NEW A-INFINITY DIAGONALS FROM CONTRACTIONS OF THE WEIGHTED ASSOCIAHEDRA

by

BO Q. PHILLIPS

A DISSERTATION

Presented to the Department of Mathematics
and the Division of Graduate Studies of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

June 2023

DISSERTATION APPROVAL PAGE

Student: Bo Q. Phillips

Title: New A-infinity Diagonals from Contractions of the Weighted Associahedra

This dissertation has been accepted and approved in partial fulfillment of the requirements for the Doctor of Philosophy degree in the Department of Mathematics by:

Degree awarded June 2023.

DISSERTATION ABSTRACT

Bo Q. Phillips

Doctor of Philosophy

Department of Mathematics

June 2022

Title: New A-infinity Diagonals from Contractions of the Weighted Associahedra

In this paper, we build on the work of Lipshitz, Ozsváth, and Thurston by constructing an algorithm that generates a weighted A_∞-diagonal given a family of contractions of the weighted associahedron complexes. Using this, we exhibit a new weighted A_∞-diagonal and relate it to the unweighted A_∞-diagonal exhibited by Masuda-Thomas-Tonks-Vallete given by so-called "right-moving trees."

To my father

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

The goal of this thesis is to expand on the work of Lipshitz, Ozsváth, and Thurston in [3] to construct new examples of weighted A_∞-diagonals. In their paper, Lipshitz et al prove the existence of such objects but do not provide any examples. Masuda-Thomas-Tonks-Vallete in [7] give an example of an unweighted A_∞-diagonal. In the present paper, we develop an algorithm that generates a new weighted A_∞-diagonal that agrees with the example given in [7].

What is meant precisely by an A_∞-diagonal and a weighted A_∞-diagonal is elaborated on in Section 3. The motivation to find an A_∞-diagonal is to define a tensor product of A_∞-algebras, and from a weighted A_∞-diagonal define accordingly a tensor product of "deformed" A_∞-algebras.

We begin with a brief review of A_∞-technology. A_∞-algebras are algebras over the A_∞-operad, also known as the Stasheff operad. Informally, an operad is a collection of *operations* stratified by the number of inputs; i.e. a collection $O(1)$ of *unary* operations, a collection $O(2)$ of *binary* operations, and so forth. In practice, there is a background symmetric monoidal category and the "collections" $O(n)$ are objects in this category. The A_∞-operad has two main guises: one in topological spaces and one in chain complexes. As an operad of topological spaces, $O(n)$ is the CW complex K_n called the $(n-2)$-dimensional *Stasheff polytope* or *generalized associahedron.* As an operad of chain complexes, $O(n)$ is the cellular chain complex $C_*(K_n)$. If we have a ground ring R, we use $C_*(K_n; R)$.

The cells of K_n correspond to planar rooted trees T with n leaves (not counting the root) and with no 2-valent vertices. K_n has the characteristic property that the boundary of its single top dimensional cell is the union of products of lower dimensional associahedra, i.e.

$$\partial K_n = \bigcup_{i+j=n+1} \bigcup_{\ell=1}^{i} K_i \times K_j$$

where $2 \leq i, j \leq n-1$. The ℓ-th copy of $K_i \times K_j$ is the subcomplex of K_n whose cells are n-leaf trees formed by grafting a tree with j leaves onto the ℓ input of a tree with i leaves. The tree Ψ_n corresponding to the top dimensional cell is an indecomposable n-ary operation, whose leaves are the inputs and whose root is the output, shown in Figure 1.1. Its boundary cells consist of all ways to form an n-ary operation as a composition of two lower arity operations.

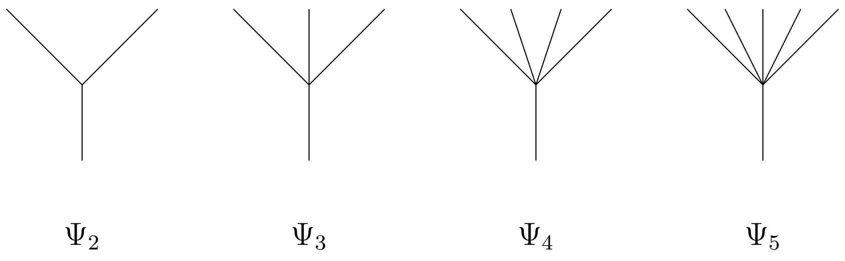


Figure 1.1: The corolla trees Ψ_n for $n \leq 5$.

For example, K_2 is isomorphic to a point. Its single cell corresponds to the unique planar rooted tree Ψ_2 on two leaves, which graphically represents some abstract binary operation μ_2. K_3 is a 1-dimensional CW complex with two 0-cells, corresponding to the two ways of forming a 3-leaf tree by grafting Ψ_2 onto itself. These two trees in turn represent the two ways to form a 3-fold product from a single binary operation μ_2 and formal input values a, b, c, namely $\mu_2(\mu_2(a, b), c)$ and $\mu_2(a, \mu_2(b, c))$. The top-dimensional cell of K_3, corresponding to the tree Ψ_3, represents a formal indecomposable ternary operation μ_3; one can interpret this cell as a path between the two bracketings of μ_2. This is saying that μ_2 is associative "up to a homotopy" mediated by the higher operation μ_3. See Figure 1.2 for an illustration of the correspondence described. Note that the leaves of the trees in Figure 1.2 are labeled with formal inputs a, b, c to better illustrate the correspondence. Strictly speaking, the leaves of all trees in K_n are unlabeled.

One usually suppresses the μ_2 and writes $(ab)c$ and $a(bc)$ for the two bracketings of μ_2 just described. In general, the 0-cells of K_n correspond to binary trees T, i.e. trees whose nonleaf, nonroot vertices all have degree 3. Such trees are in bijection with bracketings of an n-fold binary product, i.e. all ways to form an n-ary operation out of a composite of binary operations. The fact that K_n is contractible for all

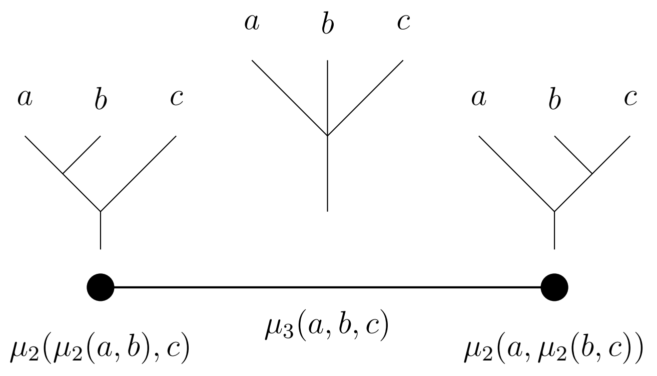


Figure 1.2: The space K_3, the trees corresponding to its cells, and the operations corresponding to those trees.

n implies in particular that all of these bracketings are path-connected; i.e. μ_2 is *homotopy associative.* For more information on the Stasheff polytopes, see [10]. For more information on the correspondence to bracketings and trees, see [11].

A_∞-algebras in the category of chain complexes can be conceptualized as "homotopical enhancements" of differential graded algebras (DGA's). A DGA is a monoid in chain complexes; it consists of a chain complex C_* and an associative multiplication $\mu : C_* \otimes C_* \to C_*$. An A_∞-algebra consists of a chain complex A_* and multiplication $\mu_2 : A_* \otimes A_* \to A_*$ that is only assumed to be associative up to homotopies governed by higher order operations. The failure of μ_2 to be associative is measured by some ternary operation $\mu_3 : A_* \otimes A_* \otimes A_* \to A_*$ which strictly speaking is not a morphism of chain complexes. More precisely, μ_3 is a graded map of degree 1, and the difference between the two bracketings of μ_2 is the differential of μ_3 in the morphism chain complex $\mathbf{Ch}(A_*^{\otimes 3}, A_*)$. From this point of view, one can interpret the correspondence in Figure 1.2 another way: if $(A_*, \{\mu_n\}_{n\geq 2})$ is an A_∞-algebra and with $a, b, c \in A_*$, then $\mu_2(\mu_2(a,b),c)$ and $\mu_2(a,\mu_2(b,c))$ are elements of A_* whose difference is

$$\partial\mu_3(a,b,c) \pm \mu_3(\partial a,b,c) \pm \mu_3(a,\partial b,c) \pm \mu_3(a,b,\partial c)$$

where the signs depend on the degrees of a, b, c. In this way, one can think of μ_2 as "homologically associative", i.e. the induced operation $HA_* \otimes HA_* \to HA_*$ on the homology of A_* is associative. The operation μ_3 may in turn satisfy a kind of "higher associativity" condition (sometimes called the "pentagon identity", see Section 7 of [1]), and its failure to satisfy this is measured by some 4-ary operation μ_4, and so forth. An A_∞-algebra on A_* is a map of operads $C_*(K_\bullet) \to \mathbf{Ch}(A_*^{\otimes \bullet}, A_*)$, which amounts to maps $f_n : C_*(K_n) \to \mathbf{Ch}(A_*^{\otimes n}, A_*)$ compatible with the operad structures. The operations μ_n correspond to the elements $f_n(\Psi_n)$, and it turns out that these data alone uniquely determine the maps f_n. More information on A_∞-algebras can be found in [2].

An A_∞-diagonal is a certain kind of map of operads $C_*(K_\bullet) \to C_*(K_\bullet) \otimes C_*(K_\bullet)$. It consists of chain maps $\Gamma_n : C_*(K_n) \to C_*(K_n) \otimes C_*(K_n)$ that are compatible with the operad structures. It turns out that these are determined by the classes $\gamma_n = \Gamma_n(\Psi_n)$, so an A_∞-diagonal can be concretely realized as linear combinations γ_n of tensor products of planar trees on n leaves, one for each dimension $n \geq 2$. The boundary conditions of K_n imply a recursive construction of γ_n in terms of the lower dimensional combinations. As aforementioned, one can use this ultimately to construct a tensor product of A_∞-algebras.

More generally, one can study deformations of an A_∞-algebra A_*. If A_* is a chain complex over a ring R, then a deformation of A_* is a chain complex $B_* = A_* \otimes_R R[[t]]$ over the ring $R[[t]]$ equipped with the structure of an A_∞-algebra in the category of chain complexes over $R[[t]]$. The operations of B_* are now stratified over formal power series

$$\mu_n = \sum_{w=0}^{\infty} \mu_n^w t^w$$

The operation μ_n^w should be thought of as an n-ary operation with a certain nonnegative integer weight w attached to it. Note that the operations μ_n^0 of weight 0 induce an A_∞-algebra structure on the chain complex A_*. Thus one should think of A_* with these operations as the *undeformed* A_∞-algebra, and the positive weight factors μ_n^w of the n-ary operation of B_* as "twisting" the unweighted n-ary operation of A_*. For more information on deformations of A_∞ algebras, see [3].

Just as for ordinary A_∞-algebras, the operations μ_n^w correspond to the top cells of certain CW complexes, namely the complexes called weighted associahedra X_n^w. These are a generalization of the Stasheff polytopes. As a space, X_n^w is an $(n+2w-2)$-dimensional CW complex whose cells correspond to *weighted planar trees*; i.e. n-leaf trees whose non-root vertices have a nonnegative integer weight assigned to them, and such that the sum of all these weights is w. We will often suppress 0-weights and depict such vertices as unweighted. In this context, a *leaf* means any weight-0, degree-1 vertex aside from the root. Degree-1 vertices with positive weight are referred to as *lollipops*. We will also use the term *k-lollipop* to refer to the rooted planar tree Ψ_0^k with only a single nonroot vertex of weight k. Figure 1.3 shows some examples of these trees for small values of k.

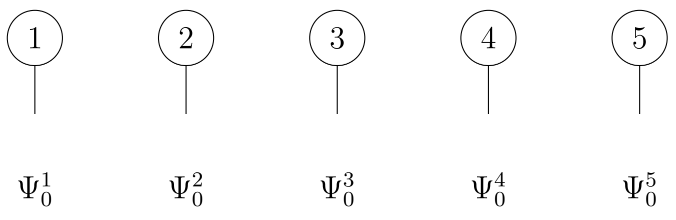


Figure 1.3: The k-lollipops for $k \leq 5$.

The boundary of X_n^w satisfies a similar condition to that of the unweighted associahedron K_n; i.e. it is a union of products $X_i^u \times X_j^v$ of lower-dimensional, lower-weight spaces X_i^u, X_j^v. For example, X_0^1 is isomorphic to a point; its single cell corresponds to the *1-lollipop*, the rooted planar tree Ψ_0^1 with only a single nonroot vertex of weight 1. This tree represents a nullary operation μ_0^1 which should be imagined as a fixed constant term in the algebra. Figure 1.4 shows the space X_1^1 along with the trees and bracketings corresponding to its cells. Once again, the formal input a is included on trees to better illustrate the correspondence. Strictly speaking, the leaves in Figure 1.4 marked by a are weight-0 vertices with no other label.

One can interpret the above correspondence as follows. If $(B_*, \{\mu_n^w\}_{n,w})$ is a deformation of the A_∞-algebra $(A_*, \{\mu_n^0\}_n)$ and a is any element of A_*, then μ_0^1 is an element of A_* such that $\mu_2^0(\mu_0^1, a)$ and $\mu_2^0(a, \mu_0^1)$ differ by the boundary terms involving $\mu_1^1(a)$; so μ_0^1 is a central element of the DGA $H(A_*)$ with multiplication induced by μ_2^0.

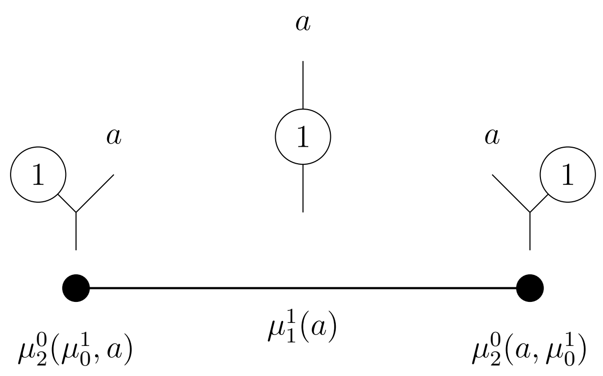


Figure 1.4: The space X_1^1, the weighted trees corresponding to its cells, and the operations corresponding to those trees.

In a way analogous to that of ordinary A_∞-algebras, one can define a tensor product of deformations of A_∞-algebras using a notion called a **weighted A_∞-diagonal**. These were introduced by Lipshitz et al in [3]. Such a diagonal is determined by terms $\gamma_n^w \in C_*(X_n^w) \otimes C_*(X_n^w)$ satisfying some compatibilities. In this context, γ_n^w will be a linear combination of order-2 tensors whose factors are weighted planar rooted trees with n leaves and total weight w, i.e. generators of the cellular chain complex $C_*(X_n^w)$ as a graded R-module. The boundary property of X_n^w implies a recursive relationship among the γ_n^w that can be exploited to generate them from a set of initial terms.

In the present paper, we develop an algorithm that generates weighted A_∞-diagonals from a particular contraction of the weighted associahedron chain complex $C_*(X_n^w)$. We first give a contraction of $C_*(K_n)$ to the subcomplex generated by a chosen basis element R_n in dimension 0. This is done by first putting a particular filtration on $C_*(K_n)$ such that the component with lowest filtration degree is precisely the subcomplex generated by R_n. Next we define an algorithm on trees in K_n and use this to produce a map α on $C_*(K_n)$ that is chain homotopic to the identity and lowers the filtration degree of generators with nonminimal filtration degree. Finally, we show that finitely many applications of α produces the desired contraction of $C_*(K_n)$.

