8\sigma_9$
$\tau_1\tau_3\tau_2$	$\sigma_6\sigma_7\sigma_7$	$\tau_1\tau_3\tau_3\tau_2$	$\sigma_8\sigma_8\sigma_9\sigma_9$
$\tau_2\tau_2\tau_1$	$\sigma_7\sigma_7\sigma_7$	$\tau_2\tau_2\tau_2\tau_1$	$\sigma_8\sigma_9\sigma_9\sigma_9$
$\tau_2\tau_3\tau_1$	$\sigma_8\sigma_8\sigma_8$	$\tau_2\tau_2\tau_3\tau_1$	$\sigma_9\sigma_9\sigma_9\sigma_9$
$\tau_2\tau_3\tau_2$	$\sigma_8\sigma_8\sigma_9$	$\tau_2\tau_2\tau_3\tau_2$	$\sigma_{10}\sigma_{10}\sigma_{10}\sigma_{10}$
$\tau_3\tau_3\tau_1$	$\sigma_8\sigma_9\sigma_9$	$\tau_2\tau_3\tau_3\tau_1$	$\sigma_{10}\sigma_{10}\sigma_{10}\sigma_{11}$
$\tau_3\tau_3\tau_2$	$\sigma_9\sigma_9\sigma_9$	$\tau_2\tau_3\tau_3\tau_2$	$\sigma_{10}\sigma_{10}\sigma_{11}\sigma_{11}$
$\tau_1\tau_1\tau_2\tau_1$	$\sigma_6\sigma_6\sigma_6\sigma_6$	$\tau_3\tau_3\tau_3\tau_1$	$\sigma_{10}\sigma_{11}\sigma_{11}\sigma_{11}$
$\tau_1\tau_1\tau_3\tau_1$	$\sigma_6\sigma_6\sigma_6\sigma_7$	$\tau_3\tau_3\tau_3\tau_2$	$\sigma_{11}\sigma_{11}\sigma_{11}\sigma_{11}$

Table 4.3: Simple prefixes of length 2, 3, and 4 with their corresponding images for $n = 3$, $w = 2$.

(b) Write $g(P)$. This will hereafter be called the "translated prefix". All letters in the translated prefix will be of the same tier (between 2 and L).

(c) Read the remainder of W from left to right. Take the first letter that has not yet been read. This will be called l.

i. If $t(l)$ exceeds the tier of the translated prefix, write l at the end of the word. This ensures l stays in the word, the translated prefix can be identified in $f(W)$, and $f(W)$ remains invalid.

ii. Otherwise, write l immediately before the translated prefix. Increase the index on each of the letters to the right of l by $n-1$. This ensures that $f(W)$ will be invalid, the translated prefix can be identified in $f(W)$, and l remains in the word.

(d) Repeat step c until all of W has been read. The output is $f(W)$.

The easiest way to understand this procedure is with an example, which will be worked out step-by-step.

Example 4.3.9. Let $w = 2$, $n = 3$. In Table 4.4, it is shown that $f(\tau_2\tau_3\tau_1\tau_2\tau_3\sigma_{13}\sigma_{16}\sigma_{19}) = \tau_2\tau_3\sigma_{13}\ \sigma_{14}\sigma_{14}\sigma_{14}\ \sigma_{16}\sigma_{19}$ (as defined in Definition 4.3.8). Note that $\tau_2\tau_3\sigma_{13}\sigma_{14}\sigma_{14}\sigma_{14}\sigma_{16}\sigma_{19}$ is an invalid word (since $t(\sigma_{14}) = 6$) and an element of $U(A_i)$ (since the indices on the τ's are in non-decreasing order). Thus $f(\tau_2\tau_3\tau_1\tau_2\tau_3\sigma_{13}\sigma_{16}\sigma_{19}) \in U'(A_i)$.

Input letter(s)	Current output	Notes
$\tau_2\tau_3\tau_1$	$\underline{\sigma_8\sigma_8\sigma_8}$	Prefix P is $\tau_2\tau_3\tau_1$. Translated prefix $g(P)$ will be underlined to distinguish it. See Table 4.3. $t(\sigma_8) = 3$
τ_2	$\tau_2\ \underline{\sigma_{10}\sigma_{10}\sigma_{10}}$	$t(\tau_2) = 1$. Since $1 < t(\sigma_8)$, we write τ_2 before the translated prefix and increase the indices by $n - 1 = 3 - 1 = 2$. $t(\sigma_{10}) = 4$
τ_3	$\tau_2\tau_3\ \underline{\sigma_{12}\sigma_{12}\sigma_{12}}$	$t(\tau_3) = 1$. Since $1 < t(\sigma_{10})$, we write τ_3 before the translated prefix and increase the indices accordingly. $t(\sigma_{12}) = 5$
σ_{13}	$\tau_2\tau_3\sigma_{13}\ \underline{\sigma_{14}\sigma_{14}\sigma_{14}}$	$t(\sigma_{13}) = 5$. Since $5 = t(\sigma_{12})$, we write σ_{13} before the translated prefix and increase the indices accordingly $t(\sigma_{14}) = 6$
σ_{16}	$\tau_2\tau_3\sigma_{13}\ \underline{\sigma_{14}\sigma_{14}\sigma_{14}}\ \sigma_{16}$	$t(\sigma_{16}) = 6$. Since $6 > t(\sigma_{14})$, we simply write σ_{16} at the end of the word.
σ_{19}	$\tau_2\tau_3\sigma_{13}\ \underline{\sigma_{14}\sigma_{14}\sigma_{14}}\ \sigma_{16}\sigma_{19}$	$t(\sigma_{19}) = 7$. Since $7 > t(\sigma_{14})$, we simply write σ_{19} at the end of the word.

Table 4.4: $f(\tau_2\tau_3\tau_1\tau_2\tau_2\sigma_{13}\sigma_{16}\sigma_{19})$

4.3.3 Words beginning with compound prefixes

Thus far, we have been able to calculate $f(W)$ for any $W \in SV(A_i)$ such that $W \in U(A_i)$ or W begins with a simple prefix (this includes cases in which W itself is a simple prefix).

We must now develop a rule for mapping words beginning with compound prefixes to $U(A_i)$, one that does not contradict the previous rules. If we were given the word $W = \tau_2\tau_1\tau_2\tau_1\sigma_1$ and $n = 3$, $w = 2$, using Definition 4.3.8 to calculate $f(W)$ would not produce an element of $U'(A_i)$ (as we would still be left with two τ's not in non-decreasing order after translating the first simple prefix $\tau_2\tau_1$). Thus we need to extend our definition to cover words beginning with compound prefixes.

We will start with the case where W itself is a compound prefix. Our strategy will be a generalization of the strategy for mapping simple prefixes to invalid words (in which all letters of the translated prefix have the same tier). A compound prefix is a sequence of simple prefixes. We will thus map each component simple prefix to a word of the same length, with all letters of the same tier. We will ensure that each of these translated simple prefixes is of a different tier, and that the final word is invalid.

