UNIVERSIDAD DE BUENOS AIRES
Facultad de Ciencias Exactas y Naturales
Departamento de Matemática

Tesis de Licenciatura

Invariantes algebraicos de links y su cálculo en GAP

Santiago Daniel Varela

Director: Marco Farinati

27 de Agosto de 2024

Agradecimientos

A mi familia, quienes me apoyaron y me apoyan día a día.

A Rocío, María, Ignacio, Cecilia, Nahuel, Javier y Lucas, los amigos con quienes tuve la suerte de compartir y disfrutar este camino académico.

A Agustin Barreto, mi mentor, docente y mi amigo. Sin duda la influencia más grande de mi recorrido matemático.

A Daniel Grimaldi, Leandro Vendramin y Daniel Carando, los docentes que me guiaron a lo que hoy tanto me apasiona.

A mi director, que confió en mi y con entusiasmo me guió en los primeros pasos de mi camino dentro de la investigación.

A los jurados, Juliana y Mariano, quienes me proporcionaron valiosas correcciones y comentarios para presentar esta tesis de la mejor manera.

Y finalmente a los lectores, ya que mi objetivo es compartir estos conocimientos, incentivando a más personas a adentrarse en el hermoso mundo que construye la teoría de nudos.

Índice general

Introducción

La teoría de nudos es una rama de la topología dedicada al estudio y la clasificación de los nudos en el espacio tridimensional. Un nudo se forma al tomar una cuerda con dos extremos, anudarla de alguna manera y luego unir esos extremos. Dos de estos nudos son equivalentes, si es posible deformar el anudamiento de uno para que se vea igual que el otro.

El propósito de esta teoría es clasificar los distintos tipos de nudos no equivalentes. Para ello, se recurre a la búsqueda de invariantes: herramientas matemáticas diseñadas para distinguir entre nudos que no son equivalentes y para demostrar propiedades sobre ellos.

La teoría de nudos tiene aplicaciones prácticas en diversos campos de estudio. Por ejemplo, en biología molecular, los científicos utilizan la teoría de nudos para comprender y modelar la estructura del ADN y las proteínas. La capacidad de clasificar diferentes formas de nudos es crucial para entender cómo se empaqueta y se replica el material genético en las células. En física teórica, los nudos se utilizan como modelos para describir fenómenos en la teoría de cuerdas y la teoría cuántica de campos. La clasificación de nudos proporciona una herramienta poderosa para identificar y estudiar las simetrías fundamentales en el universo. Además, en la criptografía y la computación cuántica, los invariantes de nudos se utilizan como herramientas para el diseño de algoritmos y protocolos de seguridad. La capacidad de distinguir entre diferentes tipos de nudos es esencial para garantizar la seguridad y la integridad de los sistemas de comunicación y cifrado en un entorno digital cada vez más complejo.

Mucha de esta teoría se generaliza a links, que son uniones finitas de nudos que pueden entrelazarse entre sí.

En esta tesis, abordaremos la teoría mediante invariantes algebraicos de links, enfocándonos particularmente en invariantes basados en estructuras algebraicas conocidas como quandles, racks y biracks. Estas estructuras emergen del estudio de los diagramas de links, una representación gráfica en el plano que permite trabajar con los links de manera más tangible y accesible.

En el primer capítulo, veremos las nociones básicas y formales de nudos y links, conoceremos el invariante más famoso, además de introducir los quandles y su relación con los diagramas de links.

En el segundo capítulo, profundizaremos en los quandles definiendo los 2-cociclos, y construiremos con ellos la función de partición, herramienta relacionada con la física que involucra varias áreas de la matemática.

En el tercer capítulo, hablaremos de nudos enmarcados, una variación del concepto de nudos usando cintas en vez de sogas. Desarrollaremos también la teoría de racks y biracks, estructuras algebraicas que permiten construir invariantes para links enmarcados. Luego de eso, veremos como nos conducen al concepto de grupo universal de 2-cociclo, un resultado fundamental desarrollado por M. Farinati y J. Galofre en [FG16].

En el cuarto capítulo, proporcionamos una guía básica del lenguaje de programación GAP y presentamos los códigos desarrollados para esta tesis, con el fin de calcular los invariantes discutidos en los capítulos anteriores.

Finalmente, en el anexo, se encuentran los códigos listos para ser utilizados en GAP, facilitando la implementación práctica de los conceptos y algoritmos presentados hasta el momento.

A través de esta tesis, esperamos proporcionar una introducción accesible y práctica a algunos de los conceptos algebraicos que subyacen en el estudio de la teoría de nudos.

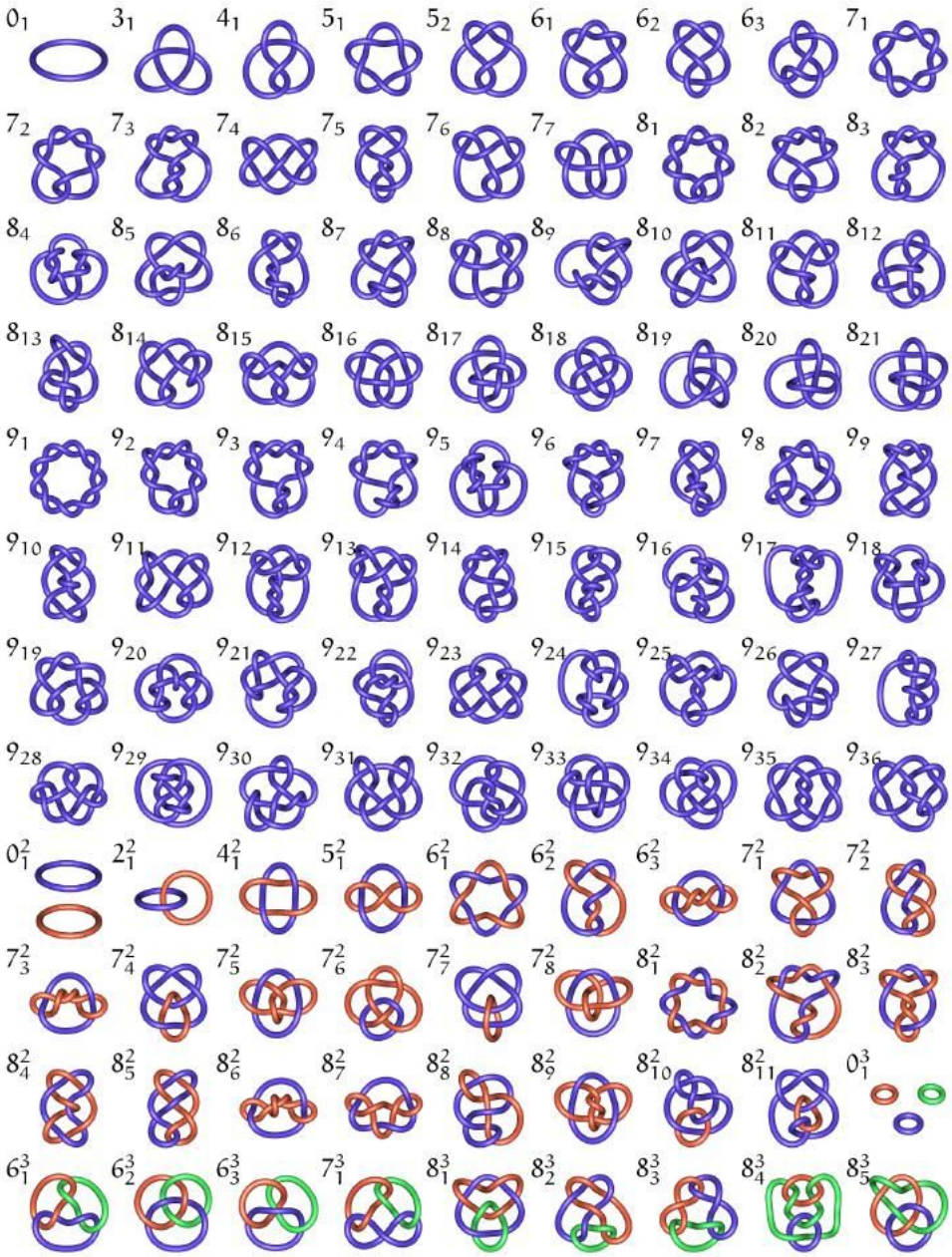


Figura 1: Tabla de clasificación de nudos elaborada por P.G.Tait

CAPÍTULO 1

Conceptos básicos

1.1. Nudos

Definición 1.1. Un **nudo** (en $\mathbb{R}^3$) es una función inyectiva y continua $\alpha : S^1 \to \mathbb{R}^3$, donde la circunferencia unitaria está dada por $S^1 = \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 = 1\}$.

Diremos que los nudos dados por las funciones α y β son **equivalentes** si y sólo si existe una isotopía entre ellas, es decir, si existe una función continua $H\colon S^1 \times [0, 1] \to \mathbb{R}^3$ tal que la función $H_t\colon z \mapsto H(z, t)$ es un nudo para todo $t \in [0, 1]$, $H_0 = \alpha$ y $H_1 = \beta$.

Definición 1.2. Un **enlace** (o **link**), es una unión de n nudos en el espacio que pueden estar anudados entre sí, es decir, es una función inyectiva y continua $\gamma : S^1 \sqcup ... \sqcup S^1 \to \mathbb{R}^3$ donde n es la cantidad de circunferencias del dominio. Cada uno de estos nudos es una **componente** del link. Dos links serán equivalentes si existe una isotopía entre ellos.

Nos referiremos indistintamente a un nudo/link como la función inyectiva y continua, su imagen o su clase de equivalencia de tales funciones. Aseguramos que esto no causará confusión alguna.

Una **orientación** se define eligiendo una dirección para viajar alrededor del nudo. Para orientar links, tomamos una orientación de cada componente. Dos links orientados serán equivalentes si existe una isotopía H entre ellos que sea compatible con las orientaciones.

Para evitar casos molestos, como puede ser un nudo que se anude sobre sí infinitamente, consideraremos únicamente nudos equivalentes a nudos dados por poligonales, o sea por unión de segmentos. Estos nudos se llaman **nudos mansos**, y los que no sean mansos se llamarán **nudos salvajes**. En la figura 1.1 mostramos un ejemplo de nudo salvaje.

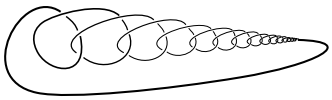


Figura 1.1: Un ejemplo de nudo salvaje.

De aquí en adelante, nuestros nudos siempre serán mansos, y trabajaremos con links que tengan como componentes nudos mansos.

En la práctica, nuestros nudos poligonales tendrán tantos segmentos que será casi imposible diferenciar a nuestra curva de una curva suave.

Sea K un link. Consideremos la proyección de K en el plano dada por

$$\pi\colon (x, y, z) \mapsto (x, y).$$

Un punto $p \in \pi(K)$ es un **punto múltiple** si $\pi^{-1}(p)$ contiene más de un punto de K. La **multiplicidad** de p se define como el cardinal del conjunto $\pi^{-1}(p) \cap K$. Una proyección de K en el plano se dice **genérica** si tiene las siguientes propiedades: 1) hay una cantidad finita de puntos de multiplicidad mayor a uno, 2) no hay puntos de multiplicidad mayor que dos, y 3) no hay puntos dobles donde uno de los puntos es un vértice. La figura 1.2 muestra algunos ejemplos de cruces no admitidos en una proyección genérica, el de la izquierda por ser un punto triple y los otros dos por presentar vértices.

Con deformaciones continuas del nudo en el espacio, se puede ver que todo nudo (manso) es equivalente a uno que admite una proyección genérica. Esto puede consultarse en [BZH14] por ejemplo.

Figura 1.2: Cruces no admitidos en el diagrama de un link dado por una poligonal.

Definición 1.3. Un **diagrama** de un link K es una proyección genérica de K en el plano donde en cada **cruce** (punto de multiplicidad dos) se puede distinguir qué segmento pasa por arriba y qué segmento pasa por debajo. Para ilustrar esta situación, el segmento que pasa por debajo se dibuja cortado. Las componentes conexas del diagrama se llaman **arcos**. Para diagramas de links orientados, agregaremos una flecha al diagrama que indique la orientación.

Definición 1.4. Cualquier nudo equivalente a $S^1 = \{(x, y, 0) \in \mathbb{R}^3 : x^2 + y^2 = 1\}$ será considerado como el **nudo trivial**. En la práctica, no siempre es fácil reconocer la trivialidad de un nudo. En la figura 1.3 vemos dos proyecciones distintas del nudo trivial. Una proyección aún más curiosa del nudo trivial puede verse en la figura 1.4.

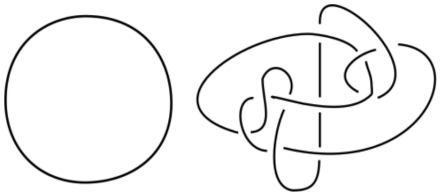


Figura 1.3: Dos diagramas del nudo trivial.

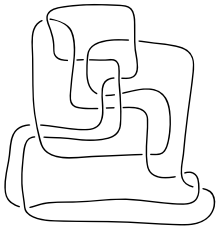


Figura 1.4: Diagrama de Thistlethwaite del nudo trivial.