In the subsequent sections, we show that there is a series of chain complex inclusions $C_*(K_w) \subset C_*(X_0^w) \subset C_*(X_n^w)$. For the chain complex $C_*(X_0^w)$, there is a filtration $F_0 \subset F_1 \subset ...$ such that F_0 corresponds to $C_*(K_w)$. By constructing an algorithm on weighted trees in X_0^w, we produce a map ξ that is chain homotopic to the identity of $C_*(X_0^w)$ and lowers the filtration degree of generators outside $C_*(K_w)$. We show that finitely many applications of ξ yields a contraction of $C_*(X_0^w)$ to the subcomplex $C_*(K_w)$, thus reducing the contraction problem for $C_*(X_0^w)$ to that of the unweighted complex $C_*(K_w)$. The general case of $C_*(X_n^w)$ is handled simiarly. We define a filtration for $C_*(X_n^w)$ whose minimal filtration degree corresponds to $C_*(X_0^w)$. We construct an algorithmically defined map ζ and show that ζ reduces the filtration degree of trees outside $C_*(X_0^w)$. By combining finitely many iterations of ζ with the contractions described for $C_*(X_0^w)$ and $C_*(K_w)$, we obtain a systematic contraction of $C_*(X_n^w)$ and ultimately a formula for generating new weighted A_∞ diagonals.

1.1 Organization of this paper

The organization of this paper is as follows. Section 2 covers all preliminary definitions about trees, the chain complexes $C_*(X_n^w)$, and their relation to each other. Several short lemmas about trees are included here that will be useful in the rest of the paper.

Section 3 gives more exposition on A_∞ diagonals and how these arise from the weighted associahedron complexes. We prove here how one can inductively construct an A_∞ diagonal or a weighted A_∞ diagonal from a chosen family of contractions of the chain complexes $C_*(K_n)$ or $C_*(X_n^w)$ respectively. The remainder of this section is devoted to developing a contraction of the unweighted associahedron chain complexes $C_*(K_n)$. This is accomplished by putting a filtration on $C_*(K_n)$ and defining an algorithm on unweighted trees which produces a chain homotopy from the identity map on $C_*(K_n)$ to the inclusion of the subcomplex of next-lowest degree. Repeated application of this algorithm is shown to yield a contraction from $C_*(K_n)$ to a certain subcomplex in dimension 0 generated by a single basis element.

Section 4 describes a similar procedure for the weighted associahedron complexes $C_*(X_n^w)$ in the special case that $n = 0$. We show that these chain complexes contain a particular subcomplex F isomorphic to the unweighted associahedron complex $C_*(K_n)$. An algorithm is then described which when applied sufficiently many times to any element in $C_*(X_0^w)$ will reduce it to an element in the subcomplex F. One may then combine this procedure with the procedure in Section 3 to obtain a total contraction of the chain complex $C_*(X_0^w)$.

Section 5 describes the general procedure for $C_*(X_n^w)$ when n, w are both positive. We show that this chain complex contains another special subcomplex G, this time isomorphic to $C_*(X_0^w)$. In this case, an algorithm is developed which after multiple iterations reduces any element in $C_*(X_n^w)$ to an element in G. When combined with the results of Sections 3 and 4, this produces a total contraction of $C_*(X_n^w)$ and ultimately the desired A_∞ diagonal.

Finally, Section 6 is devoted to an implementation of these algorithms written in SageMath, an open-source mathematics software system based on Python. This includes the source code along with the initial terms of a novel weighted A_∞ diagonal.

In what follows, all chain complexes are assumed to use $\mathbb{Z}/2$ coefficients or more generally R coefficients where R is any ring of characteristic 2. This is a matter of convenience and does not affect the generality of the algorithms that follow. For more information on how the differentials behave when sign terms are introduced, see Example 5.7 in [5].

2 Trees and the Associahedra

We first introduce some elementary notions about trees. In what follows, all trees are assumed to be rooted with a chosen embedding in the plane, so that it is meaningful to speak of "left" and "right" directions within each tree. The leaf vertices of each tree will be numbered from left to right and hereafter referred to as the *inputs* of the tree. Any vertex which is neither a leaf nor the root will be referred to as an *internal vertex*.

Let S and T be trees with m and n inputs respectively. We define composition in the following way. For $1 \leq k \leq n$, $T \circ_k S$ is the tree obtained by gluing the root edge of S to the k-th input edge of T and renumbering the inputs of the resulting tree accordingly. An illustration of this composition operation is shown in Figure 2.1. Note that this tree $T \circ_k S$ has $m + n - 1$ inputs for any k.

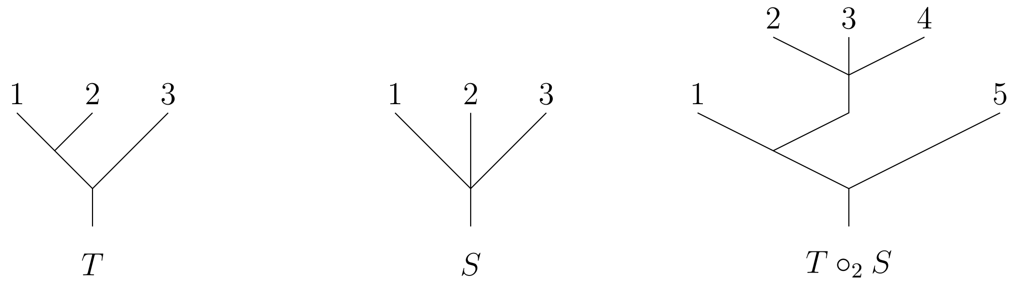


Figure 2.1: An example of tree composition.

Given a tree T, a *contraction* of T is a new tree T' obtained by contracting exactly one non-input, non-root edge in T and merging its two vertices into a single vertex v. Conversely, T is called an *expansion* of T' at the vertex v. In such an expansion, we will generally regard the "deeper" vertex (i.e. the one farther from the root) as the new vertex, while referring to the other vertex as v. Note that a tree can only be expanded if it has at least one vertex of degree greater than 3. For each $n \geq 2$, there is a unique n-leaf tree with exactly one internal vertex; we call such a tree a *corolla* and denote it by Ψ_n. Figure 2.2 shows the corolla Ψ_4 and all of its possible expansions.

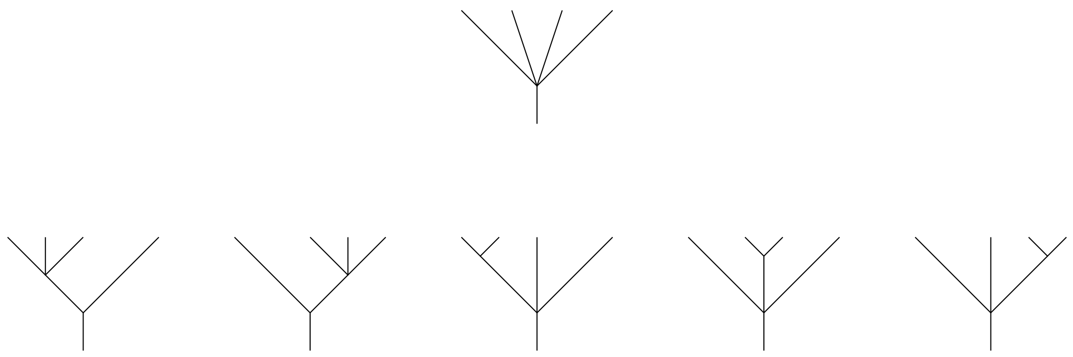


Figure 2.2: Ψ_4 and its expansions.

The following lemma is an elementary observation about tree expansion that will be useful in subsequent sections.

Lemma 2.1 *Let T' be a contraction of a tree T, formed by merging two adjacent vertices u, v in T into a single vertex v' in T'. Then*

$$[\deg(u) - 2] + [\deg(v) - 2] = \deg(v') - 2$$

For $n \geq 2$, there is an $(n-2)$-dimensional CW complex K_n called the *associahedron* of order n whose cells correspond to trees with n inputs and no 2-valent vertices. These spaces were first described by Stasheff in [10]. Shown in Figure 2.3 is the space K_4 and the trees corresponding to its cells.

The top-dimensional cell of K_n corresponds to the corolla Ψ_n, and the 0-cells correspond to all binary trees on n leaves. More generally, if T corresponds to some cell in K_n, then the boundary cells correspond to all possible expansions of T. In particular, the boundary cells of the cell corresponding to Ψ_n correspond to all composites of the form $\Psi_i \circ_k \Psi_j$ where $i + j = n + 1$. Recursively, one can show that any tree T can be written as a (not necessarily unique) composition of corollas.

In [11], Tamari showed that the set of binary trees with n leaves forms a poset whose Hasse diagram is precisely the 1-skeleton of K_n. The partial order corresponds to the transitive closure of the 1-cell relation on the 0-cells of K_n which is defined as follows. A 1-cell corresponds to a tree T with exactly one internal vertex v of degree 4, all other internal vertices having degree 3. There are exactly two expansions of

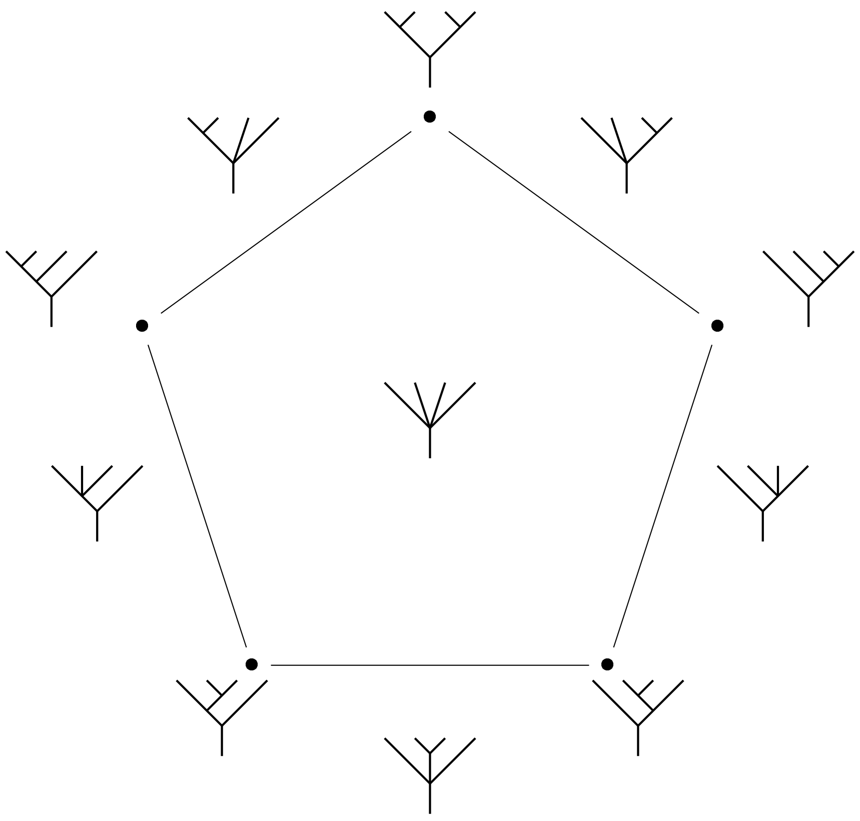


Figure 2.3: The trees corresponding to the cells of K_4.

such a tree: the tree W wherein the branches of the new vertex are the 1st and 2nd branches of v (read from left to right), or the tree S wherein the branches of the new vertex are the 2nd and 3rd branches of v. In this case, we say that $W \leq S$ under the Tamari order. One can think of this as saying that S is obtained from W by "shifting" the middle branch from the left side of v to the right.

The relation just described always has a unique maximal element, the *fully right-associated tree*, which we denote by R_n or just R if n is clear from context. Likewise, there is a unique minimal element, the *fully left-associated tree*, denoted by L_n or just L.

By the above discussion, we can think of the cellular chain complex $C_*(K_n)$ (with $\mathbb{Z}/2$ coefficients) as the graded vector space generated by the set of n-leaf trees. The generators in $C_k(K_n)$ are the trees with $n-k+1$ internal vertices. The boundary map sends a tree T to the sum of all expansions of T. The composition operations $\circ_i$ described above induce chain complex maps

$$C_*(K_{n+i-j}) \otimes C_*(K_{j-i+1}) \xrightarrow{\phi^n_{i,j}} C_*(K_n)$$

henceforth referred to as *stacking operations.* These are defined on generators $T \otimes S$ by the formulas $\phi^n_{i,j}(T \otimes S) = T \circ_i S$. Henceforth we will interchangeably use $\phi^n_{i,j}$ and $\circ_i$ for the stacking operations on chain complexes, depending on which is more convenient. For any pair of trees T, S, the sum of terms $T \circ_k S$ where k ranges over the inputs of T will also be denoted $T \circ S$; i.e. if T has $n+i-j$ leaves and S has $j-i+1$ leaves, then

$$T \circ S = \sum_{k=1}^{n+i-j} T \circ_k S$$

One can extend the maps $\circ_k$ to tensor products of trees in the following way. For pure tensors $T_1 \otimes T_2$ in $C_*(K_n)^{\otimes 2}$ and $S_1 \otimes S_2$ in $C_*(K_m)^{\otimes 2}$, let

$$(T_1 \otimes T_2) \circ_k (S_1 \otimes S_2) = (T_1 \circ_k S_1) \otimes (T_2 \circ_k S_2)$$

Extending this definition bilinearly gives operations $\circ_k : C_*(K_n)^{\otimes 2} \otimes C_*(K_m)^{\otimes 2} \to C_*(K_n) \otimes C_*(K_m)$. We will be most interested in the case when $n = m$. The map $\circ$ also extends to such an operation defined on pairs of pure tensors as follows.

$$(T_1 \otimes T_2) \circ (S_1 \otimes S_2) = \sum_{k=1}^{n} (T_1 \otimes T_2) \circ_k (S_1 \otimes S_2)$$

Throughout the rest of this paper, we will be interested in *weighted trees*; that is, trees whose non-root vertices are assigned nonnegative integer weights such that no 2-valent vertex has weight 0. If T is a weighted tree, the *total weight* of T (or simply *weight*) is the sum of the weights of each of its vertices. The *inputs* of T are the leaf vertices which have weight 0. Figure 2.4 shows an example of a weight-10 tree on 2 inputs. For convenience, we choose to depict weight-0 vertices as unweighted. Note that any unweighted tree can be regarded as a weight-0 tree on the same number of inputs.

In the context of weighted trees, we will use a slightly different convention for

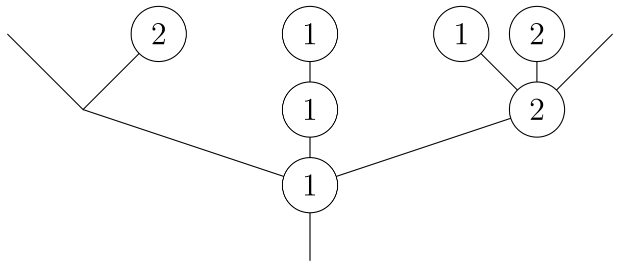


Figure 2.4: A 2-input, weight-10 tree.

internal vertices. By an *internal vertex*, we mean either a non-leaf, non-root vertex or a leaf vertex whose weight is at least 2. The tree in Figure 2.4 has 6 internal vertices (three of weight 2, two of weight 1, one of weight 0).

For each pair of integers $n, w \geq 0$ with $w \geq 1$ or $w = 0$ and $n \geq 2$, there is a unique n-input, weight-w tree with exactly one internal vertex called the *weighted corolla* and denoted Ψ_n^w. Note that Ψ_n^0 is the unweighted corolla as defined previously. The 0-input corollas are called *lollipops* and we will refer to the corolla Ψ_0^w as the *w-lollipop.* If T is any weighted tree, we will also use the term lollipop to refer to its non-input leaves.

The operations of contracting and expanding for unweighted trees generalize to the weighted case. If T is a weighted tree, a *contraction* of T is a new weighted tree T' that results from contracting exactly one non-input, non-root edge and merging its two vertices into a single vertex v. The weight of v is the sum of the weights of the original two vertices. Conversely, T is called an *expansion* of T' at the vertex v. Note the condition that T be a weighted tree; i.e. expanding T' cannot create a 2-valent, weight-0 vertex nor can it create a new input (although it may create a new non-input leaf). Figure 2.5 shows Ψ_1^3 and all of its expansions.

We state here an analog for Lemma 2.1 that will be useful in the context of weighted trees. The main difference is that when expanding at a positive weight vertex v', the original weight must be redistributed on the vertices of the newly created edge. Note however that when $\mathrm{wt}(v') = 0$, the following statement reduces to that of Lemma 2.1.