More precisely, let $W = P_1P_2P_3 \ldots P_p$, where P_j is a simple prefix of length L_j for $1 \leq j \leq p$, and let $L = \sum_{j=1}^{p} L_j$. We will show that the following procedure can be used to map W to an element of $U'(A_i)$:

1. Replace P_p with a word of length L_p composed only of tier t letters, where $L - L_p + 2 \leq t \leq L$ (with indices on the σ's in non-decreasing order). By convention, we will use the same lexiographical ordering system used for simple prefixes. For the remainder of the procedure, we will denote this value of t as t_p. This first step ensures that the final word is invalid.

2. For each j from $p - 1$ to 1 (counting downward), replace P_j with a word of length L_j composed only of tier t letters (indices on σ's in non-decreasing order), where $t_{j+1} - L_j + 1 \leq t \leq t_{j+1} - 1$. By convention, we will use the same lexiographical ordering system used for simple prefixes. For the remainder of this procedure, we will denote this value of t as t_j. This step ensures that each simple prefix is mapped to a word of a different tier.

We will denote this procedure by $\boldsymbol{g}$. Note that this is a generalization of the g we had in the previous section, so we use the same notation.

Proposition 4.3.2. *g is well-defined and injective.*

Proof. We will first show that g is well-defined. As proven in Proposition 4.3.1 the number of simple prefixes of length L_j is equal to $(L_j - 1)\binom{n+L_j-2}{L_j}$, the number of standard-form words of length L_j composed only of tier t letters (where $2 \leq t \leq L_j$). The number of words of length L_j composed only of tier t letters, where instead t ranges from $t_{j+1} - L_j + 1$ to $t_{j+1} - 1$, is also equal to $(L_j - 1)\binom{n+L_j-2}{L_j}$. Similarly the number of words of length L_p composed only of tier t letters, where t ranges from $L - L_p + 2$ to L, is equal to $(L_p - 1)\binom{n+L_p-2}{L_p}$. Furthermore, each simple prefix is guaranteed to be translated to a word of a different tier (in step 2), and it can be shown by induction on p that the first (leftmost) simple prefix will always be translated to a word of tier $p + 1$ or higher (so $g(P)$ must be invalid for every compound prefix P). Thus g is well-defined for every compound prefix.

We will next show that g is an injective map. Let P and Q be compound prefixes, and suppose $g(P) = g(Q)$. We will show that $P = Q$. Without loss of generality we will take $g(P) = W_1 W_2 \dots W_p$, where $p \geq 2$ and W_j is a standard-form word of length L_j and composed solely of tier t_j letters (for $1 \leq j \leq p$). We also need $L - L_p + 2 \leq t_p \leq L$ (where L is the total length of $g(P)$ and, by definition, P and Q as well) and, for $1 \leq j \leq p - 1$, $t_{j+1} - L_j + 1 \leq t \leq t_{j+1} - 1$. We can use the following procedure to get P given $g(P)$:

1. Make a list of simple prefixes of length L_p in lexicographical order. Then make a list of standard-form words, in lexicographical order, consisting only of letters of the same tier (with tier ranging from $L - L_p + 2$ to L). Draw a correspondence between the two sets. W_p is matched to P_p, the last simple prefix of P. Because this correspondence is one-to-one, the last simple prefix of Q must also be P_p.

2. For each j from $p - 1$ to 1 (counting downward), make a list of simple prefixes of length L_j in lexicographical order. Then make a list of standard-form words, in lexicographical order, consisting only of letters of the same tier (with tier ranging from $t_{j+1} - L_j + 1$ to $t_j - 1$). Draw a correspondence between the two sets. W_j is matched to P_j, the jth simple prefix of P. Because this correspondence is one-to-one, the jth simple prefix of Q must also be P_j.

Since all simple prefixes in both P and Q are the same, $P = Q$ and g is an injective map. □

Example 4.3.10. For $n = 3$, $w = 2$, the compound prefixes of length 4, 5 and their corresponding images are as given in Tables 4.5 and 4.6.

P	$g(P)$
$\tau_2\tau_1\tau_2\tau_1$	$\sigma_8\sigma_8\sigma_{10}\sigma_{10}$
$\tau_2\tau_1\tau_3\tau_1$	$\sigma_8\sigma_8\sigma_{10}\sigma_{11}$
$\tau_2\tau_1\tau_3\tau_2$	$\sigma_8\sigma_8\sigma_{11}\sigma_{11}$
$\tau_3\tau_1\tau_2\tau_1$	$\sigma_8\sigma_9\sigma_{10}\sigma_{10}$
$\tau_3\tau_1\tau_3\tau_1$	$\sigma_8\sigma_9\sigma_{10}\sigma_{11}$
$\tau_3\tau_1\tau_3\tau_2$	$\sigma_8\sigma_9\sigma_{11}\sigma_{11}$
$\tau_3\tau_2\tau_2\tau_1$	$\sigma_9\sigma_9\sigma_{10}\sigma_{10}$
$\tau_3\tau_2\tau_3\tau_1$	$\sigma_9\sigma_9\sigma_{10}\sigma_{11}$
$\tau_3\tau_2\tau_3\tau_2$	$\sigma_9\sigma_9\sigma_{11}\sigma_{11}$

Table 4.5: The compound prefixes of length 4 and their corresponding images for $n = 3$, $w = 2$.

We will next define our function mapping $SV'(A_i) \to U'(A_i)$ for all elements of $SV'(A_i)$ (that is, valid words in standard form) beginning with a compound prefix.

Definition 4.3.11. Let $W \in SV'(A_i)$ be a word beginning with a compound prefix. We use the following procedure to find $f(W)$:

1. If W is a compound prefix, then $f(W) = g(W)$.

2. Otherwise,

   (a) Identify the (compound) prefix of W, hereafter P.

   (b) Write $g(P)$. This will hereafter be called the "translated prefix". Keep track of which component of $g(P)$ corresponds to each simple prefix comprising P. All letters in a given component of the translated prefix will be of the same tier, and no two components will have the same tier.

   (c) Read the remainder of W from left to right. Take the first letter that has not yet been read. This will be called l.