Ejemplo 1.1. Posiblemente el más reconocido ejemplo no trivial es el **nudo trébol**, denotado por 3_1. Dos formas de representarlo se pueden ver en la figura 1.5, donde la primera es una poligonal y la segunda es la curva en el espacio dada por :

$$\{(x, y) \in \mathbb{C}/x^2 + y^2 = 0 \wedge |x|^2 + |y|^2 = \epsilon^2, \epsilon \in \mathbb{Z}\}$$

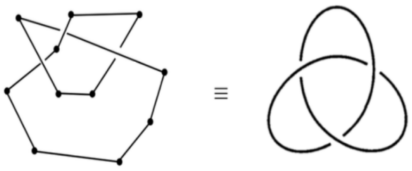


Figura 1.5: Dos proyecciones del nudo trébol

Ejemplo 1.2. El link no trivial más sencillo es el dado por dos nudos triviales enlazados de la forma más simple. Este se puede apreciar en la figura 1.6 y se conoce como **enlace de Hopf**. también tenemos otro link famoso con el mismo concepto, anudando tres nudos triviales, como se puede ver en la figura 1.7, llamado **anillos de Borromeo**.

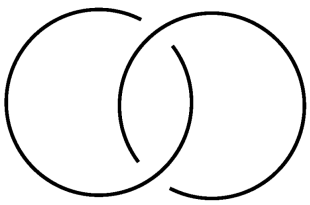


Figura 1.6: Enlace de Hopf

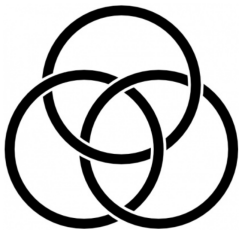


Figura 1.7: Anillos de Borromeo

Operaciones con Nudos

Definición 1.5. La **imagen especular** de un nudo se obtiene al aplicarle al nudo la transformación $(x, y, z) \mapsto (x, y, -z)$. En la figura 1.8 vemos el nudo 3_1 y su imagen especular $m(3_1)$. Más adelante, demostraremos que los nudos 3_1 y $m(3_1)$ no son equivalentes.

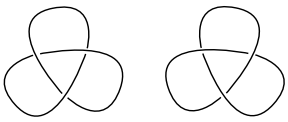


Figura 1.8: El nudo 3_1 (derecha) y su imagen especular $m(3_1)$ (izquierda).

Definición 1.6. Si K es un nudo orientado, el **reverso** $r(K)$ de K es K como conjunto pero con la orientación opuesta.

Ejemplo 1.3. El nudo 3_1 es equivalente al nudo $r(3_1)$.

Ejemplo 1.4. El nudo 4_1, comunmente llamado nudo 8 (ver figura 1.9) es **totalmente simétrico**, es decir: los nudos 4_1, $m(4_1)$, $r(4_1)$ y $rm(4_1)$ son todos equivalentes.

Figura 1.9: El nudo 4_1.

Ejemplo 1.5. El nudo 9_{32} de la figura 1.10 es **totalmente asimétrico**, es decir: los nudos 9_{32}, $m(9_{32})$, $r(9_{32})$ y $rm(9_{32})$ son todos no equivalentes.

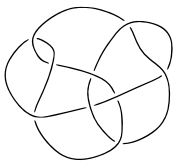


Figura 1.10: El nudo 9_{32} es totalmente asimétrico.

Definición 1.7. Dados dos nudos orientados K y L podemos obtener un nuevo nudo con el siguiente procedimiento: quitamos un pedacito de arco de cada una de las proyecciones de nuestros nudos y luego unimos los cuatro puntos finales obtenidos con dos nuevos arcos (es importante que hagamos esto sin agregar nuevos cruces) tal como muestra la figura 1.11. Esta operación se denomina **composición** de nudos. La composición de los nudos K y L se denota por $K\#L$.

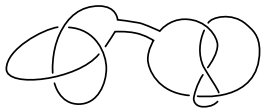


Figura 1.11: Composición de nudos.

No es difícil demostrar que la composición de nudos es una operación asociativa y conmutativa y que el nudo trivial es el neutro de esta operación.

Definición 1.8. Un nudo no trivial es **primo** si no puede descomponerse como la composición de otros nudos no triviales. Un nudo es **compuesto** si no es primo.

El problema de determinar si un nudo dado es primo es extremadamente difícil. La figura 1.12 contiene las proyecciones de los primeros nudos primos (salvo reverso e imagen especular) donde cada nudo tiene a lo sumo siete cruces.

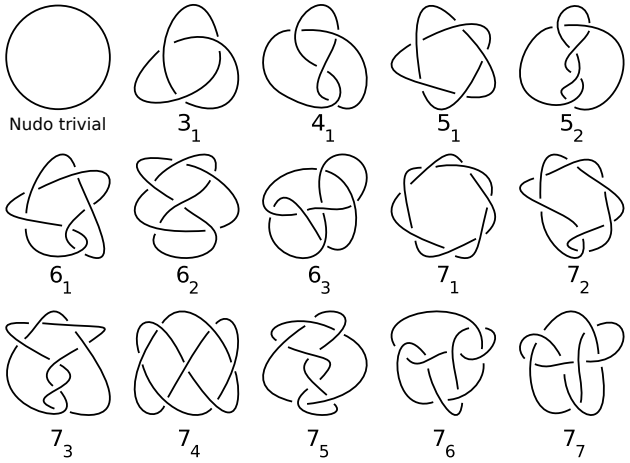


Figura 1.12: Algunos nudos primos.

Ejemplo 1.6. Consideremos el nudo que se forma al componer dos nudos 3_1. Este nudo se conoce como el **nudo de la abuela** (o *granny knot*, en inglés), se denota por $3_1\#3_1$, y se muestra a la izquierda en la figura 1.13.

Ejemplo 1.7. El nudo compuesto formado por 3_1 y su imagen especular $m(3_1)$ se conoce como el **nudo cuadrado** (o *square knot*, en inglés), se denota por $3_1\#m(3_1)$, y se muestra a la derecha en la figura 1.13.

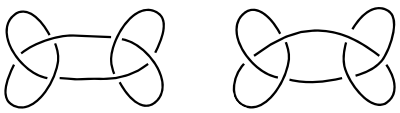


Figura 1.13: El nudo de la abuela (izquierda) y el nudo cuadrado (derecha) no son equivalentes.

Movimientos de Reidemeister

En 1926 Reidemeister vislumbró una forma combinatoria de verificar si dos nudos son equivalentes. Básicamente, dos diagramas representarán al mismo nudo si y sólo si puede pasarse de un diagrama al otro mediante una sucesión finita de ciertas transformaciones, $\mathcal{R}_1$, $\mathcal{R}_2$ y $\mathcal{R}_3$, llamadas **movimientos de Reidemeister**. Hay tres de estos movimientos:

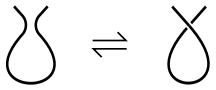


Figura 1.14: $\mathcal{R}_1$

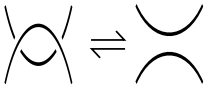


Figura 1.15: $\mathcal{R}_2$

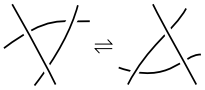


Figura 1.16: $\mathcal{R}_3$

el primero se muestra en la figura 1.14, el segundo en la figura 1.15 y el tercero en la figura 1.16.

Teorema 1.1. (de Reidemeister) Dos nudos son equivalentes si y sólo si sus diagramas están conectados por una sucesión finita de movimientos de Reidemeister.

Una demostración de este teorema se puede encontrar en [Liv94].

Ejemplo 1.8. En la figura 1.17 se muestran las movidas realizadas para transformar el diagrama del nudo trivial de la derecha de la figura 1.3 en el de la izquierda.

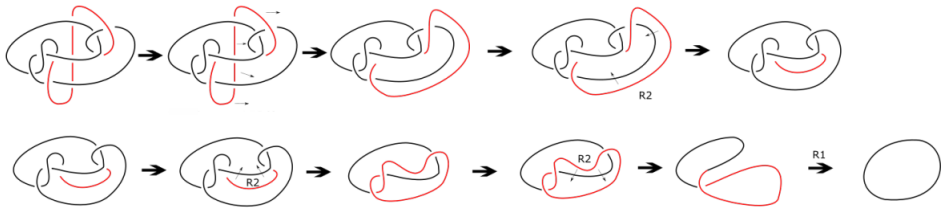


Figura 1.17: Sucesión de movimientos de Reidemeister

Observación 1.1. El mismo desarrollo vale para links, ya que la invarianza es con respecto a la configuración de cruces, independiente de lo que esté afuera de ese cruce.

Luego, dos links son equivalentes si y sólo si sus diagramas están conectados por una sucesión finita de movimientos de Reidemeister.

Grupo fundamental

Definición 1.9. El **grupo fundamental** de un link K es el grupo de homotopía

$$\pi_1(\mathbb{R}^3 \setminus K)$$

Lo notamos $\pi_1(K)$, recordando que trabajamos con su complemento.

Veamos cómo calcular el grupo fundamental de un link. Supongamos que K es un link con n cruces. Dotamos a cada componente del link con una orientación, y etiquetamos los arcos de la proyección del link con las variables $a_1, a_2, a_3 \ldots$

Como se ve en la figura 1.18, el diagrama tendrá dos tipos de cruce: cruces positivos y cruces negativos. Por cada cruce χ del diagrama como el que vemos en la figura 1.18, consideramos la **relación de Wirtinger** $r_\chi = 1$, donde $r_\chi = a_i a_j a_i^{-1} a_k^{-1}$. Como nos indica el siguiente teorema, el grupo fundamental de K es isomorfo al grupo dado por los generadores $a_1, a_2, a_3 \dots$ y las relaciones de Wirtinger. Esta presentación del grupo fundamental se conoce como la **presentación de Wirtinger**.

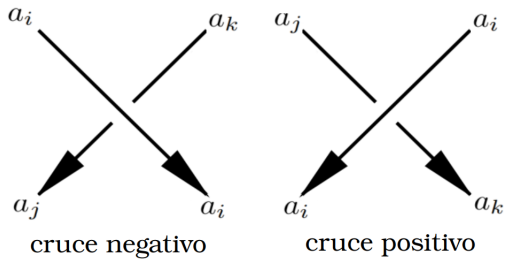


Figura 1.18: Una orientación da dos tipos de cruce. En ambos casos, la relación de Wirtinger es $a_i a_j a_i^{-1} = a_k$.

Observación 1.2. Le asociamos a cada componente del link una orientación, pero esta no influye en las relaciones que quedan al final. Así que con cualquier orientación, tendremos la misma presentación de Wirtinger.

Teorema 1.2. (de Wirtinger) Sea K un link y supongamos que K tiene una proyección con n arcos y m cruces. Entonces

$$\pi_1(K) \simeq \langle a_1, a_2, \dots, a_n : r_1, r_2, \dots, r_m \rangle \tag{1.1}$$

donde las relaciones $r_1, \dots, r_m$ están dadas por las fórmulas de Wirtinger. La ecuación (1.1) simboliza el cociente del grupo libre en $a_1, \dots, a_n$ por el menor subgrupo normal que contiene a los elementos $r_1, \dots, r_m$.

Una demostración de este teorema se puede ver en [BZH14].

Tietze fue el primero en calcular el grupo fundamental del nudo 3_1 en 1908. Ese mismo año conjeturó que dos nudos son equivalentes si y sólo si sus complementos en $\mathbb{R}^3$ son homeomorfos.

En 1915 Dehn demostró que el grupo fundamental permite detectar la trivialidad de un nudo. Más precisamente, el teorema de Dehn establece que un nudo es trivial si y sólo si el grupo fundamental del nudo es isomorfo a $\mathbb{Z}$. Es importante remarcar que, en general, es muy difícil determinar si un grupo finitamente presentado es isomorfo al grupo trivial.

Muchos años después, en 1989, Gordon y Luecke demostraron la veracidad de la conjetura. Con lo cual, este invariante distingue entre nudos no equivalentes.

Ejemplo 1.9. Vamos a calcular el grupo fundamental del nudo 3_1 que vemos en la figura 1.19. Las relaciones de Wirtinger son

$$a_1 a_2 a_1^{-1} = a_3, \quad a_2 a_3 a_2^{-1} = a_1, \quad a_3 a_1 a_3^{-1} = a_2, \tag{1.2}$$

y entonces,

$$\pi_1(3_1) \simeq \langle a_1, a_2, a_3 : \ a_1 a_2 a_1^{-1} = a_3, \ a_2 a_3 a_2^{-1} = a_1, \ a_3 a_1 a_3^{-1} = a_2 \rangle.$$

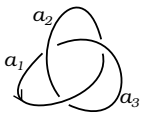


Figura 1.19: Etiquetas en los arcos de 3_1

Vamos a utilizar el grupo $\pi_1(3_1)$ para demostrar la no trivialidad de 3_1. Como existe un morfismo sobreyectivo de grupos $\pi_1(3_1) \to \mathbb{S}_3$ tal que

$$a_1 \mapsto (12), \quad a_2 \mapsto (23), \quad a_3 \mapsto (13),$$

y el grupo simétrico $\mathbb{S}_3$ no es abeliano, se sigue que $\pi_1(3_1)$ es un grupo no abeliano. En particular $\pi_1(3_1) \not\simeq \mathbb{Z}$ y entonces el nudo 3_1 no es equivalente al nudo trivial.