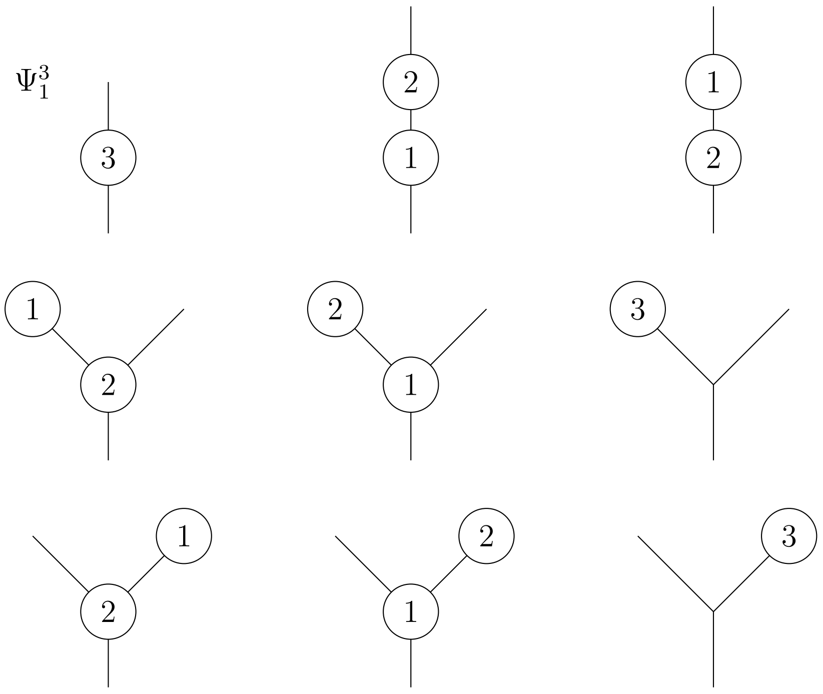


Figure 2.5: Ψ_1^3 and all of its expansions.

Lemma 2.2 *Let T' be a contraction of a weighted tree T formed by merging two adjacent vertices u, v in T into a single vertex v' in T'. Then*

$$[\deg(u) - 2] + wt(u) + [\deg(v) - 2] + wt(v) = \deg(v') - 2 + wt(v')$$

As was the case for unweighted trees, weighted trees are associated with a family of spaces X_n^w called the *weighted associahedra*. For $n, w \geq 0$ with $w \geq 1$ or $w = 0$ and $n \geq 2$, X_n^w is a CW complex of dimension $n + 2w - 2$. Its cells correspond to weight-w trees on n inputs. A weight-w tree with n input leaves and k non-input, non-root vertices corresponds to a $(n + 2w - k - 1)$-dimensional cells of X_n^w. Unlike K_n, X_n^w is generally not a polytope. The top-dimensional cell corresponds to the weighted corolla Ψ_n^w. If T is an n-input, weight-w tree, the boundary cells of the cell corresponding to T correspond to all expansions of T. See Section 8 of [3] for more information on these spaces. Figure 2.6 shows the space X_2^1 and the trees corresponding to its cells.

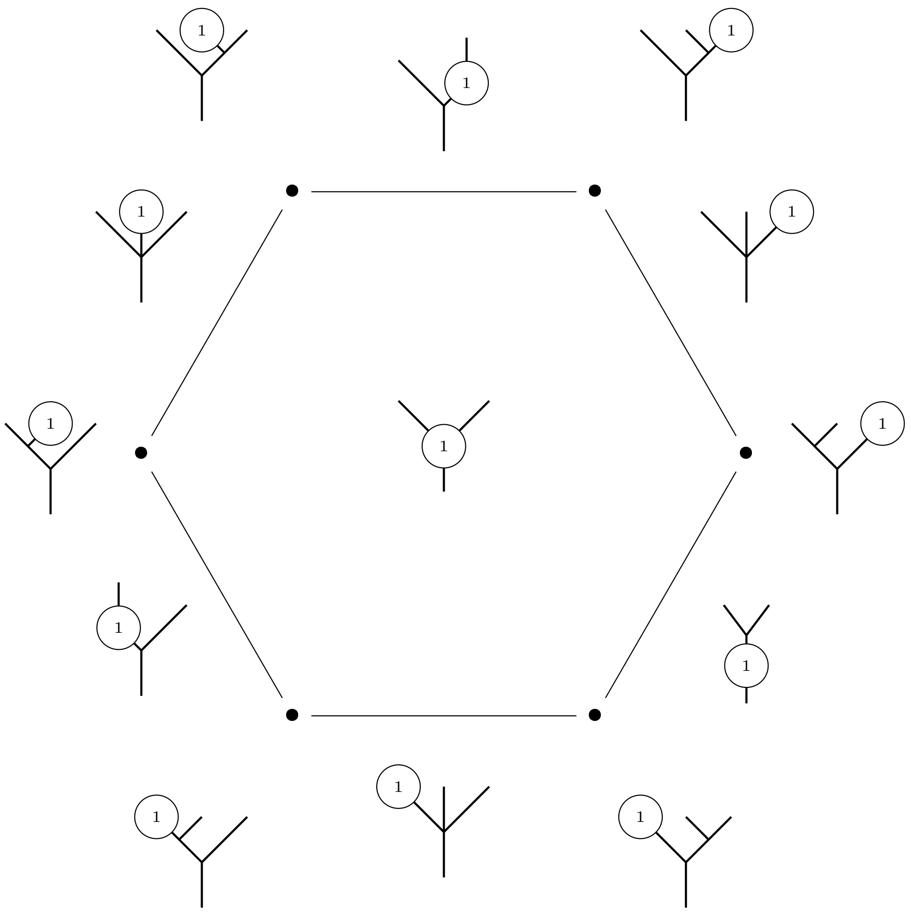


Figure 2.6: The weighted trees corresponding to the cells of X_2^1

Observe that the generators of $C_0(X_n^w)$ are the binary trees all of whose positive weight vertices are weight-1 lollipops. In other words, it is isomorphic as a vector space to a direct sum of copies of $C_0(K_{n+w})$, one for each linear arrangement of the leaves and lollipops. In general, there is no analog for weighted trees of the Tamari ordering on unweighted trees. However, it will still be useful to distinguish a basis element R_n^w of $C_0(X_n^w)$ which we call the *fully right-associated weighted tree.* Its underlying unweighted tree (i.e. the tree with all weights forgotten) is R_{n+w} and its rightmost w leaves are weight-1 lollipops.

In what follows, it will be convenient to put an order on the internal vertices of a (possibly weighted) tree. It is well-known that the vertices of a tree graph can be ordered according to *left-right depth-first search.* If T is a planar tree, the left-right depth-first search algorithm explores the vertices of the tree by starting at the root and exploring along leftmost branches as deeply as possible. After reaching a leaf, the algorithm backtracks to the deepest vertex v along its path at which there is an unexplored branch and continues down the leftmost such branch of v. This algorithm is more precisely termed *pre-order left-right depth-first search*; see Chapter 3 of [12] to contrast this with other similar search algorithms. Hereafter we will write simply the n-th vertex of T to mean the n-th vertex found in a left-right depth-first search of T as described above, restricted to the internal vertices of T. An example of ordering the internal vertices of a weighted tree is shown in Figure 2.7.

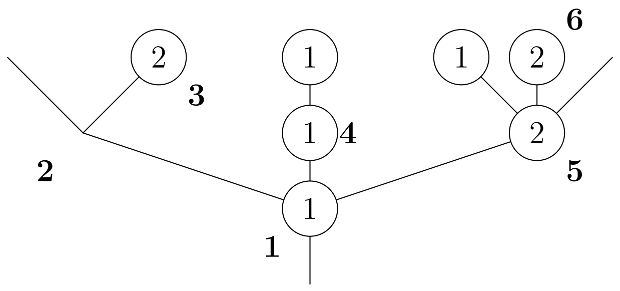


Figure 2.7: The internal vertices of a tree ordered by left-right depth-first search.

3 Associahedron Diagonals and Tree Diagonals

We briefly review the structure of operads in the category of chain complexes over $\mathbb{Z}/2$. A (nonsymmetric) *operad* $O(\bullet)$ in chain complexes consists of a sequence of chain complexes $O(1), O(2), O(3), ...$ along with composition morphisms $O(n) \otimes O(m) \xrightarrow{\circ_i} O(n+m-1)$ for $1 \leq i \leq n$ such that certain diagrams commute. As mentioned before, one should think of $O(n)$ as a collection of n-ary operations and the composition morphism $O(n) \otimes O(m) \xrightarrow{\circ_i} O(n+m-1)$ as nesting an m-ary operation inside the i-th input of an n-ary operation. There are natural associativity conditions one wishes to have when composing three or more operations together, and these follow from the commutative diagrams in the definition of such an operad. See [6] or [4] for a more precise definition and general overview of operads in an arbitrary symmetric monoidal category.

The *standard differential graded* A_∞*-operad* consists of the cellular chain complexes $C_*(K_n)$, and the composition morphism $\circ_i : C_*(K_n) \otimes C_*(K_m) \to C_*(K_{n+m-1})$ is the stacking morphism $\phi^{n+m-1}_{i,n+i-1}$ described in Section 2. These same tree compositions induce an operad structure on the chain complexes $C_*(K_n) \otimes C_*(K_n)$; its composition morphisms are given by

$$C_*(K_n)^{\otimes 2} \otimes C_*(K_m)^{\otimes 2} \xrightarrow{\cong} (C_*(K_n) \otimes C_*(K_m))^{\otimes 2} \xrightarrow{\phi^{n+m-1}_{i,n+i-1} \otimes \phi^{n+m-1}_{i,n+i-1}} C_*(K_{n+m-1})^{\otimes 2}$$

where the first morphism is the isomorphim that permutes the middle two tensor factors.

A *differential graded* A_∞*-algebra* consists of a chain complex A_* and an operad morphism from $C_*(K_\bullet)$ to $\mathbf{Ch}(A_*^{\otimes \bullet}, A_*)$, the endomorphism operad of A_*. Concretely, this consists of chain complex morphisms $\omega_n : C_*(K_n) \to \mathbf{Ch}(A_*^{\otimes n}, A_*)$ such that the following diagram commutes for all suitable index combinations n, i, j.

$$\begin{tikzcd}
C_*(K_n) \otimes C_*(K_m) \arrow[r, "\phi^{n+m-1}_{i,n+i-1}"] \arrow[d, "\omega_n \otimes \omega_m"] & C_*(K_{n+m-1}) \arrow[d, "\omega_{n+m-1}"] \\
\mathbf{Ch}(A_*^{\otimes n}, A_*) \otimes \mathbf{Ch}(A_*^{\otimes m}, A_*) \arrow[r, "\epsilon_i"] & \mathbf{Ch}(A_*^{\otimes n+m-1}, A_*)
\end{tikzcd}$$

The morphism ϵ_i is defined on pure tensors by

$$\epsilon_i(f \otimes g) = f \circ \left(\mathrm{id}^{\otimes i-1} \otimes g \otimes \mathrm{id}^{\otimes n-i}\right)$$

The operad structure of $C_*(K_\bullet)$ implies that the morphisms ω_n are determined by $\mu_n = \omega_n(\Psi_n)$, the image of the top-dimensional generator of $C_*(K_n)$. Concretely, $\mu_n : A_*^{\otimes n} \to A_*$ is a degree $n-2$ map of graded modules which we refer to simply as the *n-th operation* of A_*. As mentioned, the operations μ_n completely determine the A_∞-structure of A_*, and one can give an equivalent characterization of an A_∞-algebra in terms of such maps. See [2] for more details.

An *associahedron diagonal* is a "nondegenerate" morphism between the operads $C_*(K_\bullet)$ and $C_*(K_\bullet) \otimes C_*(K_\bullet)$. That is, it consists of degree-preserving chain maps

$$C_*(K_n) \xrightarrow{\Gamma^n} C_*(K_n) \otimes C_*(K_n)$$

which commute with the stacking operations (i.e. the following diagram commutes for all suitable index combinations n, i, j).

$$\begin{tikzcd}
C_*(K_{n+i-j}) \otimes C_*(K_{j-i+1}) \arrow[r, "\Gamma^{n+i-j} \otimes \Gamma^{j-i+1}"] \arrow[dd, "\phi^n_{i,j}"] & C_*(K_{n+i-j})^{\otimes 2} \otimes C_*(K_{j-i+1})^{\otimes 2} \arrow[d, "\cong"] \\
& (C_*(K_{n+i-j}) \otimes C_*(K_{j-i+1}))^{\otimes 2} \arrow[d, "\phi^n_{i,j} \otimes \phi^n_{i,j}"] \\
C_*(K_n) \arrow[r, "\Gamma^n"] & C_*(K_n)^{\otimes 2}
\end{tikzcd}$$

where the top right isomorphism permutes the middle tensor factors. Moreover, for nondegeneracy, we assume that Γ^2 is the standard isomorphism $\mathbb{Z}/2 \cong \mathbb{Z}/2 \otimes_{\mathbb{Z}/2} \mathbb{Z}/2$ (recall that K_2 is a single point). Because of the stacking condition and the fact that each tree is a composition of corollas, it follows that such a morphism is determined by the values $\gamma_n = \Gamma^n(\Psi_n)$. This motivates the following equivalent characterization.

An *associahedron tree diagonal* is a collection of $(n-2)$-chains $\gamma_n \in C_*(K_n) \otimes C_*(K_n)$ for each $n \geq 2$ such that $\gamma_2 = \Psi_2 \otimes \Psi_2$ and

$$\partial \gamma_n = \sum_{i+j=n+1} \gamma_i \circ \gamma_j.$$

As mentioned above, associahedron diagonals are determined by an associahedron tree diagonal, and the converse is also true. For a proof of this, see Lemma 2.19 in [3].

Saneblidze and Umble give an example of an explicit associahedron diagonal morphism in [9]. Masuda-Thomas-Tonks-Vallete give an example of an associahedron tree diagonal in [7] using a geometric construction on K_n. This second example can be alternatively characterized by taking γ_n to be the sum of all pairs of trees which are "right-moving" in dimension $n-2$. See Example 2.21 in [3] for more information on this description.

Associahedron diagonals allow us to define tensor products of A_∞-algebras in the following way: given actions $\alpha_n : C_*(K_n) \to \mathbf{Ch}_R(A^{\otimes n}, A)$ and $\beta_n : C_*(K_n) \to \mathbf{Ch}_R(B^{\otimes n}, B)$, define the action on $A \otimes B$ by the composite

$$C_*(K_n) \xrightarrow{\Gamma^n} C_*(K_n)^{\otimes 2} \xrightarrow{\alpha_n \otimes \beta_n} \mathbf{Ch}_R(A^{\otimes n}, A) \otimes \mathbf{Ch}_R(B^{\otimes n}, B) \to \mathbf{Ch}_R((A \otimes B)^{\otimes n}, A \otimes B).$$

Our next goal is to describe the analog of each of the aforementioned topics in the weighted case. For this, we need the notion of so-called *curved A_∞-algebras.* Essentially these allow for an extra 0-ary operation μ_0; ordinary A_∞-algebras are precisely those curved A_∞-algebras with $\mu_0 = 0$. For more information on curved A_∞-algebras, see section 4 of [3].

Given an A_∞-algebra A_* with operations $\{\mu_n^0\}_{n \geq 2}$, a *1-parameter deformation* of A_* is a curved A_∞-algebra structure on $B_* = A_* \otimes_R R[[t]]$ whose operations $\mu_n : B_*^{\otimes n} \to B_*$ decompose as

$$\mu_n = \sum_{w=0}^{\infty} \mu_n^w t^w$$

The nonnegative integer w is called the *weight* of the operation μ_n^w, and A_* together with the operations μ_n^0 is called the *undeformed algebra* corresponding to B_*. See Section 4 of [3] for more discussion.