P	$g(P)$
$\tau_1\tau_2\tau_1\tau_2\tau_1$	$\sigma_8\sigma_8\sigma_8\sigma_{12}\sigma_{12}$
$\tau_1\tau_2\tau_1\tau_3\tau_1$	$\sigma_8\sigma_8\sigma_8\sigma_{12}\sigma_{13}$
$\tau_1\tau_2\tau_1\tau_3\tau_2$	$\sigma_8\sigma_8\sigma_8\sigma_{13}\sigma_{13}$
$\tau_1\tau_3\tau_1\tau_2\tau_1$	$\sigma_8\sigma_8\sigma_9\sigma_{12}\sigma_{12}$
$\tau_1\tau_3\tau_1\tau_3\tau_1$	$\sigma_8\sigma_8\sigma_9\sigma_{12}\sigma_{13}$
$\tau_1\tau_3\tau_1\tau_3\tau_2$	$\sigma_8\sigma_8\sigma_9\sigma_{13}\sigma_{13}$
$\tau_1\tau_3\tau_2\tau_2\tau_1$	$\sigma_8\sigma_9\sigma_9\sigma_{12}\sigma_{12}$
$\tau_1\tau_3\tau_2\tau_3\tau_1$	$\sigma_8\sigma_9\sigma_9\sigma_{12}\sigma_{13}$
$\tau_1\tau_3\tau_2\tau_3\tau_2$	$\sigma_8\sigma_9\sigma_9\sigma_{13}\sigma_{13}$
$\tau_2\tau_2\tau_1\tau_2\tau_1$	$\sigma_9\sigma_9\sigma_9\sigma_{12}\sigma_{12}$
$\tau_2\tau_2\tau_1\tau_3\tau_1$	$\sigma_9\sigma_9\sigma_9\sigma_{12}\sigma_{13}$
$\tau_2\tau_2\tau_1\tau_3\tau_2$	$\sigma_9\sigma_9\sigma_9\sigma_{13}\sigma_{13}$
$\tau_2\tau_3\tau_1\tau_2\tau_1$	$\sigma_{10}\sigma_{10}\sigma_{10}\sigma_{12}\sigma_{12}$
$\tau_2\tau_3\tau_1\tau_3\tau_1$	$\sigma_{10}\sigma_{10}\sigma_{10}\sigma_{12}\sigma_{13}$
$\tau_2\tau_3\tau_1\tau_3\tau_2$	$\sigma_{10}\sigma_{10}\sigma_{10}\sigma_{13}\sigma_{13}$
$\tau_2\tau_3\tau_2\tau_2\tau_1$	$\sigma_{10}\sigma_{10}\sigma_{11}\sigma_{12}\sigma_{12}$
$\tau_2\tau_3\tau_2\tau_3\tau_1$	$\sigma_{10}\sigma_{10}\sigma_{11}\sigma_{12}\sigma_{13}$
$\tau_2\tau_3\tau_2\tau_3\tau_2$	$\sigma_{10}\sigma_{10}\sigma_{11}\sigma_{13}\sigma_{13}$
$\tau_3\tau_3\tau_1\tau_2\tau_1$	$\sigma_{10}\sigma_{11}\sigma_{11}\sigma_{12}\sigma_{12}$
$\tau_3\tau_3\tau_1\tau_3\tau_1$	$\sigma_{10}\sigma_{11}\sigma_{11}\sigma_{12}\sigma_{13}$
$\tau_3\tau_3\tau_1\tau_3\tau_2$	$\sigma_{10}\sigma_{11}\sigma_{11}\sigma_{13}\sigma_{13}$
$\tau_3\tau_3\tau_2\tau_2\tau_1$	$\sigma_{11}\sigma_{11}\sigma_{11}\sigma_{12}\sigma_{12}$
$\tau_3\tau_3\tau_2\tau_3\tau_1$	$\sigma_{11}\sigma_{11}\sigma_{11}\sigma_{12}\sigma_{13}$
$\tau_3\tau_3\tau_2\tau_3\tau_2$	$\sigma_{11}\sigma_{11}\sigma_{11}\sigma_{13}\sigma_{13}$
$\tau_2\tau_1\tau_1\tau_2\tau_1$	$\tau_8\tau_8\tau_{10}\tau_{10}\tau_{10}$
$\tau_2\tau_1\tau_1\tau_3\tau_1$	$\tau_8\tau_8\tau_{10}\tau_{10}\tau_{11}$
$\tau_2\tau_1\tau_1\tau_3\tau_2$	$\tau_8\tau_8\tau_{10}\tau_{11}\tau_{11}$
$\tau_2\tau_1\tau_2\tau_2\tau_1$	$\tau_8\tau_8\tau_{11}\tau_{11}\tau_{11}$
$\tau_2\tau_1\tau_2\tau_3\tau_1$	$\tau_{10}\tau_{10}\tau_{12}\tau_{12}\tau_{12}$
$\tau_2\tau_1\tau_2\tau_3\tau_2$	$\tau_{10}\tau_{10}\tau_{12}\tau_{12}\tau_{13}$
$\tau_2\tau_1\tau_3\tau_3\tau_1$	$\tau_{10}\tau_{10}\tau_{12}\tau_{13}\tau_{13}$
$\tau_2\tau_1\tau_3\tau_3\tau_2$	$\tau_{10}\tau_{10}\tau_{13}\tau_{13}\tau_{13}$
$\tau_3\tau_1\tau_1\tau_2\tau_1$	$\tau_8\tau_9\tau_{10}\tau_{10}\tau_{10}$
$\tau_3\tau_1\tau_1\tau_3\tau_1$	$\tau_8\tau_9\tau_{10}\tau_{10}\tau_{11}$
$\tau_3\tau_1\tau_1\tau_3\tau_2$	$\tau_8\tau_9\tau_{10}\tau_{11}\tau_{11}$
$\tau_3\tau_1\tau_2\tau_2\tau_1$	$\tau_8\tau_9\tau_{11}\tau_{11}\tau_{11}$
$\tau_3\tau_1\tau_2\tau_3\tau_1$	$\tau_{10}\tau_{11}\tau_{12}\tau_{12}\tau_{12}$
$\tau_3\tau_1\tau_2\tau_3\tau_2$	$\tau_{10}\tau_{11}\tau_{12}\tau_{12}\tau_{13}$
$\tau_3\tau_1\tau_3\tau_3\tau_1$	$\tau_{10}\tau_{11}\tau_{12}\tau_{13}\tau_{13}$
$\tau_3\tau_1\tau_3\tau_3\tau_2$	$\tau_{10}\tau_{11}\tau_{13}\tau_{13}\tau_{13}$
$\tau_3\tau_2\tau_1\tau_2\tau_1$	$\tau_9\tau_9\tau_{10}\tau_{10}\tau_{10}$
$\tau_3\tau_2\tau_1\tau_3\tau_1$	$\tau_9\tau_9\tau_{10}\tau_{10}\tau_{11}$
$\tau_3\tau_2\tau_1\tau_3\tau_2$	$\tau_9\tau_9\tau_{10}\tau_{11}\tau_{11}$
$\tau_3\tau_2\tau_2\tau_2\tau_1$	$\tau_9\tau_9\tau_{11}\tau_{11}\tau_{11}$
$\tau_3\tau_2\tau_2\tau_3\tau_1$	$\tau_{11}\tau_{11}\tau_{12}\tau_{12}\tau_{12}$
$\tau_3\tau_2\tau_2\tau_3\tau_2$	$\tau_{11}\tau_{11}\tau_{12}\tau_{12}\tau_{13}$
$\tau_3\tau_2\tau_3\tau_3\tau_1$	$\tau_{11}\tau_{11}\tau_{12}\tau_{13}\tau_{13}$
$\tau_3\tau_2\tau_3\tau_3\tau_2$	$\tau_{11}\tau_{11}\tau_{13}\tau_{13}\tau_{13}$

Table 4.6: The compound prefixes of length 5 and their corresponding images for $n = 3$, $w = 2$.

i. If $t(l)$ exceeds the tier of every component of the translated prefix, write l at the end of the word.

ii. Otherwise, write l immediately before the leftmost component with tier greater than or equal to $t(l)$. Increase the index on each letter to the right of l by $n - 1$ (this ensures that the $f(W)$ will be invalid).