Observación 1.3. Notamos que para cualquier link, la imagen especular es un homeomorfismo entre los complementos de los respectivos links, por lo que los grupos fundamentales de esos espacios son isomorfos y por lo tanto el $\pi_1(-)$ nunca puede distinguir a K de $m(K)$. Tampoco es posible distinguir con esta herramienta a K de $r(K)$, debido a que la orientación de la curva en el espacio no interviene en el cálculo del $\pi_1(-)$.

Observación 1.4. También es importante aclarar que el grupo fundamental es un invariante que no distingue links, y un ejemplo de esto son los links K_1 y K_2 de la figura 1.20. Ambos tienen mismo grupo fundamental (puede verse en [Rol04]), pero no son equivalentes como links. Esto lo veremos más adelante, cuando trabajemos con invariantes en GAP.

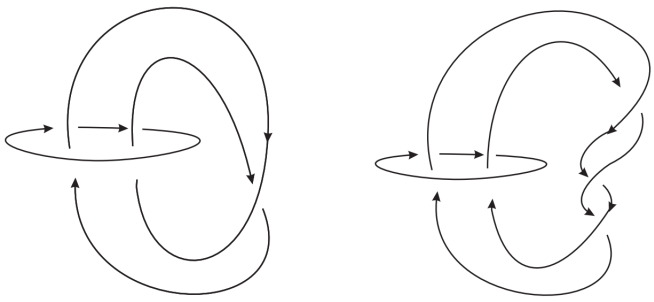


Figura 1.20: Links K_1 y K_2

1.2. Quandles

La idea de quandle fue introducida de manera formal por David Joyce en 1982, quien propuso esta estructura algebraica inspirado por los trabajos de Alexander, Conway y Wraith, acerca de conjugación de grupos. Joyce desarrolló los quandles como herramientas para producir invariantes de nudos, por las propiedades que estos tienen, y su relación con los diagramas.

Definición 1.10. Un **quandle** es un par $(X, \triangleleft)$, donde X es un conjunto no vacío con una operación binaria $X \times X \to X$, $(x, y) \mapsto x \triangleleft y$, tal que

$$\varphi_x \colon X \to X,\ y \mapsto y \triangleleft x \text{ es biyectiva}, \qquad \text{para todo } x \in X, \tag{1.3}$$

$$(x \triangleleft y) \triangleleft z = (x \triangleleft z) \triangleleft (y \triangleleft z) \qquad \text{para todo } x, y, z \in X, \tag{1.4}$$

$$x \triangleleft x = x \qquad \text{para todo } x \in X. \tag{1.5}$$

Notaremos $-\triangleleft^{-1} x$ a la inversa de φ_x.

Definición 1.11. Sean X e Y dos quandles. Una función $f \colon X \to Y$ es un **morfismo** de quandles si $f(x \triangleleft x') = f(x) \triangleleft f(x')$ para todo $x, x' \in X$.

Ejemplo 1.10. Sea X un conjunto no vacío. Entonces X es un quandle con $x \triangleleft y = x$ para todo $x, y \in X$. Este quandle se denomina **quandle trivial** sobre X.

Ejemplo 1.11. Sean G un grupo y X una clase de conjugación de G. Entonces X es un quandle con $x \triangleleft y = y^{-1}xy$ para todo $x, y \in X$.

Un grupo G con la operación $x \triangleleft y = y^{-n}xy^n$ es llamado **quandle de n-conjugación**.

El quandle asociado a la clase de conjugación de g en G se llama **quandle de conjugación** asociado a (la clase de) g en G.

Ejemplo 1.12. Sea M un $\mathbb{Z}[t, t^{-1}]$-módulo a izquierda. Definimos el **quandle de Alexander** sobre M como el quandle dado por

$$x \triangleleft y = tx + (1 - t)y \qquad \text{para todo } x, y \in M. \tag{1.6}$$

Veamos un caso particular de la contrucción del quandle de Alexander. Sea $\mathbb{F}_q$ el cuerpo de q elementos, donde q es una potencia de un número primo. Para cada $t \in \mathbb{F}_q \setminus \{0\}$ definimos el **quandle de Alexander de tipo** (q, t) como el quandle sobre $\mathbb{F}_q$ dado por $x \triangleleft y = tx + (1 - t)y$ para todo $x, y \in \mathbb{F}_q$.

Ejemplo 1.13. Dado $S \subset V$ con V un $\mathbb{F}$-espacio vectorial, que es no degenerado con respecto a la forma bilineal simétrica $\langle -, - \rangle : V \times V \to \mathbb{F}$, S es un quandle con la operación dada por

$$x \triangleleft y = y - 2\frac{\langle x, y \rangle}{\langle x, x \rangle} x$$

A esta estructura la llamamos quandle de Coxeter.

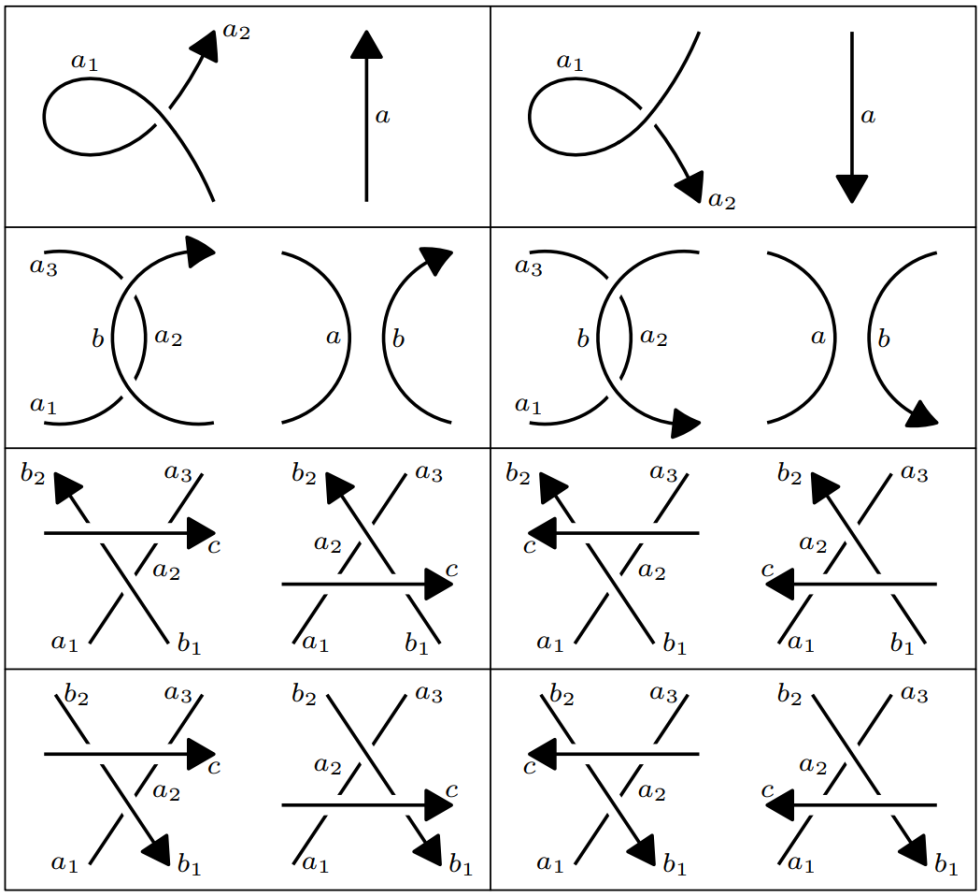


Figura 1.21: Movimientos de Reidemeister orientados

Definición 1.12. Una **coloración** del diagrama de un link orientado K con el quandle X es una asignación $\mathcal{C} : \{\text{Arcos de K}\} \to X$ que cumple la relación $a_j \triangleleft a_i = a_k$, para cada cruce negativo o positivo como muestra la Figura 1.18.

Notamos $\mathrm{Col}_X(K)$ a la cantidad de coloraciones de K por X.

Observemos que siempre existirán al menos $|X|$ coloraciones del link orientado K con el quandle X (las coloraciones triviales).

Por qué colorear? Ahora los quandles tienen color? Esto se dice así ya que es una generalización de un invariante de links llamado coloreos de Fox, que consiste en asignar colores como etiquetas a cada arco.

Teorema 1.3. $\mathrm{Col}_X(K)$ es invariante de links orientados.

Demostración. Sean D1 y D2 dos diagramas de un link K. Luego estan conectados por un número finito de movidas de Reidemeister. Considerando las posibles orientaciones de los cruces, vemos que tenemos 8 configuraciones con los arcos etiquetados, como se muestra en la figura 1.21. Para cada movimiento, tenemos que comprobar que hay una biyección en los etiquetados, es decir, que la cantidad de coloraciones es la misma para

ambos casos. Por ejemplo, al considerar el primer movimiento (esquina superior izquierda), tenemos un arco entrante a_1, y uno saliente a_2. Pero el saliente debe cumplir la relación $a_1 \triangleleft a_1 = a_2$ con lo cual $a_1 = a_2$ por propiedades de quandle. Entonces vamos a tener una sola etiqueta, al igual que en el diagrama recto de la derecha. Por lo tanto es invariante por este movimiento.

Para el segundo movimiento de Reidemeister, tomamos los diagramas de la parte izquierda de la figura 1.21. Para cualquier orientación del arco izquierdo, tenemos que $a_2 = a_1 \triangleleft^{-1} b$ y eso implica que $a_3 = a_2 \triangleleft b = a_1 \triangleleft^{-1} b \triangleleft b = a_1$. Los diagramas de la derecha son análogos, con el signo de los cruces invertido.

En cuanto al tercer movimiento, haremos sólo uno de los posibles casos, y dejaremos el resto como ejercicio para el lector, ya que todos usan la misma idea. Tomemos el primer $\mathcal{R}_3$ de la figura 1.21, que aparece del lado izquierdo. Nuevamente, para ambas orientaciones posibles, vamos a tener que las relaciones son las mismas. Por la parte izquierda, obtenemos que:

$$a_2 = a_1 \triangleleft^{-1} b_1, \qquad b_2 = b_1 \triangleleft c, \qquad a_3 = a_2 \triangleleft c = (a_1 \triangleleft^{-1} b_1) \triangleleft c$$

Mientras que para el diagrama del lado derecho obtenemos:

$$a_2 = a_1 \triangleleft c, \qquad b_2 = b_1 \triangleleft c, \qquad a_3 = a_2 \triangleleft^{-1} b_2 = (a_1 \triangleleft c) \triangleleft^{-1} (b_1 \triangleleft c)$$

Como nos interesan las expresiones finales de los arcos, y en ambos diagramas las expresiones de b_2 coinciden, nos resta ver que las de a_3 son la misma.

Aplicando $- \triangleleft (b_1 \triangleleft c)$ obtenemos por un lado:

$$(a_1 \triangleleft c) \triangleleft^{-1} (b_1 \triangleleft c) \triangleleft (b_1 \triangleleft c) = a_1 \triangleleft c$$

Mientras que por el otro lado tenemos:

$$\begin{aligned} ((a_1 \triangleleft^{-1} b_1) \triangleleft c) \triangleleft (b_1 \triangleleft c) = \\ ((a_1 \triangleleft^{-1} b_1) \triangleleft b_1) \triangleleft c = \\ a_1 \triangleleft c \end{aligned}$$

Con lo cual coinciden y por ello la cantidad de coloraciones es invariante. □

Ejemplo 1.14. Veamos que el nudo 4_1 puede colorearse de forma no trivial con un quandle de Alexander (tomando cualquier orientación, ya que es totalmente simétrico). Consideremos el cuerpo

$$\mathbb{F}_4 = \mathbb{F}_2[t]/(t^2 + t + 1) = \{0, 1, t, t + 1\},$$

y sea X el quandle de Alexander de tipo $(4, t)$. Las coloraciones con X del diagrama de la figura 1.22 son las soluciones $(a, b, c, d) \in \mathbb{F}_4$ del sistema de ecuaciones

$$\begin{aligned} (1-t)a + tc = d, \quad (1-t)b + ta = d, \\ (1-t)c + ta = b, \quad (1-t)d + tc = b. \end{aligned} \tag{1.7}$$

Como $(a, b, c, d) = (0, 1, t, t + 1)$ es una solución de (1.7), el nudo 4_1 admite entonces al menos una coloración con X no trivial. Así demostramos la no trivialidad del nudo 4_1.

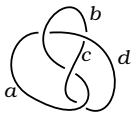


Figura 1.22: Una coloración del nudo 4_1.

CAPÍTULO 2

State sum: suma de estados

2.1. Cohomología de quandles

Sean A un grupo abeliano (escrito multiplicativamente), X un quandle, y $f\colon X\times X\to A$ una función. Sobre el conjunto $A\times X$ definimos la operación

$$(a,x)\triangleleft(b,y)=(af(x,y),x\triangleleft y)\quad\text{para todo }(x,a),(y,b)\in A\times X. \tag{2.1}$$

Es fácil demostrar que la operación (2.1) define una estructura de quandle sobre $A\times X$ si y sólo si:

$$f(x,x)=1, \qquad \text{para todo } x\in X, \tag{2.2}$$

$$f(x,z)f(x\triangleleft z,y\triangleleft z)=f(x,y)f(x\triangleleft y,z) \qquad \text{para todo } x,y,z\in X. \tag{2.3}$$

Como ejemplo, demostremos la distributividad. Sean $x,y,z\in X$ y $a,b,c\in A$. Un cálculo directo nos muestra que

$$\begin{aligned}((a,x)\triangleleft(b,y))\triangleleft(c,z)&=(af(x,y),x\triangleleft y)\triangleleft(c,z)\\&=(af(x,y)f(x\triangleleft y,z),(x\triangleleft y)\triangleleft z),\end{aligned}$$

y por otro lado,

$$\begin{aligned}((a,x)\triangleleft(c,z))\triangleleft((b,y)\triangleleft(c,z))&=(af(x,z),x\triangleleft z)\triangleleft(bf(y,z),y\triangleleft z)\\&=(af(x,z)f(x\triangleleft z,y\triangleleft z),(x\triangleleft z)\triangleleft(y\triangleleft z))\end{aligned}$$

Como X es un quandle, tenemos entonces que la condición (2.3) es equivalente a la distributividad de la operación binaria en $X\times A$.