The analog of tree diagonals for deformations of A_∞-algebras is a *weighted associahedron tree diagonal.* It consists of elements $\gamma_n^w \in C_*(X_n^w) \otimes C_*(X_n^w)$ in dimension $n + 2w - 2$ such that

$$\partial \gamma_n^w = \sum_{i+j=n+1} \sum_{u+v=w} \gamma_i^u \circ \gamma_j^v$$

.

for each pair $n, w \geq 0$. By convention $\gamma_0^0 = \gamma_1^0 = 0$. The element γ_0^1 is called the *seed* of the diagonal, which we will always take to be the generator of $C_*(X_0^1) \otimes C_*(X_0^1) \cong \mathbb{Z}/2$. More discussion of seeds can be found in Section 6 of [3].

3.1 New Results

In what follows, a *chain homotopy contraction* (or sometimes simply *contraction*) on a chain complex A_* will mean a chain homotopy from id_{A_*} to a given map $\pi : A_* \to A_*$ which projects onto a subcomplex of A_* that is 1-dimensional (as an R-module) and concentrated in dimension 0. One should think of this as analogous to a contraction of a CW complex X to a chosen 0-cell of X.

Proposition 3.1 *Let R_n be the fully right-associated tree in $C_0(K_n)$, and let h be a chain homotopy contracting $C_*(K_n)$ to R_n. Then h determines an associahedron tree diagonal $\{\gamma_n\}_{n\geq 2}$ and thus an associahedron diagonal.*

Proof: Let $R_n : C_*(K_n) \to C_*(K_n)$ denote the map that takes all generators in dimension 0 (i.e. 0-cells of K_n) to R_n and all generators in positive dimension to 0. Note that R_n is a chain map. Define a map k on $C_*(K_n)^{\otimes 2}$ by $k := \mathrm{id} \otimes h + h \otimes R_n$.

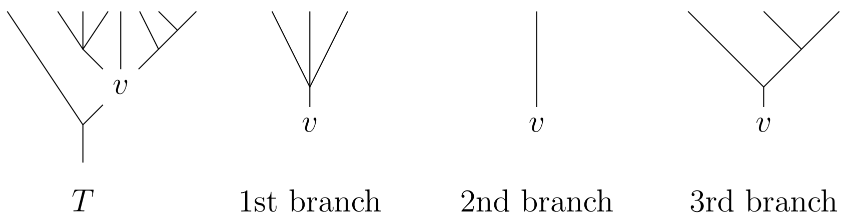


Figure 3.1: The crux v of a tree T and all of its branches

That is, k is the null homotopy on $C_*(K_n)$ induced by h that contracts the first tensor factors followed by the second. Define γ_n recursively by setting

$$\gamma_n := k\left(\sum_{i+j=n+1} \gamma_i \circ \gamma_j\right)$$

Let x denote the sum on the right. Since x is a cycle, it follows that

$$\partial\gamma_n = \partial kx = x + k\partial x + [R_n \otimes R_n]x = x + 0 + 0 = x$$

The fact that $[R_n \otimes R_n]x = 0$ follows immediately from the dimension of x whenever $n \geq 4$. A quick computation verifies that this still holds when $n = 3$. Hence γ_n as defined above satisfies the structure equation for a tree diagonal. $\square$

Essentially the same argument allows us to inductively construct weighted tree diagonals γ_n^w from a system of null homotopies of the weighted tree complexes $C_*(X_n^w)$.

Proposition 3.2 *Let R_n^w be the fully right-associated weighted tree in $C_0(X_n^w)$, and let h be a chain homotopy contracting $C_*(X_n^w)$ to R_n^w. Then h determines a weighted associahedron tree diagonal $\{\gamma_n^w\}_{n,w}$ by the recursion*

$$\gamma_n^w := (id \otimes h + h \otimes R_n^w)\left(\sum_{u+v=w}\sum_{i+j=n+1} \gamma_i^u \circ \gamma_j^v\right)$$

We conclude this section with a procedure for contracting the complexes $C_*(K_n)$. Let v be an internal vertex of a tree T. A *predecessor* of v is recursively defined as follows. v is a predecessor of itself, and if v has an input edge whose other vertex is w, then all predecessors of w are predecessors of v. By an *internal predecessor*, we mean a predecessor that is an internal vertex. Since T is planar, one can order the input edges at v from left to right. The *k-th branch* of v is the subtree induced by the predecessors of v along its k-th input edge taking v as the root. See Figure 3.1.

Algorithm 3.3 *Define a map $H : C_*(K_n) \to C_{*+1}(K_n)$ on a generator T as follows. Let v be the first (in left-right depth-first order) internal vertex of T.*

1. *If $\deg(v) > 3$, then $H(T) = 0$.*
2. *If $\deg(v) = 3$ and the left input of v is internal, $H(T)$ is the tree formed by contracting that input into v.*
3. *If $\deg(v) = 3$ but the left input of v is a leaf, skip to the next internal vertex w of T and repeat the procedure on w.*
4. *If all internal vertices of T have been skipped, then $H(T) = 0$.*

Note that Algorithm 3.3 skips every internal vertex of an n-leaf tree T if and only if $T = R_n$. If $T \neq R_n$, define the *crux* of T to be the first internal vertex v of T that Algorithm 1 does not skip (i.e. $\deg(v) > 3$ or $\deg(v) = 3$ and its left input is not a leaf). See Figure 3.1. If v is the k-th internal vertex of T in left-right depth-first order, for $1 \leq k \leq n-1$, then it follows that T can be written as $R_k \circ_k S$ where S is the right branch of the $(k-1)$-th internal vertex, i.e. the vertex immediately preceding v. Moreover, k is maximal with this property; otherwise $\deg(v) = 3$ and its left input is a leaf, contradicting the definition of v. In other words, if we let $r(T)$ denote the maximal i such that T can be written as $R_i \circ_i S$ for some subtree S, then the crux of T is precisely the $r(T)$-th internal vertex of T. For purposes hereof, we define R_1 to be the single-edge tree with one root vertex, one leaf vertex, and no internal vertices. In this way, $R_1 \circ_1 T = T$ for any tree T.

We will now define a filtration f on $C_*(K_n)$ indexed by $\mathbb{N}\times\mathbb{N}$ with the lexicographic ordering. Let T be a generator of $C_*(K_n)$. Let $f(T) = \langle f_a(T), f_b(T)\rangle$ with f_a defined as follows:

$$f_a(T) = n - r(T)$$

If v is the crux of T, then f_b is given by

$$f_b(T) = \sum_w [\deg(w) - 2]$$

where the sum ranges over all internal predecessors w of v such that w is contained in a branch of v that is not the rightmost branch. If T has no crux vertex, then $f_b(T) = 0$ by convention.
Define $f(T) = \langle f_a(T), f_b(T)\rangle$ and let $F_{(p,q)}$ be the span of all T such that $f(T) \leq (p, q)$.

Proposition 3.4 *We have a filtration $F_{(n-1,n-1)} \supset F_{(n-1,n-2)} \supset ... \supset F_{(0,1)} \supset F_{(0,0)}$ on $C_*(K_n)$ with $F_{(0,0)}$ equal to the subcomplex generated by R_n.*

Note that this filtration is always finite with $C_*(K_n) = F_{(n-1,n-1)}$ because the corolla Ψ_n always has filtration degree $(n-1, n-1)$. For any tree T, since ∂T is the sum of all expansions of T, we can deduce the statement of Proposition 3.4 from the following lemma.

Lemma 3.5 *If S is an expansion of T, then*

1. *$f_a(S) < f_a(T)$; or*
2. *$f_a(S) = f_a(T)$ and $f_b(S) \leq f_b(T)$.*

Proof: If $T = R_n$, then T has no expansions and there is nothing to prove. Otherwise let v be the crux of T, and write $T = R_{r(T)} \circ_{r(T)} T'$. If S is an expansion of a vertex u, then necessarily $u = v$ or u is an internal vertex of T'. In the second case, we clearly have $f_a(S) = f_a(T)$. We need to consider u in the calculation of f_b if

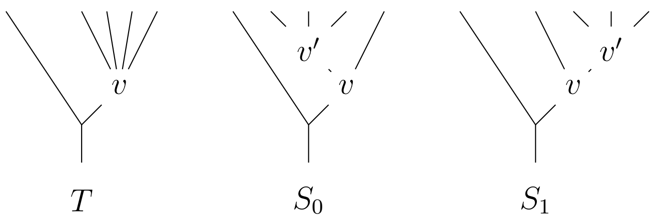


Figure 3.2: The trees in the proof of Lemma 3.7.

u is not in the rightmost branch of v; however Lemma 2.1 implies that $f_b(S) = f_b(T)$ even in this case.

For the first case, let u and u' denote the new vertices created from the expansion at v, where u occurs first in left-right depth-first order. If $\deg(u) = 3$ and its left input is internal, then $r(S) > r(T)$ and so $f_a(S) < f_a(T)$. Otherwise $f_a(S) = f_a(T)$, and $f_b(S) < f_b(T)$ if u' is in the rightmost branch of u or $f_b(S) = f_b(T)$ by Lemma 2.1 otherwise. □

Lemma 3.6 *Let $T \neq R_n$ be a generator in $C_*(K_n)$. Then $f(HT) \leq f(T)$ with equality precisely when $H(T) \neq 0$.*

Proof: When $H(T) = 0$ the statement is clear. Otherwise T has a crux vertex v and $H(T)$ merges v with its left input. Since v is still the crux of $H(T)$, we have $f_a(HT) = f_a(T)$. The fact that $f_b(HT) = f_b(T)$ follows from Lemma 2.1. □

Lemma 3.7 *Let $\alpha = id + \partial H + H\partial$, and let $T \in C_*(K_n)$ be a generator in $F_{>(0,0)}$ such that $H(T) = 0$. Then $f(\alpha(T)) < f(T)$.*

Proof: Let v be the crux of T. Since $\deg(v) > 3$, there are at least two expansions of T at v. Let S_0 be the expansion at v wherein v now has degree 3 and its left input is a new vertex v' whose branches are the original branches of v except for the rightmost branch. Let S_1 be the "mirror" expansion wherein v' is now the right input of v and whose branches are the original branches of v except for the leftmost branch. Any other expansion S of T will have $H(S) = 0$ since $\deg(v) > 3$ in S. So $H(\partial T) = HS_0 + HS_1$ and $\alpha(T) = T + HS_0 + HS_1$.

We see that $H(S_0) = T$, canceling with the id term of α. For the second tree, we clearly have $f_b(S_1) < f_b(T)$ and so $f_b(H(S_1)) < f_b(T)$ since $f_b(H(S_1)) = f_b(S_1)$ by

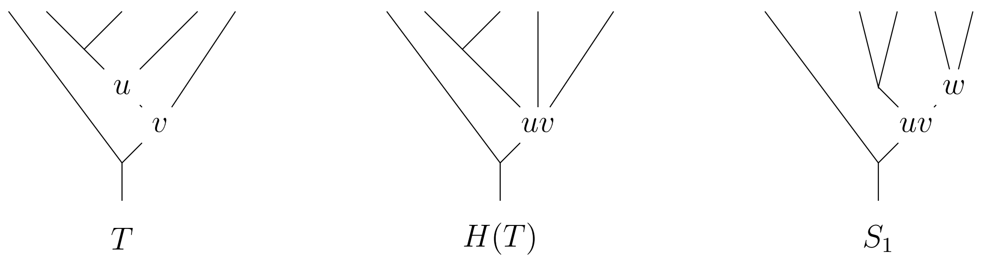


Figure 3.3: The trees in the proof of Lemma 3.8.

Lemma 2.1 if $H(S_1) \neq 0$. Likewise we have $f_a(H(S_1)) \leq f_a(S_1) \leq f_a(T)$, and thus we conclude $f(H(S_1)) < f(T)$. Therefore the result follows. $\square$

Lemma 3.8 *Let $\alpha = id + \partial H + H\partial$ and $T \in C_*(K_n)$ be a generator in $F_{>(0,0)}$ such that $H(T) \neq 0$. Then $f(\alpha(T)) \leq f(T)$ and $\alpha(T)$ is a sum of trees S such that $H(S) = 0$ or $f(S) < f(T)$.*

Proof: Let v be the crux of T. Since $H(T) \neq 0$, $\deg(v) = 3$ and its left input is an internal vertex u. Then $H(T)$ is formed by contracting the edge connecting u and v into a single vertex which will be denoted uv.

We first consider the terms in $H\partial(T)$. Suppose S is a tree in the boundary of T. Then necessarily S is an expansion of a vertex that occurs after v. Hence $H(S)$ is formed by contracting u into v as in $H(T)$, and so we will have $H(H(S)) = 0$. We also have $f(H(S)) \leq f(T)$ by Lemma 3.6.

Now consider the terms in $\partial H(T)$. These consist of expansions S of $H(T)$ which fall into three types: T itself, expansions wherein $\deg(uv) > 3$, and one other expansion S_1. The first term cancels with the id term of α, while the terms of the second type all have $f(\alpha(S)) < f(S) = f(T)$ by Lemma 3.7. The final term S_1 is the expansion of $H(T)$ at uv wherein uv now has degree 3, its leftmost input is unchanged, and its right input is a new vertex w whose branches are all of the other original branches of uv. We see that $f_b(S_1) < f_b(H(T)) = f_b(T)$ and f_a is unchanged. Hence $f(S_1) < f(T)$.

$\square$

Proposition 3.9 *The map $\alpha = id + \partial H + H\partial$ satisfies $f(\alpha^2(T)) < f(T)$ for trees $T \in C_*(K_n)$ in $F_{>(0,0)}$.*

Proof: Lemmas 3.7 and 3.8 together imply that $f(\alpha(U)) \leq f(U)$ for any such tree U. When $H(T) = 0$, we have that $f(\alpha^2(T)) \leq f(\alpha(T)) < f(T)$ by Lemma 3.7. When $H(T) \neq 0$, we have by Lemma 3.8 the previous remark for $U = \alpha(T)$, and $\alpha(T)$ is a sum of trees S such that $H(S) = 0$ or $f(S) < f(T)$. If S is a term in $\alpha(T)$ with $f(S) > (0,0)$ and $H(S) = 0$, then by Lemma 3.7 $f(\alpha(S)) < f(S) \leq f(\alpha(T)) \leq f(T)$. If S is a term in $\alpha(T)$ with $f(S) > (0,0)$ and $f(S) < f(T)$, then $f(\alpha(S)) \leq f(S) < f(T)$. For all other terms, we have $f(S) = (0,0)$ and so $f(\alpha(S)) = (0,0) < f(T)$, since we assumed T was in $F_{>(0,0)}$. This proves that every term in the expression for $\alpha^2(T)$ has filtration less than $f(T)$, and so $f(\alpha^2(T)) < f(T)$. □

Theorem 3.10 *Let $\alpha = id + \partial H + H\partial$. Repeated application of α yields a contraction of the chain complex $C_*(K_n)$ to the subcomplex generated by R_n.*

Proof: It's immediate from the definition that $\alpha \simeq \text{id}$. Hence $\alpha^k \simeq \text{id}^k \simeq \text{id}$ for all $k \geq 1$. From Proposition 3.9, we know that α^2 lowers the filtration degree of all generators in $F_{>(0,0)}$. Since the filtration is of finite length, it follows that α^N for large enough N maps $C_*(K_n)$ to the subcomplex $F_{(0,0)}$ generated by R_n, and α^N is chain homotopic to the identity on $C_*(K_n)$. □

4 Contracting the Complexes $C_*(X_0^w)$

For a weighted tree T in $C_*(X_0^w)$ all the leaves have positive weight, since there are zero inputs. If all of the internal vertices have weight 0, then all of the leaves have weight 1; so we have exactly w leaves, and therefore T is just a tree from $C_*(K_w)$ with weight 1 added to all of the leaves. In this way, we regard $C_*(K_w)$ as a subcomplex of $C_*(X_0^w)$.

In an unweighted tree, the number of leaves equals one more than the sum of $\deg(v) - 2$ where v ranges over all *branch vertices*; that is, vertices which are neither leaves nor the root. Using the fact that expansions/contractions preserve both sides of the equation, one can just check this for a corolla.

