UNIVERSIDAD
SERGIO ARBOLEDA

Sobre el método de reescritura en el estudio de operads no simétricos

Edward Julian Salamanca Valencia

Universidad Sergio Arboleda
Escuela de ciencias exactas e ingeniería
Maestría en matemáticas aplicadas

Bogotá - 2022

UNIVERSIDAD
SERGIO ARBOLEDA

Sobre el método de reescritura en el estudio de operads no simétricos

Edward Julian Salamanca Valencia

Director:
Rafael S. González D'león

Trabajo presentado como requisito para optar al título de
Magister en matemáticas aplicadas

Universidad Sergio Arboleda
Escuela de ciencias exactas e ingeniería
Maestría en matemáticas aplicadas

Bogotá - 2022

Resumen

En este trabajo estudiamos la técnica combinatoria de reescritura, como está definida por Loday y Vallette, para el estudio de la propiedad de Koszul, definida por Ginzburg y Kapranov, en operads no simétricos. En particular, nos centramos en los operads no simétricos conjuntistas binarios $\mathcal{A}s$, $\mathcal{D}ias$ y $\mathcal{T}rias$, y sus operads no simétricos duales de Koszul $\mathcal{A}s$, $\mathcal{D}end$ y $\mathcal{T}ridend$. Los operads no simétricos $\mathcal{A}s$, $\mathcal{D}ias$ y $\mathcal{T}rias$ pueden definirse de acuerdo a una estrategia de "transferencia de energía" que puede extenderse para definir un nuevo operad no simétrico que llamamos $\mathcal{T}etraas$. Este operad permite modelar la categoría de álgebras tetra-asociativas. Mostramos como ejemplo que el álgebra de descensos de Solomon admite la estructura de álgebra tetra-asociativa. Mostramos también que el operad $\mathcal{T}etraas$ es conjuntista pero no es cancelativo. Usando código en SageMath/Python implementamos el método de reescritura para chequear la propiedad de Koszul en operads no simétricos conjuntistas. Con esto chequeamos los resultados positivos para esta propiedad sobre $\mathcal{D}ias$ y $\mathcal{T}rias$, demostrados directamente por Loday y por Loday-Ronco usando técnicas homológicas. Adicionalmente, usamos esta misma técnica para demostrar que el operad $\mathcal{T}etraas$, y su dual de Koszul $\mathcal{T}etradend$, son operads no simétricos de Koszul. Finalmente, generalizamos la construcción por transferencia de energía de operads binarios a operads $(k+1)$-arios, para un entero positivo k, introduciendo los operads no simétricos $(k+1)-\mathcal{D}ias$, $(k+1)-\mathcal{T}rias$ y $(k+1)-\mathcal{T}etraas$. Mostramos que la técnica de reescritura para estos operads no es concluyente para la propiedad de Koszul y dejamos como trabajo futuro estudiar dicha propiedad por medio del uso de técnicas homológicas.

Dedicado a mis padres Martha Lady y Eduardo, a mi hermano Oscar Eduardo y a mi compañera de vida Anggy Katherin. Sin ustedes no habría sido posible.

Agradecimientos

Quiero agradecer a todas aquellas personas que de una u otra manera han contribuido en la elaboración de este documento y en el desarrollo de esta maestría.
A la Universidad Sergio Arboleda por abrirme sus puertas y por su constante acompañamiento para lograr mis metas y objetivos pero sobre todo por permitir desarrollarme como profesional en todos los aspectos.
A la doctora Marta Corzo decana del programa de Matemáticas de la Universidad Sergio Arboleda por la confianza depositada en mi tanto en el ámbito laboral como académico.
A los profesores de la Universidad por compartir todo su conocimiento y por darme herramientas para poder ver la matemática desde otro punto de vista antes desconocido para mi.
A mi director el doctor Rafael González por su apoyo incondicional desde el primer día, por asesorarme, acompañarme y orientarme en el proceso de elaboración y culminación de este documento.

Índice general

Resumen

1. Introducción **1**
   - 1.1. Especies combinatorias . . . 1
   - 1.2. Operads (no simétricos) y la propiedad de Koszul . . . 1
   - 1.3. Operads estudiados . . . 2
   - 1.4. Estructura del documento . . . 3

2. preliminares **5**
   - 2.1. Categorías y Funtores . . . 5
   - 2.2. Especies combinatorias . . . 8
   - 2.3. Operads . . . 12
   - 2.4. Método de reescritura para determinar la propiedad de Koszul . . . 16

3. Operads estudiados **21**
   - 3.1. $\mathcal{A}s - \mathcal{A}s$ . . . 21
   - 3.2. $\mathcal{D}ias - \mathcal{D}end$ . . . 21
   - 3.3. $\mathcal{T}rias - \mathcal{T}ridend$ . . . 30

4. Código en SageMath **37**
   - 4.1. Sobre SageMath . . . 37
   - 4.2. Generando árboles planares . . . 37
   - 4.3. Concatenación y composición de árboles . . . 38
   - 4.4. Generando las reglas de reescritura . . . 41
   - 4.5. Generando los monomios críticos . . . 46
   - 4.6. Aplicando las reglas de reescritura . . . 48
   - 4.7. Generando un grafo dirigido por medio de reescrituras . . . 52
   - 4.8. Chequeando confluencia . . . 53

5. $\mathcal{T}etraas$ **56**
   - 5.1. Tetra-álgebras . . . 56
   - 5.2. $\mathcal{T}etradend$ . . . 59

6. Generalizaciones y futuras líneas de trabajo **65**
   - 6.1. Otras aridades . . . 65
   - 6.2. Otras técnicas . . . 67

Índice de figuras

2.1. Re etiquetamiento de un grafo a través de la biyección f. Imagen tomada de [13]. 8
2.2. Representación de una $(F + G)$-estructura. Imagen tomada de [1]. 9
2.3. Representación de una (FG)-estructura. Imagen tomada de [1]. 10
2.4. Representación de una $(F \circ G)$-estructura. Imagen tomada de [1]. 10
2.5. Representación de una F'-estructura. Imagen tomada de [1]. 11
2.6. Composición de especies combinatorias. 12

3.1. Poset de $\mathcal{D}ias(3)$ 30

4.1. Árbol básico con raíz y etiqueta creado en SageMath. 38
4.2. Árbol con raíz y etiqueta creado en SageMath 38
4.3. Árboles de referencia 39
4.4. Lista de concatenación de árboles 40
4.5. Ejemplo de composición de árboles 41
4.6. Relaciones del operad no simétrico $\mathcal{D}ias$ 47
4.7. Monomios críticos de $\mathcal{D}ias$ 48
4.8. 49
4.9. Ejemplo de la función *is_tree_in_monomial* 50
4.10. Ejemplo de reemplazo 50
4.11. ejemplo Función rewrite_rule_at_root 51
4.12. Ejemplo de reescritura en la raíz 52
4.13. Monomio crítico en $\mathcal{D}ias$ 53
4.14. Grafo dirigido en $\mathcal{D}ias$ 53

6.1. Ejemplo de un grafo dirigido no confluente en $3 - \mathcal{D}ias$ 67

Lista de códigos

codes/arbol.py . . . 37
4.1. Ejemplo de árbol con raíz y etiqueta . . . 38
codes/arbol–2.py . . . 38
4.2. Función concatenate_ordered_trees . . . 39
codes/ejemplo–concatenar.py . . . 39
4.3. Función get_signature_at_root . . . 40
4.4. Función compose_labelled_ordered_trees . . . 40
4.5. Función generate_all_quadratic_trees . . . 41
4.6. Función generate_all_binary_quadratic_trees . . . 42
4.7. Función is_light_on . . . 42
4.8. Función leaf_addresses . . . 43
4.9. Función tree_to_path_sequence . . . 43
4.10. Función compare_path_sequences y Función compare_planar_trees . . . 44
4.11. Función light_signature . . . 44
4.12. Función get_light_equivalence_classes . . . 45
codes/dias_equivalence_classes.py . . . 45
4.13. Algoritmo *QuickSort* . . . 45
4.14. Función get_list_of_rules_using_equivalence_classes . . . 46
codes/dias_rules.py . . . 46
4.15. Función get_list_of_leaders . . . 46
4.16. Función get_critical_tree_monomials . . . 46
4.17. Función is_tree_pattern_at_root . . . 48
4.18. Función is_tree_in_monomial . . . 49
codes/ejemplo_is_tree_in_monomial.py . . . 49
4.19. Función replace_tree_at_address . . . 50
codes/ejemplo–replace_tree.py . . . 50
4.20. Función rewrite_rule_at_root . . . 51
codes/ejemplo–rewrite–rule.py . . . 51
4.21. Función rewrite_rule_at_address . . . 52
4.22. Función generate_digraph_of_relations . . . 52
codes/ejemplo_digraf_dias.py . . . 53
4.23. Función is_digraph_confluent . . . 54
4.24. Función is_operad_koszul . . . 54
4.25. Resumen del método de reescritura para el operad no simétrico $\mathcal{T}rias$ . . . 54
6.1. Confluencia de los grafos dirigidos . . . 66

1 Introducción

1.1. Especies combinatorias

La teoría de especies combinatorias fue introducida por Andre Joyal en [7] hacia 1980 con el objetivo de formalizar las operaciones que se realizan entre series generatrices cuyos coeficientes enumeran familias de objetos combinatorios. Formalmente, una *especie combinatoria (conjuntista)* es un funtor $F : \mathbb{B} \to \mathbb{S}et$, en donde $\mathbb{B}$ es la categoría de conjuntos finitos y biyecciones entre estos, y $\mathbb{S}et$ es la categoría de conjuntos en general y funciones entre estos. Dado un conjunto finito V consideramos el conjunto $F[V]$ de *F-estructuras* generadas con etiquetas en V. Por ejemplo, F puede ser la especie de permutaciones y entonces $F[V]$ es el conjunto de permutaciones de V. A una especie F, que asocie para todo conjunto finito V una colección finita $F[V]$, uno puede asociarle su *serie generatriz exponencial* $F(x) = \sum_{n\geq 0} |F[n]|\frac{x^n}{n!}$, en donde denotamos $[n] := \{1, 2, \ldots, n\}$ y $F[n] := F[[n]]$. Dadas dos especies F y G uno puede definir por ejemplo nuevas especies $F + G$, $F \cdot G$ y $F \circ G$ que corresponden respectivamente a las operaciones de suma, multiplicación y composición de sus series generatrices. Cuando en el contexto de especies cambiamos en su definición la categoria $\mathbb{S}et$ por las categorias $\mathbb{T}op$, de espacios topológicos, o $Vect_{\mathbf{k}}$, de $\mathbf{k}$-espacios vectoriales, obtenemos respectivamente los conceptos análogos de especies topológicas y especies lineales. Si en la definición de especie cambiamos la categoría $\mathbb{B}$ por la categoría $\mathbb{L}$ de ordenes lineales finitos y sus isomorfismos de orden, el tipo de especie que se obtiene se conoce con el nombre de *especie no simétrica* o *especie rígida* (ver por ejemplo [13]). A una especie no simétrica R le podemos asociar su *serie generatriz ordinaria* que está dada por $\tilde{R}(x) = \sum_{n=0}^{\infty} |R[n]|x^n$.

1.2. Operads (no simétricos) y la propiedad de Koszul

Algunas de las operaciones definidas en el conjunto de las especies dan lugar a categorias monoidales. En particular, si consideramos monoides en la categoria monoidal formada por especies bajo la operación de composición de especies llegamos al concepto de *operad*. Este concepto aparece inicialmente en el trabajo de May [12] y se ha venido estudiando y aplicando en diferentes áreas de la matemática e incluso en otras

ciencias. El concepto de *operad no simétrico* nace de considerar monoides en la categoría monoidal de $\mathbb{L}$-especies no simétricas bajo la composición ordinal como operación.
Méndez define en [13] un operad (no simétrico) conjuntista $\mathcal{O}$ como una especie combinatoria (no simétrica) junto con un mecanismo de auto reproducción $\mathcal{O}(\mathcal{O}) \xrightarrow{\eta} \mathcal{O}$ que ensambla un conjunto de $\mathcal{O}$-estructuras usando una $\mathcal{O}$-estructura externa, obteniendo una estructura más grande de la misma clase y que satisface además las propiedades monoidales de asociatividad y existencia de identidad.
La mayoría de teorías desarrolladas alrededor del concepto de operad conjuntista fueron desarrolladas por separado. Fue hasta 1981 que Joyal en [7] dio una presentación general del concepto de operad conjuntista desde la perspectiva de la teoría de especies combinatorias. Usando el enfoque de la teoría de especies combinatorias en [14], Méndez y Yang desarrollan el concepto de operad conjuntista *cancelativo*. Para este, ellos muestran que dado un operad conjuntista cancelativo $\mathcal{Q}$ se le puede asociar una familia de conjuntos parcialmente ordenados (*posets* por su nombre en inglés) que dá origen al concepto de especies de Möbius. Este concepto permite interpretar desde el punto de vista de especies la inversa composicional de la serie generatriz de un operad conjuntista cancelativo $\mathcal{Q}$.
Por otro lado Ginzburg y Kapranov en [5] introducen la teoría de dualidad de Koszul para operads algebráicos usando técnicas homológicas similares a las desarrolladas en la categoría de álgebras asociativas. Central a esta teoría es el concepto de que un operad $\mathcal{O}$ sea de *Koszul*. Esta es una condición homológica que implica que su *operad dual de Koszul* $\mathcal{O}^!$ genera una resolución óptima para $\mathcal{O}$. Para todos los conceptos no definidos aquí sobre operads algebráicos y la teoría de dualidad de Koszul, el lector puede referirse al excelente libro de Loday y Vallette [11].
Existen varios métodos combinatorios para mostrar que un operad (no simétrico) y su dual de Koszul son de Koszul. Uno de estos métodos fue definido por Vallette en [16]. En este, Vallette usa los posets definidos por Méndez previamente (llamados *posets operádicos*) para chequear si la linealización de un operad conjuntista cancelativo y su dual de Koszul son operads de Koszul. Esto se hace chequeando si los posets operádicos asociados son *Cohen-Macaulay*. Esta técnica sin embargo, solo está definida para operads algebráicos que sean la linealización de un operad conjuntista cancelativo. Otros métodos usados para concluir si un operad es de Koszul usan los conceptos de bases de Gröbner para operads definidas por Dotsenko y Khoroshkin [3] y el método de reescritura. Este último será el que principalmente exploraremos en este trabajo. Particularmente en este trabajo nos centramos en estudiar el método de reescritura aplicado a varias familias relacionadas de operads conjuntistas no simétricos.

1.3. Operads estudiados

En [5] Ginzburg y Kapranov muestran que el operad no simétrico que modela álgebras asociativas $\mathcal{A}s$ es un operad conjuntista de Koszul y ademas que es autodual, esto significa que

$$\mathcal{A}s^! \cong \mathcal{A}s.$$

Loday en [9] introduce el operad no simétrico de diálgebras asociativas $\mathcal{D}ias$ y el operad no simétrico de diálgebras dendriformes $\mathcal{D}end$, además demuestra que tanto $\mathcal{D}ias$ como $\mathcal{D}end$ son operads de Koszul y que son duales en el sentido de Koszul uno del otro, es decir

$$\mathcal{D}ias^! \cong \mathcal{D}end.$$

En [10] Loday y Ronco definen el operad no simétrico de triálgebras asociativas $\mathcal{T}rias$ y el operad no simétrico de triálgebras dendriformes $\mathcal{T}ridend$, que extienden los operads $\mathcal{D}ias$ y $\mathcal{D}end$. Ellos demuestran

que $\mathcal{T}rias$ y $\mathcal{T}ridend$ son de Koszul y que son duales de Koszul entre sí, es decir

$$\mathcal{T}rias^{!} \cong \mathcal{T}ridend.$$

Tanto en [9] como en [10] se usa el cálculo directo de la homología de lo que se conoce como el *complejo de Koszul*. Chequeando que el complejo de Koszul es acíclico permite concluir que los operads anteriores son de Koszul. En [18] se dá una extensa y detallada lista de tipos de operads conocidos junto con algunas de sus propiedades, incluyendo el hecho de si se conoce que son o no de Koszul hasta la fecha.

1.3.1. Los operads no simétricos $\mathcal{T}etraas$ y $\mathcal{T}etradend$

Resulta que los operads no simétricos $\mathcal{A}s$, $\mathcal{D}ias$ y $\mathcal{T}rias$ pueden definirse mediante una idea de "transferencia de energía" que discutimos a fondo más adelante. Basados en esta idea estudiamos un nuevo operad no simétrico de álgebras tetra-asociativas, que denotamos como $\mathcal{T}etraas$, y que extiende simultáneamente a $\mathcal{A}s$, $\mathcal{D}ias$, y $\mathcal{T}rias$. Usando dualidad de Koszul se define igualmente el operad no simétrico $\mathcal{T}etradend$ de álgebras tetradendriformes. Damos como ejemplo de un álgebra tetra-asociativa el álgebra tensorial $(T(V), \top, \dashv, \vdash, \perp)$ donde $V = \bigoplus_{n\geq 0} \mathbf{k}\omega_n$ es el $\mathbf{k}$-espacio vectorial graduado, tal que el subespacio de elementos homogeneos de grado n es el espacio vectorial de dimensión uno generado por ω_n, para $n \geq 0$. Y las operaciones $(\top, \dashv, \vdash, \perp)$ se definen como

$$\begin{aligned}
(\omega_{n_1} \otimes \cdots \otimes \omega_{n_k}) \top (\omega_{m_1} \otimes \cdots \otimes \omega_{m_s}) &= \omega_{n_1+\cdots+n_k+m_1\cdots+m_s} \\
(\omega_{n_1} \otimes \cdots \otimes \omega_{n_k}) \dashv (\omega_{m_1} \otimes \cdots \otimes \omega_{m_s}) &= \omega_{n_1} \otimes \cdots \otimes \omega_{n_{k-1}} \otimes \omega_{n_k+m_1+\cdots+m_s} \\
(\omega_{n_1} \otimes \cdots \otimes \omega_{n_k}) \vdash (\omega_{m_1} \otimes \cdots \otimes \omega_{m_s}) &= \omega_{n_1+\cdots+n_k+m_1} \otimes \omega_{m_2} \otimes \cdots \otimes \omega_{m_s} \\
(\omega_{n_1} \otimes \cdots \otimes \omega_{n_k}) \perp (\omega_{m_1} \otimes \cdots \otimes \omega_{m_s}) &= \omega_{n_1} \otimes \cdots \otimes \omega_{n_{k-1}} \otimes \omega_{n_k+m_1} \otimes \omega_{m_2} \otimes \cdots \otimes \omega_{m_s}
\end{aligned}$$

Usando, por ejemplo, una versión análoga del método de Vallete en [16] para operads conjuntistas no simétricos uno podría buscar demostrar también que $\mathcal{A}s$, $\mathcal{D}ias$ y $\mathcal{T}rias$ (y por lo tanto sus duales de Koszul $\mathcal{A}s$, $\mathcal{D}end$ y $\mathcal{T}ridend$) son de Koszul. Sin embargo, dicha técnica no aplicaría para $\mathcal{T}etraas$ (y $\mathcal{T}etradend$) ya que según el Teorema 12 este operad no simétrico no es cancelativo.
En este trabajo, implementamos computacionalmente, usando SageMath/Python, el método de reescritura, como está descrito por Dotsenko y Bremmer en [2], para demostrar el siguiente teorema.

Teorema 1. *El operad no simétrico $\mathcal{T}etraas$ y su dual de Koszul $\mathcal{T}etradend$ son de Koszul.*

Es un corolario de la demostración del Teorema 1 que los operads no simétricos $\mathcal{A}s$, $\mathcal{D}ias$ y $\mathcal{T}rias$, y sus duales de Koszul son de Koszul.

1.4. Estructura del documento

En el Capítulo 2 hacemos una revisión general de conceptos de la teoría de categorías y de funtores que serán necesarios para introducir el concepto de especie combinatoria, sobre el cuál a su vez llegamos a las definiciones de operad y operad no simétrico. Finalmente presentamos el método de reescritura como aparece en Loday y Vallete en [11].
Los operads en los que basamos nuestro estudio son los operads no simétricos $\mathcal{A}s$, $\mathcal{D}ias$ y $\mathcal{T}rias$. Estos operads junto con sus duales de Koszul $\mathcal{A}s$, $\mathcal{D}end$ y $\mathcal{T}ridend$ respectivamente son presentados en el Capítulo 3. En este capítulo también estudiamos los conceptos de *álgebra asociativa*, *diálgebra asociativa*,

triálgebra asociativa, *diálgebra dendriforme* y *triálgebra dendriforme*. Hacemos también una presentación de estas estructuras en términos de árboles planares basandonos en las ideas expuestas por Loday en [9].

En el Capítulo 4 implementamos computacionalmente en SageMath el método de reescritura presentado en [11] para probar la propiedad de Koszul en operads conjuntistas no simétricos. Con este código verificamos además que los operads $\mathcal{A}s$, $\mathcal{D}ias$ y $\mathcal{T}rias$ son de Koszul.

Mediante una idea de "transferencia de energía" con la cuál se pueden construir los operads $\mathcal{A}s$, $\mathcal{D}ias$ y $\mathcal{T}rias$, en el Capítulo 5 definimos el operad no simétrico $\mathcal{T}etraas$. Este operad codifica un nuevo tipo de ágebras que llamamos *tetra-álgebra asociativas*. Estas a su vez son una extensión de las *triálgebra asociativas* por medio de una operación adicional que denotamos por $\top$. Damos además un ejemplo concreto de tetra-álgebra asociativa. Mostramos relaciones funtoriales entre la categoría de *tetra-álgebras asociativas* denotada por $Tetraas-mod$ y las categorías de álgebras, diálgebras y triálgebras asociativas denotadas por $As-mod$, $Dias-mod$ y $Trias-mod$ respectivamente. Definimos también una *tetra-álgebra dendriforme* y su operad asociado $\mathcal{T}etradend$ de tal forma que éste sea el dual de Koszul de $\mathcal{T}etraas$, es decir que

$$\mathcal{T}etraas^{!} \cong \mathcal{T}etradend.$$

Al final del Capítulo 5 mostramos que el operad no simétrico $\mathcal{T}etraas$ no es cancelativo y por tanto la técnica que involucra verificar la propiedad de Cohen-Macaulay en el estudio de posets operádicos no sirve para probar la propiedad de Koszul en este caso. Usando código implementado en el Capítulo 4, concluimos por el método de reescritura que tanto $\mathcal{T}etraas$, como $\mathcal{T}etradend$ son de Koszul (Teorema 1).

Finalmente, en el Capítulo 6 extendemos los operads binarios no simétricos estudiados en el Capítulo 3 y en el Capítulo 5 a otras aridades. Mostramos que el método de reescritura no es concluyente en estos operads para aridades diferentes de 2 y proponemos como trabajo futuro el estudio de otras técnicas principalmente homológicas para probar la propiedad de Koszul en los operads no simétricos de aridad $(k+1)$: $(k+1)-\mathcal{D}ias$, $(k+1)-\mathcal{T}rias$ y $(k+1)-\mathcal{T}etraas$, con k un entero positivo.

2
Preliminares

2.1. Categorías y Funtores

2.1.1. Categoría

Una categoría es un sistema de objetos relacionados entre si mediante mapeos llamados flechas o morfismos. Con objetos podemos referirnos a grupos, espacios topológicos, espacios vectoriales o simplemente a conjuntos y en cada uno de estos casos los morfismos son homomorfismos, funciones continuas, transformaciones lineales y funciones respectivamente. De hecho las categorias son en sí mismas uno de estos objetos matemáticos, y los morfismos entre categorias se llaman funtores, siguiendo por esa misma linea también se tienen morfismos entre funtores los cuales son llamados transformaciones naturales [8].

La teoría de categorias fue introducida en los años cuarenta por Samuel Eilenberg y Saunders Mac Lane en sus trabajos sobre topología algebraica, pero hoy en día el uso de la teoría de categorias va mas allá, se relaciona con practicamente todas las areas de la matemática pura, de hecho también se encuentra presente en algunas aplicaciones matemáticas en otras ciencias como la física o la computación, se ha convertido en una herramienta estandar en la informática lo cual ratifica como se menciona en [8] que "*Applied mathematics is more than just applied differential equations*"las matemáticas aplicadas son mas que solo aplicar ecuaciones diferenciales.