(d) Repeat step c until all of W has been read. The output is $f(W)$.

We will work through an example step-by-step to illustrate how this procedure is applied.

Example 4.3.12. Let $w = 2$, $n = 3$. In Table 4.7 it is shown that

$$f(\tau_1\tau_2\tau_1\tau_3\tau_2\tau_1\tau_3\tau_2\tau_1\tau_2\tau_2\sigma_{16}\sigma_{20}\sigma_{27}\sigma_{30}) = \tau_1\tau_2\tau_2\sigma_{16}\sigma_{18}\sigma_{18}\sigma_{18}\sigma_{20}\sigma_{25}\sigma_{25}\sigma_{27}\sigma_{28}\sigma_{29}\sigma_{29}\sigma_{30},$$

as defined in Definition 4.3.11. Notes on Table 4.7:

1. The prefix is $\tau_1\tau_2\tau_1\tau_3\tau_2\tau_1\tau_3\tau_2$, composed of three separate simple prefixes: $\tau_1\tau_2\tau_1$, $\tau_3\tau_2$, and $\tau_1\tau_3\tau_2$. Following the procedure for translating compound prefixes,

   $$g(\tau_1\tau_2\tau_1\tau_3\tau_2\tau_1\tau_3\tau_2) = \sigma_{10}\sigma_{10}\sigma_{10}\sigma_{15}\sigma_{15}\sigma_{16}\sigma_{17}\sigma_{17}.$$

   Each part of the translated prefix has been underlined to distinguish it. $t(\sigma_{10}) = 4$, $t(\sigma_{15}) = 6$, $t(\sigma_{16}) = t(\sigma_{17}) = 7$

2. $t(\tau_1) = 1$. Since $1 < t(\sigma_{10})$, we write τ_1 before the first component of the translated prefix and increase the indices accordingly. $t(\sigma_{12}) = 5$, $t(\sigma_{17}) = 7$, $t(\sigma_{18}) = t(\sigma_{19}) = 8$

3. $t(\tau_2) = 1$. Since $1 < t(\sigma_{12})$, we write τ_2 before the first component of the translated prefix and increase the indices accordingly. $t(\sigma_{14}) = 6$, $t(\sigma_{19}) = 8$, $t(\sigma_{20}) = t(\sigma_{21}) = 9$

4. $t(\tau_2) = 1$. Since $1 < t(\sigma_{14})$, we write τ_2 before the first component of the translated prefix and increase the indices accordingly. $t(\sigma_{16}) = 7$, $t(\sigma_{21}) = 9$, $t(\sigma_{22}) = t(\sigma_{23}) = 10$

5. $t(\sigma_{16}) = 7$. Since $7 = t(\sigma_{16})$, we write σ_{16} before the first component of the translated prefix and increase the indices accordingly. $t(\sigma_{18}) = 8$, $t(\sigma_{23}) = 10$, $t(\sigma_{24}) = t(\sigma_{25}) = 11$

6. $t(\sigma_{20}) = 9$. Since $t(\sigma_{18}) < 9 < t(\sigma_{23})$, we write σ_{20} after the first component but before the second component of the translated prefix and increase the indices accordingly. $t(\sigma_{25}) = 11$, $t(\sigma_{26}) = t(\sigma_{27}) = 12$.

7. $t(\sigma_{27}) = 12$. Since $t(\sigma_{25}) < 12 \leq t(\sigma_{27})$, we write σ_{27} after the second component but before the third component of the translated prefix and increase the indices accordingly. $t(\sigma_{28}) = t(\sigma_{29}) = 13$.

8. $t(\sigma_{30}) = 14$. Since $14 > t(\sigma_{29})$, we write σ_{30} at the end of the word. Thus,

   $$f(\tau_1\tau_2\tau_1\tau_3\tau_2\tau_1\tau_3\tau_2\tau_1\tau_2\tau_2\sigma_{16}\sigma_{20}\sigma_{27}\sigma_{30}) = \tau_1\tau_2\tau_2\sigma_{16}\sigma_{18}\sigma_{18}\sigma_{18}\sigma_{20}\sigma_{25}\sigma_{25}\sigma_{27}\sigma_{28}\sigma_{29}\sigma_{29}\sigma_{30}.$$

   Note that $\tau_1\tau_2\tau_2\sigma_{16}\sigma_{18}\sigma_{18}\sigma_{18}\sigma_{20}\sigma_{25}\sigma_{25}\sigma_{27}\sigma_{28}\sigma_{29}\sigma_{29}\sigma_{30}$ is an invalid word (since $t(\sigma_16) = 7$) and an element of $U(A_i)$ (since the indices on the τ's are in non-decreasing order). Thus $f(\tau_1\tau_2\tau_1\tau_3\tau_2\tau_1\tau_3\tau_2\tau_1\tau_2\tau_2\sigma_{16}\sigma_{20}\sigma_{27}\sigma_{30}) \in U'(A_i)$.

Input Letter(s)	**Current output**	**Notes**
$\tau_1\tau_2\tau_1\tau_3\tau_2\tau_1\tau_3\tau_2$	$\underline{\sigma_{10}\sigma_{10}\sigma_{10}}\ \underline{\sigma_{15}\sigma_{15}}\ \underline{\sigma_{16}\sigma_{17}\sigma_{17}}$	1
τ_1	$\tau_1\ \underline{\sigma_{12}\sigma_{12}\sigma_{12}}\ \underline{\sigma_{17}\sigma_{17}}\ \underline{\sigma_{18}\sigma_{19}\sigma_{19}}$	2
τ_2	$\tau_1\tau_2\ \underline{\sigma_{14}\sigma_{14}\sigma_{14}}\ \underline{\sigma_{19}\sigma_{19}}\ \underline{\sigma_{20}\sigma_{21}\sigma_{21}}$	3
τ_2	$\tau_1\tau_2\tau_2\ \underline{\sigma_{16}\sigma_{16}\sigma_{16}}\ \underline{\sigma_{21}\sigma_{21}}\ \underline{\sigma_{22}\sigma_{23}\sigma_{23}}$	4
σ_{16}	$\tau_1\tau_2\tau_2\sigma_{16}\ \underline{\sigma_{18}\sigma_{18}\sigma_{18}}\ \underline{\sigma_{23}\sigma_{23}}\ \underline{\sigma_{24}\sigma_{25}\sigma_{25}}$	5
σ_{20}	$\tau_1\tau_2\tau_2\sigma_{16}\ \underline{\sigma_{18}\sigma_{18}\sigma_{18}}\ \sigma_{20}\ \underline{\sigma_{25}\sigma_{25}}\ \underline{\sigma_{26}\sigma_{27}\sigma_{27}}$	6
σ_{27}	$\tau_1\tau_2\tau_2\sigma_{16}\ \underline{\sigma_{18}\sigma_{18}\sigma_{18}}\ \sigma_{20}\ \underline{\sigma_{25}\sigma_{25}}\ \sigma_{27}\ \underline{\sigma_{28}\sigma_{29}\sigma_{29}}$	7
σ_{30}	$\tau_1\tau_2\tau_2\sigma_{16}\ \underline{\sigma_{18}\sigma_{18}\sigma_{18}}\ \sigma_{20}\ \underline{\sigma_{25}\sigma_{25}}\ \sigma_{27}\ \underline{\sigma_{28}\sigma_{29}\sigma_{29}}\ \sigma_{30}$	8