As a consequence, for any tree T in $C_0(X_0^w) \cong C_0(K_w)$, the number of branch vertices is precisely $w-1$. This may not be true for trees in positive dimensions, but $w-1$ is still an upper bound since contraction cannot increase the number of branch vertices.

If T in $C_*(X_0^w)$ has no positive weight internal vertex (hereafter abbreviated as PWIV), then it is in $C_*(K_w)$. It has w leaves, and the internal vertices are precisely the branch vertices. As previously mentioned, there will be at most $w-1$ of these.

If T has no PWIV, then set $w_0(T) = w-1$. Otherwise, let $w_0(T)$ count the number of weight-0, non-root vertices that occur before the first PWIV (with respect to the usual left-right depth-first order). Note that upon expanding T the number of weight-0 vertices cannot decrease, and so w_0 cannot decrease (this uses the previous paragraph for the "edge case" where the definition of w_0 changes).

By repeatedly expanding any tree in $C_*(X_0^w)$, we can get to a tree in $C_*(K_w)$. Since w_0 does not decrease, this shows $w_0(T) \leq w-1$ for any T.

Define $f(T) = w - 1 - w_0(T)$. Since w_0 cannot decrease upon expansion, $f(T)$ cannot increase upon expansion. So $f(S) \leq f(T)$ for any S in the boundary of T. Thus the trees T such that $f(T) \leq k$ generate a subcomplex F_k, and this defines a filtration $C_*(X_0^w) = F_{w-1} \supset F_{w-2} \supset ...$, with F_0 generated by the weight-w trees with no PWIV; i.e. F_0 is isomorphic to the ordinary associahedron chain complex $C_*(K_w)$.

These observations are summarized in the following proposition:

Proposition 4.1 *The map $f(T) = w-1-w_0(T)$ induces a filtration $F_{w-1} \supset F_{w-2} \supset \ldots \supset F_1 \supset F_0$ on $C_*(X_0^w)$ where F_0 is the subcomplex corresponding to $C_*(K_w)$.*

Our goal now is to define a procedure similar to Algorithm 3.3 that yields a contraction of $C_*(X_0^w)$ to the subcomplex $F_0 \cong C_*(K_w)$. When combined with the results in Section 3, this will ultimately produce a total contraction of the chain complex $C_*(X_0^w)$.

Algorithm 4.2 *Define a map $J : C_*(X_0^w) \to C_{*+1}(X_0^w)$ on generators as follows. Using left-right depth-first search, find the first PWIV v of T.*

1. *If* $\deg(v) = 2$ *and wt*$(v) = 1$*, the map J contracts the unique input edge of v, adding weights if necessary.*
2. *If* $\deg(v) \in \{1, 2\}$ *and wt*$(v) > 1$*, or* $\deg(v) > 2$*, then J maps T to 0.*
3. *If T has no PWIV's, then* $J(T) = 0$.

Note that the subcomplex $F_0 \cong C_*(K_w)$ is mapped to 0 by J. Let $\xi = \partial J + J\partial + \mathrm{id}$. If we could show that $f(\xi(T)) < f(T)$ for trees in positive filtration, we would have that repeated application of ξ defines a contraction of $C_*(X_0^w)$ into $C_*(K_w)$, thus giving a total contraction of $C_*(X_0^w)$ when combined with the contraction of $C_*(K_w)$. It turns out that this isn't always the case, but it's sufficient to prove a weaker result, namely that $f(\xi^2(T)) < f(T)$ for trees in positive filtration. To that end, we start with a lemma.

Lemma 4.3 *Let T be a generator of $C_*(X_0^w)$ in positive filtration. Then $f(J(T)) \leq f(T)$ with equality precisely when $J(T) \neq 0$.*

Proof: When $J(T) = 0$, the statement is clear. When $J(T) \neq 0$, T and $J(T)$ have the same first PWIV and identical structure prior to this vertex, so $f(J(T)) = f(T)$. $\square$

Lemma 4.4 *Let T be a generator of $C_*(X_0^w)$ with $f(T) > 0$, and suppose that $J(T) = 0$. Then the map $\xi = \partial J + J\partial + id$ satisfies $f(\xi(T)) < f(T)$.*

Proof: Since T has positive filtration, it has at least one PWIV. Let v be the first PWIV of T. Then $\mathrm{wt}(v) > 1$ or $\mathrm{wt}(v) = 1$ and $\deg(v) > 2$, since $J(T) = 0$. Observe that ∂T contains exactly one tree U such that $J(U) = T$, namely the result of creating a new vertex above v so that v now has weight-1 and degree-2. See Figure 4.1 for an example. Moreover any other tree S in ∂T wherein v has degree-2 will have $J(S) = 0$ since the weight of v is necessarily at least 2.

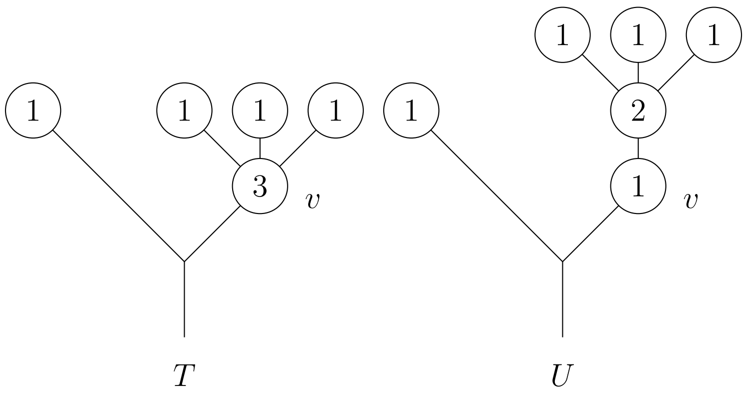


Figure 4.1: A weighted tree T in X_0^7, its first PWIV v, and the unique U in its boundary such that $J(U) = T$.

It remains to consider what other trees S can occur in the boundary of T. Note that $J(S) = 0$ if S results from changing a vertex other than v, so we need only consider those trees which result from expanding at v, because

$$\xi(T) = \partial JT + J\partial T + T = 0 + \sum_{S \text{ in } \partial T} J(S) + T = \sum_{S \text{ changes } v} J(S)$$

Case 1: v is a lollipop of weight at least 2.

Then S is created by adding a new lollipop above v. Aside from the tree U mentioned above, all such S will have $J(S) = 0$, since v is now a degree-2 vertex of weight at least 2.

Case 2: $\deg(v) \geq 2$ and S is created by adding a new lollipop at v.

Then $\deg(v)$ in S is now at least 3, so $J(S) = 0$ unless $\mathrm{wt}(v)$ is now 0. In this case, v is no longer a PWIV, so we have $f(J(S)) \leq f(S) < f(T)$ by Lemma 4.3. See Figure 4.2.

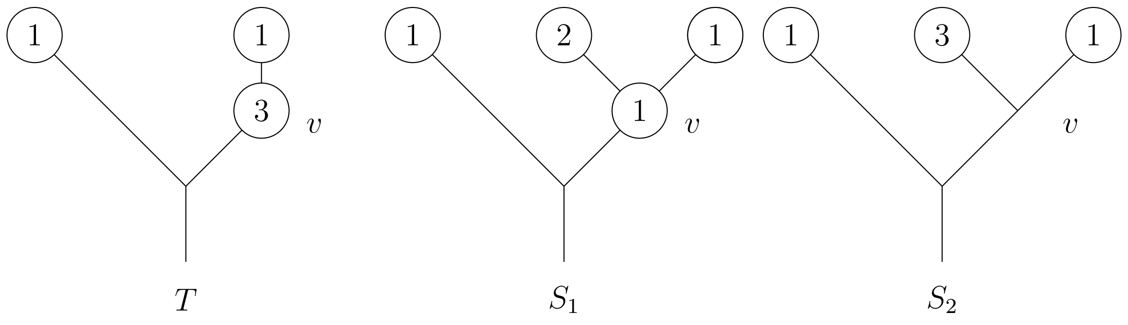


Figure 4.2: Two examples illustrating case 2 in the proof of Lemma 4.4. $J(S_1) = 0$ and $f(S_2) < f(T)$.

Case 3: $\deg(v) \geq 2$ and S is created by adding a new vertex u whose branches are some of the original branches of v and whose weight is some of the original weight of v.

The case where $\deg(v) = 2$ in S was already addressed in the first paragraph, so we will assume that $\deg(v) \geq 3$ in S. Then $J(S) = 0$ unless v is now weight-0 in S. In this case, v is no longer a PWIV and so $f(J(S)) \leq f(S) < f(T)$ by Lemma 4.3. See Figure 4.3. ☐

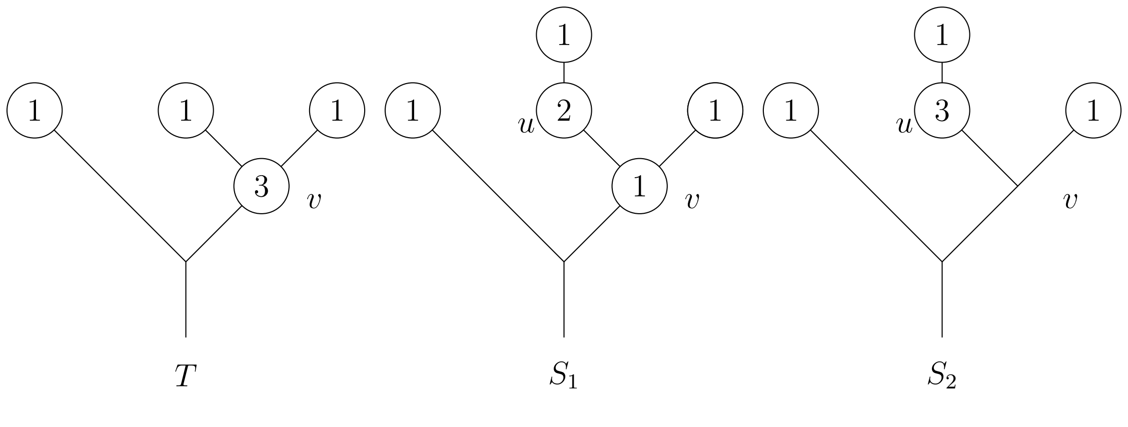


Figure 4.3: Two examples illustrating case 3 in the proof of Lemma 4.4. $J(S_1) = 0$ and $f(S_2) < f(T)$.

Lemma 4.5 *Let $\xi = \partial J + J\partial + id$ and let $T \in C_*(X_0^w)$ be a generator in positive filtration such that $J(T) \neq 0$. Then $f(\xi(T)) \leq f(T)$ and $\xi(T)$ is a sum of trees T' such that $J(T') = 0$ or $f(T') < f(T)$.*

Proof: Since $J(T) \neq 0$, T has a first PWIV of degree 2 and weight 1 which we call v. We first consider what J does to trees S that appear as terms in ∂T.

Case 1.1: S is created by adding a new lollipop at v.

This means that $\text{wt}(v)$ is now 0, so $f(J(S)) \leq f(S) < f(T)$ by Lemma 4.3.

Case 1.2: S is created by changing a vertex other than v.

Let u be the vertex corresponding to the single input of v. By definition, $H(S)$ is formed by merging the vertices u, v, adding weights if necessary. But this means that the new vertex has weight at least 2 or is weight 1 with degree at least 3. In either case, we see that $f(J(S)) \leq f(S) = f(T)$ and $J(J(S)) = 0$.

The next step is to consider the tree $J(T)$ and its boundary. Letting u be as above, the tree $J(T)$ is by definition formed by merging the vertices u, v. Let uv denote this merged vertex in $J(T)$. We have the following cases for S in $\partial J(T)$.

Case 2.1: S is created by adding a new vertex above uv so that $\deg(uv)$ is now 2.

Exactly one of these trees is T which cancels with the id term in $\xi(T)$. The weight of uv in all other such S is at least 2, and so they will have $J(S) = 0$ and $f(S) \leq f(J(T)) \leq f(T)$.

Case 2.2: S is created by adding a new lollipop at uv.

The vertex uv in $J(T)$ is either degree-2 with weight at least 2, or weight-1 with degree at least 3. Thus $J(S) = 0$ for these S unless all the weight of uv is pushed onto the new lollipop. In this case, we have $f(S) < f(J(T)) \leq f(T)$.

Case 2.3: S is created by adding a new vertex above uv so that $\deg(uv)$ is now at least 3.

All of these trees will have $J(S) = 0$ except for the tree wherein all the weight is pushed off uv onto the new vertex. In this case, $f(S) < f(J(T)) \leq f(T)$ once again.

Case 2.4: S changes a vertex other than uv.

Necessarily $J(S) = 0$, and $f(S) \leq f(J(T)) \leq f(T)$ by Lemma 4.3 again. □

Proposition 4.6 *The map $\xi = id + \partial J + J\partial$ satisfies $f(\xi^2(T)) < f(T)$ for trees $T \in C_*(X_0^w)$ in positive filtration.*

Proof: Lemmas 4.4 and 4.5 together imply that $f(\xi(U)) \leq f(U)$ for any tree U. When $J(T) = 0$, we have that $f(\xi^2(T)) \leq f(\xi(T)) < f(T)$. When $J(T) \neq 0$, we have that $f(\xi(T)) \leq f(T)$ and $\xi(T)$ is a sum of trees S such that $J(S) = 0$ or satisfying $f(S) < f(T)$. If S is a term in $\xi(T)$ with $f(S) > 0$ and $J(S) = 0$, then by Lemma 4.4 $f(\xi(S)) < f(S) \leq f(\xi(T)) \leq f(T)$. If S is a term in $\xi(T)$ with $f(S) > 0$ and $f(S) < f(T)$, then $f(\xi(S)) \leq f(S) < f(T)$. For all other terms, we have $f(S) = 0$ and so $f(\xi(S)) = 0 < f(T)$, since we assumed T was in positive filtration. This proves that every term in the expression for $\xi^2(T)$ has filtration less than $f(T)$, and so $f(\xi^2(T)) < f(T)$. $\square$

Theorem 4.7 *Let $\xi = id + \partial J + J\partial$. Repeated application of ξ yields a contraction of the chain complex $C_*(X_0^w)$ to the subcomplex generated by trees with no PWIV's.*

Proof: Similar to the proof of Theorem 3.10. $\square$

5 The General Contraction of $C_*(X_n^w)$

For positive n, there is a subcomplex $R_n \circ_n C_*(X_0^w)$ of $C_*(X_n^w)$ generated by all trees of the form $R_n \circ_n T$, where R_n is the fully right-associated tree in K_n and T is any tree in $C_*(X_0^w)$. It's not hard to see that $R_n \circ_n C_*(X_0^w)$ is isomorphic to $C_*(X_0^w)$. Thus it's enough to find a chain homotopy contracting $C_*(X_n^w)$ down to this subcomplex, and the contraction problem for general $C_*(X_n^w)$ will reduce to that for $C_*(X_0^w)$.

Algorithm 5.1 *Define a map $K : C_*(X_n^w) \to C_{*+1}(X_n^w)$ on generators as follows. Using left-right depth-first search, check each vertex v of T.*

1. *If $\deg(v) = 3$, $wt(v) = 0$, and the left input of v is not one of the input leaves of T: $K(T)$ is formed by contracting the left input edge of v.*
2. *If $\deg(v) > 3$ or $wt(v) > 0$: $K(T) = 0$.*
3. *If $\deg(v) = 3$, $wt(v) = 0$, but the left input is an input leaf: skip v and continue search.*
4. *If v is an input leaf: skip v and continue search, keeping track of the number of input leaves found in this way.*
5. *If all input leaves of T have been found: $K(T) = 0$.*

To put a filtration on $C_*(X_n^w)$, we will use indices from $\mathbb{N}^2$ with the lexicographic order just as we did for the unweighted associahedron complexes. See Figure 5.1 for an example. Given a weighted tree $T \in C_*(X_n^w)$, define its filtration degree $f(T) = \langle f_s(T), f_b(T)\rangle$ by

$$f_s(T) = \sum_{\ell} \rho(\ell)$$

where the sum is taken over all input leaves ℓ of T and $\rho(\ell)$ is the sum of the weights of all vertices occurring before ℓ in the left-right depth-first order.