Formalmente una categoría $\mathcal{C}$ está dada por:

1. Una colección de objetos $ob(\mathcal{C})$.
2. Para cada $A, B \in ob(\mathcal{C})$, una colección $\mathcal{C}(A, B)$ de flechas (mapas o morfismos) de A a B.
3. Para cada $A, B, C \in ob(\mathcal{C})$, una aplicación $\mathcal{C}(B, C) \times \mathcal{C}(A, B) \to \mathcal{C}(A, C)$ tal que $(g, f) \mapsto g \circ f$ llamada composición y que es asociativa:

$$(h \circ g) \circ f = h \circ (g \circ f).$$

4. Para cada $A \in ob(\mathcal{C})$, un elemento $1_A \in \mathcal{C}(A, A)$ llamado la identidad de A y que satisface para $f \in \mathcal{C}(A, B)$ que:
$$f \circ 1_A = f = 1_B \circ f.$$

Ejemplo 1. *A continuación algunos ejemplos explicitos de categorías:*

- *Dado un cuerpo* $\mathbf{k}$*, la categoría de* $\mathbf{k}$*-espacios vectoriales* $Vect_{\mathbf{k}}$ *es aquella en la cual los objetos* $ob(Vect_{\mathbf{k}})$ *son* $\mathbf{k}$*-espacios vectoriales y los morfismos* $Vect_{\mathbf{k}}(V, W)$ *son transformaciones lineales entre los* $\mathbf{k}$*-espacios vectoriales* V *y* W*. Si el cuerpo* $\mathbf{k}$ *está fijo, simplemente denotamos la categoría por* $Vect$*.*
- *En la categoría* $\mathbb{S}et$ *los objetos* $ob(\mathbb{S}et)$ *son conjuntos, mientras que los morfismos* $\mathbb{S}et(A, B)$ *son funciones entre los conjuntos* A *y* B*.*
- *La categoría* $\mathbb{B}$ *en donde* $ob(\mathbb{B})$ *son conjuntos finitos y* $\mathbb{B}(A, B)$ *son biyecciones entre los conjuntos* A *y* B*.*

2.1.2. Funtor

Dadas dos categorias $\mathcal{A}$ y $\mathcal{C}$. Un funtor $F : \mathcal{A} \to \mathcal{C}$ es una función que:

- Asigna a cada objeto $X \in ob(\mathcal{A})$ un objeto $F[X] \in ob(\mathcal{C})$.
- Asigna a cada morfismo $f \in \mathcal{A}(X, Y)$ un morfismo $F[f] \in \mathcal{C}(F[X], F[Y])$ y que satisface las siguientes condiciones:
  i. $F[1_X] = 1_{F[X]}$, para todo $X \in ob(\mathcal{A})$.
  ii. $F[g \circ_{\mathcal{A}} f] = F[g] \circ_{\mathcal{C}} F[f]$, para toda $f \in \mathcal{A}(X, Y)$ y toda $g \in \mathcal{A}(Y, Z)$.

Tal funtor F es llamado usualmente funtor *covariante*.
Si para el funtor $F : \mathcal{A} \to \mathcal{C}$ se tiene que:

- Asigna a cada objeto $X \in ob(\mathcal{A})$ un objeto $F[X] \in ob(\mathcal{C})$.
- Asigna a cada morfismo $f \in \mathcal{A}(X, Y)$ un morfismo $F[f] \in \mathcal{C}(F[Y], F[X])$ y satisface las siguientes condiciones:
  i. $F[1_X] = 1_{F[X]}$, para todo $X \in ob(\mathcal{A})$.
  ii. $F[g \circ_{\mathcal{A}} f] = F[f] \circ_{\mathcal{C}} F[g]$, para toda $f \in \mathcal{A}(X, Y)$ y toda $g \in \mathcal{A}(Y, Z)$.

Entonces F es llamado un funtor *contravariante*.

2.1.3. Categoría Monoidal

Una categoría monoidal $(\mathcal{C}, \square, I, \alpha, \lambda, \rho)$ está formada por:

- Una categoría $\mathcal{C}$.
- Un bifuntor $\square : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$.
- Un objeto $I \in ob(\mathcal{C})$ llamado objeto identidad o unidad.

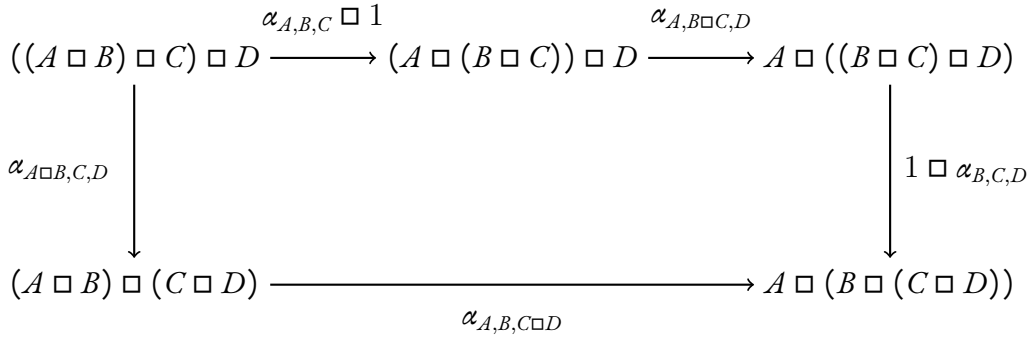
- Tres isomorfismos naturales:

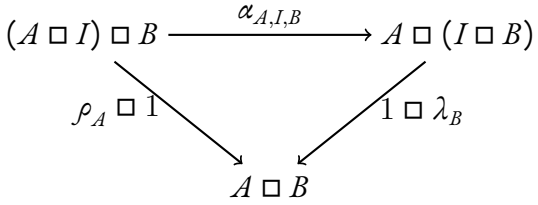
  i. $\alpha_{A,B,C} : (A \square B) \square C \simeq A \square (B \square C)$ llamado asociador.

  ii. $\lambda_A : I \square A \simeq A$ llamado unifuntor izquierdo.

  iii. $\rho_A : A \square I \simeq A$ llamado unifuntor derecho.

Estos isomorfismos naturales deben tener condiciones de coherencia, esto nos dice que para $A, B, C, D, I \in ob(\mathcal{C})$los siguientes diagramas son conmutativos:



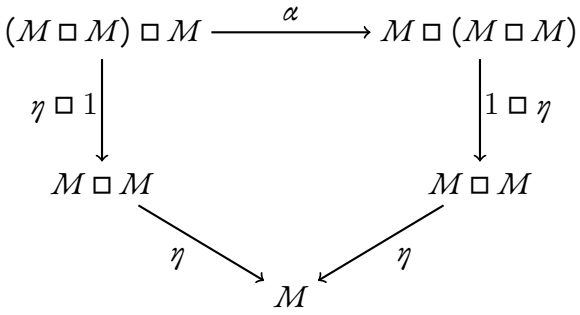


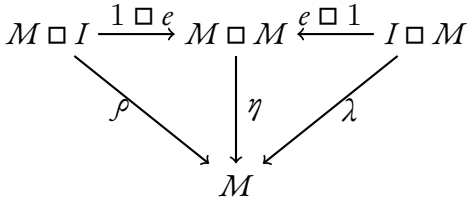
Un monoide (M, η, e) en una categoría monoidal $(\mathcal{C}, \square, I, \alpha, \lambda, \rho)$ es un objeto M de $\mathcal{C}$ junto con dos morfismos:

i. $M \square M \xrightarrow{\eta} M$ llamado producto y que es asociativo.

ii. $I \xrightarrow{e} M$ llamado unidad.

Y son tales que los siguientes diagramas conmutan:





Ejemplo 2. *Veamos algunos ejemplos de monoides:*

- *Un monoide en la categoría monoidal de conjuntos* $(\mathbb{S}et, \times, \{*\})$ *es un monoide (en el contexto de la teoría de grupos).*
- *Para un cuerpo* $\mathbf{k}$*, un monoide en la categoría monoidal de* $\mathbf{k}$*- espacios vectoriales* $(Vect_{\mathbf{k}}, \otimes, \mathbf{k})$ *es un algebra asociativa unital o con unidad.*
- *Un monoide en la categoría monoidal de endofuntores* $(EndoFunct_{\mathcal{C}}, \circ, Id_{\mathcal{C}})$ *es llamada una monada.*

2.2. Especies combinatorias

Informalmente una Especie de Estructura Combinatoria o simplemente una Especie Combinatoria es una clase de estructura finita etiquetada que es cerrada para re etiquetamientos.
Por ejemplo, un grafo $G = (V, \mathcal{E})$ está formado por un conjunto de etiquetas V para sus vértices, y un conjunto de pares no ordenados de vértices $\mathcal{E}$ llamados aristas que definen la estructura sobre V. Si cambiamos las etiquetas, es decir si definimos una biyección f entre V y otro conjunto finito W, G es automaticamente transformado por f en un nuevo grafo cuyas etiquetas están en W y cuyas aristas son transformadas por la asignación $\{a, b\} \to \{f(a), f(b)\}$, para $\{a, b\}$ en $\mathcal{E}$.
Agrupemos ahora todos los grafos que comparten el mismo conjunto de etiquetas V en un conjunto que denotaremos por $\mathcal{G}[\,V\,]$. Este es un conjunto finito, ademas toda biyección $f : V \to W$ induce otra que denotamos por $\mathcal{G}[f] : \mathcal{G}[\,V\,] \to \mathcal{G}[\,W\,]$ y que transporta a través de f cada grafo de $\mathcal{G}[\,V\,]$ en un grafo de $\mathcal{G}[\,W\,]$ como se ve en la figura 2.1.

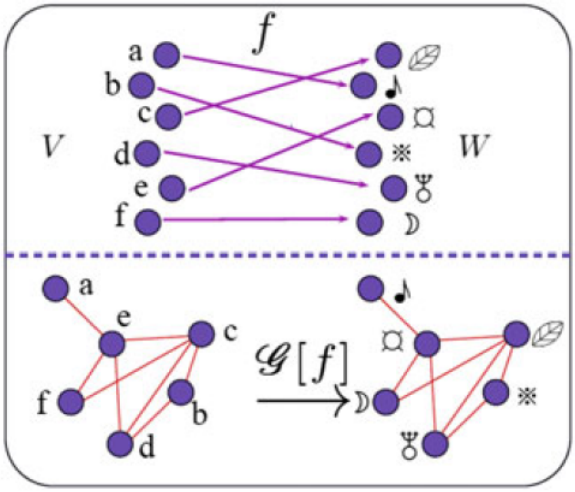


Figura 2.1: Re etiquetamiento de un grafo a través de la biyección f. Imagen tomada de [13].

Formalmente una *especie combinatoria* es un funtor covariante $F : \mathbb{B} \to \mathbb{S}et$.
Sean $V, W \in ob(\mathbb{B})$ y $\sigma \in \mathbb{B}(V, W)$. Un elemento $s \in F[\,V\,]$ es llamado una F-estructura sobre V. La función $F[\sigma]$ se llama tranasporte de F alrededor de σ.

A cada especie F se le puede asociar una serie de potencias formal llamada serie generatriz y que está relacionada con el conteo de las F-estructuras.

La serie generatriz (exponencial) de una especie F es de la forma

$$F(x) = \sum_{n=0}^{\infty} f_n \frac{x^n}{n!}$$

Donde $f_n = |F([n])|$ es el número de F- estructuras sobre el conjunto $[n] = \{1, 2, \ldots, n\}$.

Ejemplo 3. *Consideremos la especie L de órdenes lineales, sabemos que el número de órdenes lineales sobre un conjunto de n elementos está dado por $n!$, de allí que $l_n = |L([n])| = n!$ y por lo tanto la serie generatriz de esta especie está dada por*

$$L(x) = \sum_{n=0}^{\infty} l_n \frac{x^n}{n!} = \sum_{n=0}^{\infty} n! \frac{x^n}{n!} = \sum_{n=0}^{\infty} x^n = \frac{1}{1-x}.$$

En general, para una especie F usamos la notación $|F[n]|$ para referirnos al número de F-estructuras sobre $[n]$ en lugar de $|F([n])|$,esto para hacer la escritura un poco mas ligera.

Ejemplo 4. *Sea $\mathcal{P}$ la especie de partes de un conjunto, esta especie a cada conjunto V le asigna el conjunto partes (o conjunto potencia) de V.*
Como $p_n = |\mathcal{P}[n]| = 2^n$, se tiene que la serie generatriz de $\mathcal{P}$ es entonces:

$$\mathcal{P}(x) = \sum_{n=0}^{\infty} p_n \frac{x^n}{n!} = \sum_{n=0}^{\infty} 2^n \frac{x^n}{n!} = \sum_{n=0}^{\infty} \frac{(2x)^n}{n!} = e^{2x}.$$

2.2.1. Operaciones entre especies combinatorias

Definiremos una serie de operaciones en las categorías de especies y especies positivas.
Sean F y G dos especies de estructuras. La *suma* $F+G$ de F y G es la especie definida de la siguiente forma: Una $(F+G)$-estructura sobre un conjunto V es una F-estructura ó una G-estructura sobre V. Es decir

$$(F+G)[V] = F[V] + G[V]$$

Donde $F[V] + G[V]$ indica la unión disjunta de los conjuntos $F[V]$ y $G[V]$.

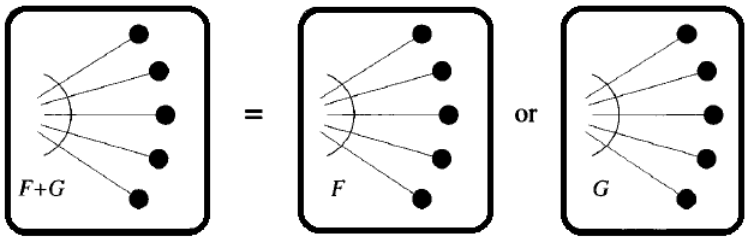


Figura 2.2: Representación de una $(F+G)$-estructura. Imagen tomada de [1].

La especie producto $F \cdot G$ o simplemente denotada FG se define como:

$$(FG)[V] = \sum_{\substack{(V_1, V_2) \\ V_1 \bigsqcup V_2 = V}} F[V_1] \times G[V_2].$$

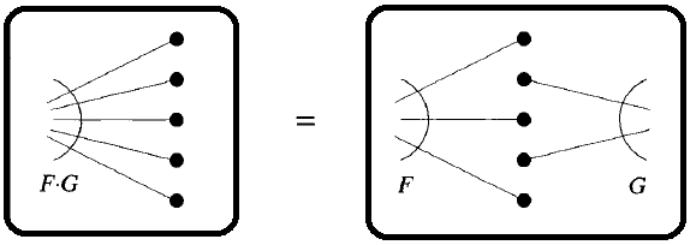


Figura 2.3: Representación de una (FG)-estructura. Imagen tomada de [1].

Si G es tal que $G[\varnothing] = \varnothing$, la especie $F \circ G$ también denotada $F[G]$ es llamada la composición de G en F y está definida por

$$(F \circ G)[V] = \sum_{\pi \in Par(V)} F[\pi] \times \prod_{p \in \pi} G[p], \tag{2.1}$$

en donde la suma recorre el conjunto de todas las particiones de V.

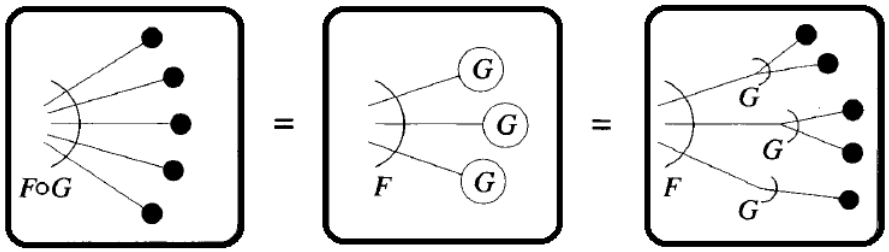


Figura 2.4: Representación de una $(F \circ G)$-estructura. Imagen tomada de [1].

La especie F' es llamada la derivada de F y está dada por:

$$F'[V] = F[V^+],$$

donde $V^+ = V + \{*\}$ y $*$ es un elemento elegido fuera de V.

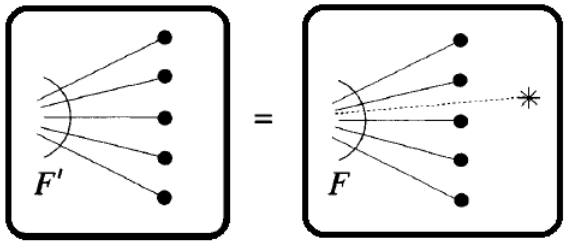


Figura 2.5: Representación de una F'-estructura. Imagen tomada de [1].

Las operaciones entre especies combinatorias son analogos combinatorios de las operaciones entre sus funciones generatrices suma, producto, sustitución (composición), y diferenciación [1]. Esto se resume en el siguiente lema.

Lema 2. *Si F y G son especies combinatorias con series generatrices $F(x)$ y $G(x)$ respectivamente, entonces las especies $F+G, FG, F\circ G, F'$ tienen series generatrices asociadas dadas por :*

a. $(F+G)(x) = F(x) + G(x)$

b. $(FG)(x) = F(x)G(x)$

c. $(F\circ G)(x) = F(G(x))$

d. $F'(x) = \dfrac{d}{dx}F(x)$

El lema anterior dice que las especies $F+G, FG, F\circ G, F'$ están definidas de tal forma que la enumeración de sus estructuras depende solamente de la enumeración de las F- estructuras y G-estructuras de la siguiente forma:

- El número de $(F+G)$-estructuras sobre un conjunto de n elementos es:

$$|(F+G)[n]| = |F[n]| + |G[n]|.$$

- El número de (FG)-estructuras sobre un conjunto de n elementos es:

$$|(FG)[n]| = \sum_{i+j=n} \frac{n!}{i!j!}|F[i]||G[j]|.$$

- El número de $(F\circ G)$-estructuras sobre un conjunto de n elementos es:

$$|(F\circ G)[n]| = \sum_{j=0}^{n} \sum_{\substack{n_1+\ldots+n_j=n \\ n_i>0}} \frac{1}{j!}\binom{n}{n_1,\ldots,n_j}|F[j]|\prod_{i=1}^{j}|G[n_i]|.$$

- El número de F'-estructuras sobre un conjunto de n elementos es:

$$|F'[n]| = |F[n+1]|.$$

Ejemplo 5. *Consideremos la especie End de endofunciones definida como* $End[V] = \{\psi | \psi : V \to V\}$, *la especie S de permutaciones o endofunciones biyectivas y la especie A de árboles con raíz.*
Sea $\psi \in End[10]$ *dada por*

$$\psi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 8 & 2 & 8 & 6 & 4 & 4 & 1 & 7 & 1 & 6 \end{pmatrix}.$$

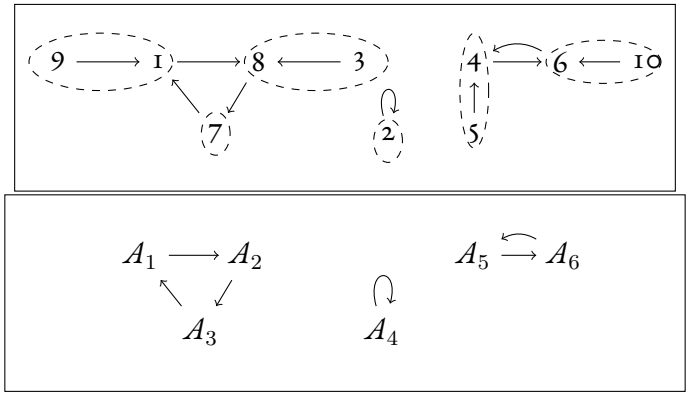


Figura 2.6: Composición de especies combinatorias.

La imagen 2.6 muestra que la endofunción ψ puede verse como una permutación de los árboles con raíz $A_1, A_2, A_3, A_4, A_5, A_6$, este hecho en general esta diciendo que que toda *End*-estructura se puede ver como una $S \circ A$- estructura, o en términos de operaciones de especies combinatorias que $End = S \circ A$.
Diremos que una especie F_+ es *positiva* si $F_+[\varnothing] = \varnothing$, es decir,si no asigna ninguna estructura al conjunto vacío. La serie generatriz asociada a una especie positiva F_+ es de la forma

$$F_+(x) = \sum_{n=1}^{\infty} |F_+[n]| \frac{x^n}{n!}.$$

Una especie G se dice una subespecie de F ($G \subseteq F$) si satisface las siguientes condiciones:

- Para $V \in ob(\mathcal{B})$ se tiene que $G[V] \subseteq F[V]$.
- Para $\sigma \in \mathbb{B}(V, W)$ se tiene que $G[\sigma] = F[\sigma]|_{G[V]}$.[1]

2.3. Operads

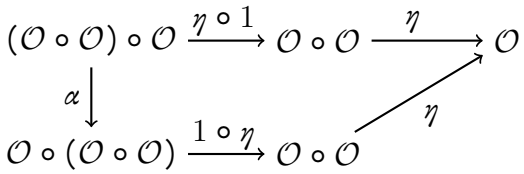
Si nos centramos en las especies combinatorias que bajo alguna operación son cerradas, llegamos al concepto de monoide dentro de cierta categoría monoidal. En particular cuando esta categoria monoidal está asociada a la composición de especies obtenemos el concepto de operad.
Consideremos la categoría monoidal de las Especies Positivas $(Sp_+, \circ, X, \alpha, \lambda, \rho)$ con la composición como operación asociativa y que tiene como identidad a la especie X definida por

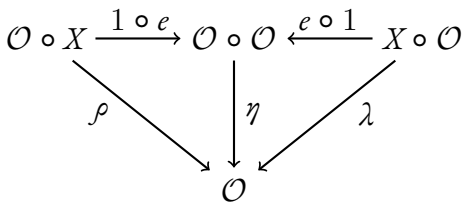
$$X[V] = \begin{cases} \{V\} & \text{si } |V| = 1, \\ \varnothing & \text{en caso contrario.} \end{cases}$$

Definición 1. *Formalmente un* operad $(\mathcal{O}, \eta, e)$ *es un monoide en esta categoría. Esto significa que:*

1. *El morfismo* $\mathcal{O} \circ \mathcal{O} \xrightarrow{\eta} \mathcal{O}$ *es un producto asociativo.*
2. *El morfismo* $X \xrightarrow{e} \mathcal{O}$ *es la identidad operadica.*

Y que los siguientes diagramas son conmutativos:





Aquí $1 : \mathcal{O} \to \mathcal{O}$ *es el morfismo identidad de* $\mathcal{O}$. *[11]*

2.3.1. Operads algebraicos y operads conjuntistas

Un *operad algebraico* es un objeto que modela las operaciones que definen cierto tipo de álgebras. Por ejemplo, existe el operad $\mathcal{A}s$ que codifica las álgebras asociativas, el operad $\mathcal{C}om$ para álgebras conmutativas y el operad $\mathcal{L}ie$ para álgebras de Lie [16].
Sea $\mathbf{k}$ un cuerpo (de característica cero). Denotamos por $Vect_{\mathbf{k}}$ a la categoría de $\mathbf{k}$-espacios vectoriales de dimensión finita y transformaciones lineales.

Una *especie Vectorial o Lineal* es un funtor covariante $\mathbb{B} \xrightarrow{R} Vect_{\mathbf{k}}$. Las *especies vectoriales* junto con las transformaciones naturales entre ellas forman una categoría (monoidal).
Las operaciones que definimos entre especies conjuntistas tienen su definicion analoga para especies vectoriales, en ese caso se debe cambiar el producto cartesiano $\times$ por el producto tensorial $\bigotimes$ y la union disjunta $\bigsqcup$ por la suma directa $\bigoplus$.
Formalmente un *operad algebraico* $(\mathcal{O}, \eta, e)$ es un monoide en la categoría monoidal de especies vectoriales positivas con respecto a la composición, esto significa que los morfismos $\eta : \mathcal{O} \circ \mathcal{O} \to \mathcal{O}$ y $e : X \to \mathcal{O}$ satisfacen los diagramas conmutativos de la definición 1.
Un *operad conjuntista* (*set operad*) o simplemente un *operad* como se definió antes es un monoide $(\mathcal{O}, \eta, e)$ en la categoría monoidal de especies positivas Sp_+.

Observación. *Sea* F *una especie combinatoria, como consecuencia de su definición se tiene que para un cojunto finito* $V \in ob(\mathbb{B})$ *el grupo simétrico de permutaciones de* V, $\mathbb{S}_V$ *actúa en* $F[V]$ *esto implica dos cosas. Por un lado se tiene una clase de acciones de los grupos simétricos* $\mathbb{S}_V \times F[V] \to F[V]$. *Por otro lado, como cualquier biyección* $f : V \to W$ *con* $W \in ob(\mathbb{B})$ *conecta los grupos* $\mathbb{S}_V$ *y* $\mathbb{S}_W$, *una permutación* $\sigma \in \mathbb{S}_V$

se transporta por conjugación con f a una única permutación $\tau \in \mathbb{S}_W$ que está dada por $\tau = f \circ \sigma \circ f^{-1}$, tenemos por tanto que la acciones de dos conjuntos de igual cardinal se conectan por la especie F:

$$F[\sigma] = F[f]^{-1} \circ F[\tau] \circ F[f].$$

Lo que se obtiene es una sucesión de acciones de los grupos simétricos $\mathbb{S}_n$:

$$\mathbb{S}_n \times F[n] \to F[n].$$

Esta sucesión de acciones es llamada un $\mathbb{S}$-conjunto.
Utilizando este concepto de $\mathbb{S}$-conjunto no es dificil mostrar que un operad conjuntista es equivalente a un monoide en la categoría monoidal de $\mathbb{S}$-conjuntos (que denotaremos $\mathbb{S}$-set) con composición definida de manera análoga a la ecuación (2.1).