Table 4.7: $f(\tau_1\tau_2\tau_1\tau_3\tau_2\tau_1\tau_3\tau_2\tau_1\tau_2\tau_2\sigma_{16}\sigma_{20}\sigma_{27}\sigma_{30})$

We have now constructed a map from $SV(A_i)$ to $U(A_i)$, defined for all words in $SV(A_i)$. It must still be shown that this map is bijective. This will be proven in the following section.

4.3.4 Showing that f is a bijection

Our definition of $f(W)$ in cases where W begins with a simple prefix is a special case of the definition for words beginning with compound prefixes. This simplifies the final definition of f.

Given a word $W \in SV(A_i)$, we calculate $f(W)$ as follows:

1. If $W \in U(A_i)$, $f(W) = W$.
2. If W is a prefix (simple or compound), then $f(W) = g(W)$
3. Otherwise, W will begin with a compound prefix:
   (a) Identify the prefix of W, hereafter P.
   (b) Write $g(P)$. This will hereafter be called the "translated prefix". Keep track of which component of $g(P)$ corresponds to each simple prefix comprising P (there will be only one component if P is simple). All letters in a given component of

the translated prefix will be of the same tier, and no two components will have the same tier.

(c) Read the remainder of W from left to right. Take the first letter that has not yet been read. This will be called l.

  i. If $t(l)$ exceeds the tier of every component of the translated prefix, write l at the end of the word.

  ii. Otherwise, write l immediately before the leftmost component with tier greater than or equal to $t(l)$. Increase the subscript on each letter to the right of l by $n-1$ (this ensures that the $f(W)$ will be invalid).

(d) Repeat step c until all of W has been read. The output is $f(W)$.

We must prove now that f is a bijection from $SV(A_i)$ to $U(A_i)$. That is f is well-defined, injective, and surjective.

Lemma 4.3.3. *f is a well defined function mapping $SV(A_i)$ to $U(A_i)$.*

Proof. This follows immediately. $f(W)$ cannot have a prefix, so the subscripts on the τ's must be in non-decreasing order. This ensures $f(W) \in U(A_i)$. The procedure for calculating $f(W)$ defines a way to map each element of $SV(A_i)$ to a single element of $U(A_i)$. This includes each type of element: elements of $SV(A_i) \cap U(A_i)$, words beginning with a simple prefix, and words beginning with a compound prefix. □

To show that f is a bijection, we will prove that f has an inverse function. First, define $h : U(A_i) \to SV(A_i)$ as follows: given a word $V \in U(A_i)$,

1. If $V \in SV(A_i)$, then $h(V) = V$.

2. Otherwise, V is invalid. We will keep track of the tier and position of each letter of V, and a sequence of letters of the same tier will be referred to as a block. Suppose V consists of b blocks $B_1, B_2, \ldots, B_b$. To calculate $h(V)$, we execute the following steps. We initially have a null word as the output and as the remainder.

   For $j = b$ to 1 (counting downwards),

(a) If the remainder is a null word,

i. If the position of the first (leftmost) letter of B_j is at least $t(B_j)$, write B_j at the beginning (far left) of the output word.

ii. If $t(B_j)$ exceeds the position of the first letter of B_j, B_j becomes the remainder. The output is not changed.

(b) Otherwise, let r be the first (leftmost) letter of the remainder.

i. If $t(r)-t(B_j) \geq L_{B_j}$ (where L_{B_j} is the length of B_j), write B_j at the beginning of the remainder. The output is not changed.

ii. Otherwise, write B_j at the beginning of the output. In the remainder, the subscript of each σ is reduced by $L_{B_j}(n-1)$.

Once this has been done for all blocks, find the prefix P such that $g(P)$ gives the remainder. Write P at the beginning of the output word. The final output is $h(V)$.

This is most easily illustrated with an example.

Example 4.3.13. We will let $w = 2$, $n = 3$ and show that

$$h(\tau_1\tau_2\tau_2\sigma_{16}\sigma_{18}\sigma_{18}\sigma_{18}\sigma_{20}\sigma_{25}\sigma_{25}\sigma_{27}\sigma_{28}\sigma_{29}\sigma_{29}\sigma_{30}) = \tau_1\tau_2\tau_1\tau_3\tau_2\tau_1\tau_3\tau_2\tau_1\tau_2\tau_2\sigma_{16}\sigma_{20}\sigma_{27}\sigma_{30}$$

The first step is to write down the tier and position of each letter in h.

τ_1	τ_2	τ_2	σ_{16}	σ_{18}	σ_{18}	σ_{18}	σ_{20}	σ_{25}	σ_{25}	σ_{27}	σ_{28}	σ_{29}	σ_{29}	σ_{30}
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	7	8	8	8	9	11	11	12	13	13	13	14

So the blocks are: $\tau_1\tau_2\tau_2$, σ_{16}, $\sigma_{18}\sigma_{18}\sigma_{18}$, σ_{20}, $\sigma_{25}\sigma_{25}$, σ_{27}, $\sigma_{28}\sigma_{29}\sigma_{29}$, σ_{30}. In Table 4.8, we process each of these blocks starting with σ_{30}. Notes on Table 4.8:

1. $t(\sigma_{30}) = 14$, and this block begins at position 15. Thus it's not a part of the remainder and will be added to the output.

2. $t(\sigma_{28}) = 13$, and this block begins at position 12. Thus it's part of the remainder.

3. The length of this block is 1, and the tier is 12. The tier of the first block of the remainder is 13. Since $13 - 12 \geq 1$, this block is not part of the remainder and will be added to the output. In the remainder, we reduce each subscript by $1(n-1) = 3-1 = 2$.

4. The length of this block is 2, and the tier is 11. The tier of the first block of the remainder is 12. Since $12 - 11 < 2$, this block is part of the remainder.