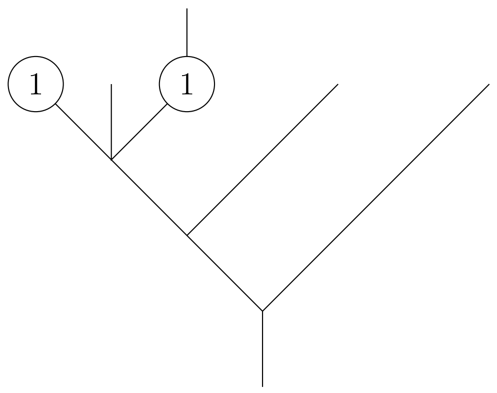


Figure 5.1: A weighted tree T in X_4^2 with $f(T) = \langle 7, 4\rangle$

The f_b filtration is defined similarly to its counterpart in the unweighted case, with some modification to account for the weights. If T is a weighted tree such that Algorithm 5.1 terminates after finding all of its input leaves, define $f_b(T) = 0$. Otherwise, there is some vertex v at which Algorithm 5.1 terminates in case 1 or 2, hereafter called the *crux* of T. Then

$$f_b(T) = \sum_w [\max(\deg(w) - 2, 0) + \mathrm{wt}(w)]$$

where w ranges over v and all internal predecessors of v that are not in its rightmost branch. Just as in the unweighted case, it should be noted that if v is the crux of a weighted tree T, then T can be written as $R_k \circ_k T'$ for some $k \geq 1$ with v corresponding to the root of T'. It should also be noted that T' always contains at least one input leaf; otherwise Algorithm 5.1 would have terminated before reaching v.

Lemma 5.2 *Let T be a generator in $C_*(X_n^w)$, and let S be an expansion of T. Then*

1. *$f_s(S) < f_s(T)$; or*
2. *$f_s(S) = f_s(T)$ and $f_b(S) \leq f_b(T)$.*

Proof: Let S result from expanding T at a vertex u, creating a new vertex u' in S. Let ℓ be any input leaf, and consider the values $\rho_T(\ell), \rho_S(\ell)$ in T, S respectively. If u occurs after ℓ in left-right depth-first order, then so does u' and we have $\rho_T(\ell) = \rho_S(\ell)$.

If u occurs before ℓ, then $\rho_T(\ell) = \rho_S(\ell)$ unless u' occurs after ℓ and $\mathrm{wt}(u') > 0$. In this case, we have $\rho_T(\ell) > \rho_S(\ell)$. Since this is true for all input leaves, we have shown that $f_s(S) \leq f_s(T)$ in all cases.

For the remainder of the proof, it's enough to show in each subcase that either $f_s(S) < f_s(T)$ or $f_b(S) \leq f_b(T)$. Let v be the crux of T, and let u, u' be as before. As mentioned in the previous discussion, T can be written as $R_k \circ_k T'$ where v corresponds to the root of T'. So u cannot occur before v in left-right depth-first order (hence neither can u'). If $u \neq v$, then v is still the crux of S and either both u, u' are in the rightmost branch of v or they are both in another branch of v. In the first case, we clearly have $f_b(S) = f_b(T)$, and Lemma 2.2 gives the same result in the second case.

If $u = v$, then v may no longer be the crux of S. If this happens, then v is now a degree-3, weight-0 vertex with leaf left input and u' is now the crux of S. If $\mathrm{wt}(u') > 0$, then $f_s(S) < f_s(T)$. Otherwise $f_s(S) = f_s(T)$ and $f_b(S) \leq f_b(T)$ by Lemma 2.2, since every branch of u' was a branch of v in T. If v is still the crux of S, then Lemma 2.2 again implies that $f_b(S) = f_b(T)$ unless u' is in the rightmost branch of v and $\mathrm{wt}(u') > 0$. In this last case, we have that $f_b(S) < f_b(T)$ by the same lemma. $\square$

Lemma 5.3 *Let T be a generator in $C_*(X_n^w)$. Then $f(K(T)) \leq f(T)$.*

Proof: It's clear that $f_s(K(T)) \leq f_s(T)$. The inequality $f_b(K(T)) \leq f_b(T)$ follows from Lemma 2.1 and a similar observation about the weights in T and $K(T)$. $\square$

Let $\zeta : C_*(X_n^w) \to C_*(X_n^w)$ be the morphism of chain complexes given by $\zeta = \partial K + K\partial + \mathrm{id}$. Lemmas 5.2 and 5.3 together imply the following useful result.

Corollary 5.4 *For any generator T in $C_*(X_n^w)$, $f(\zeta(T)) \leq f(T)$.*

Observe that the bottom filtration $F_{(0,0)}$ is precisely the subcomplex generated by $R_n \circ_n \Psi_0^w$ as desired. Thus it remains to show that repeated application of ζ lowers filtration degree on generators in $F_{>(0,0)}$, and it will follow just as in the proof of Theorem 3.10 that ζ^N maps into $F_{(0,0)}$ for sufficiently large N and is chain homotopic to the identity on $C_*(X_n^w)$. This is stated as Theorem 5.8 at the end of this section, whither we build first with several lemmas.

Lemma 5.5 *Let T be a generator of $C_*(X^w_n)$ with $f(T) > (0,0)$, and suppose that $K(T) = 0$. Then the map $\zeta = \partial K + K\partial + id$ satisfies $f(\zeta(T)) < f(T)$.*

Proof: Since $f(T) > (0,0)$, there is either an input leaf that occurs after a positive weight vertex, or $T = T' \circ_n \Psi^w_0$ where T' is not fully right-associated. In either case, T has a crux vertex v. As remarked in the discussion of Algorithm 5.1, at least one branch of v must contain an input leaf; in particular, v itself cannot be a lollipop. Since $K(T) = 0$, we have that either $\mathrm{wt}(v) > 0$ holds or $\mathrm{wt}(v) = 0$ and $\deg(v) > 3$. We consider trees S in ∂T in each of these cases separately. Note that any expansion S at a vertex other than v always has $K(S) = 0$, so we only consider expansions at v since

$$\zeta(T) = \sum_{S \in \partial T} K(S) + T$$

Case 1: $\deg(v) = 2$

Necessarily $\mathrm{wt}(v)$ is some positive integer s, and there are exactly two expansions of T which K does not map to 0: the expansion S_0 that creates a new left input of v which is a lollipop of weight s, and the expansion S_1 in which instead the s-lollipop is the new right input of v. We have that $K(S_0) = T$, canceling with the id term of ζ, so $\zeta(T) = K(S_1)$. We also have that $f_s(K(S_1)) \leq f_s(S_1) < f_s(T)$; the inequality holds because there is at least one input leaf in the original branch of v, and so the weight s appears in the calculation of $f_s(T)$ but not $f_s(S_1)$.

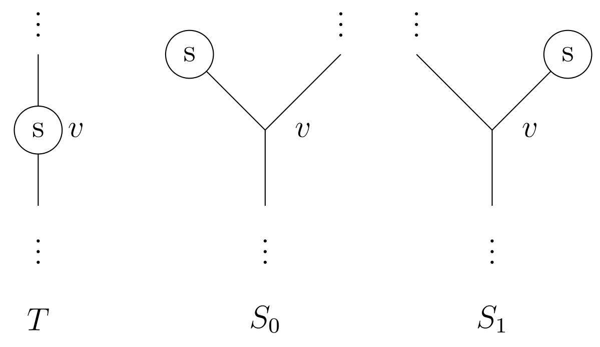


Figure 5.2: The crucial part of the trees in the proof of Lemma 5.5, Case 1.

Case 2: $\deg(v) = 3$ and $\mathrm{wt}(v) > 0$

Let $\mathrm{wt}(v) = s$ once again. There are precisely two expansions of T at v which will be nonzero under K: the expansion S_0 wherein all weight on v is pushed onto a new degree-2 vertex to the left of v, and its "mirror" S_1 wherein all weight on v is pushed onto a new degree-2 vertex to the right of v. Observe that $K(S_0) = T$, canceling with the id term of ζ, so $\zeta(T) = K(S_1)$. If there is an input leaf in the left branch of v, then $f_s(K(S_1)) \leq f_s(S_1) < f_s(T)$. Otherwise, K merges the left input of v with v, and v is now the crux of $K(S_1)$. The cruces of T and $K(S_1)$ have the same internal predecessors in the leftmost branch except for the crux of T itself, which contributes $s + 1$ to $f_b(T)$. Hence $f_s(K(S_1)) = f_s(T)$ and $f_b(K(S_1)) = f_b(T) - (s + 1) < f_b(T)$.

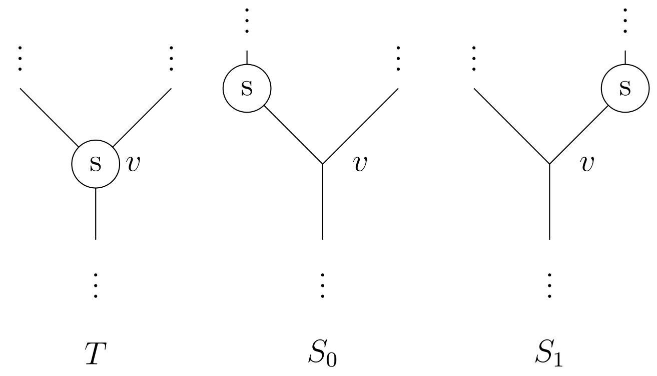


Figure 5.3: The crucial part of the trees in the proof of Lemma 5.5, Case 2.

Case 3: $\deg(v) > 3$

Let $\mathrm{wt}(v) = s$ once again where now $s \geq 0$. There are precisely two expansions of T at v which will be nonzero under K: the expansion S_0 wherein all weight on v and all but the rightmost input of v are pushed onto a new left input of v, and its "mirror" S_1 wherein all weight on v and all but the leftmost input of v are pushed onto a new right input of v. Observe that $K(S_0) = T$, canceling with the id term of ζ and giving $\zeta(T) = K(S_1)$. The second tree will have $f_s(K(S_1)) < f_s(T)$ if $s > 0$ and there was an input leaf in the left branch of v. If not, then $f_s(K(S_1)) = f_s(T)$. Moreover, in this last case, v in S_1 has lost at least one of the middle branches it had in T since we assumed $\deg(v) > 3$. Since the crux of $K(S_1)$ is either v or in one of its branches, we must have that $f_b(K(S_1)) < f_b(T)$. $\square$

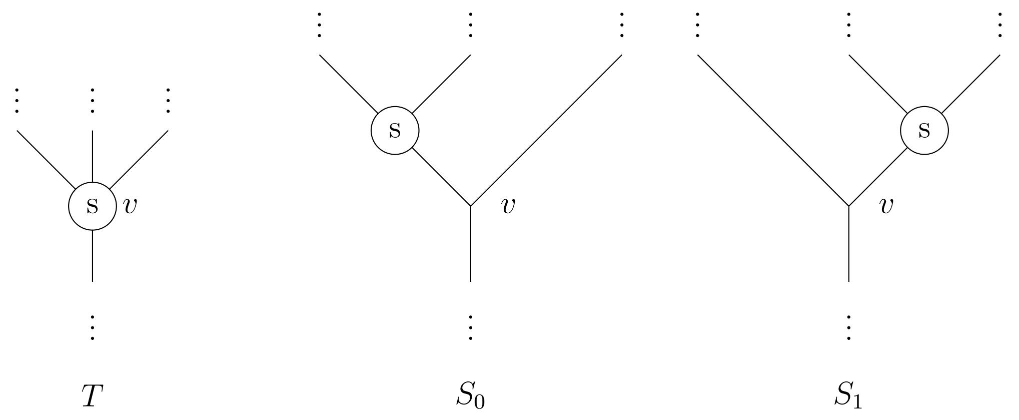


Figure 5.4: The crucial part of the trees in the proof of Lemma 5.5, Case 3.

Lemma 5.6 *Let $\zeta = \partial K + K\partial + id$ and let $T \in C_*(X_n^w)$ be a generator with $f(T) > (0,0)$ and $K(T) \neq 0$. Then $\zeta(T)$ is a sum of trees S such that $K(S) = 0$ or $f(S) < f(T)$.*

Proof: Once again, the condition $f(T) > (0,0)$ implies that T has a crux vertex v. Since $K(T) \neq 0$, we must have $\deg(v) = 3$, $\mathrm{wt}(v) = 0$, and the left input of v is either a 1-lollipop or an internal vertex. Call this left input u. First we consider the terms of ∂T. Any such tree S is an expansion of T at a vertex after v, and so v is still the crux of S. Hence v is also the crux of $K(S)$ and we have $K(K(S)) = 0$ since v now has positive weight or $\deg(v) > 3$. Moreover, $f(K(S)) \leq f(S) \leq f(T)$ by Lemmas 5.2 and 5.3. This show $f(K\partial T) \leq f(T)$ and $K\partial T$ is a sum of trees S such that $K(S) = 0$. Recall that

$$\zeta(T) = \sum_{S \in \partial K(T)} S + K\partial T + T$$

We next consider $K(T)$ and its boundary terms. We do this by looking at two cases for the vertex u.

Case 1: u is a lollipop of weight s

Then $K(T)$ has crux at v but now $\deg(v) = 2$ and $\mathrm{wt}(v) = s$. We have that $K(S) = 0$ for all trees S in the boundary of $K(T)$ with two exceptions: T itself and the tree S_1 which instead has an s-lollipop as the right input of v. The first tree cancels with the id term of ζ, while the second has $f_s(S_1) < f_s(K(T)) = f_s(T)$ since the other branch of v must contain an input leaf. The boundary trees such that $K(S) = 0$ all have $f(K(S)) \leq f(S) \leq f(T)$ by Lemmas 5.2 and 5.3.

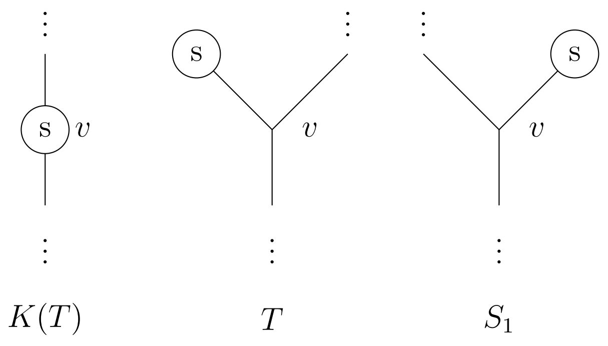


Figure 5.5: The crucial part of the trees in the proof of Lemma 5.6, Case 1.

Case 2: u has at least one branch

Then $K(T)$ merges the vertices u and v into a new vertex denoted uv. Once again $K(S) = 0$ and $f(K(S)) \leq f(S) \leq f(K(T))$ for all but two expansions S of $K(T)$. The first is T itself which cancels with the id term of ζ. The second term S_1 pushes all weight and all but the leftmost input of uv onto a new right input w of uv. If the leftmost branch of u contained an input leaf, then $f_s(S_1) < f_s(T)$. Otherwise $f_s(S_1) = f_s(T)$, and since $\deg(u) \geq 2$, we have the following subcases. Either $\deg(u) > 2$ and so at least one branch of u in T is now a branch of w in S_1; or $\deg(u) = 2$, hence $\mathrm{wt}(u) > 0$, and so some positive weight that was in the left branch of v in T has moved to the right branch of uv in S_1. Thus in either subcase we will have $f_b(S_1) < f_b(T)$, because the crux of S_1 is either uv or in one of its branches. Hence $f(S_1) < f(T)$ in any case. □

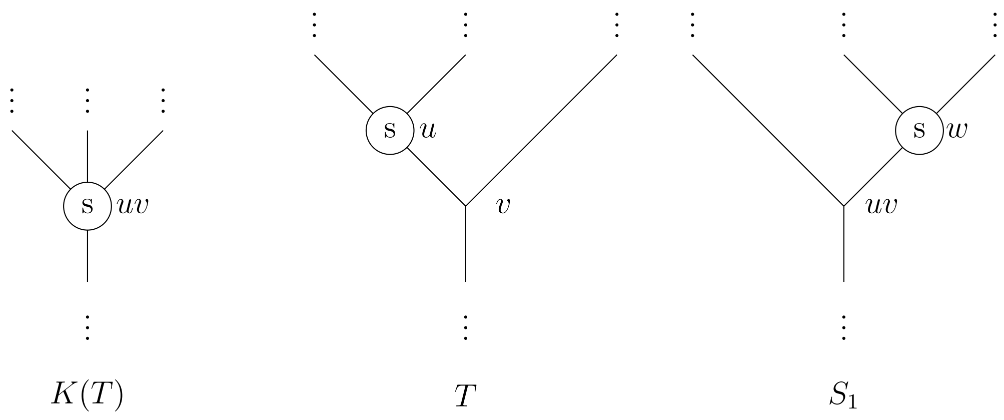


Figure 5.6: The crucial part of the trees in the proof of Lemma 5.6, Case 2.