Siguiendo esta linea podemos también definir un operad algebraico en términos de $\mathbb{S}$-módulos. Un $\mathbb{S}$-módulo sobre un cuerpo $\mathbf{k}$ es una sucesión $M = (M(0), M(1), \ldots M(n), \ldots)$ de $\mathbf{k}[\mathbb{S}_n]-$módulos $M(n)$.
Un $\mathbb{S}$-módulo M se dice finito si $M(n)$ es un espacio vectorial de dimensión finita para todo n, en donde n es llamada la *aridad* de un elemento $\mu \in M(n)$. Podemos ver a un operad algebraico como un monoide en la categoría monoidal de $\mathbb{S}$-módulos, denotada por $\mathbb{S}$-mod.
Si $\mathcal{O}$ es un operad conjuntista (set operad) entonces el $\mathbf{k}$-modulo libre $\hat{\mathcal{O}}(n) := \mathbf{k}[\mathcal{O}(n)]$ es un operad algebraico.
Un operad $\mathcal{O}$ es llamado *conexo* si es conexo como especie, esto significa que asigna solo una estructura a los singletons, es decir $|\mathcal{O}[1]| = 1$.
Si $\mathcal{R}$ es una subespecie de un operad $\mathcal{O}$. Decimos que $\mathcal{R}$ es un *ideal operadico* si la imagen de $\mathcal{O} \circ \mathcal{R}$ y la imagen de $\mathcal{R} \circ \mathcal{O}$ por η están contenidas en $\mathcal{R}$.
Sea $\mathcal{G}$ una especie conexa, y sea $\mathcal{G}_{2^+} = \sum_{k \geq 2} \mathcal{G}_k$ la especie de $\mathcal{G}$-estructuras excluyendo al singleton $\mathcal{G}_1$. Teniendo en cuenta que $\mathcal{G}_1$ es isomorfo a X, $\mathcal{G}$ puede escribirse como

$$\mathcal{G} = \mathcal{G}_1 + \mathcal{G}_{2^+} = X + \mathcal{G}_{2^+}.$$

El *operad libre conexo* generado por $\mathcal{G}$ es la especie lineal $\mathcal{F}_{\mathcal{G}}$ que satisface

$$\mathcal{F}_{\mathcal{G}} = X + \mathcal{G}_{2^+} \circ \mathcal{F}_{\mathcal{G}}.$$

La construcción del operad libre sobre V es dada por árboles cuyos vértices están indexados por los elementos de V. El operad libre está equipado con una graduación que corresponde al número de vértices de los árboles

$$\mathcal{F}_{\mathcal{G}} = \sum_{k=0}^{\infty} {\mathcal{F}_{\mathcal{G}}}^{(k)}.$$

${\mathcal{F}_{\mathcal{G}}}^{(k)}$ es la especie de árboles decorados con estructuras en $\mathcal{G}$ que tienen k vértices internos.
Sea $\mathcal{G}$ una especie conexa. Una relación cuadrática $\mathcal{R}$ en $\mathcal{F}_{\mathcal{G}}$ es un ideal operadico que está generado por una subespecie R de ${\mathcal{F}_{\mathcal{G}}}^{(2)}$(árboles con dos vértices).
Un operad de la forma $\mathcal{O} = \mathcal{F}_{\mathcal{G}}/\mathcal{R}$ es llamado *cuadrático* si $\mathcal{R} = \langle R \rangle$ es una relación cuadrática (de grado dos). El operad $\mathcal{O}$ es llamado *binario* si la especie de generadores es de cardinal 2, o sea $\mathcal{G} = \mathcal{G}_2$ [13].

2.3.2. Operads no simétricos

Un *orden lineal (total)* (no estricto) en un conjunto ℓ_X es una relación $\leq$ en $X \times X$ que es reflexiva, antisimétrica, transitiva, y que para cualquier par $x, y \in \ell_X$ se tiene que $x \leq y$ o $y \leq x$ (todos sus elementos son *comparables*). Si $f : \ell_X \to \ell_Y$ es una función entre dos órdenes lineales ℓ_X y ℓ_Y, decimos que f es *monótona* o *preservante de orden* si para todo $x \leq y$ en ℓ_X se tiene que $f(x) \leq f(y)$ en ℓ_Y. Si adicionalmente f es biyectiva decimos que f es un *isomorfismo* de órdenes lineales. Denotemos por $\mathbb{L}$ a la categoría de órdenes lineales finitos cuyos morfismos son isomorfismos de órdenes lineales. Es decir $ob(\mathbb{L})$ son órdenes lineales ℓ_X en donde X es un conjunto finito, y $\mathbb{L}(\ell_X, \ell_Y)$ esta formado por la única biyección que preserva los órdenes totales ℓ_X y ℓ_Y si $|X| = |Y|$, o es vacío en el caso de que $|X| \neq |Y|$.
Una *especie no simétrica* o una $\mathbb{L}$*-especie* es un funtor covariante $R : \mathbb{L} \to \mathbb{S}et$.

Observación. *Nótese que en* $\mathbb{B}$ *para* $X, Y \in ob(\mathbb{B})$ *tales que* $|X| = |Y| = n$ *se tiene que* $|\mathbb{B}(X, Y)| = n!$*, mientras que en* $\mathbb{L}$*, para* $\ell_X, \ell_Y \in ob(\mathbb{L})$ *tales que* $|X| = |Y|$ *tenemos* $|\mathbb{L}(X, Y)| = 1$*. Esto refleja el hecho de que en la categoría* $\mathbb{L}$ *no haya una acción de* $\mathbb{S}_{|X|}$ *sobre* $R[\ell_X]$ *cuando* R *es una especie no simétrica.*

A una $\mathbb{L}$-especie R podemos asociar dos clases de funciones generatrices:

- La función generatriz exponencial:

$$R(x) = \sum_{n=0}^{\infty} |R[n]| \frac{x^n}{n!}.$$

- Y la función generatriz ordinaria:

$$\tilde{R}(x) = \sum_{n=0}^{\infty} |R[n]| x^n.$$

Definimos dentro de la categoría de $\mathbb{L}$-especies no simétricas la *sustitución ordinal* $F \diamond G$ de G en F, también denotada $F\langle G \rangle$, y está definida por

$$(F \diamond G)[\ell_X] = \sum_{\ell_1 + \cdots + \ell_k = \ell_X} F[\ell_k] \times \prod_{i=1}^{k} G[\ell_i], \tag{2.2}$$

en donde la suma recorre el conjunto de todas las descomposiciones $\ell_1 \oplus \cdots \oplus \ell_k = \ell_X$ de ℓ_X como suma ordinal de órdenes lineales, y ℓ_k es el orden lineal inducido en las partes de la descomposición.
Un *operad no simétrico* $\mathcal{O}$ es un monoide en la categoría de $\mathbb{L}$-especies positivas con la composición ordinal como operación. Formalmente podemos decir que un operad no simétrico es una $\mathbb{L}$-especie positiva $\mathcal{O}$ junto con morfismos $\eta : \mathcal{O} \diamond \mathcal{O} \to \mathcal{O}$ y $e : X \to \mathcal{O}$ que satifacen las propiedades de asociatividad e identidad de la misma forma que ocurre en la Definición 1.

Observación. *Si en esta definición cambiamos la definición de composición ordinal en* $\mathbb{L}$*-especies positivas por una noción de composición similar a la de la ecuación* (2.1)*, conocida como shuffle, obtenemos otra clase de operads llamados* operads shuffle.

2.4. Método de reescritura para determinar la propiedad de Koszul

Un *árbol* T es un grafo orientado conexo sin ciclos. Sean T_1 y T_2 dos árboles (vistos como espacios topológicos de dimensión 1). Un morfismo de T_1 a T_2 es una función sobreyectiva y continua $f : T_1 \to T_2$ con las siguientes propiedades:

- f envía cada vértice de T_1 a un vértice de T_2 y cada arista de T_1 a una arista o vértice de T_2.
- f es monótona (preserva orientación).
- La imagen inversa de un punto de T_2 bajo f es un subárbol conexo en T_1.

Bajo estas condiciones tenemos entonces una categoría formada por árboles y que denotaremos *Trees* [5].

2.4.1. Dualidad de Koszul y operads de Koszul

La teoría de la dualidad de Koszul, llamada así en honor al matemático *Jean-Louis Koszul* en principio fue desarrollada para álgebras asociativas, pero luego fue extendida a operads binarios cuadráticos por *Ginzburg y Kapranov* en [5].
Para un operad cuadrático $\mathcal{O}$ hay asociado otro operad cuadrático, llamado el operad dual y denotado por $\mathcal{O}^!$. Uno de los principales resultados de la teoría de la dualidad de Koszul para operads es mostrar la existencia de un diferencial sobre el operad $(\mathcal{O}^!)^* \circ \mathcal{O} = \mathcal{O}^{\text{¡}} \circ \mathcal{O}$, lo cual da lugar al complejo de Koszul $(\mathcal{O}^{\text{¡}} \circ \mathcal{O}, \partial)$.
En lo que sigue definiremos lo que es el complejo de Koszul asociado a un operad cuadrático.
A un $\mathbb{S}$-módulo V se le puede asociar otro $\mathbb{S}$-módulo llamado el *dual de Czech* de V y denotado por $V^\vee$, por la fórmula $V^\vee(n) = V(n)^* \otimes sgn_{\mathbb{S}_n}$, donde V^* es el espacio lineal dual de V, y $sgn_{\mathbb{S}_n}$ es la representación signo de $\mathbb{S}_n$.
Sea $\mathcal{O} = \mathcal{F}(V)/\mathcal{R}$ un operad cuadrático. Su *operad dual de Koszul* es el operad cuadrático generado por el $\mathbb{S}$-módulo $V^\vee$ y las relaciones $\mathcal{R}^\perp$ generadas por las relaciones ortogonales a los generadores de $\mathcal{R}$. Este operad se denota por $\mathcal{O}^! = \mathcal{F}(V^\vee)/\langle R^\perp \rangle$.
Un *cooperad* es un operad en la categoría opuesta $\mathbb{S}$-mod^{op}, formalmente es una tripla $(\mathcal{C}, \Delta, \varepsilon)$ donde $\mathcal{C}$ es un $\mathbb{S}$-módulo, $\Delta : C \to C \circ C$ es un morfismo coasociativo y $\varepsilon : C \to I$ es una counidad [16].
Sea $\mathcal{O} = \mathcal{F}(V)/\mathcal{R}$ un operad cuadrático generado por un $\mathbb{S}$-módulo V de dimensión finita. El *cooperad dual de Koszul* de $\mathcal{O}$ está definido por el dual de Czech de $\mathcal{O}^!$ y se denota por $\mathcal{O}^{\text{¡}} = (\mathcal{O}^!)^\vee$ [16].
A un operad cuadrático $\mathcal{O}$ podemos asociar el siguiente morfismo sobre el $\mathbb{S}$-módulo $\mathcal{O}^{\text{¡}} \circ \mathcal{O}$:

$$\partial : \mathcal{O}^{\text{¡}} \circ \mathcal{O} \xrightarrow{\Delta'} \mathcal{O}^{\text{¡}} \circ \mathcal{O}^! \circ \mathcal{O} \xrightarrow{1_{\mathcal{O}^{\text{¡}}} \circ (I+\beta) \circ 1_{\mathcal{O}}} \mathcal{O}^{\text{¡}} \circ \mathcal{O} \circ \mathcal{O} \xrightarrow{1_{\mathcal{O}^{\text{¡}}} \circ \eta} \mathcal{O}^{\text{¡}} \circ \mathcal{O},$$

donde Δ' es la proyección del coproducto $\Delta = \eta_{\mathcal{O}^{!\vee}}$ del cooperad $\mathcal{O}^{\text{¡}}$ sobre el $\mathbb{S}$-módulo $\mathcal{O}^! \circ \mathcal{O}^!$ con solo un elemento de $\mathcal{O}^!$ en la derecha y β es la siguiente composición:

$$\beta : \mathcal{O}^{\text{¡}} = (\mathcal{F}(V^\vee)/\langle R^\perp \rangle)^\vee \twoheadrightarrow (V^\vee)^\vee \simeq V \hookrightarrow \mathcal{O} = \mathcal{F}(V)/\mathcal{R}.$$

En conclusión diremos que un operad cuadrático $\mathcal{O}$ es *Koszul* [11] si su complejo de Koszul asociado $(\mathcal{O}^{\text{¡}} \circ \mathcal{O}, \partial)$ es acíclico. Esto significa que su homología es el funtor identidad.

Se puede ver que si $\mathcal{O}$ es Koszul, entonces también lo es $\mathcal{O}^!$, en este caso las series generatrices respectivas satisfacen la ecuación funcional

$$F_{\mathcal{O}^!}(-F_{\mathcal{O}}(-x)) = x.$$

Existen varios operads de típo Koszul tales como $\mathcal{A}s$, $\mathcal{C}om$, $\mathcal{D}end$ y $\mathcal{L}ie$, algunos de estos los estudiaremos en detalle en las proximas secciones.

Ejemplo 6. *Los operads asociados a álgebras asociativas, álgebras conmutativas y álgebras de Lie denotados por $\mathcal{A}s$, $\mathcal{C}om$ y $\mathcal{L}ie$ respectivamente son Koszul, mas aún se tiene que:*

- $\mathcal{A}s^! \cong \mathcal{A}s$.
- $\mathcal{C}om^! \cong \mathcal{L}ie$.
- $\mathcal{L}ie^! \cong \mathcal{C}om$.

Es bien conocida la teoría de homología para estos operads. En el caso de $\mathcal{A}s$ se trata de la homología de Hochschild para álgebras asociativas, en $\mathcal{C}om$ es la homología de Harrison para álgebras conmutativas y para el operad $\mathcal{L}ie$ es la homología de Chevalley - Eilenberg para álgebras de Lie.
Además de las técnicas homológicas hay otras maneras de concluir que un operad es de Koszul. En las siguientes secciones mostraremos algunos métodos y técnicas tanto combinatorias como algebráicas para probar la *Koszulidad* de un operad.

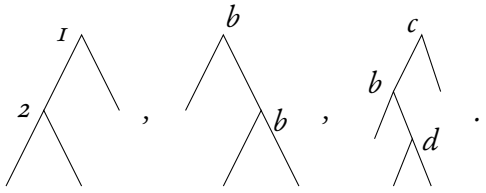
2.4.2. Orden Monomial

En esta sección se extiende el concepto de orden monomial descrito en [2] al caso de árboles monomiales no simétricos. Tambien mostramos como asociar a un árbol monomial una sucesión de palabras en un alfabeto χ.
Una colección de órdenes totales Ξ_n de $Trees(n)$ (árboles de aridad n) con $n \geq 0$, se dice un *orden monomial* si se satisfacen las siguientes condiciones:

- Cada Ξ_n es un buen orden.
- Cada composición no simétrica, es una función estrictamente creciente en sus argumentos; esto es: Si $T_0, T_0{}' \in Trees(r)$, $T_1, T_1{}' \in Trees(n_1), \ldots, T_r, T_r{}' \in Trees(n_r)$, entonces
  - $T_0 \circ (T_1, \ldots, T_r) < T_0{}' \circ (T_1, \ldots, T_r)$, si $T_0 < T_0{}'$.
  - $T_0 \circ (T_1, \ldots, T_i, \ldots, T_r) < T_0 \circ (T_1, \ldots, T_i{}', \ldots, T_r)$, si $T_i < T_i{}'$.

Para un árbol monomial T, las etiquetas de los vértices internos de cada camino desde la raíz hasta cada punto final (hoja) forman una palabra en el alfabeto χ que tiene un orden monomial. La sucesión de estas palabras es llamada la *sucesión de caminos* del árbol monomial T.

Ejemplo 7. *Consideremos los siguientes árboles monomiales*

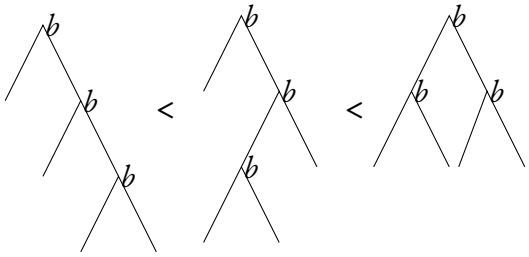


Las sucesiones de caminos son $(12, 12, 1), (b, bb, bb), (cb, cbd, cbd, c)$. *respectivamente.*

Observemos que un árbol monomial T está unicamente determinado por su sucesión de caminos, esto significa que si dos sucesiones de caminos son iguales, entonces los árboles de los cuáles provienen dichas sucesiones son iguales.
Supongamos que Ξ es un orden monomial. La *extensión de camino* de Ξ es el orden lexicográfico graduado o deglex en las sucesiones de caminos derivadas de Ξ, es decir:

- Si para T_1 y T_2 árboles monomiales, el número de puntos finales de T_1 es menor que el de T_2, entonces se tiene que $T_1 < T_2$.
- Si T_1 y T_2 tienen el mismo número de puntos finales, se comparan las sucesiones de caminos de cada árbol, palabra por palabra, usando el orden Ξ.

Ejemplo 8. *Los siguientes árboles han sido ordenados después de observar el orden entre sus sucesiones de caminos:*



$$(b, b^2, b^3, b^3) < (b, b^3, b^3, b^2) < (b^2, b^2, b^2, b^2).$$

2.4.3. Método de reescritura

El método de reescritura para operads no simétricos es una extensión del método de reescritura para álgebras asociativas desarrollado en [11], llevado a operads cuadráticos no simétricos. Consiste en un algoritmo que usa las relaciones que definen al operad como si fueran reglas de reescritura, este algoritmo permite concluir si un operad no simétrico es de Koszul.
Sea $\mathcal{O}(G, R)$ un operad cuadrático y no simétrico, en donde el espacio G de generadores es homogéneo de grado m y R es el espacio de relaciones de $\mathcal{O}$.

Paso 1. Consideremos una base ordenada para el espacio de operaciones G dada por: $\{\mu_1, \mu_2, ..., \mu_k\}$. El orden $\mu_1 < \mu_2 < ... < \mu_k$ será importante en el desarrollo de este método.

Paso 2. Las operaciones que generan el nivel 2 del operad libre no simétrico son de la forma $\mu_i \circ_a \mu_j$, con $a = 1, 2, \ldots, m$. Impondremos un orden total en este conjunto de la siguiente forma:

- $\mu_i \circ_a \mu_j < \mu_i \circ_b \mu_j$, para todo i, j y $a > b$,
- $\mu_i \circ_a \mu_j < \mu_k \circ_a \mu_l$, siempre que $i < k$, para todo $a = 1, \ldots, m$, y para todo j, l,
- $\mu_i \circ_a \mu_j < \mu_i \circ_a \mu_l$, siempre que $j < l$ para todo $a = 1, \ldots, m$.

Esta estrategia define un orden total en el conjunto $\{\mu_i \circ_a \mu_j \mid i,j = 1,\dots,k\,,\ a = 1,\dots,m\}$. Las relaciones del operad $\mathcal{O}$ que están determinadas por el espacio R pueden ser escritas en términos de los elementos de la base como:

$$r = \lambda\mu_i \circ_a \mu_j - \sum \lambda_{k,b,l}\mu_k \circ_b \mu_l,$$

con $\lambda, \lambda_{k,b,l} \in \mathbb{K}$, $\lambda \neq 0$, y en donde $\mu_i \circ_a \mu_j > \mu_k \circ_b \mu_l$ para todo (k,b,l) tal que $\lambda_{k,b,l} \neq 0$.

El término $\mu_i \circ_a \mu_j$ se conoce como el *término líder* del polinomio r. Dividiendo por λ podemos suponer que $\lambda = 1$. Si los términos líderes del conjunto de polinomios que generan R son todos distintos se dice que esta es la forma normalizada de la presentación.

Paso 3. Observemos que cada polinomio r de este tipo da lugar a una *regla de reescritura* en el operad $\mathcal{O}$, así:

$$\mu_i \circ_a \mu_j \mapsto \sum \lambda_{k,b,l}\mu_k \circ_b \mu_l.$$

Un monomio *crítico* es un árbol con tres vertices internos en el cuál sus dos subárboles de dos vértices internos son términos líderes.

Paso 4. Hay al menos dos formas de reescribir un monomio crítico hasta que no se pueda aplicar ninguna regla de reescritura. Si todas estas formas llevan al mismo elemento, entonces se dice que el monomio crítico es *confluente*.

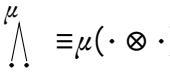
El siguiente teorema, cuya demostración puede consultarse en [11] es una herramienta poderosa para probar la *Koszulidad* de un operad no simétrico.

Teorema 3 (cf. [11]). *Sea $\mathcal{O}(G,R)$ un operad cuadrático no simétrico. Si su espacio de generadores G admite una base ordenada para la cuál existe un orden sobre los árboles planares de tal forma que todo monomio crítico es confluente, entonces el operad no simétrico $\mathcal{O}$ es de Koszul.*

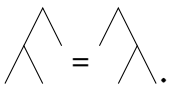
En este caso el operad no simétrico $\mathcal{O}$ admite una base formada por árboles planares libres de monomios líderes. Esta base se conoce como *base PBW* (Poincare-Birkhoff-Witt).

Ejemplo 9. *Un álgebra asociativa sobre un cuerpo* $\mathbf{k}$ *es un espacio vectorial A junto con una operación binaria $A \otimes A \xrightarrow{\mu} A$ dada por $\mu(a \otimes b) = ab$ que satisface la relación $(ab)c = a(bc)$.*

El operad no simétrico $\mathcal{A}s$ que es generado por la operación binaria *codifica las álgebras asociativas haciendo la identificación:*



$\mathcal{A}s$ satisface entonces la relación:

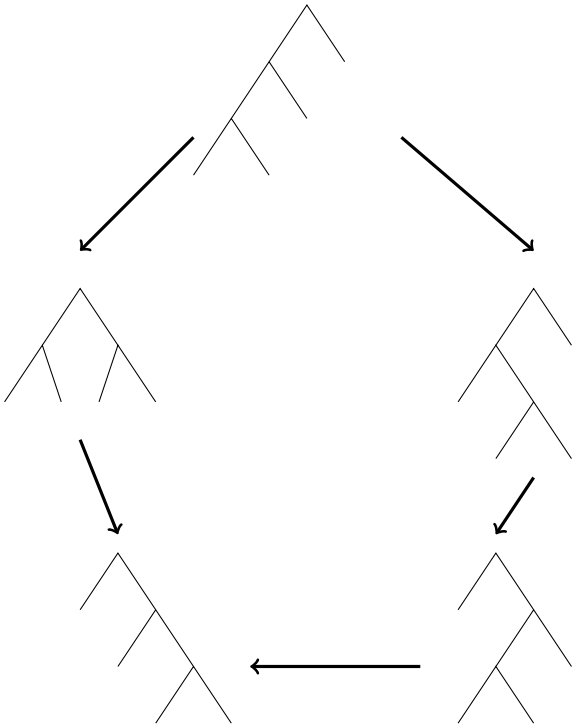


El término líder es

y tenemos solo el monomio crítico

.

Aplicando el método de reescritura se genera el siguiente grafo confluente



Así, por el teorema 3 concluimos que $\mathcal{A}s$ es un operad de Koszul.

Definición 2. *Dado un operad conexo $\mathcal{Q}$, en virtud de la ecuación 2.1 sean $(q_\pi, (a_p)_{p\in\pi})$ y $(q'_\pi, (a_p)_{p\in\pi})$ que pertenecen a $(\mathcal{Q}\circ\mathcal{Q})[n]$, con $q_\pi, q'_\pi \in \mathcal{Q}[\pi]$ y $a_p \in \mathcal{Q}[P]$, $\forall p \in \pi$.*
Si $\eta(q_\pi, (a_p)_{p\in\pi}) = \eta(q'_\pi, (a_p)_{p\in\pi})$ implica que $q_\pi = q'_\pi$ entonces el operad $\mathcal{Q}$ se dice cancelativo *o* básico.

Una técnica muy usada para probar la propiedad de Koszul de un operad cancelativo es la que involucra el estudio de posets y la función de Möbius.

Observación. *Asociado a un operad cancelativo $\mathcal{Q}$ existe una familia de posets $P_{\mathcal{Q}}[n]$ cuya función generatriz de Möbius*

$$M\ddot{o}b[P_{\mathcal{Q}}](x) = \sum_{n\geq 1} |P_{\mathcal{Q}}[n]|_{\mu} \frac{x^n}{n!}$$

es la inversa composiciónal de $\mathcal{Q}(x)$, donde $|P_{\mathcal{Q}}[n]|$ es el cardinal de Möbius definido en [13].
Vallette en [16] muestra la relación existente entre la estructura de los posets $P_{\mathcal{Q}}[n]$ y la propiedad de Koszul del operad cancelativo $\mathcal{Q}$. Cuando $\mathcal{Q}$ es un operad de Koszul, los cardinales de Möbius de los posets $P_{\mathcal{Q}}[n]$ son (salvo signos) las dimensiones del operad dual de Koszul $\mathcal{Q}^!$.

3
Operads estudiados

En este capítulo describiremos algunas características y propiedades de ciertos operad no simétricos. En particular nos centraremos en $\mathcal{A}s$ que codifica *álgebras asociativas*, $\mathcal{D}ias$ para las *álgebras diasociativas* y $\mathcal{T}rias$ para las *álgebras triasociativas*. Además introduciremos un nuevo operad no simétrico que denominaremos $\mathcal{T}etraas$ que modela un típo de álgebras llamadas álgebras tetra-asociativas y que extiende simultáneamente a los operads $\mathcal{A}s$, $\mathcal{D}ias$ y $\mathcal{T}rias$.