Definición 2.1. Una función $f : X \times X \to A$ que satisface las condiciones (2.2) y (2.3) se llama 2**-cociclo del quandle** X con coeficientes en A.

Ejemplo 2.1. Para el quandle trivial $x \triangleleft y = x$, cualquier función que cumpla

$$f(x, x) = 1 \qquad \forall x \in X$$

es un 2-cociclo.

Ejemplo 2.2. Sea X el quandle de conjugación asociado a la clase de (1234) en $\mathbb{S}_4$ y sea $A = \langle\sigma\rangle = \{1, \sigma, \sigma^2, \sigma^3\}$ el grupo cíclico de orden cuatro (escrito multiplicativamente). La función $f\colon X \times X \to A$ dada por el cuadro 2.1, donde las filas son los x y las columnas son las y:

f	(1234)	(1432)	(1342)	(1243)	(1324)	(1423)
(1234)	1	σ	σ^2	σ^2	σ	σ^3
(1432)	σ	1	σ^2	1	σ^3	σ^3
(1342)	σ^2	σ	1	σ	σ^2	σ^3
(1243)	σ^3	σ^2	σ	1	1	σ^3
(1324)	σ	σ	σ	σ	1	σ
(1423)	1	1	1	1	σ	1

Cuadro 2.1: 2-cociclo f

es un 2-cociclo de X con coeficientes en A.

Definición 2.2. El quandle sobre $X \times A$ definido por (2.1) se llama **extensión abeliana** de X por el grupo abeliano A y el 2-cociclo f, y se denota por $X \times_f A$.

Ejemplo 2.3. Veremos que el nudo de la abuela $3_1\#3_1$ y el nudo cuadrado $3_1\#m(3_1)$ de la figura 2.1 no son equivalentes. Sean X el quandle de conjugación asociado a la clase de (1234) en $\mathbb{S}_4$ y f el 2-cociclo de X del cuadro 2.1. Consideremos la extensión $X \times_f A$ dada por

$$(x, \sigma^i) \triangleleft (y, \sigma^j) = (x \triangleleft y, \sigma^j f(x, y)) \quad \text{para todo } x, y \in X,\ i, j \in \{0, \dots 3\}.$$

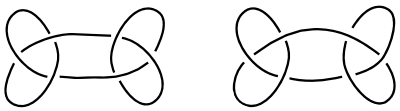


Figura 2.1: El nudo de la abuela $3_1\#3_1$ (izquierda) y el nudo cuadrado $3_1\#m(3_1)$ (derecha)

La extensión abeliana $X \times_f A$ nos permite distinguir el nudo de la abuela del nudo cuadrado. Con ayuda computacional (que veremos en el capítulo 4), se puede chequear que la cantidad de coloreos para el nudo de la abuela con este quandle es:

$$\mathrm{Col}_{X \times_f A}(3_1\#3_1) = 24, \tag{2.4}$$

mientras que para el nudo cuadrado, tenemos:

$$\mathrm{Col}_{X\times_f A}(3_1\#m(3_1)) = 408. \tag{2.5}$$

Con lo cual tenemos que son nudos no triviales y no equivalentes.

Cada 2-cociclo de un quandle X nos permite definir una extensión abeliana de X. Estos 2-cociclos son en realidad 2-cociclos en una teoría de cohomología de quandles. Tal como pasa en la teoría de grupos, las clases de equivalencia de extensiones abelianas del quandle X por el grupo abeliano A están en correspondencia biyectiva con las clases de equivalencia de 2-cociclos de X con coeficientes en A.

2.2. Invariante por 2-cociclos

La función de partición es un concepto que proviene de la mecánica estadística y física matemática, asociada con la teoría de campos topológica llamada teoría de Chern-Simons. Esta misma es una cantidad que resume toda la información sobre los estados cuánticos del sistema.

Definición 2.3. Fijemos un grupo abeliano A (escrito multiplicativamente) y un link orientado K. Sean X un quandle finito, $f\colon X\times X\to A$ un 2-cociclo de X con coeficientes en A y $\mathcal{C}\colon \{\text{Arcos de K}\}\to X$ una coloración de K. En cada cruce como el que vemos en la figura 2.2 se define el **peso de Boltzmann** $\omega_f(\mathcal{C},\chi)$ (con respecto al 2-cociclo f, a la coloración $\mathcal{C}$ y al cruce χ) como el elemento de A dado por la expresión:

$$\omega_f(\mathcal{C},\chi) = \begin{cases} f(a_j,a_i)^{-1} & \text{si el cruce es negativo} \\ f(b_j,b_i) & \text{si el cruce es positivo} \end{cases}$$

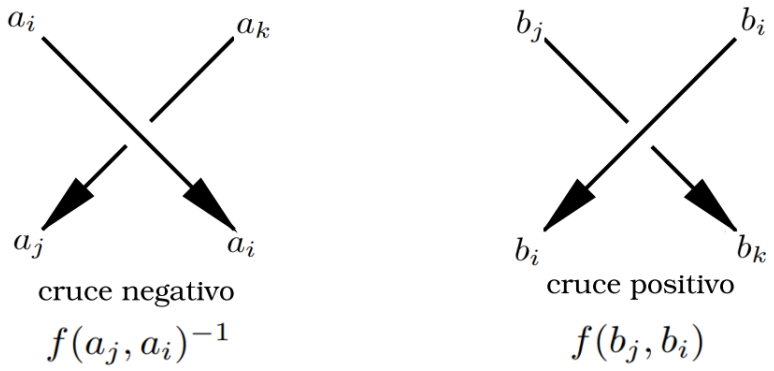


Figura 2.2: Pesos asociados a cada cruce

Definición 2.4. La **función de partición** (o **state-sum**) $Z_{X,f}(K)$ del link orientado K, asociada al quandle X y al 2-cociclo f, es la expresión

$$Z_{X,f}(K) = \sum_{\mathcal{C}} \prod_{\chi} \omega_f(\mathcal{C}, \chi), \tag{2.6}$$

donde el producto se toma sobre todos los cruces χ que tiene el diagrama del link K y la suma se toma sobre todos las coloraciones $\mathcal{C}$ de K dadas por el quandle X. La fórmula (2.6) define un elemento de $\mathbb{Z}[A]$, el anillo de grupo de A.

Ejemplo 2.4. Consideramos el nudo trébol $K = 3_1$ representado con el diagrama de trenza de la Figura (2.3). Sea $X = \mathbb{Z}_2[t]/(t^2 + t + 1)$ el quandle de Alexander con $x \triangleleft y = tx + (1 - t)y$. Se puede comprobar facilmente que etiquetando con cualquier par de elementos de X los arcos superiores del primer cruce del nudo, queda determinado un coloreo del nudo por X (ver Figura 2.4). Entonces $Col_X(K) = 4^2$.

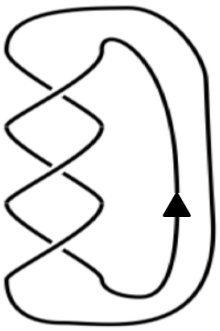


Figura 2.3: Nudo trébol trenzado K

Sea A el grupo abeliano (en forma aditiva) dado por X pero con la operación del módulo. Consideramos la función $\phi : X \times X \to A$ dada por $\phi(x, y) = (x - y)^2 y$. Veamos que es un 2-cociclo (escrito de forma aditiva):

Primero vemos que $\phi(x, x) = (x - x)^2 x = 0$, así que cumple (2.2). Para chequear que cumple (2.3), uso las propiedades de A, como que $t^3 = 1$, $(x \triangleleft y)^2 = (tx)^2 + ty^2$ o que $(x + y)^2 = x^2 + y^2$.

Con esto obtenemos que

$$\begin{aligned} \phi(x, y) + \phi(x \triangleleft y, z) &= (x - y)^2 y + \phi(x \triangleleft y, tx + (1 - t)y, z) \\ &= (x^2 + y^2)y + ((tx + (1 + t)y)^2 + z^2)z \\ &= x^2 y + y^3 + (tx)^2 z + ty^2 z + z^3 \end{aligned} \tag{2.7}$$

Por otro lado,

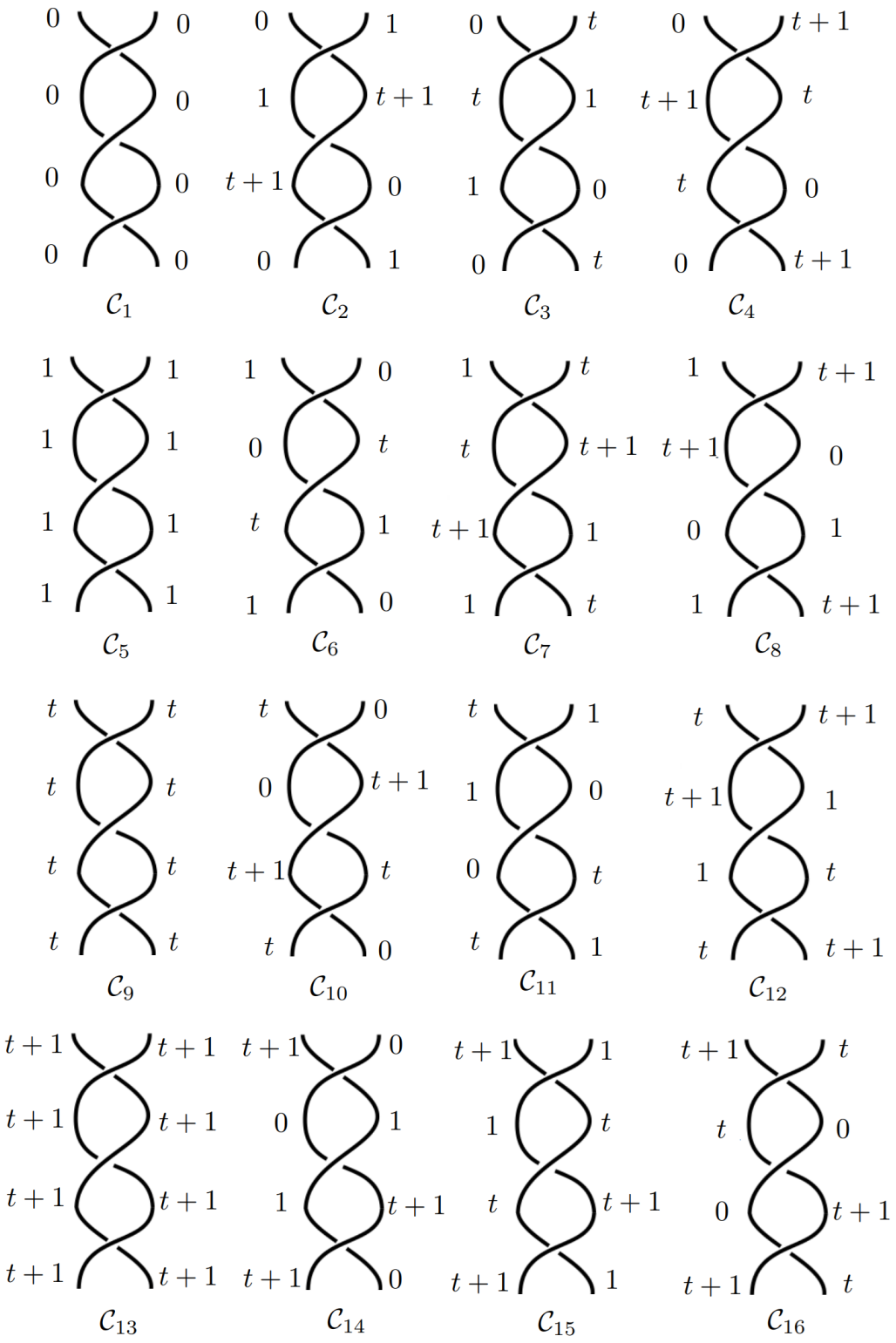


Figura 2.4: Coloraciones del nudo trenzado 3_1

$$\begin{aligned}\phi(x,z)+\phi(x\triangleleft z, y\triangleleft z) &= x^2z+z^3+[(x\triangleleft z)^2+(y\triangleleft z)^2(y\triangleleft z)\\ &= x^2z+z^3+[((tx)^2+tz^2)+((ty)^2+tz^2)](ty+z+tz)\\ &= x^2z+z^3+[(t^3x^2y+(tz)^2y)+((ty)^3+(tz)^2y)]\\ &+[((tx)^2z+tz^3)+((ty)^2z+tz^3)]+[(t^3x^2z+t^2z^3)+(t^3y^2z+t^2z^3)]\\ &= x^2z+z^3+x^2y+y^3+(tx)^2z+(ty)^2z+x^2z+y^2z\\ &= z^3+x^2y+y^3+(tx)^2z+ty^2z\\ &= x^2y+y^3+(tx)^2z+ty^2z+z^3\end{aligned} \tag{2.8}$$

así que coinciden y cumplen las condiciones de ser 2-cociclo.