Proposition 5.7 *The map $\zeta = id + \partial K + K\partial$ satisfies $f(\zeta^2(T)) < f(T)$ for trees $T \in C_*(X_n^w)$ in $F_{>(0,0)}$.*

Proof: By Corollary 5.4, $f(\zeta(T)) \leq f(T)$ for any tree T. When $K(T) = 0$, we have that $f(\zeta^2(T)) \leq f(\zeta(T)) < f(T)$. When $K(T) \neq 0$, we have that $f(\zeta(T)) \leq f(T)$ and $\zeta(T)$ is a sum of trees S such that $K(S) = 0$ or satisfying $f(S) < f(T)$. If S is a term in $\zeta(T)$ with $f(S) > (0,0)$ and $K(S) = 0$, then by Lemma 5.5 $f(\zeta(S)) < f(S) \leq f(\zeta(T)) \leq f(T)$. If S is a term in $\zeta(T)$ with $f(S) > (0,0)$ and $f(S) < f(T)$, then $f(\zeta(S)) \leq f(S) < f(T)$. For all other terms, we have $f(S) = (0,0)$ and so $f(\zeta(S)) = (0,0) < f(T)$, since we assumed T was in $F_{>(0,0)}$. This proves that every term in the expression for $\zeta^2(T)$ has filtration less than $f(T)$, and so $f(\zeta^2(T)) < f(T)$. $\square$

Theorem 5.8 *Let $\zeta = id + \partial K + K\partial$. Repeated application of ζ yields a contraction of the chain complex $C_*(X_n^w)$ to the subcomplex generated by $R_n \circ_n \Psi_0^w$.*

Proof: Similar to the proof of Theorem 3.10. $\square$

6 Resulting Tree Diagonals

We conclude this paper with an implementation of the algorithms described in the previous sections used to produce weighted tree diagonals γ_n^w. The following outputs were generated in SageMath using the Abstract Labeled Trees library. The source code used to generate them is included at the end of the section. Tables 1, 2, and 3 summarize the number of terms in γ_n^w for small values of n and w along with the runtime needed to compute them. All computations were done on a Hewlett-Packard laptop with a 64-bit operating system, a 2.30 GHz processor, and 4 GB of RAM.

All trees are displayed in ASCII art using SageMath's Abstract Labeled Trees library. This prints the nonroot vertices of the tree as nonnegative integers according to their label, with the first vertex (in depth-first search order) printed in the first line and each other vertex printed below the vertex of which it is a branch.

We remark that the initial terms of our results in the unweighted case agree with the results in [7]. It seems reasonable to suspect that they are the same given the similarities in the definition of Algorithm 5.1 and in [7], although we have no theoretical basis on which to justify this.

It's also worth remarking that the numbers of terms in γ_n listed below agree with the terms of the sequence $a(n) = \frac{2(3n)!}{(2n+1)!(n+1)!}$ when offset by 1. This sequence has been used to count various combinatorial sets of trees (see sequence A000139 at [8]), so it seems reasonable to conjecture that $a(n-1)$ also counts the number of pure tensors in γ_n.

n	Number of terms in γ_n	Runtime (sec)
2	1	0
3	2	0.00347
4	6	0.0607
5	22	0.5824
6	91	4.3637
7	408	32.8650
8	1938	240.8907

Table 1: Numbers of terms in weight-0 tree diagonals.

```
---------------
    0__  times  _0__
   /  /        / / /
  0_ 0        0 0 0
 / /
0 0
---------------
  _0__  times  _0__
 / / /        /   /
0 0 0        0   0_
                / /
               0 0
---------------
```

Figure 6.1: The 2 terms in γ_3 displayed with ASCII art.

```
---------------
      _0__  times   __0___
     /   /         / / / /
    0__ 0         0 0 0 0
   /  /
  0_ 0
 / /
0 0
---------------
     _0__  times   __0____
    /   /         /   /  /
  _0__ 0         0   0_ 0
 / / /              / /
0 0 0              0 0
---------------
     _0__  times   _0___
    /   /         /    /
  _0__ 0         0   _0__
 / / /              / / /
0 0 0              0 0 0
---------------
```

```
---------------
    _0___  times   __0___
   /  / /         / /   /
  0_ 0 0         0 0   0_
 / /                  / /
0 0                  0 0
---------------
  __0____  times   _0___
 /   /  /         /    /
0   0_ 0         0   _0__
   / /              / / /
  0 0              0 0 0
---------------
  __0___  times   _0___
 / / / /         /    /
0 0 0 0         0   _0__
                   /   /
                  0   0_
                     / /
                    0 0

---------------
```

Figure 6.2: The 6 terms in γ_4 displayed with ASCII art.

```
---------------
         _0__  times   ___0____
        /   /         / / / / /
      _0__ 0         0 0 0 0 0
     /   /
    0__ 0
   /  /
  0_ 0
 / /
0 0
---------------
        _0__  times   ___0_____
       /   /         /   /  / /
     _0__ 0         0   0_ 0 0
    /   /              / /
  _0__ 0              0 0
 / / /
0 0 0
---------------
        _0__  times   ___0_____
       /   /         /    /   /
     _0__ 0         0   _0__ 0
    /   /              / / /
  _0__ 0              0 0 0
 / / /
0 0 0
---------------
        _0__  times   __0___
       /   /         /     /
     _0__ 0         0   __0___
    /   /              / / / /
  _0__ 0              0 0 0 0
 / / /
0 0 0
---------------
  ___0_____  times   _0___
 / /   /  /         /    /
0 0   0_ 0         0   _0___
     / /              /    /
    0 0              0   _0__
                        / / /
                       0 0 0
---------------
  ___0____  times   _0___
 / / / / /         /    /
0 0 0 0 0         0   _0___
                     /    /
                    0   _0__
                       /   /
                      0   0_
                         / /
                        0 0
---------------
```

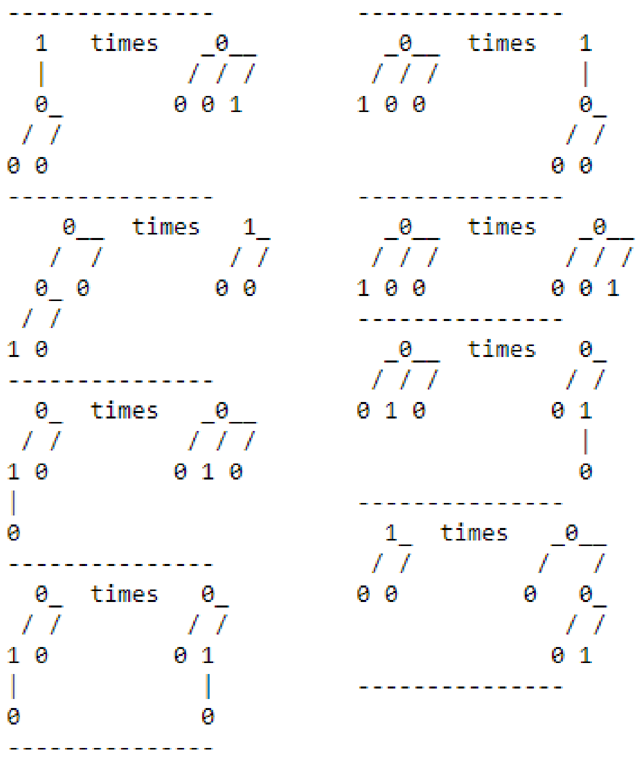
```
---------------
       _0___  times   ___0_____
      /    /         / /   /  /
    _0___ 0         0 0   0_ 0
   /  / /                / /
  0_ 0 0                0 0
 / /
0 0
---------------
      __0___  times   ___0_____
     /     /         /    /   /
  __0____ 0         0   _0__ 0
 /   /  /              / / /
0   0_ 0              0 0 0
   / /
  0 0
---------------
      __0___  times   __0___
     /     /         /     /
  __0____ 0         0   __0___
 /   /  /              / / / /
0   0_ 0              0 0 0 0
   / /
  0 0
---------------
      _0___  times   ____0_____
     /    /         /    /    /
  __0___ 0         0   _0__  0
 / / / /              /   /
0 0 0 0              0   0_
                        / /
                       0 0
---------------
     __0___  times   ___0_____
    /   / /         /   /    /
  _0__ 0 0         0   0_   0_
 / / /                / /  / /
0 0 0                0 0  0 0
---------------
     __0___  times   __0___
    /   / /         /     /
  _0__ 0 0         0   __0___
 / / /                / /   /
0 0 0                0 0   0_
                          / /
                         0 0
---------------
  ___0_____  times   __0___
 /    /   /         /     /
0   _0__ 0         0   __0____
   / / /              /   /  /
  0 0 0              0   0_ 0
                        / /
                       0 0
---------------
  ___0_____  times   _0___
 /    /   /         /    /
0   _0__ 0         0   _0___
   / / /              /    /
  0 0 0              0   _0__
                        / / /
                       0 0 0
---------------
```

```
---------------
      _0___  times   __0___
     /    /         /     /
  __0___ 0         0   __0____
 / / / /              /   /  /
0 0 0 0              0   0_ 0
                        / /
                       0 0
---------------
      _0___  times   _0___
     /    /         /    /
  __0___ 0         0   _0___
 / / / /              /    /
0 0 0 0              0   _0__
                        / / /
                       0 0 0
---------------
      __0___  times   ___0____
     /   / /         / / /   /
    0__ 0 0         0 0 0   0_
   /  /                    / /
  0_ 0                    0 0
 / /
0 0
---------------
  ____0_____  times   __0___
 /     /   /         /     /
0     0__ 0         0   __0___
     /  /              / / / /
    0_ 0              0 0 0 0
   / /
  0 0
---------------
       _0___  times   __0____
      /    /         / /    /
    _0___ 0         0 0   _0__
   /  / /                / / /
  0_ 0 0                0 0 0
 / /
0 0
---------------
    ___0____  times   __0____
   /    /  /         / /    /
  0_   0_ 0         0 0   _0__
 / /  / /                / / /
0 0  0 0                0 0 0
---------------
    __0____  times   __0____
   /  / / /         / /    /
  0_ 0 0 0         0 0   _0__
 / /                    /   /
0 0                    0   0_
                          / /
                         0 0
---------------
  ___0_____  times   __0___
 /   /  / /         /     /
0   0_ 0 0         0   __0___
   / /                / /   /
  0 0                0 0   0_
                          / /
                         0 0
---------------
```

Figure 6.3: The 22 terms in γ_5 displayed with ASCII art.

n	Number of terms in γ_n^1	Runtime (sec)
0	1	0
1	2	0.00396
2	8	0.0782
3	38	0.9958
4	196	10.7997
5	1062	102.2154

Table 2: Numbers of terms in weight-1 tree diagonals.

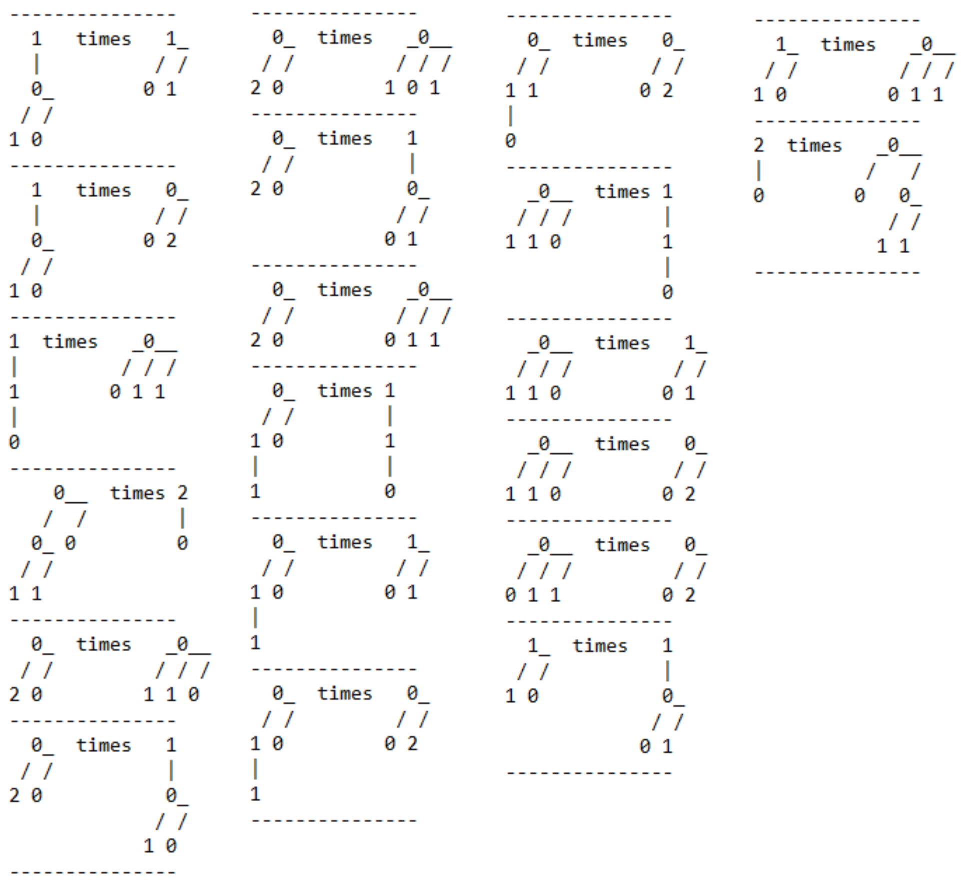


```
---------------          ---------------
  1   times   _0__         _0__  times   1
  |          / / /        / / /          |
  0_        0 0 1        1 0 0           0_
 / /                                    / /
0 0                                    0 0
---------------          ---------------
    0__  times   1_        _0__  times   _0__
   /  /         / /       / / /         / / /
  0_ 0           0 0      1 0 0         0 0 1
 / /                     ---------------
1 0                        _0__  times   0_
---------------           / / /         / /
  0_  times   _0__       0 1 0         0 1
 / /         / / /                       |
1 0         0 1 0                        0
|                        ---------------
0                          1_  times   _0__
---------------           / /         /   /
  0_  times   0_         0 0         0   0_
 / /         / /                        / /
1 0         0 1                        0 1
|             |          ---------------
0             0
---------------
```

Figure 6.4: The 8 terms in γ_2^1 displayed with ASCII art.

n	Number of terms in γ_n^2	Runtime (sec)
0	2	0.0128
1	20	0.3734
2	147	6.7881
3	1023	94.1467
4	7007	1141.8786

Table 3: Numbers of terms in weight-2 tree diagonals.



```
---------------
  1   times   1_
  |          / /
  0_        0 1
 / /
1 0
---------------
  1   times   0_
  |          / /
  0_        0 2
 / /
1 0
---------------
1  times   _0__
|         / / /
1        0 1 1
|
0
---------------
    0__  times 2
   /  /        |
  0_ 0         0
 / /
1 1
---------------
  0_  times   _0__
 / /         / / /
2 0         1 1 0
---------------
  0_  times   1
 / /          |
2 0           0_
             / /
            1 0
---------------
```

```
---------------
  0_  times   _0__
 / /         / / /
2 0         1 0 1
---------------
  0_  times   1
 / /          |
2 0           0_
             / /
            0 1
---------------
  0_  times   _0__
 / /         / / /
2 0         0 1 1
---------------
  0_  times 1
 / /        |
1 0         1
|           |
1           0
---------------
  0_  times   1_
 / /         / /
1 0         0 1
|
1
---------------
  0_  times   0_
 / /         / /
1 0         0 2
|
1
---------------
```

```
---------------
  0_  times   0_
 / /         / /
1 1         0 2
|
0
---------------
  _0__  times 1
 / / /        |
1 1 0         1
              |
              0
---------------
  _0__  times   1_
 / / /         / /
1 1 0         0 1
---------------
  _0__  times   0_
 / / /         / /
1 1 0         0 2
---------------
  _0__  times   0_
 / / /         / /
0 1 1         0 2
---------------
  1_  times   1
 / /          |
1 0           0_
             / /
            0 1
---------------
```

```
---------------
  1_  times   _0__
 / /         / / /
1 0         0 1 1
---------------
2  times   _0__
|         /   /
0        0   0_
            / /
           1 1
---------------
```

Figure 6.5: The 20 terms in γ_1^2 displayed with ASCII art.