3.1. $\mathcal{A}s - \mathcal{A}s$

Como vimos en el Ejemplo 9, el operad no simétrico $\mathcal{A}s$ que modela álgebras asociativas es de Koszul. Si A es un álgebra asociativa la operación $\mu : A \otimes A \to A$ satisface la relación $\mu \circ (\mu \otimes 1_A) = \mu \circ (1_A \otimes \mu)$, en donde $1_A : A \to A$ es la función identidad. En esta relación las variables están en el mismo orden, esto nos indica que $\mathcal{A}s$ es un operad no simétrico.
Si denotamos por μ_n a la operación n-aria $\mu_n(a_1, \ldots, a_n) = a_1 \ldots a_n$, el espacio de operaciones n-arias $\mathcal{A}s(n)$ es de dimensión 1 y está generado por μ_n. Se tiene entonces que $\mathcal{A}s = \langle \mu_n \rangle$, de allí que $dim(\mathcal{A}s(n)) = 1$ y por tanto la serie generatriz ordinaria del operad no simétrico $\mathcal{A}s$ es:

$$F_{\mathcal{A}s}(x) = \frac{x}{1-x}.$$

En [11] Loday muestra que $\mathcal{A}s^! \cong \mathcal{A}s$.

Teorema 4. *El operad no simétrico $\mathcal{A}s$ es conjuntista y cancelativo.*

Demostración. La demostración se puede consultar en [16]. □

3.2. $\mathcal{D}ias - \mathcal{D}end$

Una *diálgebra* asociativa (álgebra diasociativa) es una estructura algebraica que es asociativa con respecto a dos operaciones que denotaremos $\dashv$ y $\vdash$, las diálgebras se estudian en detalle en [9].

Formalmente una diálgebra sobre un cuerpo $\mathbf{k}$ es un $\mathbf{k}$-espacio vectorial D junto con dos funciones lineales

$$\dashv, \vdash : D \otimes D \to D,$$

tal que para todo $a, b, c \in D$ se satisfacen las siguientes relaciones:

1. $(a \dashv b) \dashv c = a \dashv (b \dashv c)$
2. $(a \dashv b) \dashv c = a \dashv (b \vdash c)$
3. $(a \vdash b) \dashv c = a \vdash (b \dashv c)$
4. $(a \dashv b) \vdash c = a \vdash (b \vdash c)$
5. $(a \vdash b) \vdash c = a \vdash (b \vdash c)$.

Notemos que si $(D, \cdot)$ es un álgebra asociativa, y definimos para todo $a, b \in D$ $a \dashv b := a \cdot b$ y $a \vdash b := a \cdot b$, entonces $(D, \dashv, \vdash)$ es una diágebra. Se revela entonces la existencia de un funtor de la categoría de álgebras asociativas en la categoría de diálgebras:

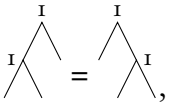
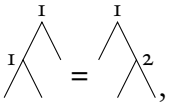
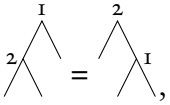
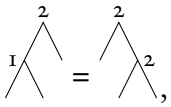
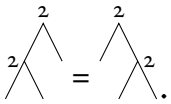
$$As - mod \to Dias - mod.$$

Por otro lado a partir de una diálgebra $(D, \dashv, \vdash)$ se tiene que $(D, \dashv)$ y $(D, \vdash)$ son álgebras asociativas, lo que implica que hay dos funtores de olvido que van de la categoría de diálgebras en la categoría de álgebras asociativas:

$$\begin{aligned} Dias - mod &\to As - mod \\ (D, \dashv, \vdash) &\mapsto (D, \dashv) \\ (D, \dashv, \vdash) &\mapsto (D, \vdash). \end{aligned}$$

Para simplificar la notación, denotaremos $\dashv = \circ_1$ como la operación 1 y $\vdash = \circ_2$ como la operación 2.

Se define entonces el operad no simétrico $\mathcal{D}ias$ como el generado por las operaciones binarias  y  , sujeto a las 5 relaciones anteriores. Este operad modela la categoría de diálgebras.
En términos de árboles binarios se tienen entonces las siguientes relaciones:

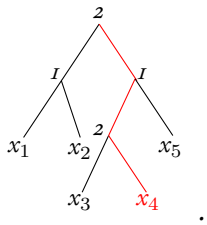
1.  ,
2.  ,
3.  ,
4.  ,
5.  .

3.2.1. Interpretación alternativa de las relaciones en $\mathcal{D}ias$

Podemos darle una interpretación alternativa a las relaciones en el operad $\mathcal{D}ias$. Si pensamos que las etiquetas de los vértices de cada uno de estos árboles transportan energía con la siguiente regla: Iniciando en

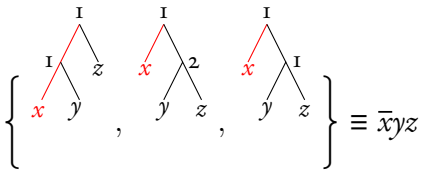
la raíz del árbol la energía va bajando siguiendo la ruta indicada por la operación que está en cada vértice, $\dashv \equiv 1$ hacia el primer hijo (izquierda) y $\vdash \equiv 2$ hacia el segundo hijo (derecha) hasta llegar finalmente a un término x_i.
Dado un monomio $x_1x_2 \ldots x_i \ldots x_k$, definimos el término *punteado* x_i como aquel en donde finaliza la energía descrita por el algortimo anterior. De esta manera un monomio en $\mathcal{D}ias$ puede verse como un monomio no conmutativo en donde una de sus variables aparece marcada: $x_1x_2 \ldots \overline{x_i} \ldots x_k$.

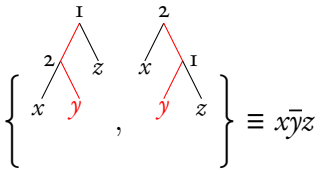
Ejemplo 10. *Consideremos el árbol*

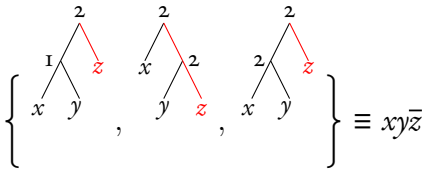


Siguiendo la idea anterior, la energía se origina en la raíz y va bajando hasta llegar al término punteado x_4. Podemos identificar entonces a este árbol con el monomio $x_1x_2x_3\overline{x_4}x_5$.

Si aplicamos esta regla de "energía" a los árboles que definen las 5 relaciones de $\mathcal{D}ias$ podemos ver que se generan 3 clases de equivalencia haciendo la identificación en cada caso con monomios que tienen una variable marcada, así:

1.  $\equiv \bar{x}yz$

2.  $\equiv x\bar{y}z$

3.  $\equiv xy\bar{z}$

Por lo tanto tenemos que $\mathcal{D}ias(3) = \langle \bar{x}yz, x\bar{y}z, xy\bar{z} \rangle$ y $dim(\mathcal{D}ias(3)) = 3$, en general se tiene que $dim(\mathcal{D}ias(n)) = n$.

Teorema 5 (Loday [9]). *Sea D una diálgebra y sean $x_1, \ldots, x_n \in D$, entonces:*

1. *Toda parentización de*

$$x_1 \circ_2 \cdots \circ_2 x_{i-1} \circ_2 x_i \circ_1 x_{i+1} \circ_1 \cdots \circ_1 x_n, \text{ con } \circ_1 \equiv 1 \text{ y } \circ_2 \equiv 2$$

da como resultado el mismo elemento en D que denotamos

$$x_1 \dots \overline{x_i} \dots x_n,$$

un monomio no conmutativo con la variable x_i marcada.

2. *Sea $m = x_{j_1} \dots x_{j_k}$ un monomio en D. Si x_{j_i} es su término punteado, entonces se tiene que :*

$$m = x_{j_1} \dots \overline{x_{j_i}} \dots x_{j_k}.$$

3. *En D se tienen las siguientes fórmulas:*

   - $(x_1 \dots \overline{x_i} \dots x_k) \circ_1 (x_{k+1} \dots \overline{x_j} \dots x_n) = x_1 \dots \overline{x_i} \dots x_k x_{k+1} \dots x_j \dots x_n.$
   - $(x_1 \dots \overline{x_i} \dots x_k) \circ_2 (x_{k+1} \dots \overline{x_j} \dots x_n) = x_1 \dots x_i \dots x_k x_{k+1} \dots \overline{x_j} \dots x_n.$

Este teorema muestra de forma explicita como hacer cálculos en una diálgebra D, su demostración puede consultarse en [9].

Ejemplo 11. *Sea D una diálgebra y sean $x_1, x_2, x_3, x_4, x_5, x_6 \in D$, entonces usando el teorema anterior tenemos:*

$$(x_1 \circ_1 (x_2 \circ_1 x_3)) \circ_2 ((x_4 \circ_2 x_5) \circ_1 x_6) = (\overline{x_1} x_2 x_3) \circ_2 (x_4 \overline{x_5} x_6) = x_1 x_2 x_3 x_4 \overline{x_5} x_6.$$

3.2.2. *Dend*

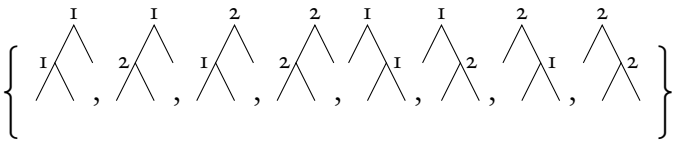
Una diálgebra *dendriforme* E sobre $\mathbf{k}$ es un $\mathbf{k}$-espacio vectorial junto con dos operaciones binarias

$$\prec, \succ : E \otimes E \to E,$$

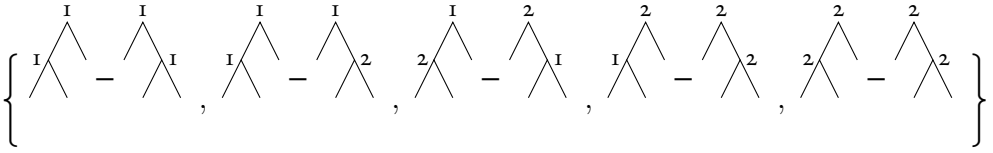
y que para $a, b, c \in E$ satisface:

1. $(a \prec b) \prec c = a \prec (b \prec c) + a \prec (b \succ c)$
2. $(a \succ b) \prec c = a \succ (b \prec c)$
3. $(a \prec b) \succ c + (a \succ b) \succ c = a \succ (b \succ c).$

El término *dendriforme* proviene de la palabra árbol en griego que se traduce como *dendro*, esto pues la estructura de esta álgebra se puede describir a través de árboles planares. Notemos ademas que las anteriores relaciones que definen una diálgebra dendriforme tienen las variables en el mismo orden, esto indica que el operad asociado $\mathcal{D}end$ es un operad no simétrico.
Recordemos que si $\mathcal{O} = \mathcal{F}(V)/\mathcal{R}$ es un operad cuadrático generado por un $\mathbb{S}$-módulo V y sujeto a la relación cuadrática $\mathcal{R}$, se define el operad dual de Koszul $\mathcal{O}^!$ de $\mathcal{O}$ como el operad generado por el $\mathbb{S}$-módulo dual de Czech $V^\vee$ y relaciones cuadráticas $\mathcal{R}^\perp$ generadas por elementos ortogonales a los generadores de $\mathcal{R}$. Para definir el operad dual de Koszul de $\mathcal{O}$ debemos entonces estudiar el complemento ortogonal del espacio generado por las relaciones que lo definen.
En el caso de $\mathcal{D}ias$, sabemos que está generado por los 8 árboles binarios



sujeto a las 5 relaciones que lo definen. Para hallar el dual de Koszul de $\mathcal{D}ias$ denotado por $\mathcal{D}ias^!$ buscamos el complemento ortogonal $\mathcal{R}^\perp$ del espacio de relaciones $\mathcal{R}$ el cuál está generado por



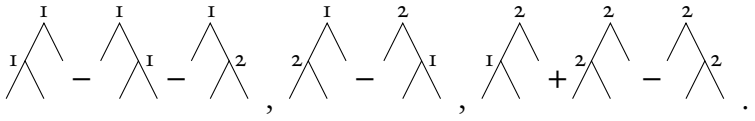
Con respecto a esta base, la matriz asociada es de tamaño 5×8 (5 relaciones y 8 árboles binarios), y está dada por

$$\begin{bmatrix} 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix}$$

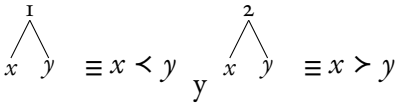
Después de escalonarla se obtiene la matriz

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \end{bmatrix}$$

Ahora, podemos hallar las relaciones generadoras de $\mathcal{R}^\perp$ buscando una base para el espacio nulo de esta matriz. Obtenemos así las siguientes relaciones que definen al operad $\mathcal{D}ias^!$, que adicionalmente consideran un signo menos $(-)$ que adquieren los árboles derechos gracias a las reglas de signos de Koszul (ver [11]):



Haciendo las identificaciones



las 3 relaciones que definen $\mathcal{D}ias^!$ toman la forma

$$\begin{aligned} (x \prec y) \prec z - x \prec (y \prec z) - x \prec (y \succ z) &= 0 \\ (x \succ y) \prec z - x \succ (y \prec z) &= 0 \\ (x \prec y) \succ z + (x \succ y) \succ z - x \succ (y \succ z) &= 0 \end{aligned}$$

y por tanto se tiene que $\mathcal{D}ias^! \cong \mathcal{D}end$. Como se mostró antes, las relaciones de $\mathcal{D}ias(3)$ están agrupadas en 3 clases de equivalencia, por esta razón $\mathcal{D}end(3)$ está generado por 3 relaciones.
En cuanto a dimensiones, se ve la relación existente entre el operad $\mathcal{D}ias$ y $\mathcal{D}end$; $\mathcal{D}end(3)$ está generado por 3 relaciones y $dim(\mathcal{D}ias(3)) = 3$, mientras que $\mathcal{D}ias(3)$ está generado por 5 relaciones y $dim(\mathcal{D}end(3)) = 5$.

3.2.3. Series generatrices de $\mathcal{D}ias$ y $\mathcal{D}end$

La serie generatriz de un operad no simétrico $\mathcal{O}$ en general es de la forma

$$F_{\mathcal{O}}(x) = \sum_{n\geq 1} f_n x^n,$$

donde $f_n = dim(\mathcal{O}(n))$.
Para el caso particular del operad no simétrico $\mathcal{D}ias$ vimos que $dim(\mathcal{D}ias(n)) = n$, por esta razón su serie generatriz está dada por:

$$\begin{aligned} F_{\mathcal{D}ias}(x) &= \sum_{n\geq 1} nx^n \\ &= x\sum_{n\geq 1} nx^{n-1} \\ &= x\frac{d}{dx}\sum_{n\geq 0} x^n \\ &= x\frac{d}{dx}\left(\frac{1}{1-x}\right) \\ &= \frac{x}{(1-x)^2}. \end{aligned}$$

Basandonos en esta serie hallaremos la serie generatriz del operad no simétrico$\mathcal{D}end$ para así identificar $dim(\mathcal{D}end(n))$, esto se puede hacer usando el Teorema 6 de la inversión de Lagrange.

Teorema 6 (cf. [4]). *[Teorema de inversión de Lagrange].*
Sea $\Phi(u) = \sum_{k\geq 0} \Phi_k u^k$ *una serie de potencias de* $\mathbb{C}[[u]]$ *con* $\Phi_0 \neq 0$. *Entonces, la ecuación* $y = z\Phi(y)$ *admite una única solución en* $\mathbb{C}[[z]]$ *cuyos coeficientes están dados por:*

$$y(z) = \sum_{n=1}^{\infty} y_n z^n, \text{ donde } \; y_n = \frac{1}{n}[u^{n-1}]\Phi(u)^n.$$

Este teorema es demostrado en [4].
Como sabemos $\mathcal{D}end$ es el operad dual de Koszul de $\mathcal{D}ias$, luego sus serie generatriz debe satisfacer la relación

$$F_{\mathcal{D}ias}(-F_{\mathcal{D}end}(-x)) = x \Leftrightarrow F_{\mathcal{D}ias}(-F_{\mathcal{D}end}(x)) = -x$$

Sea $g = F_{\mathcal{D}end}(x)$, entonces

$$F_{\mathcal{D}ias}(-g) = -x \Leftrightarrow \frac{g}{(1+g)^2} = x \Leftrightarrow x(1+g)^2 = g \Leftrightarrow x\Phi(g) = g$$

Donde $\Phi(g) = (1+g)^2$.
Si $g(x) = \sum_{n\geq 1} g_n x^n$, gracias al teorema de inversión de Lagrange tenemos:

$$\begin{aligned} g_n &= \frac{1}{n}[x^{n-1}]\Phi(x)^n \\ &= \frac{1}{n}[x^{n-1}](1+x)^{2n} \\ &= \frac{1}{n}[x^{n-1}]\sum_{k=0}^{2n}\binom{2n}{k}x^k, \textit{ haciendo } k = n-1 \\ &= \frac{1}{n}\binom{2n}{n-1} \\ &= \frac{1}{n+1}\binom{2n}{n} \\ &= c_n. \end{aligned}$$

Este número c_n se conoce como el n-ésimo número de Catalan. Concluimos que

$$F_{\mathcal{D}end}(x) = \sum_{n\geq 1} c_n x^n$$

Y por lo tanto $dim(\mathcal{D}end(n)) = c_n$.

3.2.4. Órdenes monomiales en $\mathcal{D}ias$

Como se vió en la Sección 2.4.2 podemos codificar árboles binarios decorados unívocamente por su sucesión de caminos, estas sucesiones se deben comparar en cada una de las cinco relaciones que definen el operad $\mathcal{D}ias$ para determinar cuales son los árboles líderes. Si fijamos un orden en el cual $\dashv < \vdash$, es decir $1 < 2$, tenemos para cada relación:

1. Las sucesiones de caminos correspondientes a la primer relación son $(11, 11, 1)$ y $(1, 12, 12)$. Comparando las longitudes de la primer coordenada de las sucesiones vemos que la longitud de la palabra 11 es mayor que la de 1, se tiene entonces que $(11, 11, 1) > (1, 12, 12)$ y por tanto el árbol

   es un término líder.

2. Por el mismo argumento del caso anterior, para esta relación tenemos que $(11, 11, 1) > (1, 11, 11)$ pero ya sabíamos que la sucesión $(11, 11, 1)$ genera un término líder; se deben entonces comparar las sucesiones $(1, 12, 12)$ y $(1, 11, 11)$. La longitud de la primer coordenada de ambas sucesiones es la misma, debemos entonces comparar la segunda coordenada, en este caso se tiene que $12 > 11$ pues se había fijado que $2 > 1$ por lo tanto $(1, 12, 12) > (1, 11, 11)$ y así el árbol

es un término líder.

3. La longitud de la primer coordenada de las sucesiones de camino de esta relación nos indican que $(12, 12, 1) > (2, 21, 21)$, concluimos entonces que el término líder es el árbol

4. Para esta relación, de nuevo por la longitud de la primer coordenada de las sucesiones de camino tenemos que $(21, 21, 2) > (2, 22, 22)$, de allí que el árbol

es un término líder.

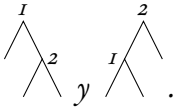
5. Finalmente para la última relación tenemos que el término líder es el árbol

esto pues las sucesiones de camino nos indican que $(22, 22, 2) > (2, 22, 22)$.

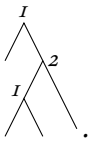
Todas las posibles composiciones entre estos términos lideres forman los monomios críticos. En total tenemos 14 monomios críticos que hallamos computacionalmente con el código en Python que será discutido en el capítulo 4. También chequeamos computacionalmente que todos los grafos (diamantes) derivados de estos monomios críticos son confluentes de acuerdo a el Teorema 3, recuperando el siguiente resultado.

Teorema 7 (Loday [9]). *El operad Dias de diálgebras y su dual de Koszul, el operad Dend de álgebras dendriformes, son operads de Koszul.*

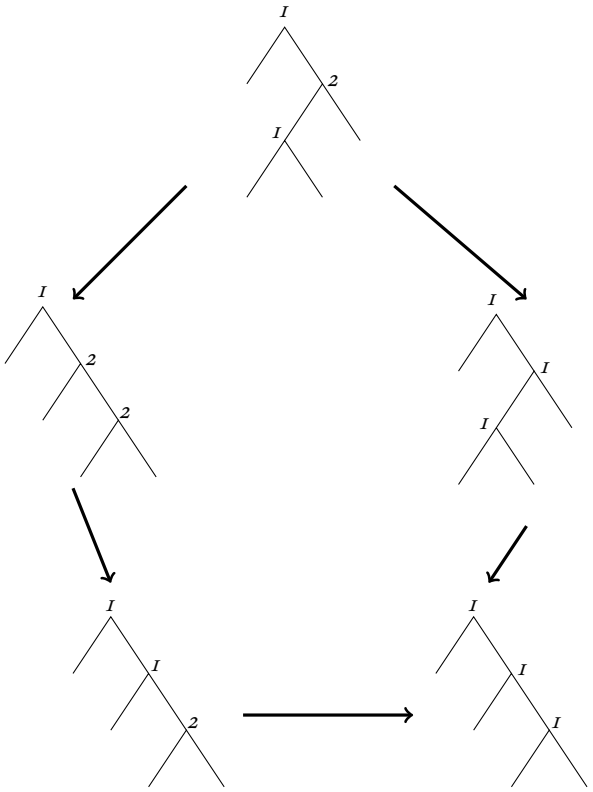
Ejemplo 12. *Consideremos los siguientes términos líderes:*



Al componerlos, obtenemos el monomio crítico:



Aplicando el método de reescritura y usando las relaciones que definen al operad Dias vemos que este monomio crítico es confluente como se puede ver en el siguiente grafo:



Este mismo proceso debe hacerse con todos los monomios críticos; que para el caso *Dias* son en total 14. En la Sección 4 se discute como obtener todos los monomios críticos y como verificar su confluencia computacionalmente.

Teorema 8. *El operad no simétrico Dias es conjuntista y cancelativo.*

Demostración. La demostración se puede consultar en [16]. □

Ejemplo 13. *Por ser Dias un operad cancelativo y en virtud de la observación de la definición 2, construyendo el poset asociado es posible verificar su dimensión y la de Dend.*

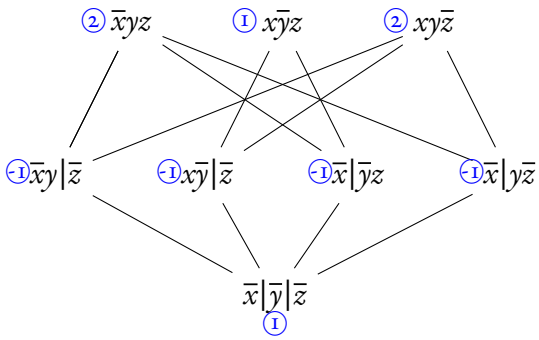


Figura 3.1: Poset de $\mathcal{D}ias(3)$

En el tope del poset hay 3 *elementos* $\bar{x}yz, x\bar{y}z, xy\bar{z}$ *los cuales son los generadores de* $\mathcal{D}ias(3)$ *esto nos confirma que* $dim(\mathcal{D}ias(3)) = 3$.
El cardinal de Möbius de este poset o la suma de los valores de la función de Möbius en los topes del poset es $5 = 2 + 1 + 2$ *lo que nos dice que* $dim(\mathcal{D}end(3)) = c_3 = 5$.