Para calcular los pesos de Boltzmann, conseguimos los valores de $\phi(x,y)$, donde las filas son los x y las columnas son las y:

ϕ	0	1	t	$t+1$
0	0	1	1	1
1	0	0	$t+1$	t
t	0	t	0	$t+1$
$t+1$	0	$t+1$	t	0

Ahora, podemos considerar los pesos de Boltzmann en cada cruce y armarnos la función de partición. Solo que el grupo A está escrito en forma aditiva. Pero esto lo solucionamos facilmente con un cambio de notación:

$$e^a.e^b := e^{a+b} \quad \forall a,b\in A$$

Esta nueva escritura, nos permite considerar a $(A,\cdot)$ con este producto, y cada sumando de la función será un elemento de $\mathbb{Z}[A]$, dado por alguna de estas coloraciones:

$$\prod_{\chi}\omega_f(\mathcal{C}_1,\chi)=\phi(0,0).\phi(0,0).\phi(0,0)=e^0.\,e^0.\,e^0=e^0$$
$$\prod_{\chi}\omega_f(\mathcal{C}_2,\chi)=\phi(0,1).\phi(1,t+1).\phi(t+1,0)=e^1.\,e^t.\,e^0=e^{t+1}$$
$$\prod_{\chi}\omega_f(\mathcal{C}_3,\chi)=\phi(0,t).\phi(t,1).\phi(1,0)=e^1.\,e^t.\,e^0=e^{t+1}$$
$$\prod_{\chi}\omega_f(\mathcal{C}_4,\chi)=\phi(0,t+1).\phi(t+1,t).\phi(t,0)=e^1.\,e^t.\,e^0=e^{1+t}$$
$$\prod_{\chi}\omega_f(\mathcal{C}_5,\chi)=\phi(1,1).\phi(1,1).\phi(1,1)=e^0.\,e^0.\,e^0=e^0$$
$$\prod_{\chi}\omega_f(\mathcal{C}_6,\chi)=\phi(1,0).\phi(o,t).\phi(t,1)=e^0.\,e^1.\,e^t=e^{t+1}$$

$$\prod_{\chi}\omega_f(\mathcal{C}_7,\chi)=\phi(1,t).\phi(t,t+1).\phi(t+1,1)=e^{t+1}.\,e^{t+1}.\,e^{t+1}=e^{t+1}$$
$$\prod_{\chi}\omega_f(\mathcal{C}_8,\chi)=\phi(1,t+1).\phi(t+1,0).\phi(0,1)=e^{t}.\,e^{0}.\,e^{1}=e^{t+1}$$
$$\prod_{\chi}\omega_f(\mathcal{C}_9,\chi)=\phi(t,t).\phi(t,t).\phi(t,t)=e^{0}.\,e^{0}.\,e^{0}=e^{0}$$
$$\prod_{\chi}\omega_f(\mathcal{C}_{10},\chi)=\phi(t,0).\phi(0,t+1).\phi(t+1,t)=e^{0}.\,e^{1}.\,e^{t}=e^{t+1}$$
$$\prod_{\chi}\omega_f(\mathcal{C}_{11},\chi)=\phi(t,1).\phi(1,0).\phi(0,t)=e^{t}.\,e^{0}.\,e^{1}=e^{t+1}$$
$$\prod_{\chi}\omega_f(\mathcal{C}_{12},\chi)=\phi(t,t+1).\phi(t+1,1).\phi(1,t)=e^{t+1}.\,e^{t+1}.\,e^{t+1}=e^{t+1}$$
$$\prod_{\chi}\omega_f(\mathcal{C}_{13},\chi)=\phi(t+1,t+1).\phi(t+1,t+1).\phi(t+1,t+1)=e^{0}.\,e^{0}.\,e^{0}=e^{0}$$
$$\prod_{\chi}\omega_f(\mathcal{C}_{14},\chi)=\phi(t+1,0).\phi(0,1).\phi(1,t+1)=e^{0}.\,e^{1}.\,e^{t}=e^{t+1}$$
$$\prod_{\chi}\omega_f(\mathcal{C}_{15},\chi)=\phi(t+1,1).\phi(1,t).\phi(t,t+1)=e^{t+1}.\,e^{t+1}.\,e^{t+1}=e^{t+1}$$
$$\prod_{\chi}\omega_f(\mathcal{C}_{16},\chi)=\phi(t+1,t).\phi(t,0).\phi(0,t+1)=e^{t}.\,e^{0}.\,e^{1}=e^{t+1}$$

y entonces la función de partición de 3_1 y ϕ, elemento de $\mathbb{Z}[A]$, es

$$Z_{X,\phi}(3_1)=4e^{0}+12e^{t+1}$$

Probemos ahora que realmente esta definición es invariante.

Teorema 2.1. La función de partición $Z_{X,f}$ es un invariante de links orientados.

Demostración. Tenemos que demostrar que el producto de los pesos de Boltzmann es invariante bajo las versiones orientadas de los movimientos de Reidemeister.

Consideremos el primer movimiento de Reidemeister. Tal como muestra la parte superior de la Figura 1.21, hay dos orientaciones posibles para tener en cuenta.

En ambos casos, si suponemos que el arco superior tiene etiqueta $a_1 \in X$, entonces en el único cruce χ que tiene el diagrama tendremos el valor $f(a_1,a_1)^{\text{signo}(\chi)}$. Como f es un 2-cociclo, $f(a_1,a_1)=1$, así que como factor multiplicativo, aporta lo mismo que el arco recto sin cruces. Luego, el primer movimiento de Reidemeister deja invariante al producto de los pesos de Boltzmann.

Consideremos ahora el segundo movimiento. Aquí tenemos cuatro posibles diagramas orientados. Por ejemplo, en el de la figura 2.5, si etiquetamos el arco que pasa por arriba con $a_1 \in X$ y el arco entrante que pasa por debajo con $a_2 \in X$ entonces, como para

ambos diagramas tenemos un cruce positivo y uno negativo, el producto de los pesos de Boltzmann es $f(a_2, a_1)f(a_2, a_1)^{-1} = 1$. Luego, el segundo movimiento de Reidemeister también deja invariante al producto de los pesos de Boltzmann.

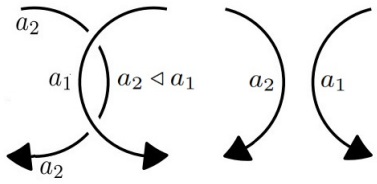


Figura 2.5: Configuración posible para el segundo movimiento de Reidemeister.

Para finalizar, tenemos que demostrar que el tercer movimiento de Reidemeister deja invariante al producto de los pesos de Boltzmann. Hay ocho casos para verificar, pero hagamos como ejemplo el caso que se corresponde con la figura 2.6 y dejemos el resto como ejercicio. Una cuenta detallada de estos casos, un poco más general (sirve para el capítulo siguiente también), se puede ver en [FG16].

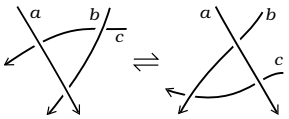


Figura 2.6: Otra de las configuraciones posibles para el tercer movimiento de Reidemeister.

Si calculamos el producto de los pesos de Boltzmann sobre los tres cruces que tienen los diagramas de la figura 2.6 vemos que este producto es invariante por el tercer movimiento de Reidemeister si y sólo si

$$f(a \triangleleft b, c)f(b, c)f(a, b) = f(b, c)f(a \triangleleft c, b \triangleleft c)f(a, c).$$

Como A es un grupo abeliano, al cancelar $f(b, c)$ en ambos miembros obtenemos el resultado deseado, debido a que f es un 2-cociclo. □

Observación 2.1. La ecuación anterior es otra posible motivación de la definición de 2-cociclo, de forma independiente de la teoría de extensiones:

> Se puede definir un 2-cociclo como una función tal que si se la utiliza como peso, entonces resulte invariante por los movimientos de Reidemeister.

Notablemente esta definición resulta equivalente al punto de vista de la construcción de extensiones.

Ejemplo 2.5. Vamos a dar otra prueba de que el nudo de la abuela no es equivalente al nudo cuadrado. Vamos a utilizar los mismos quandle X y 2-cociclo f que vimos cuando mostramos que no son equivalentes en el Ejemplo 2.3 (el quandle de conjugación y el

2-cociclo del Cuadro 2.1). El invariante dado por X y el 2-cociclo f para el nudo de la abuela nos da un elemento de $\mathbb{Z}[A]$ que es:

$$Z_{X,f}(3_1\#3_1) = 6 + 48\sigma + 96\sigma^2,$$

mientras que para el nudo cuadrado es:

$$Z_{X,f}(3_1\#m(3_1)) = 102 + 24\sigma + 24\sigma^3.$$

Esto nos muestra que el nudo de la abuela no es equivalente al nudo cuadrado.

Ejemplo 2.6. En este ejemplo vamos a distinguir el nudo 3_1 de su imagen especular $m(3_1)$, que podemos ver en la figura 2.7. Volvemos a considerar el quandle X y el 2-cociclo f de X que vimos en el ejemplo anterior. Computacionalmente, obtenemos que:

$$Z_{X,f}(3_1) = 6 + 24\sigma^3, \quad Z_{X,f}(m(3_1)) = 6 + 24\sigma.$$

Esto nos dice que los nudos 3_1 y $m(3_1)$ no son equivalentes.

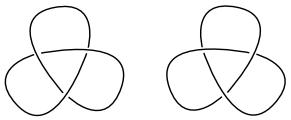


Figura 2.7: El nudo 3_1 (derecha) y su imagen especular $m(3_1)$ (izquierda).

CAPÍTULO 3

Generalizaciones

Nuestro siguiente paso es generalizar las ideas de invariantes de links por 2-cociclos de quandles para links enmarcados, usando racks y posteriormente biracks, generalizaciones de los quandles.

3.1. Nudos enmarcados

Definición 3.1. Un nudo **enmarcado** (framed knot) es un par (K, V) donde K es un nudo, y V es un campo vectorial nunca nulo, normal a K, llamado **enmarcación** (ver Figura 3.1). La longitud de los vectores es irrelevante (así que la asumimos uniforme).

Un nudo enmarcado puede verse como una cinta que fue anudada en el espacio, permitiendo hacer un número par de medios giros, y pegando finalmente los extremos (ver Figuras 3.2 y 3.3).

Observemos que el número de medios giros es par para asegurar la continuidad del campo vectorial.

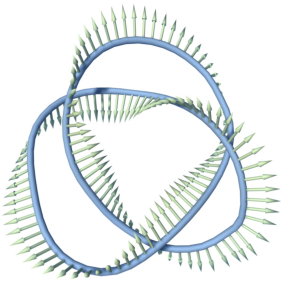


Figura 3.1: Nudo enmarcado

Figura 3.2: Cinta Anudada

Un **link enmarcado** es un link cuyas componentes son nudos enmarcados.

Definición 3.2. Dos links enmarcados son equivalentes si existe una isotopía entre ellos.

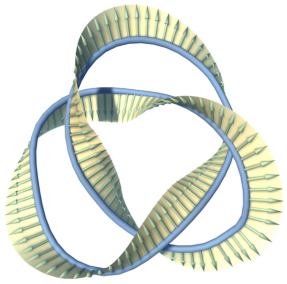


Figura 3.3: Equivalencia entre nudos enmarcados y cintas

Observación 3.1. Podemos plantear los movimientos de Reidemeister para links enmarcados, pero tendríamos que cambiar el primero, ya que hacer el movimiento $\mathcal{R}_1$ para una cinta, es girar la cinta 360 grados, como muestra la Figura 3.4. A estos giros, los llamamos **twists**, y una idea visual de estos aparece en la misma figura, como una parte de la estructura de hélice del ADN.

Entonces, vamos a considerar un movimiento $\mathcal{R}_1'$ que sea una equivalencia entre la cinta recta y la cinta con dos giros opuestos consecutivos, como vemos en la figura 3.5

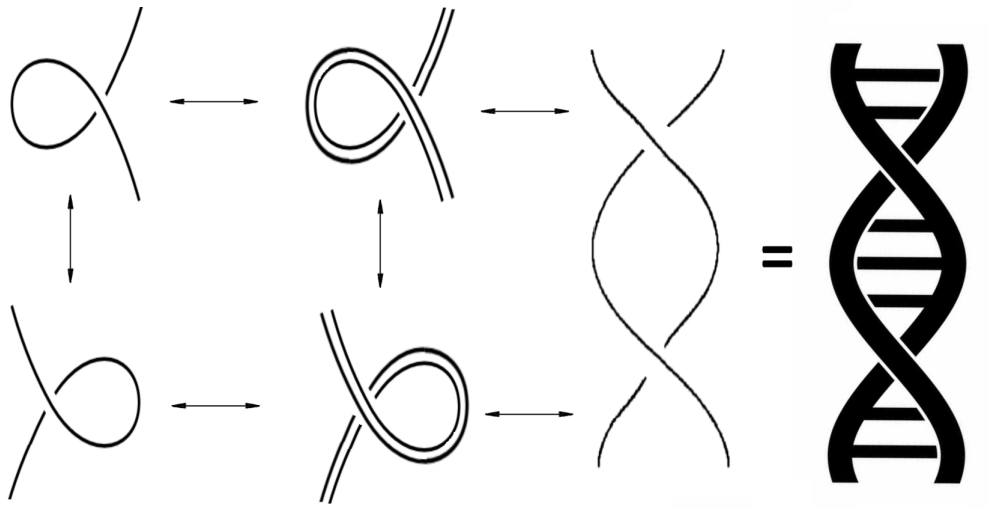


Figura 3.4: Equivalencias del twist en la cinta, y el twist, con sus diagramas planares (a izquierda)

Observación 3.2. Podemos además, mediante movimientos de tipo $\mathcal{R}_1$, llevar un diagrama dado de K (con posibles giros en la cinta) a un **diagrama planar**, donde podemos pensarlo como una cinta plasmada sobre una hoja, con estos twists.