6.1 Source Code

```
import functools

#shorthand for building trees
def LOT(xs,n): return LabelledOrderedTree(xs, label=n)

#the zero tree
zeroTree=LOT([],label=-1)

#contract leftmost input and add weights
def contractLeft(T):
    branches = [t for t in iter(T[0])] + [t for t in iter(T[1:])]
    new_label = T.label() + T[0].label()
    return LabelledOrderedTree(branches, label=new_label)

#tree sum
#mod-2 reduces a list of elements
def mod2Reduce(ts):
    rs=[]
    for t in ts:
        l1 = len(ts) #length with t
        ts = [u for u in ts if u != t]
        l0 = len(ts) #length without
        if (l1-l0) % 2 == 1:
            rs.append(t)
        #ts = us
    return rs

#proper union function of a list of lists
def overUnion(xss):
    if xss == []: return []
    else: return functools.reduce((lambda x,y:x+y), xss)

#expand a weighted tree at a given location
def expandAt(S,path):
    #expanding at the root vs elsewhere
    if path == ():
        T=S
    else:
        T=S[path]

    l=len(T)
    w=T.label()
    #ts0 and ts1 only need to be considered when weight is positive
    if w>0:
    #expansions that create a new lollipop
        #expanding higher degree vertices
        if l>1:
            ts0=[LOT(T[:i]+[LOT([],u)]+T[i:], w-u) for i in range(l+1) for u in range(1,w+1)]
        #expanding a degree-2
        elif l==1:
            ts0=[LOT(T[:i]+[LOT([],u)]+T[i:], w-u) for i in range(l+1) for u in range(1,w+1)]
        #expanding a degree-1 vertex
        #degree-2 vertices must have positive weight, so u=w is excluded
        else:
            ts0=[LOT(T[:i]+[LOT([],u)]+T[i:], w-u) for i in range(l+1) for u in range(1,w)]

    #expansions that create a degree 2 vertex
        #if the degree of the original vertex is 1, then there is nothing to create here
        if l==0:
            ts1=[]
        #if the degree of the original vertex is 2,
        #there is only one new vertex that can be created
        elif l==1:
            ts1=[LOT([LOT([t for t in T], u)], w-u) for u in range(1,w)]
        #if the degree of the original vertex is at least 3,
        #create a vertex along each branch including "below"
        else:
            ts1a=[LOT([LOT([t for t in T], u)], w-u) for u in range(w)]
            ts1b=[LOT(T[:i]+[LOT([T[i]],u)]+T[i+1:], w-u) for i in range(l) for u in range(1,w+1)]
            ts1 = ts1a + ts1b
    else:
        ts0=[]
        ts1=[]
    #expansions of the remaining types
    #only need to consider when degree is at least 4
    if l==3:
        #ts3 is redundant in this case
        ts2=[LOT([LOT(T[:l-1],u),T[l-1]], w-u) for u in range(w+1)]
        ts3=[]
        ts4=[LOT([T[0],LOT(T[1:],u)], w-u) for u in range(w+1)]
    elif l>3:
        ts2=[LOT([LOT(T[:l-1],u),T[l-1]], w-u) for u in range(w+1)]
        ts3=[LOT(T[:i]+[LOT(T[i:i+2],u)]+T[i+2:], w-u) for i in range(l-1) for u in range(w+1)]
        ts4=[LOT([T[0],LOT(T[1:],u)], w-u) for u in range(w+1)]
    else:
        ts2=[]
        ts3=[]
        ts4=[]

    #ts0 may have repetitions, so we need to compute its sum mod-2
    ts=mod2Reduce(ts0)+ts1+ts2+ts3+ts4

    #now append each of these to S
    es = []
    for t in ts:
        with S.clone() as U:
            if path == ():
                U = t
            else:
                U[path] = t
            es.append(U)
    return es

#boundary map for weighted or unweighted trees
def boundary(T):
    return overUnion([expandAt(T,path) for path in T.paths()])

#check if vertex (path in tree) is internal (i.e. not a leaf or a 1-lollipop)
def isInternal(v,T):
    if (len(T[v]) == 0) & (T[v].label()<2): return False
    else: return True
#H procedure
def HAlg(T):
    for v in T.paths():
        if isInternal(v,T)==False:
            continue
        if len(T[v])>2:
            return zeroTree
        if len(T[v])==2:
            #check left input and contract if valid
            if isInternal(0,T[v])==False:
                continue
            else:
                #contracting at the root vs elsewhere
                if v == ():
                    return contractLeft(T)
                else:
                    with T.clone() as S:
                        S[v]=contractLeft(S[v])
                        return S
    #if all vertices have been exhauted, H(T)=0
    return zeroTree

#alpha map
def alpha(T):
    ts = mod2Reduce([T]+boundary(HAlg(T))+[HAlg(S) for S in boundary(T)])
    if ts == []: return [zeroTree]
    else: return ts
#linearly extend over sums
def alphaLE(ts):
    us = []
    for t in ts:
        x = alpha(t)
        us.append(x)
    return mod2Reduce(overUnion(us))

#total contraction for trees in K_n
def TotalH(T):
    hs=[]
    Us = [T]
    while (alphaLE(Us) != [zeroTree]) & (set(alphaLE(Us)) != set(Us)):
        for U in Us:
            hs.append(HAlg(U))
        Us = alphaLE(Us)
    return mod2Reduce(hs)

#fully right-associated unweighted tree on n leaves
def RAT(n):
    if n<1: return zeroTree
    if n==1: return LOT([], 0)
    if n==2: return LOT([LOT([],0), LOT([],0)], 0)
    else: return LOT([LOT([],0), RAT(n-1)], 0)

#tree comp operators
def comp(T,i,S):
    j=1
    for v in T.paths():
        if (len(T[v])==0) & (T[v].label()==0):
            #composing when T is RAT(1)
            if v==():
                return S
            elif j==i:
                with T.clone() as U:
                    U[v]=S
                    return U
            j+=1
    #returns 0 if i invalid
    return zeroTree

#count number of input leaves
def leafCount(T):
    j=0
    for v in T.paths():
        if (len(T[v])==0) & (T[v].label()==0): j+=1
    return j

def totalComp(T,S):
    return [comp(T,i+1,S) for i in range(leafCount(T))]

#projection to R_n
#returns 0 if not a binary tree (i.e. 0-cell of K_n)
def toRAT(T):
    n = leafCount(T)
    for v in T.paths():
        if len(T[v])>2: return zeroTree
    return RAT(n)

#reduce sums of binary tensors mod-2
def tensorReduce(ts):
    xs = []
    for t in ts:
        if (t[0]==zeroTree) | (t[1]==zeroTree): continue
        xs.append(t)
    xs = mod2Reduce(xs)
    if xs == []: return [(zeroTree,zeroTree)]
    else: return xs

#total comp operator on binary tensors T=(t0,t1) and S=(s0,s1)
#assumes all four trees have the same number of input leaves
def totalTComp(T,S):
    return [(comp(T[0],i+1,S[0]), comp(T[1],i+1,S[1])) for i in range(leafCount(T[0]))]

#linearly extended total comp over binary tensors
def totalTCompLE(Ts,Ss):
    return tensorReduce(overUnion([totalTComp(T,S) for T in Ts for S in Ss]))

#unweighted diagonal constructor
def Diag(n):
    if n<2: return []
    elif n==2: return [(RAT(2),RAT(2))]
    else:
        tss = [totalTCompLE(Diag(i),Diag(n+1-i)) for i in range(2,n)]
        ts = overUnion(tss)
        gamma1 = [(t[0],h) for t in ts for h in TotalH(t[1])]
        gamma2 = [(h,toRAT(t[1])) for t in ts for h in TotalH(t[0])]
        gamma = gamma1 + gamma2
        return tensorReduce(gamma)

#check if vertex (path in tree) is internal (i.e. not a leaf or a 1-lollipop)
def isPWIV(v,T):
    if T[v].label()>1: return True
    elif (T[v].label()==1) & (len(T[v])>0): return True
    else: return False
#J procedure
def JAlg(T):
    for v in T.paths():
        if isPWIV(v,T)==False:
            continue
        if (len(T[v])==1) & (T[v].label()==1):
            #contracting at the root vs elsewhere
            if v == ():
                return contractLeft(T)
            else:
                with T.clone() as S:
                    S[v]=contractLeft(S[v])
                    return S
        return zeroTree
    #if no PWIV's, we need to do HAlg to the underlying unweighted tree
    return HAlg(T)

#xi map
def xi(T):
    ts = mod2Reduce([T]+boundary(JAlg(T))+[JAlg(S) for S in boundary(T)])
    if ts == []: return [zeroTree]
    else: return ts
#linearly extend over sums
def xiLE(ts):
    us = []
    for t in ts:
        x = xi(t)
        us.append(x)
    return mod2Reduce(overUnion(us))

#total contraction for trees in X_0^w
def TotalJ(T):
    hs=[]
    Us = [T]
    while (set(xiLE(Us)) != set(Us)):
        for U in Us:
            hs.append(JAlg(U))
        Us = xiLE(Us)
    return mod2Reduce(hs)

#fully right-associated tree on n leaves whose weights are all 1
def JRAT(n):
    if n<1:
        return zeroTree
    elif n==1:
        return LOT([],1)
    else:
        with (RAT(n)).clone() as T:
            leaves = [v for v in T.paths() if len(T[v])==0]
            for v in leaves:
                T.set_label(v, 1)
            return T
#count number of 1-lollipops
def lollipopCount(T):
    j=0
    for v in T.paths():
        if (len(T[v])==0) & (T[v].label()==1): j+=1
    return j
#projection to JRAT(n)
#returns 0 if there is a PWIV (i.e. not a 0-cell of X_0^w)
def toJRAT(T):
    n = lollipopCount(T)
    for v in T.paths():
        if isPWIV(v,T): return zeroTree
    return JRAT(n)

#check if v is a weight 0 leaf
def isLeaf(v,T):
    if (len(T[v])==0) & (T[v].label()==0): return True
    else: return False
```

```
#fully right-associated tree on n+w leaves whose rightmost w leaf weights are all 1
def KRAT(n,w):
    if (n<0) | (w<0):
        return zeroTree
    elif n==0:
        return JRAT(w)
    elif w==0:
        return RAT(n)
    else:
        return LOT([LOT([], 0), KRAT(n-1,w)], 0)

#projection to KRAT(n,w)
#returns 0 if there is a PWIV or vertex of degree > 3 (i.e. not a 0-cell of X_n^w)
def toKRAT(T):
    n = leafCount(T)
    w = lollipopCount(T)
    for v in T.paths():
        if isPWIV(v,T) | (len(T[v]) > 2): return zeroTree
    return KRAT(n,w)

#K procedure
def KAlg(T):
    n = leafCount(T)
    if n==0:
        return JAlg(T)
    m = 0
    for v in T.paths():
        subT = T[v]
        #if all leaves have been found, then apply JAlg to remainder of tree
        if m==n:
            R = RAT(n+1)
            J = JAlg(subT)
            return comp(R,n+1,J)
        if isLeaf(v,T):
            m += 1
            continue
        if (len(subT)==2) & (subT.label()==0):
            #check that left input isn't an input leaf
            if isLeaf(0,subT) == False:
                #contracting at the root vs elsewhere
                if v == ():
                    return contractLeft(T)
                else:
                    with T.clone() as S:
                        S[v]=contractLeft(S[v])
                        return S
            #skip v if left input is an input leaf
            else: continue
        if (len(subT)>2) | (subT.label()>0):
            return zeroTree

#zeta map
def zeta(T):
    ts = mod2Reduce([T]+boundary(KAlg(T))+[KAlg(S) for S in boundary(T)])
    if ts == []: return [zeroTree]
    else: return ts
#linearly extend over sums
def zetaLE(ts):
    us = []
    for t in ts:
            x = zeta(t)
            us.append(x)
        return mod2Reduce(overUnion(us))

#total contraction for trees in X_n^w
def TotalK(T):
    hs=[]
    Us = [T]
    while (set(zetaLE(Us)) != set(Us)):
        for U in Us:
            hs.append(KAlg(U))
        Us = zetaLE(Us)
    return mod2Reduce(hs)

#valid pairs of tuples in the recursion for wDiag(n,w)
def diagRecursion(n,w):
    ds = []
    for i in range(n+2):
        for u in range(w+1):
            if ( ((i,u)!=(0,0) ) & ( (i,u)!=(1,0) ) & ( (i,u)!=(n+1,w) ) & ( (i,u)!=(n,w) ) ):
                ds.append( ((i,u),(n+1-i,w-u)) )
    return ds

#weighted diagonal constructor
def wDiag(n,w):
    if w<0: return []
    elif w==0: return Diag(n)
    elif n<0: return []
    elif (w==1) & (n==0): return [(JRAT(1),JRAT(1))]
    else:
        ds = diagRecursion(n,w)
        tss = [totalTCompLE(wDiag(x[0][0],x[0][1]),wDiag(x[1][0],x[1][1])) for x in ds]
        ts = overUnion(tss)
        gamma1 = [(t[0],h) for t in ts for h in TotalK(t[1])]
        gamma2 = [(h,toKRAT(t[1])) for t in ts for h in TotalK(t[0])]
        gamma = gamma1 + gamma2
        return tensorReduce(gamma)
```

7 References

[1] A. Joyal and R. Street. Braided Tensor Categories. *Advances in Mathematics*, 102(1):20–78, 1993.

[2] Bernhard Keller. Introduction to A-infinity algebras and modules. *Homology, Homotopy and Applications*, 3(1):1 – 35, 2001.

[3] Robert Lipshitz, Peter Ozsváth, and Dylan Thurston. Diagonals and A-infinity tensor products. 2020. preprint, Accessed 04-13-2023.

[4] Michael A. Mandell. Operads and operadic algebras in homotopy theory. In *Stable categories and structured ring spectra*, volume 69 of *Math. Sci. Res. Inst. Publ.*, pages 183–247. Cambridge Univ. Press, Cambridge, 2022.

[5] Martin Markl. Models for operads. *Comm. Algebra*, 24(4):1471–1500, 1996.

[6] Martin Markl. Operads and PROPs. In *Handbook of algebra. Vol. 5*, volume 5 of *Handb. Algebr.*, pages 87–140. Elsevier/North-Holland, Amsterdam, 2008.

[7] Naruki Masuda, Hugh Thomas, Andy Tonks, and Bruno Vallette. The diagonal of the associahedra. *J. Éc. polytech. Math.*, 8:121–146, 2021.

[8] OEIS Foundation Inc. The On-Line Encyclopedia of Integer Sequences, 2023. Published electronically at http://oeis.org/A000139.

[9] Samson Saneblidze and Ronald Umble. Diagonals on the permutahedra, multiplihedra and associahedra. *Homology Homotopy Appl.*, 6(1):363–411, 2004.

[10] Jim Stasheff. Homotopy associativity of H-spaces. II. *Transactions of the American Mathematical Society*, 108:293–312, 1963.

[11] Dov Tamari. The algebra of bracketings and their enumeration. *Nieuw Archief voor Wiskunde*, 3:131–146, 1962.

[12] Gabriel Valiente. *Algorithms on Trees and Graphs.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.