5. The length of this block is 1, and the tier is 9. The tier of the first block of the remainder is 11. Since $11 - 9 \geq 1$, this block is not part of the remainder and will be added to the output. In the remainder, we reduce each subscript by $1(n-1) = 3-1 = 2$.

6. The length of this block is 3, and the tier is 8. The tier of the first block of the remainder is 10. Since $10 - 8 < 3$, this block is part of the remainder.

7. The length of this block is 1, and the tier is 7. The tier of the first block of the remainder is 8. Since $8 - 7 \geq 1$, this block is not part of the remainder and will be added to the output. In the remainder, we reduce each subscript by $1(n-1) = 3-1 = 2$.

8. The length of this block is 3, and the tier is 1. The tier of the first block of the remainder is 7. Since $7 - 1 \geq 3$, this block is not part of the remainder and will be added to the output. In the remainder, we reduce each subscript by $3(n-1) = 3(3-1) = 6$.

9. The final remainder is $\sigma_{10}\sigma_{10}\sigma_{10}\sigma_{15}\sigma_{15}\sigma_{16}\sigma_{17}\sigma_{17}$. As demonstrated in Example 4.3.12, this corresponds to the prefix $\tau_1\tau_2\tau_1\tau_3\tau_2\tau_1\tau_3\tau_2$. Thus,

$$h(\tau_1\tau_2\tau_2\sigma_{16}\sigma_{18}\sigma_{18}\sigma_{18}\sigma_{20}\sigma_{25}\sigma_{25}\sigma_{27}\sigma_{28}\sigma_{29}\sigma_{29}\sigma_{30}) = \tau_1\tau_2\tau_1\tau_3\tau_2\tau_1\tau_3\tau_2\tau_1\tau_2\tau_2\sigma_{16}\sigma_{20}\sigma_{27}\sigma_{30}.$$

This example is summarized in Figure 4.3.

We must now prove that h is well-defined (that is, $h(V)$ has exactly one value for every word $V \in U(A_i)$) and that h is an inverse of f.

Lemma 4.3.4. *h is well-defined.*

Proof. There are three cases that need to be considered: cases where V is valid, cases where (after reading all blocks of V) the remainder consists of only one block, and cases in which the remainder consists of two or more blocks.

Block	**Current output**	**Current remainder**	**Notes**
σ_{30}	σ_{30}	None	1
$\sigma_{28}\sigma_{29}\sigma_{29}$	σ_{30}	$\sigma_{28}\sigma_{29}\sigma_{29}$	2
σ_{27}	$\sigma_{27}\sigma_{30}$	$\sigma_{26}\sigma_{27}\sigma_{27}$	3
$\sigma_{25}\sigma_{25}$	$\sigma_{27}\sigma_{30}$	$\sigma_{25}\sigma_{25}\sigma_{26}\sigma_{27}\sigma_{27}$	4
σ_{20}	$\sigma_{20}\sigma_{27}\sigma_{30}$	$\sigma_{23}\sigma_{23}\sigma_{24}\sigma_{25}\sigma_{25}$	5
$\sigma_{18}\sigma_{18}\sigma_{18}$	$\sigma_{20}\sigma_{27}\sigma_{30}$	$\sigma_{18}\sigma_{18}\sigma_{18}\sigma_{23}\sigma_{23}\sigma_{24}\sigma_{25}\sigma_{25}$	6
σ_{16}	$\sigma_{16}\sigma_{20}\sigma_{27}\sigma_{30}$	$\sigma_{16}\sigma_{16}\sigma_{16}\sigma_{21}\sigma_{21}\sigma_{22}\sigma_{23}\sigma_{23}$	7
$\tau_1\tau_2\tau_2$	$\tau_1\tau_2\tau_2\sigma_{16}\sigma_{20}\sigma_{27}\sigma_{30}$	$\sigma_{10}\sigma_{10}\sigma_{10}\sigma_{15}\sigma_{15}\sigma_{16}\sigma_{17}\sigma_{17}$	8
Prefix	$\tau_1\tau_2\tau_1\tau_3\tau_2\tau_1\tau_3\tau_2\tau_1\tau_2\tau_2\sigma_{16}\sigma_{20}\sigma_{27}\sigma_{30}$	$\sigma_{10}\sigma_{10}\sigma_{10}\sigma_{15}\sigma_{15}\sigma_{16}\sigma_{17}\sigma_{17}$	9

Table 4.8: $h(\tau_1\tau_2\tau_2\sigma_{16}\sigma_{18}\sigma_{18}\sigma_{18}\sigma_{20}\sigma_{25}\sigma_{25}\sigma_{27}\sigma_{28}\sigma_{29}\sigma_{29}\sigma_{30})$.

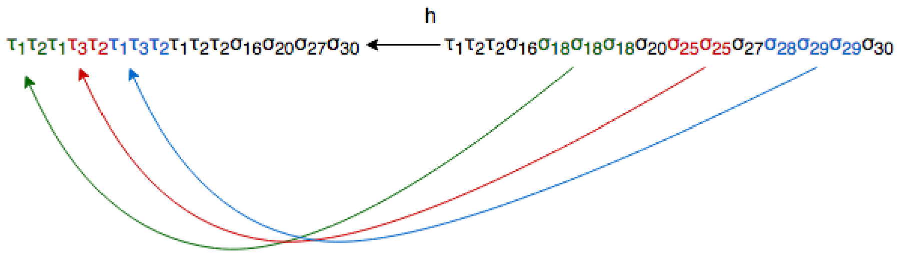

τ1τ2τ1τ3τ2τ1τ3τ2τ1τ2τ2σ16σ20σ27σ30
h
τ1τ2τ2σ16σ18σ18σ18σ20σ25σ25σ27σ28σ29σ29σ30

Figure 4.3: Mapping $U'(A_i) \to SV'(A_i)$

The first case follows immediately from the definition. Every valid word without a prefix is an element of $SV(A_i)$, so $h(V) \in SV(A_i)$ for any valid word V.

For the second case, let $V = V_1BV_2$, where B is the block that will comprise the remainder (V_1 and V_2 may have any number of blocks, or may be null words). Let L_1, L_B, and L_2 be the lengths of V_1, B, and V_2 respectively. Clearly $t(B) > L_1 + 1$ (as otherwise V would be valid), so $t(B) - L_1 > 1$. That is, $t(B) - L_1 \geq 2$. Thus the tier of V' is at least 2. On the other hand, $t(B)$ must be less than the tier of the first block of V_2. This value is at most $L_1 + L_B + 1$. Thus $t(B) < L_1 + L_B + 1$, or $t(B) - L_1 < L_B + 1$. That is, $2 \leq t(B) - L_1 \leq L_B$. So the tier of B ranges between 2 and L_B (the length of the block). When finding $h(V)$, the final step of the algorithm leaves B as the remainder and V_1V_2 as the current output. By Proposition 4.3.1, we can map every possible remainder B to a simple prefix P. Thus $h(V) = PV_1V_2$.