Esto nos da una mejor idea de como es el diagrama general, ya que podemos agrupar todos estos twists como una sucesion de giros, de los cuales puede llegar a haber simplificaciones mediante el movimiento $\mathcal{R}_1'$.

Finalmente a la cantidad de twists restantes luego de simplificar, se la conoce como

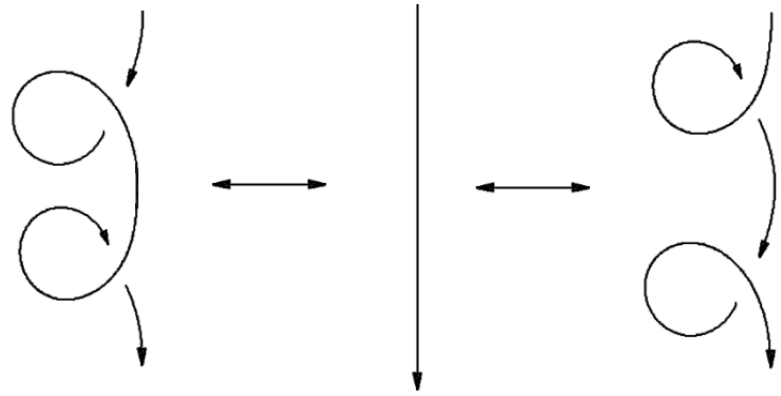


Figura 3.5: Nuevo movimiento $\mathcal{R}_1{}'$

número de twist del nudo enmarcado, es decir, la cantidad de giros de 360° que presenta su campo vectorial.

Ejemplo 3.1. En la figura 3.6 podemos ver un link enmarcado y su diagrama planar correspondiente

Figura 3.6: Un link enmarcado y su diagrama planar

3.2. Racks

Definición 3.3. Un **rack** es un par $(X, \triangleleft)$, donde X es un conjunto no vacío con una operación binaria $X \times X \to X$, $(x, y) \mapsto x \triangleleft y$, tal que

$$\varphi_x \colon X \to X,\ y \mapsto y \triangleleft x \text{ es biyectiva}, \qquad \text{para todo } x \in X, \tag{3.1}$$

$$(x \triangleleft y) \triangleleft z = (x \triangleleft z) \triangleleft (y \triangleleft z) \qquad \text{para todo } x, y, z \in X. \tag{3.2}$$

Observemos que un quandle es un rack que cumple $x \triangleleft x = x$ para todo $x \in X$.

Ejemplo 3.2. Un conjunto X con la biyección $\sigma : X \to X$ es un rack con la operación $x \triangleleft y = \sigma(x)$. Este se llama rack de acción constante.

Notar que si $\sigma \neq Id$, entonces X no será un quandle.

Ejemplo 3.3. Dados un grupo G y un elemento fijo $u \in G$, podemos darle estructura de rack con la operación $x \triangleleft y := y^{-1}uyx$.

Efectivamente, por un lado tenemos:

$$(x \triangleleft y) \triangleleft z = (yuy^{-1}x) \triangleleft z = zuz^{-1}yuy^{-1}x$$

y por otro tenemos que vale:

$$\begin{aligned}(x \triangleleft z) \triangleleft (y \triangleleft z) &= (zuz^{-1}x) \triangleleft (zuz^{-1}y) \\ &= (zuz^{-1}y)u(zuz^{-1}y)^{-1}(zuz^{-1}x) \\ &= (zuz^{-1}y)u(y^{-1}zu^{-1}z^{-1})(zuz^{-1}x) \\ &= zuz^{-1}yuy^{-1}x\end{aligned}$$

Además, es claro que este rack es quandle si y sólo si u es el neutro del grupo (y en este caso la operación es la identidad en x). También, podemos observar que si G es abeliano, nos queda un rack de permutación.

Ejemplo 3.4. Consideramos $\Lambda = \mathbb{Z}[t, t^{-1}, s]/\langle s^2 - (1-t)s\rangle$. Luego, cualquier M que sea un Λ-módulo, es un rack bajo la operación

$$x \triangleleft y = tx + sy$$

A este tipo de racks se los llama (t, s)-racks. Por ejemplo, podemos tomar $M = \mathbb{Z}_n$ y $t, s \in M$ tal que $\text{mcd}(n, t) = 1$ y $s^2 = (1-t)s$.

Observemos que esta construcción generaliza los quandles de Alexander, ya que estos son el caso en el que $s = (1-t)$.

Ejemplo 3.5. Un ejemplo donde un (t, s)-rack no es un quandle, es $X = \mathbb{Z}_4 = \{1, 2, 3, 4\}$ con $t = 1$ y $s = 4$. Esto ya que $x \triangleleft x = 3x \neq x$.

A continuación, veremos un resultado técnico que nos será de utilidad un poco más adelante.

Definición 3.4. Sean X e Y dos racks. Una función $f\colon X \to Y$ es un **morfismo** de racks si $f(x \triangleleft x') = f(x) \triangleleft f(x')$ para todo $x, x' \in X$.

Definición 3.5. Dado un rack X se define la función $\varphi : X \to X$ como $\varphi(x) = x \triangleleft x$.

Teorema 3.1. Sea X un rack, entonces

$$\begin{aligned}\varphi(x) \triangleleft y &= \varphi(x \triangleleft y) \\ x \triangleleft \varphi(y) &= x \triangleleft y\end{aligned}$$

en particular, φ es morfismo de racks.

Demostración. Para la primera igualdad tenemos:

$$\varphi(x) \triangleleft y = (x \triangleleft x) \triangleleft y = (x \triangleleft y) \triangleleft (x \triangleleft y) = \varphi(x \triangleleft y)$$

Para la segunda, como $- \triangleleft y$ es un morfismo de racks, $- \triangleleft^{-1} y$ también lo es, y luego

$$x \triangleleft \varphi(y) = x \triangleleft y$$

equivale a

$$(x \triangleleft \varphi(y)) \triangleleft^{-1} y = (x \triangleleft y) \triangleleft^{-1} y = x$$

Pero

$$(x \triangleleft \varphi(y)) \triangleleft^{-1} y = (x \triangleleft^{-1} y) \triangleleft (\varphi(y) \triangleleft^{-1} y) = (x \triangleleft^{-1} y) \triangleleft ((y \triangleleft y) \triangleleft^{-1} y) = (x \triangleleft^{-1} y) \triangleleft y = x$$

□

Pero además, φ pertenece a una familia de morfismos que tienen muy buenas propiedades, y eso nos permitirá concluir que es un automorfismo.

Empecemos definiendo esta familia:

Definición 3.6. Un morfismo de racks $f : X \to X$ que verifica

$$f(x \triangleleft y) = f(x) \triangleleft y$$

se llamará un **morfismo lineal a derecha**. En particular, si es automorfismo de racks, esto implica $x \triangleleft y = x \triangleleft f(y)$.

Proposición 3.1. Si X es un rack y $f : X \to X$ un automorfismo de racks que es lineal a derecha, entonces $(X, \triangleleft_f)$ con la operación

$$x \triangleleft_f y := f(x) \triangleleft y = f(x \triangleleft y) = f(x) \triangleleft f(y)$$

es un rack. Además, f resulta un automorfismo de racks lineal a derecha para esta nueva estructura.

Demostración.

$$(x \triangleleft_f y) \triangleleft_f z = (f(x) \triangleleft y) \triangleleft_f z = f(f(x) \triangleleft y) \triangleleft z = (f^2(x) \triangleleft y) \triangleleft z$$

A su vez,

$$(x \triangleleft_f z) \triangleleft_f (y \triangleleft_f z) = (f(x) \triangleleft z) \triangleleft_f (f(y) \triangleleft z) = (f^2(x) \triangleleft z) \triangleleft (f(y) \triangleleft z)$$

y usando la ecuación de rack

$$= (f^2(x) \triangleleft f(y)) \triangleleft z = (f^2(x) \triangleleft y) \triangleleft z$$

Esto muestra que es un rack. Para ver que f sea lineal a derecha con la estructura $\triangleleft_f$, usamos su definicion:

$$f(x \triangleleft_f y) = f(f(x) \triangleleft y) = f^2(x) \triangleleft y$$

y a su vez

$$f(x) \triangleleft_f y = f(f(x)) \triangleleft y$$

□

Lema 3.1. Si $(X, \triangleleft)$ es un rack, entonces $(X, \triangleleft^{-1})$ también.

Demostración. Es claro que $- \triangleleft^{-1} y$ es biyectiva, ahora de la ecuación de rack,

$$(x \triangleleft y) \triangleleft z = (x \triangleleft z) \triangleleft (y \triangleleft z)$$

vemos que $- \triangleleft z$ es un morfismo de racks, por lo tanto $- \triangleleft^{-1} z$ también:

$$(x \triangleleft y) \triangleleft^{-1} z = (x \triangleleft^{-1} z) \triangleleft (y \triangleleft^{-1} z)$$

y si hacemos el cambio de variable

$$(x, y, z) \longleftrightarrow (x', y, z) = (x \triangleleft y, y, z)$$

tenemos $x = x' \triangleleft^{-1} y$, y la ecuación anterior se escribe como

$$(x \triangleleft y) \triangleleft^{-1} z = (x \triangleleft^{-1} z) \triangleleft (y \triangleleft^{-1} z) \iff x' \triangleleft^{-1} z = ((x' \triangleleft^{-1} y) \triangleleft^{-1} z) \triangleleft (y \triangleleft^{-1} z)$$

Si aplicamos $\triangleleft^{-1}(y \triangleleft^{-1} z)$ en ambos lados de la igualdad, obtenemos

$$(x' \triangleleft^{-1} z) \triangleleft^{-1} (y \triangleleft^{-1} z) = (x' \triangleleft^{-1} y) \triangleleft^{-1} z$$

□

Observación 3.3. Si $f : X \to X$ es un morfismo de rack para $\triangleleft$, lo es para $\triangleleft^{-1}$ ya que,

$$f(x) = f((x \triangleleft^{-1} y) \triangleleft y) = f(x \triangleleft^{-1} y) \triangleleft f(y)$$

y aplicando $\triangleleft^{-1} f(y)$ en ambos miembros obtenemos

$$f(x) \triangleleft^{-1} f(y) = f(x \triangleleft^{-1} y)$$

Similarmente, si f además es lineal a derecha, entonces

$$f(x) = f((x \triangleleft^{-1} y \triangleleft y) = f(x \triangleleft^{-1} y)$$

y por lo tanto

$$f(x) \triangleleft^{-1} y = f(x \triangleleft^{-1} y)$$

Es decir, f es lineal a derecha para la estructura $\triangleleft^{-1}$. Lo mismo ocurre si es lineal a izquierda.

Corolario: La función $i : X \to X$ dada por $i(x) := x \triangleleft^{-1} x$ es un morfismo de racks que es lineal a derecha y que es inverso de $\varphi(x) = x \triangleleft x$. En consecuencia, tanto i como φ son automorfismos.

Demostración. Llamemos $i_\triangleleft$ para indicar la dependecia de la estructura de rack. Notamos que $i_\triangleleft = \varphi_{\triangleleft^{-1}}$, por lo tanto $i_\triangleleft$ es morfismo lineal a derecha tanto para $\triangleleft$ como para $\triangleleft^{-1}$.

Ahora calculamos

$$i\varphi(x) = i(x \triangleleft x) = i(x) \triangleleft x = (x \triangleleft^{-1} x) \triangleleft x = x$$

Análogamente

$$\varphi i(x) = \varphi(x \triangleleft^{-1} x) = \varphi(x) \triangleleft^{-1} x = x \triangleleft x \triangleleft^{-1} x = x$$

□

Observación 3.4. Podemos desarrollar con los racks, una teoría análoga a quandles aplicados a links, pero ahora para links enmarcados.

Definición 3.7. Una **coloración** del diagrama de un link enmarcado orientado K con el rack X es una asignación $\mathcal{C}$: {Semi-arcos de K} $\to X$ que cumple en cada cruce la relación $a_j \triangleleft a_i = a_k$, para cada cruce negativo o positivo como muestra la Figura 1.18. Notamos $\mathrm{RCol}_X(K)$ a la cantidad de coloraciones de K por X.

Teorema 3.2. $\mathrm{RCol}_X(K)$ es invariante de links enmarcados orientados.

Demostración. Para demostrar esto, basta chequearlo para el movimiento $\mathcal{R}_1$', ya que los demás, son iguales al caso de quandle.