3.3. *Trias – Tridend*

Una triálgebra asociativa $\mathcal{A}$ es un espacio vectorial junto con tres operaciones binarias y asociativas

$$\dashv, \vdash, \perp : \mathcal{A} \otimes \mathcal{A} \to \mathcal{A},$$

tal que para todo $a, b, c \in \mathcal{A}$ se satisfacen las siguientes relaciones:

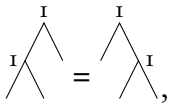
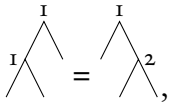
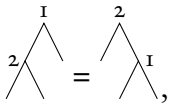
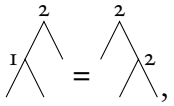
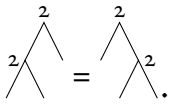
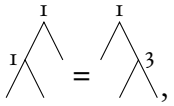
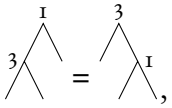
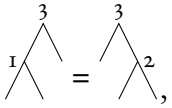
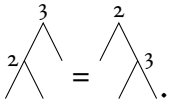
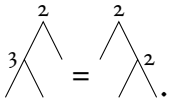
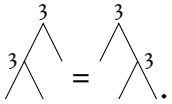
1. $(a \dashv b) \dashv c = a \dashv (b \dashv c)$
2. $(a \dashv b) \dashv c = a \dashv (b \vdash c)$
3. $(a \vdash b) \dashv c = a \vdash (b \dashv c)$
4. $(a \dashv b) \vdash c = a \vdash (b \vdash c)$
5. $(a \vdash b) \vdash c = a \vdash (b \vdash c)$
6. $(a \dashv b) \dashv c = a \dashv (b \perp c)$
7. $(a \perp b) \dashv c = a \perp (b \dashv c)$
8. $(a \dashv b) \perp c = a \perp (b \vdash c)$
9. $(a \vdash b) \perp c = a \vdash (b \perp c)$
10. $(a \perp b) \vdash c = a \vdash (b \vdash c)$
11. $(a \perp b) \perp c = a \perp (b \perp c)$

Las triálgebras han sido ampliamente estudiadas por Loday y Ronco en [10] y tienen una estrecha relación con el complejo simplicial asociado al asociaedro. De forma análoga a como sucede con las diálgebras, si $(A, \cdot)$ es un álgebra asociativa y si para todo $a, b \in A$ definimos $a \dashv b = a \perp b = a \vdash b := a \cdot b$, se tiene un funtor de la categoría de álgebras asociativas en la categoría de triálgebras:

$$As - mod \to Trias - mod.$$

También dada una triálgebra $(\mathcal{A}, \dashv, \vdash, \perp)$ en partícular se tiene que $(\mathcal{A}, \dashv, \vdash)$ tiene estructura de diálgebra y que $(\mathcal{A}, \vdash), (\mathcal{A}, \dashv), (\mathcal{A}, \perp)$ tienen estructura de álgebras asociativas. Esto nos muestra la

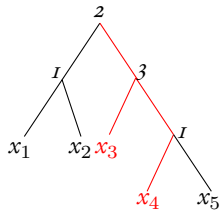
existencia de un funtor de olvido de la categoría de triálgebras en la categoría de diálgebras y tres funtores de la categoría de triálgebras en la categoría de álgebras asociativas.
Denotaremos $\perp = \circ_3$ como la operación 3. Definimos $\mathcal{T}rias$ como el operad no simétrico generado por los árboles binarios  , y sujeto a las relaciones que definen una triálgebra. Estas 11 relaciones podemos expresarlas en términos de árboles binarios como sigue:

1.  ,
2.  ,
3.  ,
4.  ,
5.  .
6.  ,
7.  ,
8.  ,
9.  .
10.  .
11.  .

3.3.1. Interpretación alternativa de las relaciones en $\mathcal{T}rias$

Extendiendo la interpretación alternativa que se dio a las relaciones en $\mathcal{D}ias$ en donde la operación $\circ_1$ indica que la energía baja desde la raíz del árbol hacia el primer hijo y la operación $\circ_2$ hacia el segundo hijo. Para el caso de $\mathcal{T}rias$ diremos que la operación $\circ_3$ transporta energía desde la raíz del árbol hacia los dos hijos (izquierda y derecha) de manera simultanea. Como se dijo antes, dado un monomio $x_1x_2\ldots x_k$, una variable punteada $\overline{x_i}$ será aquella en donde finaliza la energía. Así un monomio en $\mathcal{T}rias$ se puede identificar con un monomio no conmutativo con al menos una de sus variables marcadas. Por esta razón los 18 árboles cuadráticos que generan $\mathcal{T}rias$ dan lugar a 7 clases de equivalencia y por tanto se puede afirmar que $\mathcal{T}rias(3) = \langle \bar{x}yz, x\bar{y}z, xy\bar{z}, \overline{xy}z, \overline{x\bar{y}}z, x\overline{y\bar{z}}, \overline{xyz}\rangle$ y $dim(\mathcal{T}rias(3)) = 7$, en general se tiene que $dim(\mathcal{T}rias(n)) = 2^n - 1$.

Ejemplo 14. *Con la regla de la transferencia de energía, el siguiente árbol en $\mathcal{T}rias$*



se identifica con el monomio $x_1x_2\overline{x_3x_4}x_5$.

El siguiente teorema muestra como hacer cálculos en una triálgebra A.

Teorema 9. *Sea* $\mathcal{A}$ *una triálgebra y sean* $x_1, \ldots x_n \in \mathcal{A}$, *entonces:*

1. *Toda parentización completa de*

$$(x_1 \circ_2 \cdots \circ_2 x_i) \circ_2 (x_{i+1} \circ_1 \cdots \circ_1 x_j) \circ_3 (x_{j+1} \circ_1 \cdots \circ_1 x_k) \circ_3 \cdots \circ_3 (x_l \circ_1 \cdots \circ_1 x_n)$$

   da como resultado el mismo elemento en $\mathcal{A}$ *que denotamos*

$$x_1 \ldots x_i\overline{x_{i+1}} \ldots x_j\overline{x_{j+1}} \ldots x_k \ldots \overline{x_l} \ldots x_n.$$

2. *En* $\mathcal{A}$ *se tienen las siguientes fórmulas:*

   - $(x_1 \ldots \overline{x_i} \ldots x_k) \circ_1 (x_{k+1} \ldots \overline{x_j} \ldots x_n) = x_1 \ldots \overline{x_i} \ldots x_kx_{k+1} \ldots x_j \ldots x_n.$
   - $(x_1 \ldots \overline{x_i} \ldots x_k) \circ_2 (x_{k+1} \ldots \overline{x_j} \ldots x_n) = x_1 \ldots x_i \ldots x_kx_{k+1} \ldots \overline{x_j} \ldots x_n.$
   - $(x_1 \ldots \overline{x_i} \ldots x_k) \circ_3 (x_{k+1} \ldots \overline{x_j} \ldots x_n) = x_1 \ldots \overline{x_i} \ldots x_kx_{k+1} \ldots \overline{x_j} \ldots x_n.$

La demostración de este teorema se puede consultar en [10].

Ejemplo 15. *Sean* $x_1, x_2, x_3, x_4, x_5, x_6, x_7$ *elementos en una triálgebra* $\mathcal{A}$, *usando el teorema anterior se tiene:*

$$((x_1 \circ_2 x_2) \circ_1 (x_3 \circ_3 x_4)) \circ_2 ((x_5 \circ_1 x_6) \circ_3 x_7) = (x_1\overline{x_2}x_3x_4) \circ_2 (\overline{x_5}x_6\overline{x_7}) = x_1x_2x_3x_4\overline{x_5}x_6\overline{x_7}.$$

3.3.2. *Tridend*

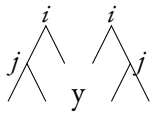
Una triálgebra dendriforme o un álgebra tridendriforme sobre $\mathbf{k}$ es un $\mathbf{k}$-espacio vectorial T dotado con 3 operaciones binarias

$$\prec, \succ, \cdot : T \otimes T \to T,$$

tal que para $a, b, c \in T$ satisface las siguientes relaciones:

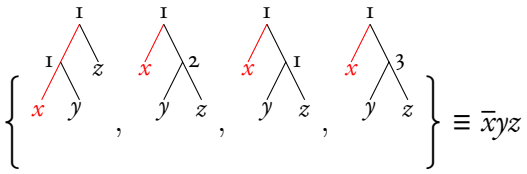
1. $(a \prec b) \prec c = a \prec (b * c)$
2. $(a \succ b) \prec c = a \succ (b \prec c)$
3. $(a * b) \succ c = a \succ (b \succ c)$
4. $(a \succ b) \cdot c = a \succ (b \cdot c)$
5. $(a \prec b) \cdot c = a \cdot (b \succ c)$
6. $(a \cdot b) \prec c = a \cdot (b \prec c)$
7. $(a \cdot b) \cdot c = a \cdot (b \cdot c)$,

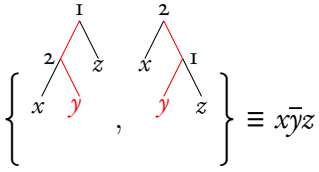
donde $a * b = a \prec b + a \succ b + a \cdot b$.
Notemos que las relaciones que definen las triálgebras dendriformes tienen las variables en el mismo orden, esto indica que el operad asociado a las triálgebras dendriformes $\mathcal{T}ridend$ es un operad no simétrico.
Recordemos que el operad $\mathcal{T}rias$ está generado por los 18 árboles binarios de la forma

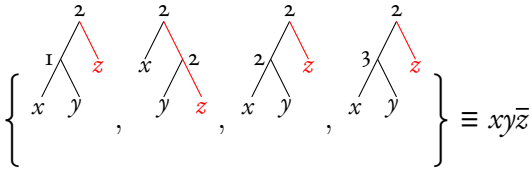


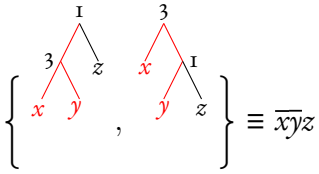
, con $i,j \in [3]$,

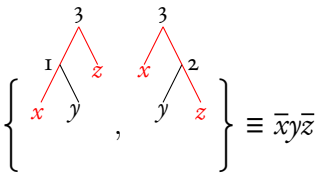
sujeto a las 11 relaciones que lo definen. Buscaremos ahora el complemento ortogonal del espacio de relaciones que definen $\mathcal{T}rias$ para hallar el dual de Koszul de $\mathcal{T}rias^!$.
De forma explicita las 7 clases de equivalencia en $\mathcal{T}rias(3)$ provenientes de los 18 árboles binarios y las 11 relaciones son:

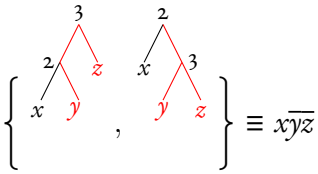
1.  $\equiv \bar{x}yz$

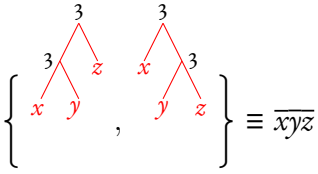
2.  $\equiv x\bar{y}z$

3.  $\equiv xy\bar{z}$

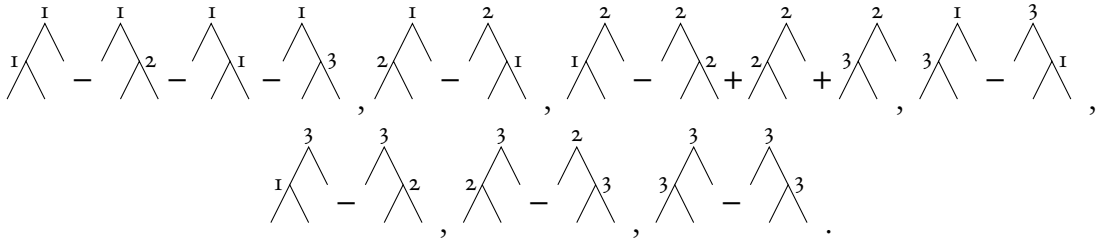
4.  $\equiv \overline{xy}z$

5.  $\equiv \bar{x}y\bar{z}$

6.  $\equiv x\overline{yz}$

7.  $\equiv \overline{xyz}$

A partir de cada una de estas clases de equivalencia y teniendo en cuenta que los árboles derechos tienen signo negativo por las reglas de signos de Koszul se tienen las siguientes relaciones que definen al operad $\mathcal{T}rias^!$:



Después de hacer la identificación

$\equiv x \cdot y$

y recordando que $x * y = x \prec y + x \succ y + x \cdot y$ vemos que las relaciones que definen a $\mathcal{T}rias^!$ coinciden con las de $\mathcal{T}ridend$, por tanto tenemos que $\mathcal{T}rias^! \cong \mathcal{T}ridend$.
El hecho de que $\mathcal{T}ridend(3)$ esté generado por 7 relaciones confirma que $dim(\mathcal{T}rias(3)) = 7$, mientras que al estar $\mathcal{T}rias(3)$ generado por 11 relaciones tenemos que $dim(\mathcal{T}ridend(3)) = 11$.

3.3.3. Series generatrices de $\mathcal{T}rias$ y $\mathcal{T}ridend$

Sabemos que $dim(\mathcal{T}rias(n)) = 2^n - 1$, de allí que la serie generatriz del operad no simétrico $\mathcal{T}rias$ esté dada por:

$$\begin{aligned} F_{\mathcal{T}rias}(x) &= \sum_{n\geq 1}(2^n - 1)x^n \\ &= \sum_{n\geq 1}(2x)^n - \sum_{n\geq 1}x^n \\ &= \sum_{n\geq 0}(2x)^{n+1} - \sum_{n\geq 0}x^{n+1} \\ &= (2x)\sum_{n\geq 0}(2x)^n - x\sum_{n\geq 0}x^n \\ &= \frac{2x}{1-2x} - \frac{x}{1-x} \\ &= \frac{x}{(1-x)(1-2x)}. \end{aligned}$$

Utilizaremos este resultado para hallar la serie generatriz del operad no simétrico $\mathcal{T}ridend$.
Como el dual de Koszul de $\mathcal{T}rias$ es $\mathcal{T}ridend$ su serie generatriz $F_{\mathcal{T}ridend}(x)$ debe satisfacer la relación

$$F_{\mathcal{T}rias}(-F_{\mathcal{T}ridend}(-x)) = x \iff F_{\mathcal{T}rias}(-F_{\mathcal{T}ridend}(x)) = -x$$

Por facilidad en la notación, sea $g = F_{\mathcal{T}ridend}(x)$, por lo tanto:

$$F_{\mathcal{T}rias(-g)} = -x \iff \frac{-g}{(1+g)(1+2g)} = -x \iff g = x(1+g)(1+2g) \iff g = x\Phi(g)$$

Con $\Phi(g) = (1+g)(1+2g)$.
Usando el teorema de inversión de Lagrange, si $g(x) = \sum_{n\geq 1} g_n x^n$, tenemos que:

$$
\begin{aligned}
g_n &= \frac{1}{n}[x^{n-1}]\Phi(x)^n \\
&= \frac{1}{n}[x^{n-1}](1+x)^n(1+2x)^n \\
&= \frac{1}{n}[x^{n-1}]\sum_{k=0}^{n}\binom{n}{k}x^k\sum_{j=0}^{n}\binom{n}{j}(2x)^j \\
&= \frac{1}{n}[x^{n-1}]\sum_{k\geq 0}\sum_{j\geq 0}\binom{n}{k}\binom{n}{j}2^j x^{k+j}, \textit{ haciendo k+j=r} \\
&= \frac{1}{n}[x^{n-1}]\sum_{r\geq 0}\left[\sum_{k=0}^{r}\binom{n}{k}\binom{n}{r-k}2^{r-k}\right]x^r, \textit{ cuando r=n-1} \\
&= \frac{1}{n}\sum_{k=0}^{n-1}2^{n-k-1}\binom{n}{k}\binom{n}{n-(k+1)} \\
&= \sum_{k=0}^{n-1}\frac{1}{n}2^{n-k-1}\binom{n}{k}\binom{n}{k+1}.
\end{aligned}
$$

Luego

$$
F_{\mathcal{T}ridend}(x) = \sum_{n\geq 1}\left[\sum_{k=0}^{n-1}\frac{1}{n}2^{n-k-1}\binom{n}{k}\binom{n}{k+1}\right]x^n.
$$

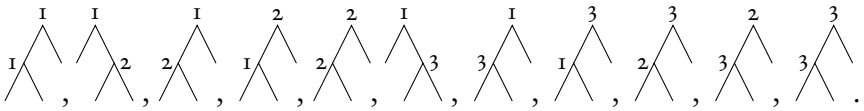
De donde se concluye que

$$
dim(\mathcal{T}ridend(n)) = \sum_{k=0}^{n-1}\frac{1}{n}2^{n-k-1}\binom{n}{k}\binom{n}{k+1} = C_n.
$$

El número C_n es conocido como el n-ésimo número super Catalan.

3.3.4. Órdenes monomiales en $\mathcal{T}rias$

Comparando las sucesiones de caminos correspondientes a las once relaciones que definen al operad $\mathcal{T}rias$ se obtienen los monomios o árboles líderes, tomando el orden $\dashv < \vdash < \perp$, esto equivale a tener $1 < 2 < 3$. Es importante resaltar que al estar $\mathcal{T}rias$ definido por mas relaciones que $\mathcal{D}ias$, el proceso de comparación es mas largo, razón por la cuál se vuelve conveniente hacerlo de forma computacional con ayuda de código en Python el cual sera tratado en detalle en la siguiente sección.
Los 11 términos líderes son:



Después de realizar todas las posibles composiciones entre estos términos lideres obtenemos 45 monomios críticos. Chequeando computacionalmente la confluencia de cada uno de los diamantes correspondientes a estos monomios críticos y usando el Teorema 3, llegamos a una demostración alternativa del siguiente teorema de Loday y Ronco en [10], el cuál fué demostrado directamente mostrando que el complejo de Koszul es acíclico.

Teorema 10 (Loday-Ronco [10]). *Los operads no simétricos $\mathcal{T}rias$ y $\mathcal{T}ridend$ son de Koszul.*

Teorema 11. *El operad no simétrico $\mathcal{T}rias$ es conjuntista y cancelativo.*

Demostración. La demostración se puede consultar en [16]. □

4
Código

En este capítulo describiremos el código elaborado para verificar la propiedad de Koszul en los operads estudiados en capítulos anteriores mediante el método de reescritura descrito en la sección 2.4.

4.1. Sobre SageMath

SageMath o Sage es un sistema algebraico computacional. Es un software libre escrito en lenguaje Python lo cual lo hace una muy buena alternativa para otros software matemáticos como Maple, Mathematica o Matlab. El nombre SAGE proviene de sus siglas en ingles *System for Algebra and Geometry Experimentation* y fue desarrollado por el matemático estadounidense William Stein.
Su primera versión fue publicada en el año 2005 y el objetivo principal era eliminar la dependencia del software matemático cerrado. El hecho de que SAGE utilice lenguaje Python lo hace muy popular en el mundo científico pues se basa en programación orientada a objetos.
Originalmente Sage como su nombre lo indica fue pensado para trabajar en temas de experimentación en geometría, pero el hecho de ser libre lo ha popularizado tanto entre los matemáticos que en la actualidad cuenta con herramientas propias de diversas áreas de la matemática como álgebra, álgebra lineal, teoría de grafos, teoría de grupos, combinatoria, cálculo, geometría algebraica, teoría de números entre otras; incluso también es compatible con el lenguaje de programación R enfocado al análisis estadístico.
Particularmente en el desarrollo de este documento utilizamos la versión de SageMath 9.0 [15] y paquetes especializados en combinatoria, teoría de grafos y álgebra como *Ordered Rooted Trees* desarrollado por Florent Hivert y Frédéric Chapoton en el 2010.

4.2. Generando árboles planares

Para crear árboles con raíz y etiquetas en SageMath vamos a usar la clase `LabelledOrderedTree`. Usaremos el comando

```
1 LOT = LabelledOrderedTree
```

para referirnos al constructor de la clase LabelledOrderedTree de ahora en adelante como `LOT`. Esto es útil por simplicidad y para hacer mas ligera la notación.

Ejemplo 16. *Las líneas de código en el Código 4.1 generan el árbol con raíz y etiquetas que se ilustra en la Figura 4.1. La función* `ascii_art` *nos ayuda a mostrar de forma gráfica el árbol que se creó.*

Código 4.1: Ejemplo de árbol con raíz y etiqueta

```
LOT = LabelledOrderedTree
ejemplo=(LOT([[],[]],label=1))
ascii_art(ejemplo)
```

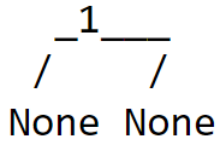


Figura 4.1: Árbol básico con raíz y etiqueta creado en SageMath.

Como los operads estudiados en este documento son no simétricos, no es necesario especificar las etiquetas de las hojas. Por esta razón aparece "None" en cada una de las hojas de estos árboles.
Usando la clase `LOT` de forma recursiva se pueden crear árboles con raíz y etiquetas con mas hijos y variantes. Por ejemplo la linea

```
ascii_art(LOT([LOT([[],[],[]],label=5),[]],label=3))
```

genera el árbol en la Figura 4.2.

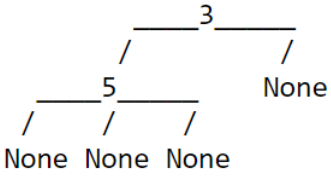


Figura 4.2: Árbol con raíz y etiqueta creado en SageMath

4.3. Concatenación y composición de árboles

La primer función que vamos a definir es la que permite concatenar dos árboles. Cuando hablamos de "concatenar" un árbol `tree1` con un árbol `tree2` estamos haciendo referencia al proceso de insertar el árbol `tree2` en cada una de las hojas del árbol `tree1`.
La función `concatenate_ordered_trees(LOT1, LOT2)` implementada en el Código 4.2 genera una lista de todos los posibles árboles que resultan de concatenar el árbol `LOT1` con el árbol `LOT2`.

Código 4.2: Función concatenate_ordered_trees

```
def concatenate_ordered_trees(LOT1, LOT2):
    list_of_concatenations = []
    for i, subtree in enumerate(LOT1):
        if subtree == LOT([]):
           list_of_concatenations.append(LOT([LOT1[j] if j != i else
                                              LOT2 for j in range(len(LOT1))], label=LOT1.label())
                                                  )
        else:
            for tree in concatenate_ordered_trees(subtree, LOT2):
                list_of_concatenations.append(LOT([LOT1[j] if j != i else
                                                   tree for j
                                                   in range(len(LOT1))], label=LOT1.label()))
    return list_of_concatenations
```

Ejemplo 17. *Las líneas de código*

```
A=LOT([LOT([[],[]],label=1),[]],label=2)
B=LOT([[],LOT([[],[]],label=5)],label=1)
ascii_art(concatenate_ordered_trees(A,B))
```

generan la lista de árboles que resultan de la concatenación de los árboles A y B de la Figura 4.3. Estos se muestran en la Figura 4.4.

(a) árbol *A* **(b)** árbol *B*

Figura 4.3: Árboles de referencia

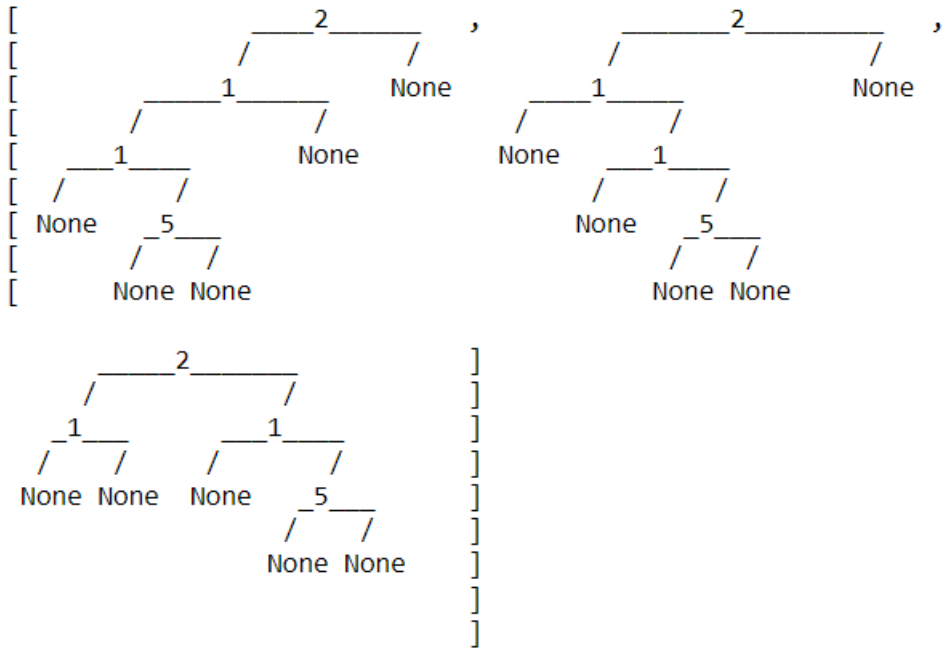


Figura 4.4: Lista de concatenación de árboles

La función `get_signature_at_root` evaluada en un árbol *tree* genera una pareja en donde la primer coordenada es una lista que indica cuales de los hijos de la raíz no son hojas y la segunda coordenada indica cual es la etiqueta de la raíz. Se usa el valor 0 para indicar que el hijo es una hoja y 1 en caso contrario. La función se define en el Código 4.3.

Código 4.3: Función get_signature_at_root

```
def get_signature_at_root(tree):
    return ([0 if item == LOT([]) else 1 for item in tree], tree.label())
```

Esta función es importante pues es utilizada para definir la función `compose_labelled_ordered_trees` del Código 4.4. `compose_labelled_ordered_trees(tree1, tree2)` genera una lista formada por los árboles que resultan de *componer* el árbol `tree1` con el árbol `tree2` y genera una lista vacía en caso de que los árboles dados no se puedan componer.