For the third case, let b be the total number of blocks in the remainder and let $V = V_1B_1V_2B_2 \dots V_bB_bV_{b+1}$ (where $B_1, B_2, \dots, B_b$ will be the blocks that comprise the remainder and $V_1, V_2, \dots, V_{b+1}$ are words consisting of any number of blocks). Let L_i be the length of V_i and L_{B_i} be the length of B_i. In the final remainder then, the tier of the final block will be $t(B_b) - \sum_{j=1}^{b} L_i$.

For B_b to be part of the remainder we need $t(B_b) > 1 + \sum_{i=1}^{b-1} L_{B_i} + \sum_{j=1}^{b} L_i$, so

$$t(B_b) - \sum_{j=1}^{b} L_i \geq 2 + \sum_{i=1}^{b-1} L_{B_i}.$$

On the other hand, $t(B_b)$ must be less than or equal to the total length of V (less the length of V_{b+1}). That is,

$$t(B_b) \leq \sum_{i=1}^{b} L_{B_i} + \sum_{j=1}^{b} L_j$$
$$t(B_b) - \sum_{j=1}^{b} L_j \leq \sum_{i=1}^{b} L_{B_i}.$$

Putting these together, we get

$$2 + \sum_{i=1}^{b-1} L_{B_i} \leq t(B_b) - \sum_{j=1}^{b} L_i \leq \sum_{i=1}^{b} L_{B_i}$$
$$2 + \sum_{i=1}^{b} L_{B_i} - L_{B_i} \leq t(B_b) - \sum_{j=1}^{b} L_i \leq \sum_{i=1}^{b} L_{B_i}.$$

Compare this with the output of $g(P)$ for an arbitrary prefix P composed of b simple prefixes (as in Proposition 4.3.2). The tier of the final block of the translated prefix is between $L_P - L_b + 2$ and L_P, where L_P is the total length of the prefix and L_b is the length of the final simple prefix. This is exactly the range we have calculated above.

Next, consider an arbitrary block B_j, where $1 \leq j < b$. On the one hand, for B_j to be part of the remainder we need

$$(t(B_{j+1}) - L_{j+1}) - t(B_j) < L_{B_j}$$

$$\begin{aligned}
(t(B_{j+1}) - L_{j+1}) - L_{B_j} &< t(B_j) \\
t(B_j) &> (t(B_{j+1}) - L_{j+1}) - L_{B_j} \\
t(B_j) &\geq 1 + (t(B_{j+1}) - L_{j+1}) - L_{B_j} \\
t(B_j) - \sum_{i=1}^{j} L_i &\geq 1 + t(B_{j+1}) - \sum_{i=1}^{j} L_i - L_{j+1} - L_{B_j} \\
t(B_j) - \sum_{i=1}^{j} L_i &\geq 1 + (t(B_{j+1}) - \sum_{i=1}^{j+1} L_i) - L_{B_j}.
\end{aligned}$$

On the other hand $t(B_j)$ must be less than the tier of the first block of V_{j+1}, which we will denote by $V_{1,j+1}$. Since V_{j+1} is part of the output, we need $t(B_{j+1}) - t(V_{1,j+1}) \geq L_{j+1}$. That is,

$$\begin{aligned}
t(B_{j+1}) - t(V_{1,j+1}) &\geq L_{j+1} \\
t(B_{j+1}) - L_{j+1} &\geq t(V_{1,j+1}) \\
t(B_j) < t(V_{1,j+1}) &\leq t(B_{j+1}) - L_{j+1} \\
t(B_j) &\leq t(B_{j+1}) - L_{j+1} - 1 \\
t(B_j) - \sum_{i=1}^{j} L_i &\leq t(B_{j+1}) - L_{j+1} - \sum_{i=1}^{j} L_i - 1 \\
t(B_j) - \sum_{i=1}^{j} L_i &\leq (t(B_{j+1}) - \sum_{i=1}^{j+1} L_i) - 1.
\end{aligned}$$

Combining these inequalities, we obtain:

$$1 + (t(B_{j+1}) - \sum_{i=1}^{j+1} L_i) - L_{B_j} \leq t(B_j) - \sum_{i=1}^{j} L_i \leq (t(B_{j+1}) - \sum_{i=1}^{j+1} L_i) - 1.$$

Compare this with the output of $g(P)$ for an arbitrary prefix P composed of b simple prefixes (as in Proposition 4.3.2). The tier of the jth block of the translated prefix (where $1 \leq j < b$), denoted by t_j, is between $t_{j+1} - L_j + 1$ and $t_{j+1} - 1$, where L_j is the length of the jth simple prefix. This is exactly the range we have calculated above.

Thus we can find a prefix P such that $g(P)$ gives the final remainder. This means $h(V)$ is well-defined for any $V \in U(A_i)$. $\square$

Lemma 4.3.5. *f and h are inverse functions.*

Proof. We will prove that $h(f(W)) = W$ for all $W \in SV(A_i)$. This is most easily seen if we look at the equivalent definition of f in terms of blocks:

Given a word $W \in SV(A_i)$, we calculate $f(W)$ as follows:

1. If $W \in U(A_i)$, $f(W) = W$.

2. If W is a prefix, then $f(W) = g(W)$.

3. Otherwise, W will begin with a simple prefix or a compound prefix:

   (a) Identify the prefix of W, hereafter P. It is composed of the simple prefixes $P_1, P_2, \ldots, P_p$, where $p \geq 1$.

   (b) Write $g(P)$. This will hereafter be called the "translated prefix", Q. Q will be composed of the blocks $Q_1, Q_2, \ldots, Q_p$. These will hereafter be referred to as the Q-blocks.

   (c) The remainder of W will be the blocks $B_1, B_2, \ldots, B_k$, where $k \geq 1$. For each j from 1 to k, write B_j to the right of B_{j-1} (if $j > 1$), to the left of any Q-blocks of tier less than $t(B_j)$, and to the right of any Q-blocks of tier greater than or equal to $t(B_j)$. For Q-blocks to the right of B_j, increase the tier of each of these blocks by the length of B_j (that is, add $L_{B_j}(n-1)$ to the index on each letter).

   (d) The output is $f(W)$.

Now, to calculate $h(f(W))$ we read the word from right to left (one block at a time). Every block will either be a B-block or a Q-block (in the latter case, the tier will have been increased). We must ensure that h correctly identifies which blocks are B-blocks and which are Q-blocks, and that the original translated prefix Q is all that remains after all blocks have been read. We know from the definition of g, the prefix translation function, that the tier of the rightmost Q-block must be greater than the position of its first (leftmost) letter. Conversely, any block that meets this criterion cannot be a B-block (as that would result in an invalid word after applying h). After the right-most Q-block has been identified, we continue reading blocks from right to left, determining whether the tier of the next block exceeds the tier of the leftmost known Q-block by at least the length of the block. If so, we reduce the tier on all known Q-blocks by the length of the block. This is, again, consistent with the above definition of f as well as the prefix translation function. Every step of h undoes a step of f, but in the reverse order. Thus we eventually do get Q as our remainder and W as the final result. Likewise, we could similarly reason that $f(h(W)) = W$. $\square$

This proves that there exists a bijection between $SV(A_i)$ and $U(A_i)$. Thus, $|R_i| = |SV(A_i)| = |U(A_i)| = \binom{ni+1+w(n-1)}{i}$.