Hagámoslo para el primer caso de la Figura 3.5, el otro es análogo. Consideramos etiquetas en cada arco, y usando la relación de rack que se impone en cada cruce, tenemos un etiquetado como el de la Figura 3.7. Esto ya que el cruce superior me indica que el arco del centro es $a \triangleleft^{-1} a = i(a)$. Y en el siguiente cruce, este arco central me brinda la información de que $b = i(a) \triangleleft b$, y por ser un automorfismo lineal a derecha, vale que $b = i(a \triangleleft b)$ y aplicándole la inversa de i obtenemos que $b \triangleleft b = \varphi(i(a \triangleleft b)) = a \triangleleft b$. Finalmente, usamos que $- \triangleleft z : X \to X, x \mapsto x \triangleleft z$ es biyectivo para concluir que $a = b$.

□

Definición 3.8. Dados A un grupo abeliano (escrito multiplicativamente) y X un rack, una función $f : X \times X \to A$ que cumple

$$f(x,z)f(x \triangleleft z, y \triangleleft z) = f(x,y)f(x \triangleleft y, z) \quad \text{para todo } x,y,z \in X.$$

se llama 2**-cociclo del rack** X con coeficientes en A.

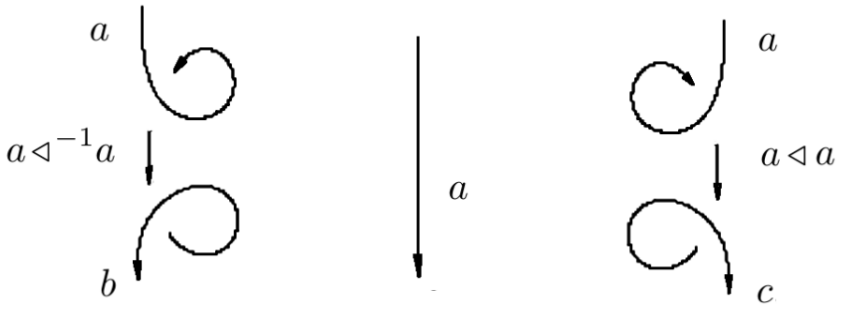


Figura 3.7: Invarianza por el $\mathcal{R}_1$'

Definición 3.9. Fijemos un grupo abeliano A (escrito multiplicativamente) y un nudo enmarcado orientado K. Sean X un rack finito, $\mathcal{C}\colon R(K) \to X$ una coloración de K y $f\colon X \times X \to A$ un 2-cociclo de X con coeficientes en A. En cada cruce como el que vemos en la figura 2.2 (igual que el caso quandle) se define el **peso de Boltzmann** $R\omega_f(\mathcal{C}, \chi)$ (con respecto a la coloración $\mathcal{C}$, al 2-cociclo f y al cruce χ) como el elemento de A dado por la expresión

$$R\omega_f(\mathcal{C}, \chi) = \begin{cases} f(a_j, a_i)^{-1} & \text{si el cruce es negativo} \\ f(b_j, b_i) & \text{si el cruce es positivo} \end{cases}$$

Definición 3.10. La **función de partición** $RZ_{X,f}(K)$ del link enmarcado orientado K (asociada al rack X y al 2-cociclo f) es la expresión

$$RZ_{X,f}(K) = \sum_{\mathcal{C}} \prod_{\chi} R\omega_f(\mathcal{C}, \chi), \tag{3.3}$$

donde el producto se toma sobre todos los cruces χ que tiene el diagrama del link enmarcado K y la suma se toma sobre todos las coloraciones $\mathcal{C}$ de K dadas por el rack X. La fórmula (3.3) define un elemento de $\mathbb{Z}[A]$, el anillo de grupo de A.

Teorema 3.3. La función de partición $RZ_{X,f}$ es un invariante de link enmarcados orientados.

La demostración es análoga al caso de quandles, solo que esta vez tenemos $\mathcal{R}_1$'. Pero este diagrama consiste en dos giros opuestos consecutivos, e igual que en el caso de $\mathcal{R}_2$, esto da pesos de Boltzmann inversos, que se cancelan entre sí.

3.3. Ecuación de trenzas y biracks

La **ecuación de trenzas** (ver figura 3.8), es una condición de composición de operaciones de conjuntos, que gráficamente es similar a nuestro movimiento de Reidemeister $\mathcal{R}_3$. Además, tiene una estrecha relación con la llamada **ecuación de Yang-Baxter**, que proviene de la física. Esta última no es la misma ecuación, pero se puede demostrar que

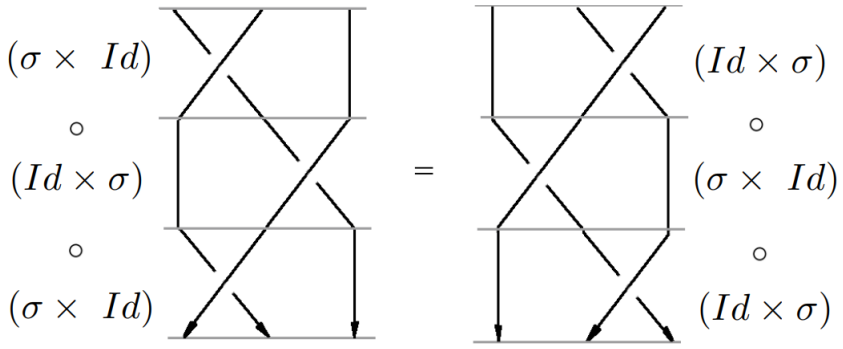


Figura 3.8: Ecuación de trenzas

es equivalente encontrar soluciones para cualquiera de las dos ecuaciones. Para más información, se puede consultar [ESS99].

Por esta razón, nos referiremos a la ecuación de trenzas y a la ecuación de Yang-Baxter indistintamente.

Definición 3.11. Una **solución conjuntista de la ecuación de Yang–Baxter** es un par (X, σ) donde X es un conjunto y $\sigma : X \times X \to X \times X$ es una función biyectiva que satisface:

$$(Id \times \sigma)(\sigma \times Id)(Id \times \sigma) = (\sigma \times Id)(Id \times \sigma)(\sigma \times Id)$$

Notamos $\sigma(x, y) = (\sigma_1(x, y), \sigma_2(x, y))$.

Definición 3.12. Una solución (X, σ) se dice **birack** si satisface:

1. Para todo $x, z \in X$ existe un único $y \in X$ tal que $\sigma_1(x, y) = z$
2. Para todo $y, t \in X$ existe un único $x \in X$ tal que $\sigma_2(x, y) = t$

A estas dos condiciones las llamamos condiciones de **no degeneración**. Observemos que pedir la no degeneración es equivalente a pedir que se cumpla:

- Existe una única función inversible $S : X \times X \to X \times X$ que cumple

$$S(x, \sigma_1(x, y)) = (y, \sigma_2(x, y)) \quad \forall x, y \in X \tag{3.4}$$

Por comodidad en algunas cuentas, usaremos también $\sigma_1(x, y) = g_x(y)$ y $\sigma_2(x, y) = f_y(x)$, para enfatizar que se trata de funciones biyectivas.

Definición 3.13. Un birack se dice **biquandle** si dado $x_0 \in X$ existe un único $y_0 \in X$ tal que $\sigma(x_0, y_0) = (x_0, y_0)$. Es decir, si existe una función biyectiva $s : X \to X$ tal que

$$\{(x, y) \in X \times X \ : \ \sigma(x, y) = (x, y)\} = \{(x, s(x)) \in X \times X \ : \ x \in X\}$$

También observamos que los biquandles y biracks generalizan a los quandles y los racks ya que $\sigma : X \times X \to X \times X$ de la forma $\sigma(x, y) = (y, f(x, y))$ es un birack si y solo si $x \triangleleft y := f(x, y)$ brinda a X estructura de rack; y es biquandle si y solo si, brinda estructura de quandle.

Definición 3.14. Un birack (X, σ) se dice **diagonalmente biyectivo** si la función S definida en la ecuación 3.4 y su inversa S^{-1}, cumplen que las composiciones $S_1^{\pm 1} \circ \Delta$ y $S_2^{\pm 1} \circ \Delta$ son biyectivas, donde $\Delta : X \to X \times X$ es la función diagonal, dada por $\Delta(x) = (x, x)$.

Veremos más adelante que si X es finito, la propiedad de ser diagonalmente biyectivo se sigue de la definición de birack. Pero primero, veamos algunos ejemplos:

Ejemplo 3.6. Dado X un conjunto arbitrario con la operación $\sigma(x, y) = (y, x)$, obtenemos un birack llamado Flip. Además es un biquandle con $s = Id$.

Ejemplo 3.7. Sea $X = \{x, y\}$ con la operación σ tal que

$$\sigma(x, x) = (x, x)\ , \quad \sigma(x, y) = (y, x)\ , \quad \sigma(y, x) = (y, x)\ , \quad \sigma(y, y) = (y, y)$$

Este es un birack (un biquandle de hecho), que no es de la forma $\sigma(x, y) = (y, x \triangleleft y)$ para $\triangleleft$ una estructura de rack.

Ejemplo 3.8. Sea M un $k[r^{\pm 1}, t^{\pm 1}]$-módulo, entonces

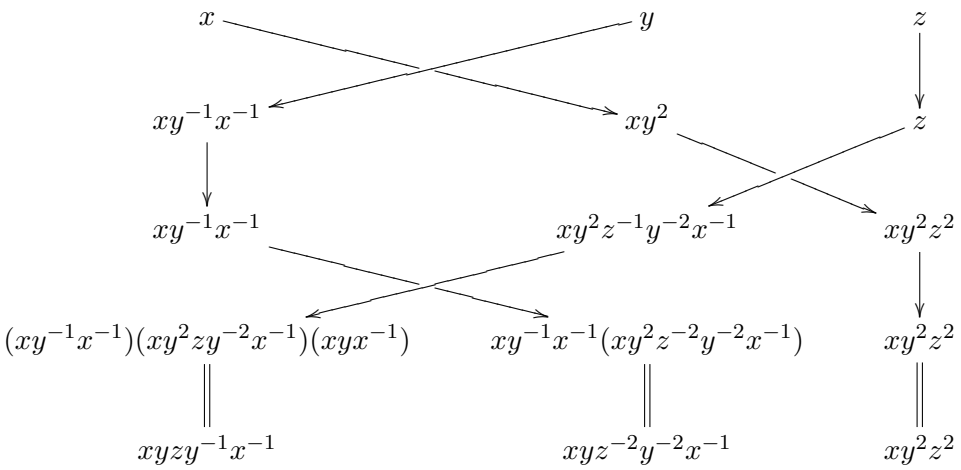
$$\sigma(x, y) = \big(rx, tx + (1 - rt)y\big)$$

es un birack, llamado bialexander. Además es un biquandle con la función $s(x) = r^{-1}x$, y generaliza el quandle de Alexander, que se obtiene tomando $r = 1$.

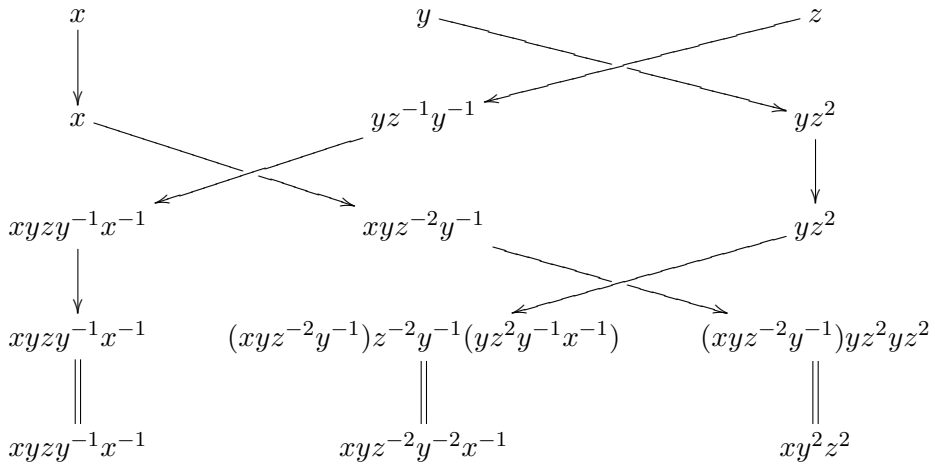
Ejemplo 3.9. Si G es un grupo, el birack de Wada de G es el dado por la operación:

$$\sigma(x, y) = (xy^{-1}x^{-1}, xy^2)$$

De hecho, este birack es un biquandle con $s(x) = x^{-1}$. La condición de no degeneración es evidente, y que satisface Y-B se sigue de los siguientes diagramas:



y por el otro lado:



Notar que en caso de que G sea abeliano, da un bialexander con $r = -1$ y $t = 1$

Como en casos anteriores con quandles y racks, queremos etiquetar a los diagramas con esta nueva estructura de birack. Tenemos una función que toma pares de X y devuelve pares de X, así que en este caso vamos a poder asignar elementos a cada semi-arco que se forma con un cruce del diagrama (separando la linea contigua superior en dos semi-arcos).

Definición 3.15. Dado X un birack y un diagrama planar de un link enmarcado orientado K, definimos una **coloración** de K por X como una asignación $\mathcal{C} : \{\text{Semi-arcos de K}\} \to X$ que cumple en cada cruce las condiciones de la Figura 3.9

Notamos $Col_X(K)$ a la cantidad de coloraciones de K por el birack X.