Código 4.4: Función compose_labelled_ordered_trees

```
def compose_labelled_ordered_trees(tree1, tree2):
    list_of_composed_trees = []
    (signature1, label1) = get_signature_at_root(tree1)
    (signature2, label2) = get_signature_at_root(tree2)
    if label1 == label2 and len(signature1) == len(signature2):
        if 1 not in [signature1[i]*signature2[i] for i in range(len(signature1))]:
            list_of_composed_trees.append(LOT([tree1[i] if signature1[i] == 1 \
                                                else tree2[i] for i in range(len(tree1))],\
                                               label=label1))
    for j in range(len(tree1)):
        list_of_composed_trees = list_of_composed_trees \
        + [LOT([tree1[i] if i != j else tree for i in range(len(tree1))], label=label1)\
           for tree in compose_labelled_ordered_trees(tree1[j], tree2)]
    return list_of_composed_trees
```

Es necesario hacer enfasis en que se entiende por *componer*. Para que dos árboles `tree1` y `tree2` se puedan componer, la etiqueta de un vértice v de `tree1` debe coincidir con la etiqueta de la raíz r de `tree2`, v y r

deben tener la misma aridad y deben tener hijos en hojas complementarias. Así la composición de `tree1` con `tree2` es una lista con todos aquellos árboles en los cuales se reemplaza el árbol `tree2` por el vértice o los vértices de `tree1` que coinciden con la raíz de `tree2` de esta manera.

Ejemplo 18. *Consideremos los árboles* `A` *y* `B` *que aparecen en la Figura 4.3. Si ejecutamos el comando* `get_signature_at_root(A)` *se obtiene la pareja* $([1,0],2)$. *Esta pareja en su primer coordenada tiene la lista* $[1,0]$. *El número* 1 *indica que el hijo izquierdo del árbol* `A` *no es una hoja, mientras que como el hijo derecho si es una hoja se usa el número* 0. *En la segunda coordenada de la lista se generó el número* 2, *esto pues la raíz del árbol tiene al* 2 *como etiqueta.*
Por otro lado, el comando `ascii_art(compose_labelled_ordered_trees(A,B))` *genera la lista de árboles que resultan al componer el árbol* `A` *con el árbol* `B`. *En este caso esta lista tiene un solo árbol que se muestra en la Figura 4.5.*

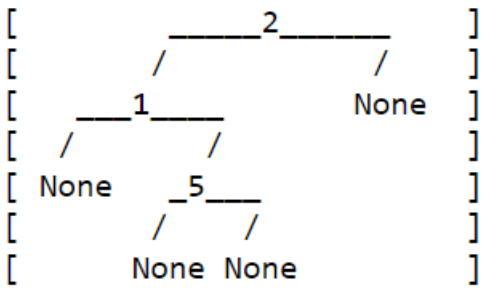


Figura 4.5: Ejemplo de composición de árboles

4.4. Generando las reglas de reescritura

Para aplicar el método de reescritura presentado en la sección 2.4 a un operad es necesario conocer todos sus árboles. Para ello en el Código 4.5 definimos la función `generate_all_quadratic_trees` que dada una lista de etiquetas y una aridad fija genera todos los posibles árboles cuadráticos bajo tales condiciones.

Código 4.5: Función generate_all_quadratic_trees

```
def generate_all_quadratic_trees(labels, arity):
    #todos los arboles
    general_all_trees = []
    for i in labels:
        for j in labels:
            for p in range(arity):
                general_all_trees.append(LOT([LOT([]) if k != p else
                                              LOT([[] for m in range(arity)], label=j)for
                                              k in range(arity)], label=i))
    return general_all_trees
```

En el caso de los operads no simétricos que se han estudiado en los capítulos anteriores y el operad no simétrico *Tetraas* que se estudiará en el próximo capítulo todos son operads binarios o de aridad 2.

Ejemplo 19. *Usando la función* `generate_all_quadratic_trees`, *podemos hallar todos los posibles árboles binarios de los operads As, Dias y*

- `generate_all_quadratic_trees([1],2)` *genera los* 2 *únicos árboles binarios del operad As.*

- `generate_all_quadratic_trees([1,2],2)` *genera los* 8 *árboles binarios del operad* $\mathcal{D}ias$.
- `generate_all_quadratic_trees([1,2,3],2)` *genera los* 18 *árboles binarios del operad* $\mathcal{T}rias$.

  En general para un operad no simétrico con n *etiquetas esta función genera todos los* $2n^2$ *posibles árboles. Por lo que*
- `generate_all_quadratic_trees([0,1,2,3],2)` *genera los* $2*5^2 = 50$ *árboles binarios del operad* $\mathcal{T}etraas$. *Este operad se definirá en el Capítulo 5.*

Como todos los operads que estudiamos en este documento son binarios o de aridad 2, en el Código 4.6 definimos una función basada en `generate_all_quadratic_trees` que genera todos los árboles binarios dada una lista de etiquetas.

Código 4.6: Función generate_all_binary_quadratic_trees

```
def generate_all_binary_quadratic_trees(labels):
  return generate_all_quadratic_trees(labels, 2)
```

Así por ejemplo, basta con calcular `generate_all_binary_quadratic_trees([1,2])` para obtener todos los árboles binarios que son generadores del operad $\mathcal{D}ias$.
Es importante saber en cada uno de los operads estudiados como se distribuye la energía según la interpretación presentada en la sección 3.2.1. Una vez se han generado todos los árboles binarios en cada uno de estos operads se agruparán en clases de equivalencia de acuerdo a la distribución de la energía para después compararlos entre sí y así obtener las reglas de reescritura, los monomios lideres y a partir de éstos los monomios críticos.
La función `is_light_on` definida en el Código 4.7 indica como se distribuye la energía para los árboles dentro de un operad determinado. Esta función necesita como argumentos una lista de direcciones, las etiquetas u operaciones del operad y un número de hijos.

Código 4.7: Función is_light_on

```
def is_light_on(address, labels, number_of_children, light_on=True):
    if light_on:
        on_pattern = [int(item) for item in bin(labels[0])[2:]]
        on_pattern.reverse()
        on_pattern = (on_pattern + (number_of_children[0]-len(on_pattern))*[0])
        if  on_pattern[address[0]] == 1:
            if len(labels) == 1:
                return True
            else:
                return is_light_on(address[1:], labels[1:], number_of_children[1:])
    return False
```

Recordemos que según los estudiado en capítulos anteriores para operads no simétricos, la etiqueta (operación) 1 indíca que la energía se distribuye hacia el hijo izquierdo del árbol, es decir en la dirección $[0]$. La etiqueta (operación) 2 indica que la energía se distribuye hacia el hijo derecho, es decir en la dirección $[1]$. Por su parte la etiqueta (operación) 3 indica que la energía se distribuye para el hijo izquierdo y el derecho de manera simultanea.

Ejemplo 20. *En el operad no simétrico* $\mathcal{D}ias$ *hay dos operaciones* 1 *y* 2. *Ejecutando el comando*

`is_light_on([1],[2],[2],is_light_on=True)`

se obtiene como resultado `True`, *esto pues como se indicó antes la operación* 2 *distribuye la energía hacia el hijo derecho, es decir en la dirección* $[1]$.

Por su parte, si se ejecuta el comando

```
is_light_on([0],[2],[2],is_light_on=True)
```

el resultado que se obtiene es `False` *pues la operación* 2 *no envía energía en la dirección* $[0]$.

La función `leaf_addresses` definida en el Código 4.8 genera una lista con la dirección de las hojas de un árbol dado. La función inicia en la raíz y va bajando hasta llegar a las hojas; cada uno de esos movimientos para llegar a la hoja aporta un número en la dirección. Si el movimiento se hace hacia el lado izquierdo se genera el número 0 en la dirección, mientras que si el movimiento se realiza hacia el lado derecho el número que se genera en la dirección es el 1.

Código 4.8: Función leaf_addresses

```
def leaf_addresses(tree):
    addresses = []
    if tree == LOT([]):
        addresses.append([])
    else:
        for i, subtree in enumerate(tree):
            new_addresses = [[i]+ addr for addr in leaf_addresses(subtree)]
            addresses = addresses + new_addresses
    return addresses
```

Ejemplo 21. *Para el árbol* `A` *de la Figura 4.3a al ejecutar el comando* `leaf_addresses(A)` *se obtiene como resultado la lista* $[[0,0],[0,1],[1]]$ *que indica las direcciones en las cuales se encuentran las hojas de éste árbol.*

Mostraremos ahora como a partir de un árbol determinado se puede obtener su sucesión de caminos. Este proceso será fundamental pués después dadas dos sucesiones de caminos las compararemos con el orden introducido en la sección 2.4.2.
Dado un árbol `tree` la función definida en el código 4.9 genera la sucesión de caminos asociada a tal árbol.

Código 4.9: Función tree_to_path_sequence

```
def tree_to_path_sequence(tree):
    return ([[tree[address[:i]].label() for i in range(len(address))]
             for address in leaf_addresses(tree)])
```

Ejemplo 22. *Al ejecutar el comando* `tree_to_path_sequence(A)` *se obtiene la sucesión de caminos*

$$[[2,1],[2,1],[2]]$$

asociada al árbol `A` *de la Figura 4.3a.*
Si ejecutamos el comando `tree_to_path_sequence(B)` *la sucesión de caminos que se obtiene es*

$$[[1],[1,5],[1,5]]$$

que está asociada al árbol B que se muestra en la Figura 4.3b.

Dados dos árboles planares estos se pueden comparar mediante sus sucesiones de caminos tomando el orden definido en la sección 2.4.2. La función `compare_path_sequences` evaluada en dos sucesiones de camino `s1` y `s2` es una función por partes dada por:

$$\texttt{compare\_path\_sequences(s1,s2)} = \begin{cases} 0, si \ \texttt{s1 = s2} \\ -1, si \ \texttt{s1 < s2} \\ 1, si \ \texttt{s1 > s2} \end{cases}.$$

Esta función es necesaria para definir la función `compare_planar_trees` que evaluada en dos árboles `tree1` y `tree2` los compara utilizando sus sucesiones de caminos asociadas.
Las funciones `compare_path_sequences` y `compare_planar_trees` están implementadas en el Código 4.10.

Código 4.10: Función compare_path_sequences y Función compare_planar_trees

```
def compare_path_sequences(s1, s2):
    if s1 == s2:
        return 0
    elif len(s1) < len(s2):
        return -1
    elif len(s1) > len(s2):
        return 1
    else:
        for i, item in enumerate(s1):
            if len(s1[i]) < len(s2[i]):
                return -1
            elif len(s1[i]) > len(s2[i]):
                return 1
            else:
                if s1[i] < s2[i]:
                    return -1
                elif s1[i] > s2[i]:
                    return 1
def compare_planar_trees(tree1, tree2):
    return compare_path_sequences(tree_to_path_sequence(tree1), tree_to_path_sequence(tree2))
```

Ejemplo 23. *Para los árboles* `A` *y* `B` *que aparecen en la Figura 4.3 al compararlos se tiene que* `A > B`. *Por esta razón luego de ejecutar el comando* `compare_planar_trees(A,B)` *se obtiene como resultado un* 1.

La función `light_signature` del Código 4.11 evaluada en un árbol binario de alguno de los operads no simétricos estudiados en este documento indica como se distribuye la energía en tal árbol, esto de acuerdo a lo introducido en la sección 3.2.1 . Si la energía llega a una hoja, esta función asigna un " True" (Verdadero) de lo contrario asigna un " False" (Falso).

Código 4.11: Función light_signature

```
def light_signature(tree):
    light_pattern = []
    for i in range(len(leaf_addresses(tree))):
        address = leaf_addresses(tree)[i]
        labels = tree_to_path_sequence(tree)[i]
        number_of_children = [len(tree[leaf_addresses(tree)[i][:j]]) for
                              j in range(len(leaf_addresses(tree)[i]))]
        light_pattern.append(is_light_on(address, labels, number_of_children))
    return tuple(light_pattern)
```

Ejemplo 24. *En el árbol* `A` *de la Figura 4.3a la energía solamente llega a la hoja de la dirección* $[1]$ *por lo que después de ejecutar el comando* `light_signature(A)` *se obtiene como resultado la tupla*

$$(False, False, True).$$

La función definida en el Código 4.12 dada una lista de árboles genera un diccionario que los agrupa en clases de equivalencia de acuerdo a la forma en que se distribuye la energía. Adicionalmente cada clase la entrega como una lista ordenada usando el criterio de orden deseado. En particular, si se tiene que `ordering_function=None` entonces los árboles no estarán ordenados.

Código 4.12: Función get_light_equivalence_classes

```
def get_light_equivalence_classes(list_of_trees, ordering_function=None):
  dict_trees = {}
  for tree in list_of_trees:
      if light_signature(tree) in dict_trees.keys():
          dict_trees[light_signature(tree)].append(tree)
      else:
          dict_trees[light_signature(tree)] = [tree]
  if ordering_function is not None:
    for key in dict_trees:
        dict_trees[key] = ordering_function(dict_trees[key], compare_planar_trees)
  return dict_trees
```

Ejemplo 25. *En la sección 3.2.1 para el operad no simétrico Dias todos sus árboles binarios se agruparon en tres clases de equivalencia de acuerdo a como se distribuye la energía en tales árboles.*
Estas clases de equivalencia se pueden obtener después de haber generado todos los árboles binarios del operad no simétrico Dias evaluando la función `generate_all_binary_quadratic_trees` *definida en el Código 4.6 con las etiquetas* $[1, 2]$ *como argumento y luego ejecutando la función* `get_light_equivalence_classes` *definida en el Código 4.12 como se muestra en las lineas de código*

```
dias_trees=generate_all_binary_quadratic_trees([1,2])
dias_classes=get_light_equivalence_classes(dias_trees,ordering_function=quick_sort)
```

A la lista que contiene a todos los árboles binarios de Dias la llamaremos de acá en adelante `dias_trees` *mientras que al diccionario con las clases de equivalencia del operad no simétrico Dias lo llamaremos* `dias_classes`.
Al imprimir los elementos del diccionario `dias_classes` *obtenemos las* 3 *clases de equivalencia del operad no simétrico Dias:* $(True, False, False)$, $(False, True, False)$, $(False, False, True)$. *Cada una de esas clases contiene una lista con los árboles que pertenecen a la clase.*

Dentro de los argumentos de la función `get_light_equivalence_classes` está la posibilidad de usar `ordering_function=quick_sort`. En este caso estamos haciendo referencia al algoritmo de ordenación *QuickSort* que implementamos en el Código 4.13. Este algoritmo es en la actualidad uno de los mas rápidos y eficientes metodos de ordenamiento existentes.

Código 4.13: Algoritmo *QuickSort*

```
def quick_sort(list_of_items, compare_function):
    if len(list_of_items) == 1 or len(list_of_items) == 0:
        return list_of_items
    pivot = len(list_of_items)//2
    higher_list = [item for item in list_of_items
                   if compare_function(list_of_items[pivot], item) == -1]
    lower_list = [item for item in list_of_items
                  if compare_function(list_of_items[pivot], item) == 1]
    return (quick_sort(lower_list, compare_function)
            + [list_of_items[pivot]]
            + quick_sort(higher_list, compare_function))
```

Teniendo la lista ordenada de una clase de equivalencia, construiremos un conjunto de reglas emparejando cada uno de los árboles de la clase con el menor de todos ellos. La función definida en el Código 4.14 genera una lista con estas relaciones.

Código 4.14: Función get_list_of_rules_using_equivalence_classes

```
def get_list_of_rules_using_equivalence_classes(dict_of_classes):
    rules = []
    for key in dict_of_classes:
      if len(dict_of_classes[key]) > 1:
        for i in range(1, len(dict_of_classes[key])):
          rules.append((dict_of_classes[key][i], dict_of_classes[key][0]))
    return rules
```

Ejemplo 26. *Para el caso del operad no simétrico Dias como en el ejemplo 25 ya generamos el diccionario de clases de equivalencia* `dias_classes`*. Aplicando la función* `get_list_of_rules_using_equivalence_classes` *definida en el Código 4.14 obtenemos la lista de reglas de reescritura de este operad que llamaremos* `dias_rules` *como se muestra en la linea de código*

```
dias_rules=get_list_of_rules_using_equivalence_classes(dias_classes)
```

Todos los elementos de esta lista son pares de árboles de la forma (*monomio líder*, *reemplazo*)*. Imprimiendo los elementos de la lista* `dias_rules` *obtenemos todas las relaciones que generan al operad Dias como se muestra en la Figura 4.6*

4.5. Generando los monomios críticos

En el Código 4.15 se define la función `get_list_of_leaders`. Esta función forma una lista con la primer coordenada de cada una de las parejas de la lista de reglas de reescritura obtenida con la función que se definió en el Código 4.14.

Código 4.15: Función get_list_of_leaders

```
def get_list_of_leaders(list_of_rules):
    return [rule[0] for rule in list_of_rules]
```

Ejemplo 27. *Para el operad no simétrico Dias la lista de monomios líderes que llamaremos* `dias_leaders` *se obtiene a partir de la función definida en el Código 4.15 ejecutando el comando*

```
dias_leaders=get_list_of_leaders(dias_rules)
```

A partir de la lista de monomios líderes se generan los monomios críticos realizando todas las posibles composiciones entre ellos. El Código 4.16 muestra la función `get_critical_tree_monomials` que se usa para generar los monomios críticos.

Código 4.16: Función get_critical_tree_monomials

```
def get_critical_tree_monomials(list_of_leaders):
    list_of_critical_monomials = []
    for tree1 in list_of_leaders:
        for tree2 in list_of_leaders:
            list_of_critical_monomials = (list_of_critical_monomials +
                                          compose_labelled_ordered_trees(tree1, tree2))
    return list(set(list_of_critical_monomials))
```

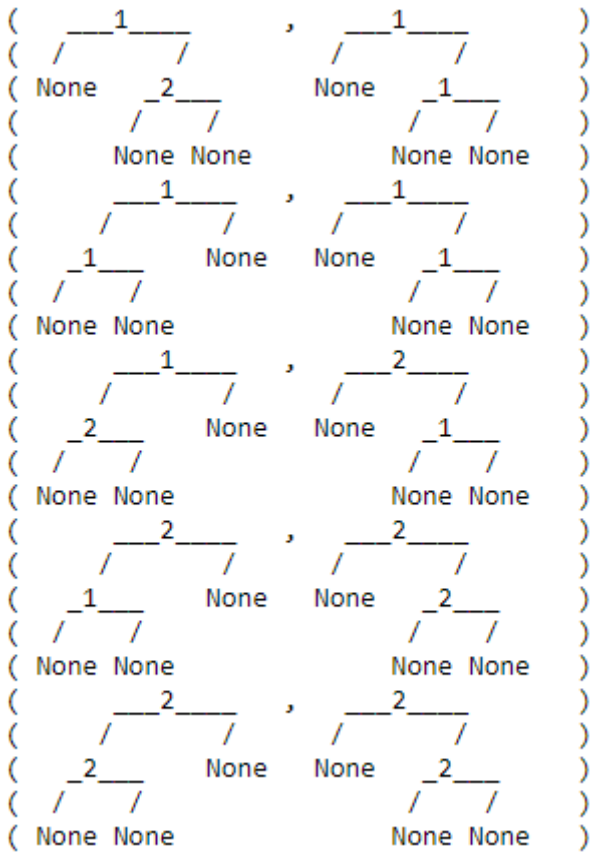


Figura 4.6: Relaciones del operad no simétrico *Dias*

Ejemplo 28. *En el ejemplo 27 se generaron todos los monomios líderes del operad Dias y se guardaron en una lista llamada* `dias_leaders`. *Con la función definida en el Código 4.16 se generan todos los monomios críticos y los guardaremos en una lista que llamaremos* `dias_criticals`. *Estos monomios críticos se obtienen ejecutando el comando*

```
dias_criticals=get_critical_tree_monomials(dias_leaders)
```

Todos los monomios críticos que se obtienen como resultado se muestran en la Figura 4.7

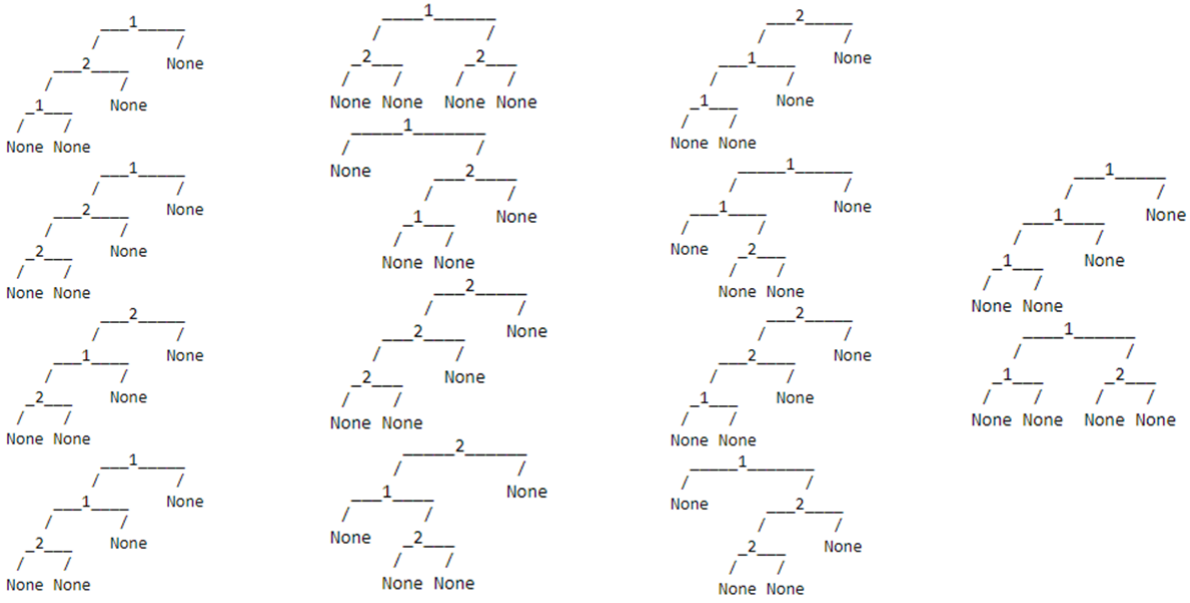


Figura 4.7: Monomios críticos de *Dias*

4.6. Aplicando las reglas de reescritura

Teniendo identificadas la reglas de rescritura debemos ver como aplicarlas en el proceso del método de reescritura. Para ello definiremos funciones que permitan aplicar tales reglas y así verificar la confluencia de los monomios críticos como se mostrará en la sección 4.8.
Para dos árboles `tree1` y `tree2` la función definida en el Código 4.17 nos indica si el árbol `tree2` es el patrón que aparece en la raíz del árbol `tree1`. La función retorna *Verdadero* (True) o *Falso* (False) según sea el caso.

Código 4.17: Función is_tree_pattern_at_root

```
def is_tree_pattern_at_root(tree1, tree2):
    pattern = True
    if tree2 == LOT([]):
        pass
    else:
        if len(get_signature_at_root(tree1)[0]) != len(get_signature_at_root(tree2)[0]) \
        or get_signature_at_root(tree1)[1] != get_signature_at_root(tree2)[1]:
```

```
            pattern = False
        else:
            for i, subtree in enumerate(tree1):
                if not is_tree_pattern_at_root(tree1[i], tree2[i]):
                    pattern = False
                    break
    return pattern
```

La función `is_tree_in_monomial` definida en el Código 4.18 indica en que dirección se encuentra el árbol `tree` dentro de `monomial`. Esta función retorna la lista de coordenadas correspondientes a la dirección. El proceso de busqueda de la función inicia en la raíz del árbol `monomial` y va bajando hasta encontrar al árbol `tree`. Si para llegar a `tree` fue necesario bajar por la dirección izquierda entonces esto se representa por un 0, mientras que si fue necesario bajar por la derecha se representa por un 1. Finalmente si el árbol `tree` no está en el árbol `monomial` la función retorna una lista vacía.

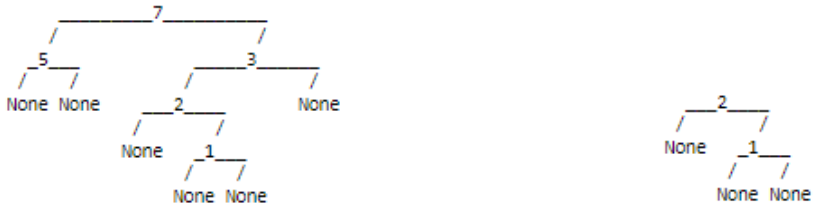
Código 4.18: Función is_tree_in_monomial

```
def is_tree_in_monomial(monomial, tree):
    list_of_directions = []
    if monomial == tree:
        list_of_directions.append([])
    else:
        if monomial != LOT([]):
            if is_tree_pattern_at_root(monomial, tree):
                list_of_directions.append([])
            for i, submonomial in enumerate(monomial):
                list_of_directions = (list_of_directions +
                                      [[i]+address for address in
                                       is_tree_in_monomial(submonomial, tree)])
    return list_of_directions
```

Ejemplo 29. *En la Figura 4.9 ilustramos un ejemplo en el que el árbol* `D` *está en el árbol* `C`*. Esto para los árboles* `C` *y* `D` *que aparecen en la Figura 4.8. Para encontrar a* `D` *dentro de* `C` *fue necesario bajar desde la raíz de* `C` *primero por la dirección derecha llegando al vértice* 3 *y luego por la izquierda. Esta dirección se representa por la lista* $[[1,0]]$.
En las lineas de código

```
C=LOT([LOT([[],[]],label=5),LOT([LOT([[],LOT([[],[]],label=1)],label=2),[]],label=3)],label=7)
D=LOT([[],LOT([[],[]],label=1)],label=2)
is_tree_in_monomial(C,D)
```

se muestra como obtener la dirección $[[1,0]]$ *usando la función* `is_tree_in_monomial(C,D)` *evaluada en los árboles* `C` *y* `D`.