4.4 Summary

We have formulated a way to write the elements of R_i as words over a given alphabet and to identify whether a given word over this alphabet actually represents an element of R_i. We then identified two properties that allowed us to identify words that represent the same element of R_i and to define a unique standard form for words. The number of valid words in standard form is then equal to $|R_i|$. We were able to develop a bijection between the set of valid words in standard form and another set of words (over the same alphabet) that could be enumerated using the occupancy problem. This proved Main Theorem 2.

Chapter 5
Conclusions and future work

We began this thesis by constructing a representation of the free nonsymmetric operad and the principal operad ideal in terms of a composition system. We were then able to present the enumeration problem on spanning sets (R_i) of the principal operad ideal in terms of trees, and give a recursive algorithm for generating these spanning sets. We then introduced a conjectured solution based on empirical data. Known results on the enumeration of complete n-ary plane trees and the n-ary Catalan numbers allowed us to prove Main Theorem 1, which enumerates the elements of R_i where the starting weight is 1. We then used properties of the recursive algorithm, along with known results on plane forests of complete n-ary trees, to generate a recurrence relation for enumerating the elements of R_i in the case where $w > 1$. The conjectured value of $|R_i|$ satisfied this recurrence relation, proving Main Theorem 2. We were also able to develop a non-recursive algorithm for generating the elements of R_i for an arbitrary value of i, without generating duplicate elements. We then presented a word representation for elements of R_i, which led to an additional proof of Main Theorem 2 by the means of occupancy problems.

We provided two different independent proofs of Main Theorem 2. A natural way to connect these two perspectives is not obvious, and may be an interesting area for future research. This may lead to a way to relate forests of n-ary trees to occupancy problems. Notably, the formula for $|R_i|$ for the $w > 1$ case appears to be very similar to the formula for the number of plane forests of $1 + w(n-1)$ complete n-ary trees with a total of i internal nodes, after removing the $\frac{t}{ni+t}$ factor (where t is the number of trees):

$$|R_i| = \binom{(n-1)(w+i)+i+1}{i} = \binom{ni+(w(n-1)+1)}{i}$$

$$
\begin{aligned}
&= \frac{w(n{-}1){+}1}{ni{+}(w(n{-}1){+}1)} \cdot \binom{ni{+}(w(n{-}1){+}1)}{i} \cdot \frac{ni{+}(w(n{-}1){+}1)}{w(n{-}1){+}1} \\
&= F(i, 1{+}w(n{-}1); n) \cdot \frac{ni{+}(w(n{-}1){+}1)}{w(n{-}1){+}1}
\end{aligned}
$$

.

Rearranging this equation, we get the following:

$$|R_i| \cdot (w(n{-}1){+}1) = F(i, 1{+}w(n{-}1); n) \cdot [ni{+}(w(n{-}1){+}1].$$

This result could provide some further insight into the connection between $|R_i|$ and plane forests, as each element of $|R_i|$ can be thought of as a forest of n-ary trees satisfying certain conditions.

Additionally, our scope was limited to principal ideals on nonsymmetric operads. It may be of interest to expand or generalize these results to ideals with more than one generator. In terms of trees, this would begin with an R_0 set that contains more than one sequence. Likewise, there are likely avenues for applying this research to symmetric operads.

References

[1] P.C. Biswal. *Discrete Mathematics and Graph Theory.* PHI Learning, 2015.

[2] M. Bremner and V. Dotsenko. *Algebraic Operads: An Algorithmic Companion.* CRC Press, Taylor & Francis Group, 2016.

[3] M. Bremner and J. Sánchez-Ortega. Quadratic nonsymmetric quaternary operads. *Linear and Multilinear Algebra*, 65(8):1683–1703, 2017.

[4] W. Chu. Elementary proofs for convolution identities of Abel and Hagen–Rothe. *Electronic Journal of Combinatorics*, 17(1):5, 2010.

[5] N. Dershowitz and S. Zaks. The cycle lemma and some applications. *European Journal of Combinatorics*, 11(1):35–40, 1990.

[6] R. Diestel. *Graph Theory*, volume 173 of *Graduate texts in mathematics.* Springer, 2016.

[7] P. Hilger and N. Poncin. Lectures on algebraic operads. `https://orbilu.uni.lu/bitstream/10993/14381/1/LecturesAlgebraicOperads.pdf`, 2011. Course notes, University of Luxembourg.

[8] J. Hopcroft, R. Motwani, and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Pearson, 3rd edition, 2006.

[9] B. Kanti and S. Kumar. *Combinatorics and Graph Theory.* PHI Learning Pvt. Ltd, 2016.

[10] V. Karavirta and C. Shaffer. TDDD86: Data Structures, Algorithms and Programming Paradigms. `http://www.ida.liu.se/opendsa/OpenDSA/Books/TDDD86_2014/html/BinaryTreeFullThm.html`, December 2014. Course notes, Linköping University.

[11] J. Loday and B. Vallette. *Algebraic Operads.* Springer-Verlag Berlin Heidelberg, 2012.

[12] M. Markl, S. Shnider, and J.D. Stasheff. *Operads in Algebra, Topology and Physics.* Mathematical surveys and monographs. American Mathematical Society, 2007.

[13] J.P. May. *The Geometry of Iterated Loop Spaces.* Number no. 271 in Lecture notes in mathematics. Springer-Verlag, 1972.

[14] G. Mohanty. *Lattice Path Counting and Applications.* Probability and mathematical statistics. Academic Press, 1979.

[15] T.V. Narayana. *Lattice Path Combinatorics, with Statistical Applications.* Mathematical expositions. University of Toronto Press, 1979.

[16] M. Renault. Four proofs of the ballot theorem. *Mathematics Magazine*, 80(5):345–352, 12 2007.

[17] F. Roberts and B. Tesman. *Applied Combinatorics, Second Edition.* CRC Press, 2009.

[18] M. Spivey. Combinatorial proofs of two Hagen-Rothe identities in concrete mathematics. `https://mikespivey.wordpress.com/2015/09/04/combinatorial-proofs-of-two-hagen-rothe-identities-in-concrete-mathematics/`, 2015.

[19] H. Wilf. *Algorithms and Complexity.* A. K. Peters, Ltd., 2002.

[20] D. Zeilberger. *Princeton Companion to Mathematics*, chapter Enumerative and Algebraic Combinatorics, pages 550–561. Princeton University Press, Princeton, 2008.