(a) árbol C **(b)** árbol D

Figura 4.8

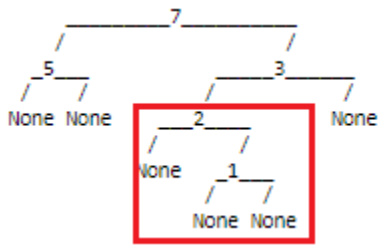


Figura 4.9: Ejemplo de la función *is_tree_in_monomial*

La función `replace_tree_at_address(tree, replacement_tree, address)` definida en el Código 4.19 reemplaza al árbol `replacement_tree` por el hijo del árbol `tree` que se encuentra en la dirección `address`.

Código 4.19: Función replace_tree_at_address

```
def replace_tree_at_address(tree, replacement_tree, address):
    try:
        tree[address]
    except IndexError:
        raise IndexError("The tree does not have such subtree")
    if address == []:
        new_tree = replacement_tree
    else:
        new_tree = LOT([replace_tree_at_address(subtree, replacement_tree, address[1:])
                        if address[0] == i else subtree for i, subtree in enumerate(tree)],
                       label=tree.label())
    return new_tree
```

Ejemplo 30. *Para los árboles C y D de la Figura 4.8, las lineas de código*

```
C=LOT([LOT([[],[]],label=5),LOT([LOT([[],LOT([[],[]],label=1)],label=2),[]],label=3)],label=7)
D=LOT([[],LOT([[],[]],label=1)],label=2)
ascii_art(replace_tree_at_address(C,D,[0]))
```

generan el árbol que resulta de reemplazar el árbol *D* por el hijo del árbol *C* que está en la dirección [0]. El árbol resultante se muestra en la Figura 4.10.

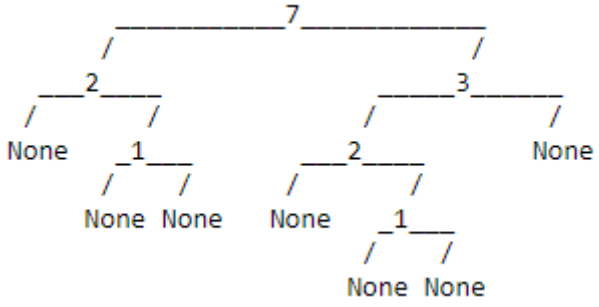


Figura 4.10: Ejemplo de reemplazo

La función `rewrite_rule_at_root` que se define en el Código 4.20 tiene como argumentos un árbol `tree` y una regla de reescritura `rule`. Si la raíz del árbol `tree` coincide con la primer coordenada de la regla de reescritura, es decir con el *monomio lider* esta debe reemplazarse por la segunda coordenada de la

regla de reescritura es decir el *reemplazo*. En caso de que la raíz del árbol `tree` no coincida con la primer coordenada de la regla de reescritura no es posible reeescribir por lo que la función retorna el error *Pattern is not a root*.

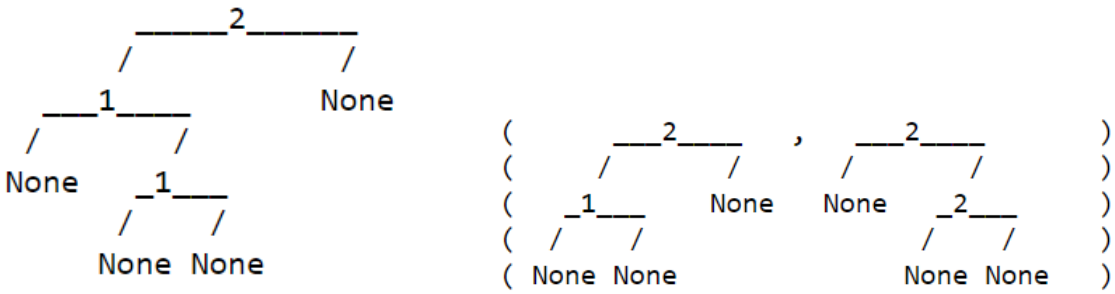
Código 4.20: Función rewrite_rule_at_root

```
def rewrite_rule_at_root(tree, rule):
  if is_tree_pattern_at_root(tree, rule[0]):
      addresses_main = leaf_addresses(rule[0])
      addresses_replacement = leaf_addresses(rule[1])
      if len(addresses_main) != len(addresses_replacement):
          raise IOError("rule is not consistent in the number of children")
      else:
          new_tree = rule[1]
          for j, item in enumerate(addresses_main):
              new_tree = replace_tree_at_address(new_tree,
                                                 tree[addresses_main[j]],
                                                 addresses_replacement[j])
  else:
      raise IOError("Pattern is not at root")
  return new_tree
```

Ejemplo 31. *Consideremos el árbol T que se muestra en la Figura 4.11a y la regla de reescritura del operad Dias que está en la lista de reglas* `dias_rules` *y que se muestra en la Figura 4.11b. Usando la función definida en el código 4.20, las lineas de código*

```
T=LOT([LOT([[],LOT([[],[]],label=1)],label=1),[]],label=2)
dias_rules[3]
ascii_art(rewrite_rule_at_root(T,dias_rules[3]))
```

generan el árbol que resulta de reemplazar el árbol monomio líder por el árbol reemplazo de la regla de reescritura en la raíz del árbol T. Este árbol se muestra en la figura 4.12.



(a) árbol de referencia T **(b)** regla de reescritura en $\mathcal{D}ias$

Figura 4.11: ejemplo Función rewrite_rule_at_root

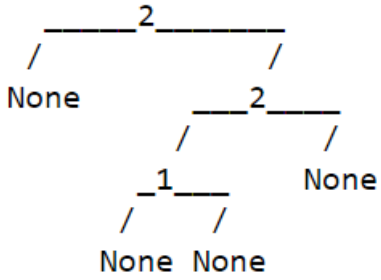


Figura 4.12: Ejemplo de reescritura en la raíz

La función `rewrite_rule_at_address` definida en el Código 4.21 extiende la función `rewrite_rule_at_root` del Código 4.20. Esta función tiene tres argumentos: el árbol `tree`, una pareja regla de reescritura `rule` y además una dirección `address` en la cual, de ser posible se realiza el proceso de reescritura.

Código 4.21: Función rewrite_rule_at_address

```
def rewrite_rule_at_address(tree, rule, address):
    return replace_tree_at_address(tree, rewrite_rule_at_root(tree[address], rule), address)
```

4.7. Generando un grafo dirigido por medio de reescrituras

La función `generate_digraph_of_relations` definida en el Código 4.22 a un árbol inicial `starting_tree` le aplica la función de reescritura definida en el Código 4.21 en todas las direcciones posibles utilizando las reglas dadas en el argumento `rules`. Este proceso genera un grafo dirigido cuyos vértices son el árbol inicial `starting_tree` y todos los árboles que se generan al aplicar la función de reescritura. Cada vez que se aplica la función de reescritura se genera una arista. Cada arista es un par de árboles de la forma (*antes de reescribir*, *después de reescribir*).

Código 4.22: Función generate_digraph_of_relations

```
def generate_digraph_of_relations(starting_tree, rules, graph=None):
    if graph is None:
        graph = DiGraph([[], []])
    applicable_rules = {}
    for rule in rules:
        addresses = is_tree_in_monomial(starting_tree, rule[0])
        if addresses != []:
            applicable_rules[rule] = addresses
    for rule in applicable_rules:
        for address in applicable_rules[rule]:
            new_tree = rewrite_rule_at_address(starting_tree, rule, address)
            graph.add_edge((starting_tree, new_tree))
            generate_digraph_of_relations(new_tree, rules, graph=graph)
    return graph
```

Ejemplo 32. *Generemos el grafo dirigido en el operad no simétrico Dias asociado al monomio crítico que se muestra en la Figura 4.13*

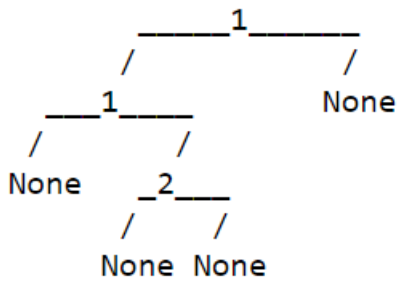


Figura 4.13: Monomio crítico en *Dias*

ejecutando las lineas de código

```
G=generate_digraph_of_relations(dias_critical[9],dias_rules,graph=None)
G.show()
```

Se genera el grafo dirigido tomando como árbol inicial el monomio crítico de la Figura 4.13 y que se encuentra en la entrada 9 de la lista `dias_rules`. El grafo resultante se muestra en la Figura 4.14.

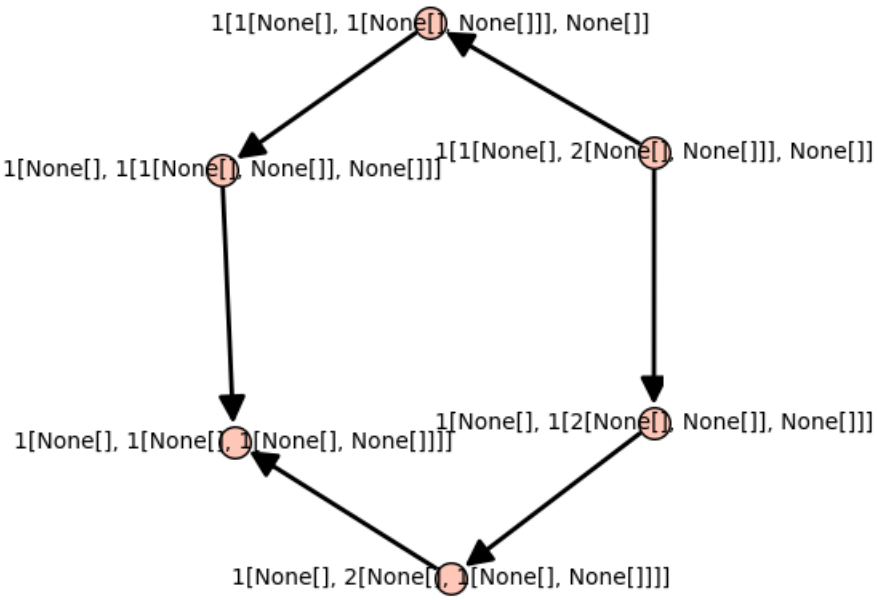


Figura 4.14: Grafo dirigido en *Dias*

4.8. Chequeando confluencia

Verificamos ahora la confluencia del grafo dirigido asociado a un monomio crítico y generado por medio de todas las relaciones de reescritura. Si esta propiedad la satisfacen todos los monomios críticos entonces, como lo indica el Teorema 3, podemos concluir que el operad no simétrico dado es de Koszul.
La función definida en el Código 4.23 evaluada en un grafo dirigido *graph* indica si este es o no confluente basandose en el número de *sinks* o *sumideros* que tiene el grafo dado. Si el número de sinks es 1 el grafo es confluente de lo contrario no lo es.

Código 4.23: Función is_digraph_confluent

```
def is_digraph_confluent(graph):
    return True if len(graph.sinks()) == 1 else False
```

Ejemplo 33. *Como se observa en la Figura 4.14 el grafo dirigido* `G` *tiene un único sink por ende es confluente. Ejecutando el comando* `is_digraph_confluent(G)` *se obtiene como resultado* `True` *lo que comprueba tal afirmación.*

La función `is_operad_koszul` definida en el Código 4.24 indica a través del método de reescritura presentado en la sección 2.4 si un operad no simétrico es de Koszul. El argumento de la función es una lista de reglas de reescritura que se obtienen con la función `get_list_of_rules_using_equivalence_classes` del Código 4.14. En esta lista están las relaciones que generan al operad no simétrico en cuestión.
Dada una lista de reglas de reescritura la función `is_operad_koszul` construye todos los monomios críticos y luego genera para cada monomio crítico su grafo dirigido asociado según el método de reescritura. Finalmente verifica si estos grafos dirigidos son o no confluentes. Si todos los grafos resultan confluentes entonces el operad no simétrico del cual provienen las reglas de reescritura es de Koszul. Si se encuentra por lo menos un grafo no confluente entonces el criterio no es concluyente.

Código 4.24: Función is_operad_koszul

```
def is_operad_koszul(rules):
    new_list_of_leaders = get_list_of_leaders(rules)
    list_of_criticals = get_critical_tree_monomials(new_list_of_leaders)
    koszul = True
    for critical in list_of_criticals:
        graph = generate_digraph_of_relations(critical, rules)
        if is_digraph_confluent(graph) == False:
            koszul = 'The criterion is inconclusive'
            break
    return koszul
```

Ejemplo 34. *Las reglas de reescritura del operad no simétrico Dias que hemos llamado* `dias_rules` *se generaron en el Ejemplo 26. Ejecutando el comando* `is_operad_koszul(dias_rules)` *se obtiene como resultado True lo cual confirma que Dias es un operad de Koszul.*

Ejemplo 35. *En el Código 4.25 se muestra un resumen de los cálculos y comandos necesarios para aplicar el método de reescritura en el operad no simétrico Trias que definimos en la sección 3.3.*

Código 4.25: Resumen del método de reescritura para el operad no simétrico *Trias*

```
# Parameters
arity = 2
labels = [1,2,3]

# Calculations
all_trees=generate_all_quadratic_trees(labels,arity)
print('There are', len(all_trees), 'quadratic trees')
#all_trees
dict_trees = get_light_equivalence_classes(all_trees, ordering_function=quick_sort)
#print(dict_trees.keys()
print("There are", len(dict_trees),"equivalence classes")
rules_new = get_list_of_rules_using_equivalence_classes(dict_trees)
print("There are", len(rules_new), "rules and monomial leaders")
new_list_of_leaders = get_list_of_leaders(rules_new)
list_of_criticals = get_critical_tree_monomials(new_list_of_leaders)
print("The number of critical monomials is", len(list_of_criticals))
print("Is this operad Koszul?", is_operad_koszul(rules_new))
```

Después de ejecutar el Código 4.25 se obtienen los siguientes resultados:

- `There are 18 quadratic trees`
- `There are 7 equivalence classes`
- `There are 11 rules and monomial leaders`
- `The number of critical monomials is 45`
- `Is this operad Koszul? True`

Lo que confirma mediante el método de reescritura que el operad no simétrico Trias es de Koszul.

Observación. *En la linea 2 del Código 4.25 la aridad es 2 pues los operads no simétricos que se han estudiado en este documento son binarios: As, Dias, Trias y Tetraas que se estudiará en el capítulo 5.*
Para aplicar el método de reescritura a los demas operads no simétricos mencionados basta con modificar la linea 3 del Código 4.25, así:

- *Para el operad no simétrico As se debe fijar* `labels=[1]`.
- *Para Dias,* `labels=[1,2]`.
- *Para Trias,* `labels=[1,2,3]`.
- *Para el operad no simétrico Tetraas, que se presentará en el capítulo 5 se debe fijar* `labels=[0,1,2,3]`.

Después de realizar las respectivas modificaciones podemos afirmar que los operads no simétricos As, Dias, Trias y Tetraas son de Koszul.

5 *Tetraas*

En este capítulo usamos la idea de "transferencia de energía" descrita en la sección 3.2.1, para extender naturalmente los operads $\mathcal{A}s$, $\mathcal{D}ias$ y $\mathcal{T}rias$, definiendo una nueva operación $\top \equiv \circ_0$ que actúa "bloqueando" el flujo de energía, es decir que cuando en el vértice de un árbol esté la operación 0 no transporte energía hacia ninguno de sus dos hijos, ni izquierdo ni derecho. Usando esta operación definimos un tipo de álgebras llamadas *Tetra-álgebras* y su operad asociado al que llamamos $\mathcal{T}etraas$. Gracias al teorema 3, con ayuda del código descrito en el capítulo 4 mostramos que este operad es de Koszul. Usando métodos similares a los que se usaron en la sección 3.3.2 mostramos que el dual de Koszul de $\mathcal{T}etraas$ es el operad $\mathcal{T}etradend$ asociado a las álgebras *Tetra-dendriformes* que también definiremos en este capítulo.

5.1. Tetra-álgebras

Una *tetra-álgebra* asociativa M es un espacio vectorial junto con cuatro operaciones binarias y asociativas

$$\top, \dashv, \vdash, \perp : M \otimes M \to M,$$

tal que para todo $a, b, c \in M$ se satisfacen las siguientes relaciones:

1. $(a \dashv b) \dashv c = a \dashv (b \dashv c)$
2. $(a \dashv b) \dashv c = a \dashv (b \vdash c)$
3. $(a \vdash b) \dashv c = a \vdash (b \dashv c)$
4. $(a \dashv b) \vdash c = a \vdash (b \vdash c)$
5. $(a \vdash b) \vdash c = a \vdash (b \vdash c)$
6. $(a \dashv b) \dashv c = a \dashv (b \perp c)$
7. $(a \perp b) \dashv c = a \perp (b \dashv c)$
8. $(a \dashv b) \perp c = a \perp (b \vdash c)$
9. $(a \vdash b) \perp c = a \vdash (b \perp c)$
10. $(a \perp b) \vdash c = a \vdash (b \vdash c)$
11. $(a \perp b) \perp c = a \perp (b \perp c)$
12. $(a \dashv b) \dashv c = a \dashv (b \top c)$

13. $(a \dashv b) \dashv c = a \bot (b \top c)$
14. $(a \top b) \vdash c = a \vdash (b \vdash c)$
15. $(a \top b) \bot c = a \vdash (b \vdash c)$
16. $(a \top b) \top c = a \top (b \top c)$
17. $(a \dashv b) \top c = a \top (b \top c)$
18. $(a \vdash b) \top c = a \top (b \top c)$
19. $(a \bot b) \top c = a \top (b \top c)$
20. $(a \top b) \dashv c = a \top (b \top c)$
21. $(a \top b) \top c = a \top (b \dashv c)$
22. $(a \top b) \top c = a \top (b \vdash c)$
23. $(a \top b) \top c = a \top (b \bot c)$
24. $(a \top b) \top c = a \vdash (b \top c)$

Estas tetra-álgebras nacen como una extensión de las triálgebras con la operación $\top$.
Si $(A, \cdot)$ es un álgebra asociativa, definiendo $a \top b = a \dashv b = a \vdash b = a \bot b := a \cdot b$ para todo $a, b \in A$ obtenemos un funtor

$$As - mod \longrightarrow Tetraas - mod$$

de la categoría de álgebras asociativas en la categoría de tetra-álgebras.
Por otro lado, sí $(A, \top, \dashv, \vdash, \bot)$ es una tetra-álgebra, se puede ver la existencia de los siguientes funtores de olvido:

- De la categoría de tetra-álgebras en la categoría de triálgebras

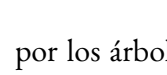
$$\begin{aligned} Tetraas - mod &\longrightarrow Trias - mod \\ (A, \top, \dashv, \vdash, \bot) &\longmapsto (A, \dashv, \vdash, \bot) \end{aligned}$$

- De la categoría de tetra-álgebras en la categoría de diálgebras

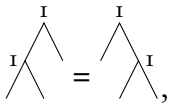
$$\begin{aligned} Tetraas - mod &\longrightarrow Dias - mod \\ (A, \top, \dashv, \vdash, \bot) &\longmapsto (A, \dashv, \vdash) \end{aligned}$$

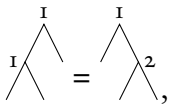
- De la categoría de tetra-álgebras en la categoría de álgebras asociativas

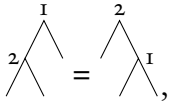
$$\begin{aligned} Tetraas - mod &\longrightarrow As - mod \\ (A, \top, \dashv, \vdash, \bot) &\longmapsto (A, \top) \\ (A, \top, \dashv, \vdash, \bot) &\longmapsto (A, \dashv) \\ (A, \top, \dashv, \vdash, \bot) &\longmapsto (A, \vdash) \\ (A, \top, \dashv, \vdash, \bot) &\longmapsto (A, \bot) \end{aligned}$$

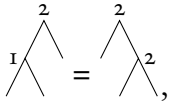
Si denotamos la operación $\top = \circ_0$ como la operación 0. Definimos el operad $\mathcal{T}etraas$ como el generado por los árboles binarios 

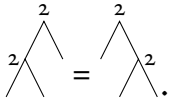
sujeto a las 24 relaciones que definen una tetra-álgebra, tales relaciones en términos de árboles binarios toman la siguiente forma:

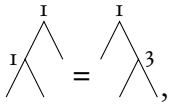
1.  ,

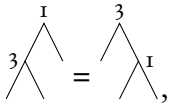
2.  ,

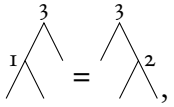
3.  ,

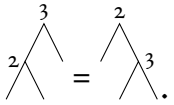
4.  ,

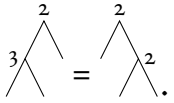
5.  .

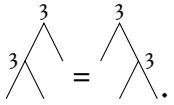
6.  ,

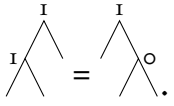
7.  ,

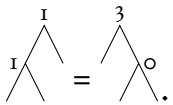
8.  ,

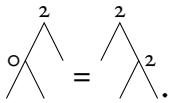
9.  .

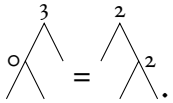
10.  .

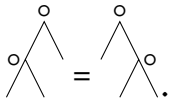
11.  .

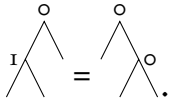
12.  .

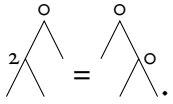
13.  .

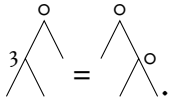
14.  .

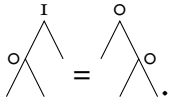
15.  .

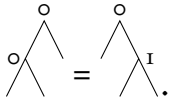
16.  .

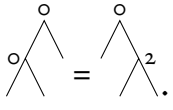
17.  .

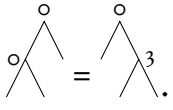
18.  .

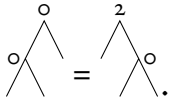
19.  .

20.  .

21.  .

22.  .

23.  .

24.  .

Ejemplo 36. *Sea* $V = \bigoplus_{n\geq 0} \mathbf{k}\omega_n$ *el* $\mathbf{k}$*-espacio vectorial graduado, tal que el subespacio de elementos homogeneos de grado n es el espacio vectorial de dimensión uno generado por* ω_n*, para* $n \geq 0$*. El álgebra tensorial* $(T(V), \top, \dashv, \vdash, \perp)$ *con las operaciones:*

$$
\begin{aligned}
(\omega_{n_1} \otimes \cdots \otimes \omega_{n_k}) \top (\omega_{m_1} \otimes \cdots \otimes \omega_{m_s}) &= \omega_{n_1+\cdots+n_k+m_1\cdots+m_s} \\
(\omega_{n_1} \otimes \cdots \otimes \omega_{n_k}) \dashv (\omega_{m_1} \otimes \cdots \otimes \omega_{m_s}) &= \omega_{n_1} \otimes \cdots \otimes \omega_{n_{k-1}} \otimes \omega_{n_k+m_1+\cdots+m_s} \\
(\omega_{n_1} \otimes \cdots \otimes \omega_{n_k}) \vdash (\omega_{m_1} \otimes \cdots \otimes \omega_{m_s}) &= \omega_{n_1+\cdots+n_k+m_1} \otimes \omega_{m_2} \otimes \cdots \otimes \omega_{m_s} \\
(\omega_{n_1} \otimes \cdots \otimes \omega_{n_k}) \perp (\omega_{m_1} \otimes \cdots \otimes \omega_{m_s}) &= \omega_{n_1} \otimes \cdots \otimes \omega_{n_{k-1}} \otimes \omega_{n_k+m_1} \otimes \omega_{m_2} \otimes \cdots \otimes \omega_{m_s}
\end{aligned}
$$

tiene estructura de tetra-álgebra asociativa, con $n_1, \ldots n_k, m_1 \ldots m_s \geq 0$*. Para observar esto basta con chequear una a una las* 24 *relaciones descritas previamente. Nótese que* $(T(V), \perp)$ *corresponde a la estructura de álgebra asociativa en el álgebra de descensos de Solomon (ver por ejemplo [10]).*

5.2. *Tetradend*

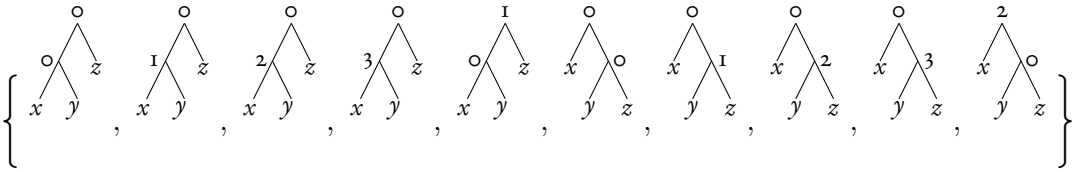
Una tetra-álgebra dendriforme o un álgebra tetradendriforme sobre un cuerpo $\mathbf{k}$ es un $\mathbf{k}$-espacio vectorial N junto con 4 operaciones binarias

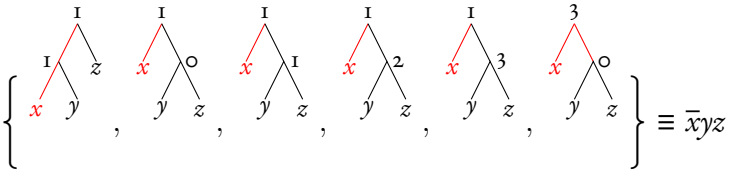
$$\odot, \prec, \succ, \cdot : N \otimes N \to N$$

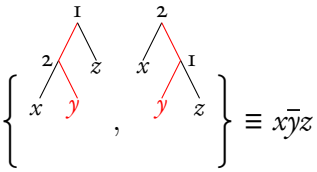
Que satisfacen para $a, b, c \in N$ las siguientes relaciones:

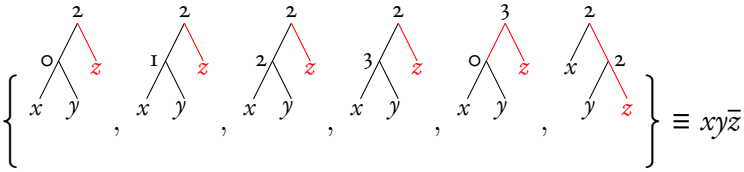
1. $(a * b) \odot c + (a \odot b) \prec c = a \odot (b * c) + a \succ (b \odot c)$
2. $(a \prec b) \prec c = a \prec (b * c) + a \cdot (b \odot c)$
3. $(a \succ b) \prec c = a \succ (b \prec c)$
4. $(a * b) \succ c + (a \odot b) \cdot c = a \succ (b \succ c)$
5. $(a \cdot b) \prec c = a \cdot (b \prec c)$
6. $(a \prec b) \cdot c = a \cdot (b \succ c)$
7. $(a \succ b) \cdot c = a \succ (b \cdot c)$
8. $(a \cdot b) \cdot c = a \cdot (b \cdot c)$

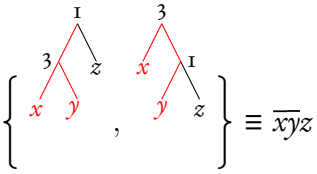
donde $a * b = a \odot b + a \prec b + a \succ b + a \cdot b$.
Como en el caso de las diálgebras y triálgebras dendriformes, en las tetra-álgebras dendriformes las variables presentes en las relaciones que la definen están en el mismo orden, esto nos dice que el operad *Tetradend* asociado a las tetra-álgebras dendriformes es un operad no simétrico.
Los 32 árboles binarios que generan al operad no simétrico *Tetraas* se pueden agrupar en 8 clases de quivalencia según la interpretación de la transferencia de energía que hemos dado a las relaciones que lo definen, estas clases son:

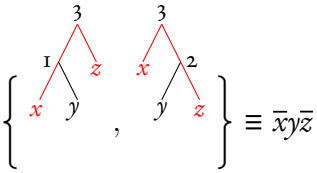
1.  $\equiv xyz$

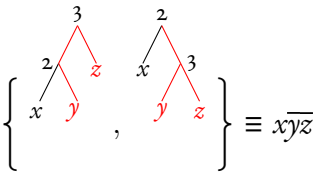
2.  $\equiv \overline{x}yz$

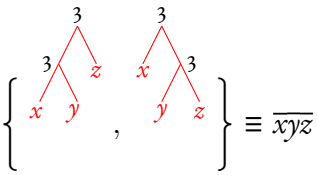
3.  $\equiv x\overline{y}z$

4.  $\equiv xy\overline{z}$

5.  $\equiv \overline{xy}z$

6.  $\equiv \overline{x}y\overline{z}$

7.  $\equiv x\overline{yz}$

8.  $\equiv \overline{xyz}$

Cada uno de los árboles binarios que generan $\mathcal{T}etraas$ se puede identificar con un monomio en n variables, cada variable tiene dos alternativas, estar o no marcada, de allí que $dim(\mathcal{T}etraas(n)) = 2^n$ y en particular $dim(\mathcal{T}etraas(3)) = 8$, esto lo confirma el hecho de que se tengan 8 clases de equivalencia.

En base a cada una de estas clases de equivalencia y recordando el signo de Koszul para los árboles derechos se obtienen las relaciones que definen al operad no simétrico $\mathcal{T}etraas^!$; siguiendo un proceso similar al realizado en 3.3.2 y extendiendolo al caso en el que se tienen cuatro operaciones: $0, 1, 2$ y 3. Haciendo la identificación

$\equiv x \odot y$

se puede ver facilmente que las relaciones que definen al operad dual de Koszul de $\mathcal{T}etraas$ coinciden con las relaciones que definen a las tetra-álgebras dendriformes. Podemos entonces afirmar que $\mathcal{T}etraas^! \cong \mathcal{T}etradend$.
$\mathcal{T}etradend(3)$ está generado por 8 relaciones, esto de nuevo se debe a que $dim(\mathcal{T}etraas(3)) = 8$ de la misma forma que $dim(\mathcal{T}etradend(3)) = 24$, esto pues el operad $\mathcal{T}etraas$ está generado por 24 relaciones.

5.2.1. Series generatrices de $\mathcal{T}etraas$ y $\mathcal{T}etradend$

Para hallar las dimensiones de $\mathcal{T}etraas(n)$ y $\mathcal{T}etradend(n)$ con n un entero positivo, es útil conocer la serie generatriz de cada uno de estos operads. Para el caso de $\mathcal{T}etraas$ ya conocemos $dim(\mathcal{T}etraas(n))$ con esto hallaremos la forma cerrada de su serie generatriz y en base a esta hallaremos la serie generatriz del operad $\mathcal{T}etradend$ para así hallar $dim(\mathcal{T}etradend(n))$.
Sea $F_{\mathcal{T}etraas}(x) = \sum_{n=0}^{\infty} f_n x^n$ la serie generatriz del operad no simétrico $\mathcal{T}etraas$. Como se menciono antes

$$dim(\mathcal{T}etraas(n)) = f_n = \begin{cases} n, n \leq 1 \\ 2^n, n > 1 \end{cases}$$

Luego la serie generatriz del operad $\mathcal{T}etraas$ está dado por:

$$\begin{aligned} F_{\mathcal{T}etraas}(x) &= x + \sum_{n\geq 2} 2^n x^n \\ &= x + \sum_{k\geq 0} (2x)^{k+2} \\ &= x + (2x)^2 \sum_{k\geq 0} (2x)^k \\ &= x + \frac{4x^2}{1-2x} \\ &= \frac{2x^2 + x}{1-2x}. \end{aligned}$$

Utilizaremos esta serie generatriz para hallar la serie generatriz de $\mathcal{T}etradend$. Recordemos que al ser $\mathcal{T}etradend$ el operad dual de Koszul de $\mathcal{T}etraas$, se debe saisfacer la relación

$$F_{\mathcal{T}etraas}(-F_{\mathcal{T}etradend}(-x)) = x \iff F_{\mathcal{T}etraas}(-F_{\mathcal{T}etradend}(x)) = -x.$$

Por simplicidad en la notación llamemos $g = F_{\mathcal{T}etradend}(x)$, se debe tener entonces que

$$F_{\mathcal{T}etraas}(-g) = -x \iff \frac{2(-g)^2 + (-g)}{1-2(-g)} = -x$$

$$\frac{2g^2-g}{1+2g} = -x \Leftrightarrow 2g^2 - g = -x - 2xg \Leftrightarrow 2g^2 + (2x-1)g + x = 0$$

Dividiendo por x ambos lados de esta última ecuación cuadrática obtenemos:

$$2\frac{g^2}{x} + (2x-1)\frac{g}{x} + 1 = 0$$

Haciendo la sustitución $g = tx$ la anterior igualdad toma la forma:

$$2\frac{(tx)^2}{x} + (2x-1)\frac{tx}{x} + 1 = 0 \Leftrightarrow 2xt^2 + (2x-1)t + 1 = 0$$

Ahora con la sustitución $t = h + 1$ tenemos:

$$2x(h+1)^2 + (2x-1)(h+1) + 1 = 0$$

$$\Leftrightarrow$$

$$2x(h^2 + 2h + 1) + 2xh - h + 2x - 1 + 1 = 0 \Leftrightarrow 2x(h^2 + 3h + 2) = h$$

$$\Leftrightarrow$$

$$x(2(h+1)(h+2)) = h$$

Sea $\Phi(h) = 2(h+1)(h+2)$, entonces la ecuación se escribe como

$$x\Phi(h) = h$$

Si $h(x) = \sum_{n\geq 1} h_n x^n$, por el teorema de inversión de Lagrange

$$\begin{aligned}
h_n &= \frac{1}{n}[x^{n-1}]\Phi(x)^n \\
&= \frac{1}{n}[x^{n-1}]2^n(x+1)^n(x+2)^n \\
&= \frac{1}{n}[x^{n-1}]2^n \sum_{j=0}^{n}\binom{n}{j}x^j \sum_{k=0}^{n}\binom{n}{k}2^{n-k}x^k \\
&= \frac{1}{n}[x^{n-1}]2^n \sum_{j\geq 0}\sum_{k\geq 0}\binom{n}{j}\binom{n}{k}2^{n-k}x^{k+j}, \textit{haciendo } k+j=r \\
&= \frac{1}{n}[x^{n-1}]\sum_{r\geq 0}\left[\sum_{k=0}^{r}\binom{n}{k}\binom{n}{r-k}2^{2n-k}\right]x^r, \textit{cuando r=n-1} \\
&= \frac{1}{n}\sum_{k=0}^{n-1}\binom{n}{k}\binom{n}{n-1-k}2^{2n-k} \\
&= \frac{1}{n}\sum_{k=0}^{n-1}\binom{n}{k}\binom{n}{n-(k+1)}2^{2n-k} \\
&= \sum_{k=0}^{n-1}\frac{1}{n}\binom{n}{k}\binom{n}{k+1}2^{2n-k}.
\end{aligned}$$

Como $t = h + 1$ y $g = tx$, entonces $F_{\mathcal{T}etradend}(x) = g = (h+1)x = x + xh$, luego

$$g = x + x\sum_{n\geq 1} h_n x^n = x + \sum_{n\geq 1} h_n x^{n+1} = x + \sum_{n\geq 2} h_{n-1}x^n.$$

Así, podemos afirmar que

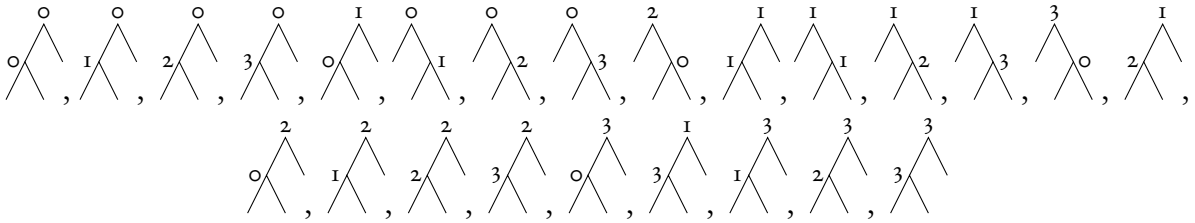
$$F_{\mathcal{T}etradend}(x) = x + \sum_{n\geq 2}\left[\sum_{k=0}^{n-2}\frac{1}{n-1}\binom{n-1}{k}\binom{n-1}{k+1}2^{2(n-1)-k}\right]x^n.$$

Y por lo tanto

$$dim(\mathcal{T}etradend(n)) = \begin{cases} 1, n = 1 \\ \sum_{k=0}^{n-2}\frac{1}{n-1}\binom{n-1}{k}\binom{n-1}{k+1}2^{2(n-1)-k}, n > 1 \end{cases}$$

5.2.2. Ordenes monomiales en $\mathcal{T}etraas$

La operación $\top$ satisface que $\top < \dashv < \vdash < \bot$, es decir $0 < 1 < 2 < 3$. Extendiendo el proceso realizado en la sección 3.3.4 con la operación $\top$ y con el código descrito en el capítulo 4 se comparan las sucesiones de caminos que definen las 24 relaciones que definen al operad no simétrico $\mathcal{T}etraas$ obteniendo así los términos líderes:



Estos 24 términos líderes deben componerse entre sí de todas las formas posibles para obtener los monomios críticos, este proceso se realiza usando el código del capítulo 4 lo cual nos arroja como resultado un total de 176 monomios críticos.

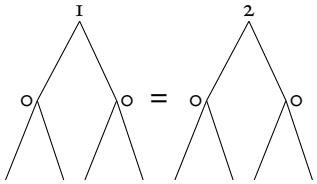
Teorema 11. *El operad no simétrico $\mathcal{T}etraas$ y su dual de Koszul $\mathcal{T}etradend$ son de Koszul.*

Demostración. Para demostrar este teorema basta con chequear que los 176 diamantes correspondientes a los monomios críticos de $\mathcal{T}etraas$ son confluentes, luego gracias al teorema 3 se tiene el resultado. Este proceso se realiza con ayuda del código mostrado en el capítulo 4.

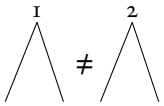
□

Teorema 12. *El operad no simétrico $\mathcal{T}etraas$ no es cancelativo.*

Demostración. Para demostrar este teorema basta considerar el siguiente contraejemplo



Pero claramente



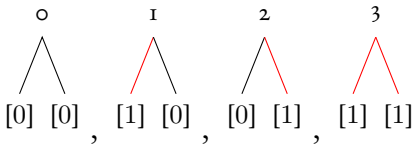
□

Por no ser *Tetraas* un operad cancelativo no es posible aplicar las técnicas que usan posets para probar que este es un operad de Koszul.

6
Generalizaciones y futuras líneas de trabajo

6.1. Otras aridades

Los operads que se estudiaron en este documento son todos binarios o de aridad 2. Con ayuda del código mostrado en el capítulo 4 se mostró que todos estos operads no simétricos: *As*, *Dias*, *Trias*, *Tetraas* son de Koszul.
Si en la linea 2 del Código 4.25 cambiamos la aridad a 3 por ejemplo; se concluye que el Método de reescritura no funciona como lo mostraremos a continuación.
La interpretación de la transferencia de energía que dimos a las operaciones desarrolladas en capítulos anteriores para los operads binarios se pueden presentar por medio de árboles binarios de la siguiente forma:



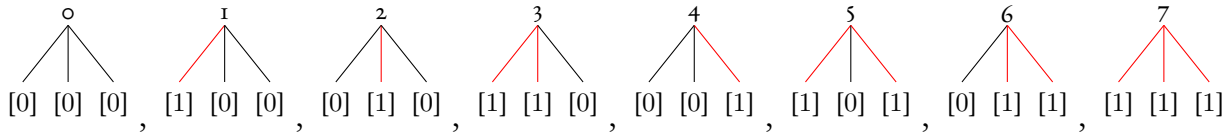
Donde la etiqueta $[0]$ indica que la energía no se transporta en esa dirección mientras que la etiqueta $[1]$ indica que la energía se transportó en la dirección de la hoja correspondiente. Con las etiquetas de las hojas se obtiene la representación reversa del número que está en la etiqueta de la raíz (operación) escrito en forma binaria. Es decir

- $0 = 0 \cdot 2^1 + 0 \cdot 2^0 \equiv [0][0]$
- $1 = 0 \cdot 2^1 + 1 \cdot 2^0 \equiv [1][0]$
- $2 = 1 \cdot 2^1 + 0 \cdot 2^0 \equiv [0][1]$
- $3 = 1 \cdot 2^1 + 1 \cdot 2^0 \equiv [1][1]$

Extendiendo esta idea para árboles de aridad 3 tenemos 8 posibles operaciones

- $0 = 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 \equiv [0][0][0]$
- $1 = 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \equiv [1][0][0]$
- $2 = 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \equiv [0][1][0]$
- $3 = 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \equiv [1][1][0]$
- $4 = 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 \equiv [0][0][1]$
- $5 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \equiv [1][0][1]$
- $6 = 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \equiv [0][1][1]$
- $7 = 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \equiv [1][1][1]$

Estas operaciones $\circ_0, \circ_1, \circ_2, \circ_3, \circ_4, \circ_5, \circ_6, \circ_7$ surgen a partir de los siguientes árboles:



Siguiendo la idea de transporte de energía que se dió para el operad *Dias* en la sección 3.2.1 definimos el operad $3-$*Dias* como el operad no simétrico de aridad 3 generado por las operaciones $\circ_1$, $\circ_2$ y $\circ_4$. Si en el Código 4.25 cambiamos las lineas 2 y 3 por:

$$\texttt{arity = 3}\ y\ \texttt{labels=[1,2,4]}$$

obtenemos el siguiente resumen del método de reescritura para el operad $3-$*Dias*:

- `There are 27 quadratic trees`
- `There are 5 equivalence classes`
- `There are 22 rules and monomial leaders`
- `The number of critical monomials is 213`
- `Is this operad Koszul? The criterion is inconclusive`

Como el criterio no es concluyente significa que existe por lo menos un grafo dirigido no confluente. En el Código 6.1 mostramos como verificar para cada uno de los grafos dirigidos que se generan si son o no confluentes. En caso de que alguno de los grafos dirigidos no sea confluente éste se visualizará.

Código 6.1: Confluencia de los grafos dirigidos

```
for tree in list_of_criticals:
    graph = generate_digraph_of_relations(tree, rules_new)
    print("Is this graph confluent? ", is_digraph_confluent(graph))
    if not is_digraph_confluent(graph):
            graph.show()
```

Ejemplo 37. *En la Figura 6.1 mostramos uno de los grafos dirigidos del operad* $3-\mathcal{D}ias$ *que no es confluente y que se obtuvo ejecutando el Código 6.1.*

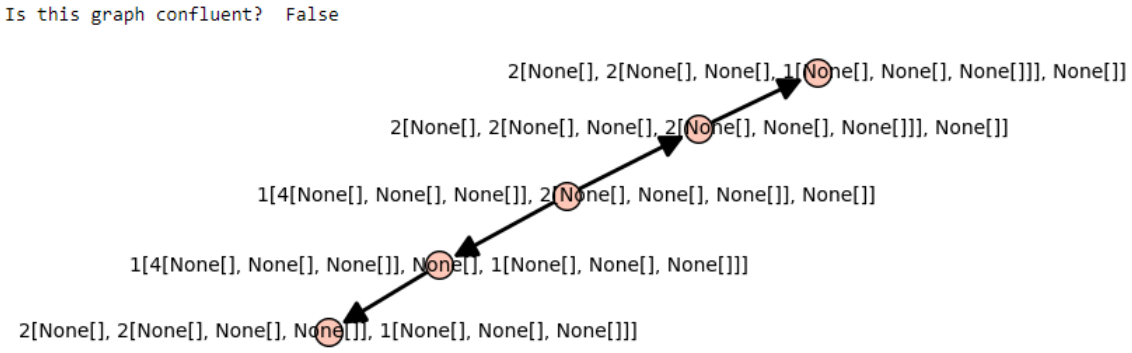


Figura 6.1: Ejemplo de un grafo dirigido no confluente en $3-\mathcal{D}ias$

Extendiendo la manera en la cual se definió el operad $3-\mathcal{D}ias$ podemos definir los operads no simétricos $3-\mathcal{A}s$, $3-\mathcal{T}rias$, $3-\mathcal{T}etraas$ de la siguiente forma:

- $3-\mathcal{A}s$ es el operad no simétrico de aridad 3 generado por la operación $\circ_1$.
- $3-\mathcal{T}rias$ es el operad no simétrico de aridad 3 generado por las operaciones $\circ_1$, $\circ_2$, $\circ_3$, $\circ_4$, $\circ_5$, $\circ_6$ y $\circ_7$.
- $3-\mathcal{T}etraas$ es el operad no simétrico de aridad 3 generado por las operaciones $\circ_0$, $\circ_1$, $\circ_2$, $\circ_3$, $\circ_4$, $\circ_5$, $\circ_6$ y $\circ_7$.

Modificando el Código 4.25 para cada uno de estos casos se concluye que:

- $3-\mathcal{A}s$ es un operad de Koszul. En general cuando la aridad es $(k+1)$ para k un entero mayor o igual que 1 este operad se conoce como operad de *álgebras totalmente asociativas* y se denota por $t\mathcal{A}s_{(k+1)}$. En [6] Gnedbaye muestra que $t\mathcal{A}s_{(k+1)}$ es un operad de Koszul y que su dual de Koszul es el operad de *álgebras parcialmente asociativas* de aridad $(k+1)$ denotado por $p\mathcal{A}s_{(k+1)}$.
- El criterio del método de reescritura aplicado al operad $3-\mathcal{T}rias$ no es concluyente.
- El criterio del método de reescritura aplicado al operad $3-\mathcal{T}etraas$ no es concluyente.

Esta misma idea puede generalizarse para definir operads no simétricos de aridad $(k+1)$: $(k+1)-\mathcal{D}ias$, $(k+1)-\mathcal{T}rias$ y $(k+1)-\mathcal{T}etraas$, con k un entero mayor o igual que 1.

6.2. Otras técnicas

La pregunta que surge naturalmente es si con otras aridades un determinado operad no simétrico es de Koszul. En otras palabras: por el teorema 3 sabemos que si en un operad no simétrico todo monomio crítico es confluente, entonces el operad es de Koszul. Pero si no todo monomio crítico resulta confluente esto no implica que el operad no sea de Koszul. En estos casos hay que tratar de aplicar otras técnicas para verificar la propiedad de Koszul en un operad no simétrico y no binario.

Por ejemplo si un operad no simétrico no binario es cancelativo entonces mediante las técnicas que involucran los posets asociados y la función de Möbius dadas en la definición 2.1 podría probarse la propiedad de Koszul.
Otra opción podría ser mediante técnicas de álgebra homológica construyendo los complejos de Koszul asociados a tales operads y observando su comportamiento. Esto pues en [2] Bremer y Dotsenko definen un operad de Koszul como un operad para el cual su complejo de Koszul asociado es acíclico. Estas técnicas homológicas son tratadas en detalle en textos especializados en álgebra homológica como por ejemplo en [17].

Bibliografía

[1] Bergeron, F., Bergeron, F., Labelle, G., & Leroux, P. (1998). *Combinatorial species and tree-like structures*, volume 67. Cambridge University Press.

[2] Bremner, M. R. & Dotsenko, V. (2016). *Algebraic operads: an algorithmic companion*. CRC Press.

[3] Dotsenko, V. & Khoroshkin, A. (2010). Gröbner bases for operads. *Duke Math. J.*, 153(2), 363–396.

[4] Flajolet, P. & Sedgewick, R. (2009). *Analytic combinatorics*. cambridge University press.

[5] Ginzburg, V. & Kapranov, M. (1994). Koszul duality for operads. *Duke Math. J.*, 76(1), 203–272.

[6] Gnedbaye, A. V. (1997). Opérades des algèbres $(k+1)$-aires. In *Operads: Proceedings of Renaissance Conferences (Hartford, CT/Luminy, 1995)*, volume 202 of *Contemp. Math.* (pp. 83–113).: Amer. Math. Soc., Providence, RI.

[7] Joyal, A. (1981). Une théorie combinatoire des séries formelles. *Advances in mathematics*, 42(1), 1–82.

[8] Leinster, T. (2014). *Basic category theory*, volume 143. Cambridge University Press.

[9] Loday, J.-L. (2001). Dialgebras. In *Dialgebras and related operads* (pp. 7–66). Springer.

[10] Loday, J.-L., Ronco, M., et al. (2004). Trialgebras and families of polytopes. *Contemporary Mathematics*, 346, 369–398.

[11] Loday, J.-L. & Vallette, B. (2012). *Algebraic operads*, volume 346. Springer Science & Business Media.

[12] May, J. (1972). The geometry of iterated loop spaces (1972).

[13] Méndez, M. A. (2015). *Set operads in combinatorics and computer science*. Springer.

[14] Méndez, M. A. & Yang, J. (1991). Möbius species. *Advances in Mathematics*, 85(1), 83–128.

[15] Stein, W. et al. (2020). *Sage Mathematics Software (Version 9.0)*. The Sage Development Team. `http://www.sagemath.org`.

[16] Vallette, B. (2007). Homology of generalized partition posets. *Journal of Pure and Applied Algebra*, 208(2), 699–725.

[17] Weibel, C. A. (1995). *An introduction to homological algebra*. Number 38 in Cambridge Studies in Advanced Mathematics. Cambridge university press.

[18] Zinbiel, G. W. (2012). Encyclopedia of types of algebras 2010. *Operads and universal algebra*, 9, 217–297.

This thesis was typeset using LaTeX, originally developed by Leslie Lamport and based on Donald Knuth's TeX. The body text is set in 12 point Egenolff-Berner Garamond, a revival of Claude Garamont's humanist typeface.

This document style template was adapted by Francisco A. Mayorga Cetina for use by students at Sergio Arboleda University, based on the original Harvard template made by Jordan Suchow. That template can be used to format a PhD thesis with this look and feel that has been published under the permissive MIT (X11) license, and can be found online at github.com/suchow/Dissertate