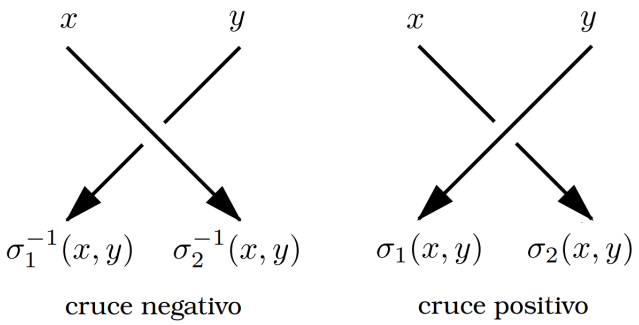


Figura 3.9: Regla de coloreos por biracks

Observación 3.5. La condición de no degeneración del birack, nos brinda una muy útil interpretación gráfica a la hora de pintar los arcos de un cruce, como se puede ver en las siguientes figuras:

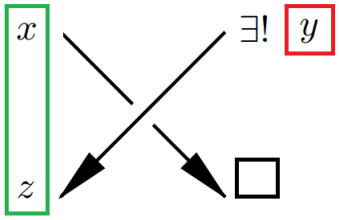


Figura 3.10: Condición 1

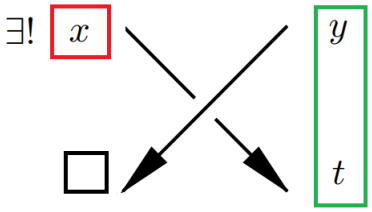


Figura 3.11: Condición 2

- La Figura 3.10 se podria interpretar en palabras como: z es el resultado de pasar con el semiarco y, sobre x; es decir, aplicarle la función "pasar por arriba de x" a y. Precisamente, $z = g_x(y)$ con lo cual $g_x^{-1}(z) = y$ por ser biyección esta función. Finalmente completamos el recuadro faltante con $t = f_{g_x^{-1}(z)}(x)$.
- Análogamente, la Figura 3.11 sería: t es el resultado de pasar x por abajo de y; es decir, aplicarle la función "pasar por debajo de y a x". Esto es $t = f_y(x)$, o sea $x = f_y^{-1}(t)$. Finalmente completamos el recuadro faltante con $z = g_{f_y^{-1}(t)}(y)$.

Observación 3.6. La propiedad de que un birack sea diagonalmente biyectivo, también se puede interpretar gráficamente pintando los semiarcos de un cruce, como se puede ver en las siguientes figuras:

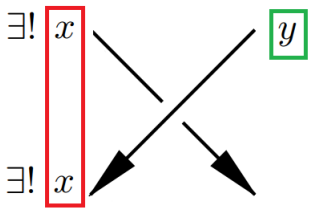


Figura 3.12: Dado y

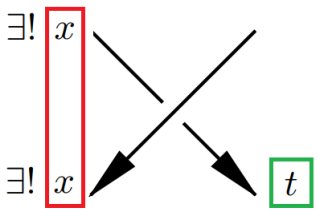


Figura 3.13: Dado t

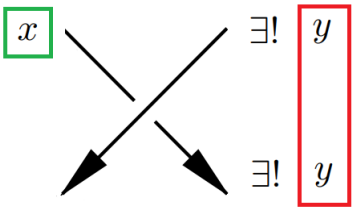


Figura 3.14: Dado x

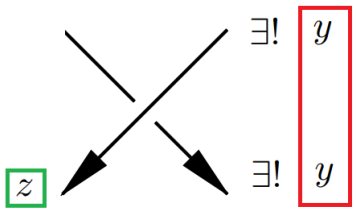


Figura 3.15: Dado z

- La Figura 3.12 nos muestra que $(S_1 \circ \Delta)(x) = y$, con lo cual $x = (S_1 \circ \Delta)^{-1}(y)$ por la biyección diagonal.

- La Figura 3.13 nos muestra que $(S_2 \circ \Delta)(x) = t$, con lo cual $x = (S_2 \circ \Delta)^{-1}(t)$.
- La Figura 3.14 nos muestra que $(S_1^{-1} \circ \Delta)(y) = x$, con lo cual $y = (S_1^{-1} \circ \Delta)^{-1}(x)$.
- La Figura 3.15 nos muestra que $(S_2^{-1} \circ \Delta)(y) = z$, con lo cual $y = (S_2^{-1} \circ \Delta)^{-1}(z)$.

Teorema 3.4. Dado X un birack finito, $Col_X(K)$ es un invariante de links enmarcados orientados.

Demostración. Como X es solución de la ecuación de trenzas, colorear los arcos es por definición invariante por $\mathcal{R}_3$.

Consideremos los diagramas de $\mathcal{R}_2$ que vemos en la Figura 3.16. Para ambos, usamos solamente la definición de coloreo junto con la biyectividad de σ.

Para las orientaciones de la Figura 3.17 usamos la condición de ser no degenerado. En el diagrama de la izquierda, conseguimos ver que $t = x$ y $z = f_y(x)$. En el diagrama de la derecha, conseguimos $t = g_x(y)$ y $z = x$.

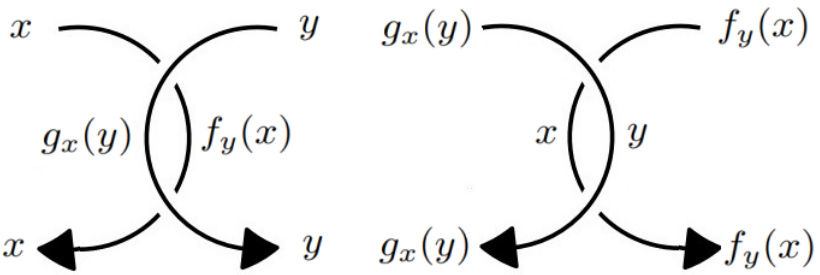


Figura 3.16

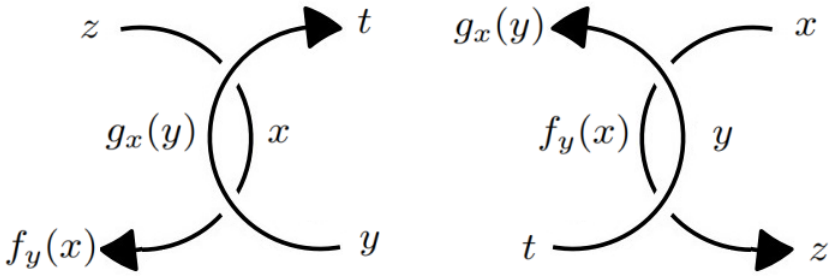


Figura 3.17

Finalmente, para ver que es invariante por $\mathcal{R}_1$', usaremos que X finito es diagonalmente biyectivo (lo demostraremos en la siguiente sección). Consideremos los posibles diagramas:

- Comenzando por arriba en el diagrama de la Figura 3.18, tenemos un y arriba a la derecha, y usando la biyección $(S_1 \circ \Delta)^{-1}$, obtenemos quién es la etiqueta del

twist. Para el siguiente semiarco, tenemos la etiqueta dada por $\varphi(y)$, donde $\varphi = S_2 \circ \Delta \circ (S_1 \circ \Delta)^{-1}$. Usando ahora la biyección diagonal (para cruces negativos ahora), obtenemos que $w = x$, y de eso podemos concluir que $t = y$.

- En la Figura 3.19, tenemos la misma situación, pero con los cruces invertidos, con lo cual, tenemos que $z = \varphi^{-1}(y)$.
- Con respecto a la Figura 3.20, empezamos con y arriba a la izquierda. Por la biyección $(S_1^{-1} \circ \Delta)^{-1}$, obtenemos la etiqueta del primer twist. De ahi, obtenemos el siguiente semiarco con $z = \pi(y)$, donde $\pi = S_2^{-1} \circ \Delta \circ (S_1^{-1} \circ \Delta)^{-1}$. Por biyección diagonal, obtenemos que $w = x$ y concluyo que $t = y$.
- En la Figura 3.21, tenemos nuevamente la figura anterior, pero con los cruces invertidos, así que la única diferencia es que $z = \pi^{-1}(y)$.

Con todo esto concluimos que si arrancamos con y, terminamos con y, pasando por semiarcos unívocamente determinados. Tenemos entonces que la cantidad de coloreos es invariante por $\mathcal{R}_1$'. □

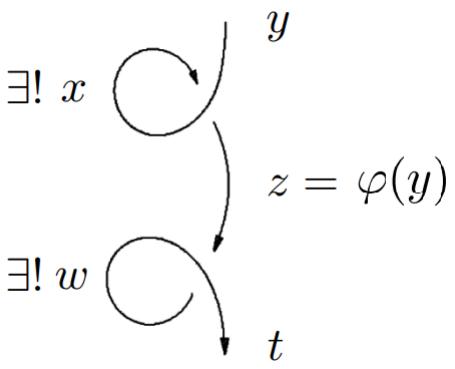


Figura 3.18

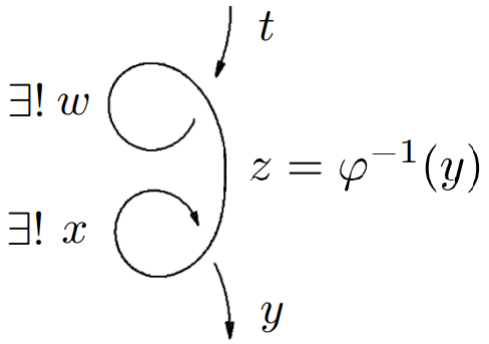


Figura 3.19

Observación 3.7. En la demostración anterior, mostramos que el semiarco intermedio del movimiento $\mathcal{R}_1$', lo podemos etiquetar con $z = \varphi(y)$, con $z = \pi(y)$, y con sus inversas. Podemos incluso probar que estas funciones son iguales:

Nos enfocamos en el diagrama de la figura 3.22, comenzando desde arriba, etiquetando por y, pasamos por el twist derecho positivo aplicando π y luego por el twist izquierdo negativo aplicando φ^{-1}.

Pero lo que podemos hacer, es usar el movimiento $\mathcal{R}_2$ para pasar el semiarco superior del twist izquierdo por encima del twist derecho, y así obtener un diagrama que podemos llevar a una linea recta mediante $\mathcal{R}_3$.

Con lo cual concluimos que $\varphi = \pi$.

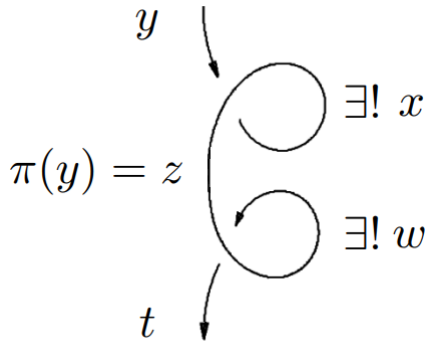


Figura 3.20

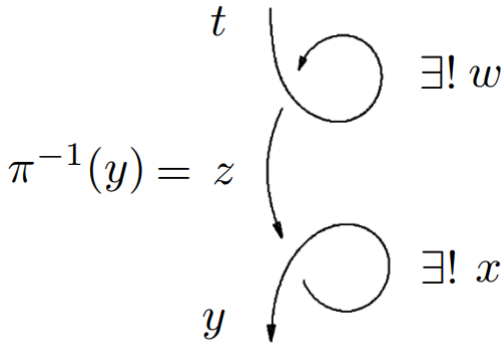


Figura 3.21

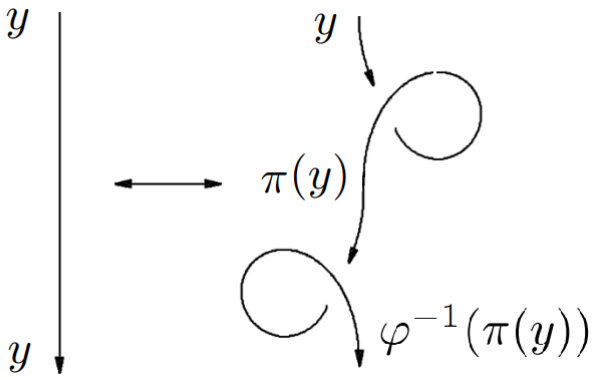


Figura 3.22: Podemos llevar un diagrama a otro con $\mathcal{R}_2$ y $\mathcal{R}_3$

Definición 3.16. Dado X un birack, podemos construir su **rack asociado**: Tenemos una biyección dada por $\Phi : X \times X \to X \times X$ donde $\Phi(x, y) = (x, g_x(y))$, y se la aplicamos al diagrama de coloreo por birack, como muestra la Figura 3.23. Luego obtenemos en la parte inferior:

$$\Phi(g_x(y), fy(x)) = (g_x(y), g_{g_x(y)}(f_y(x)))$$

Notemos en el diagrama, que una vez aplicada Φ, las etiquetas de los extremos del arco continuo son ambas $g_x(y)$, lo que nos lleva a querer tener una operación $x \triangleleft g_x(y)$ en el recuadro faltante, que cumpla la condición de rack.

Pero si queremos que el diagrama conmute, necesitamos que

$$x \triangleleft g_x(y) = g_{g_x(y)}(f_y(x))$$

Para ello observamos que

$$g_{g_x(y)}(f_y(x)) = g_{g_x(y)}(f_{g_x^{-1}}(g_x(y))(x))$$

Entonces, si definimos en X la estructura $(X, \triangleleft_\sigma)$ con $u \triangleleft_\sigma v = g_v(f_{g_u^{-1}}(v(u))$, entonces el diagrama entre estructuras mediante la biyección Φ conmuta.

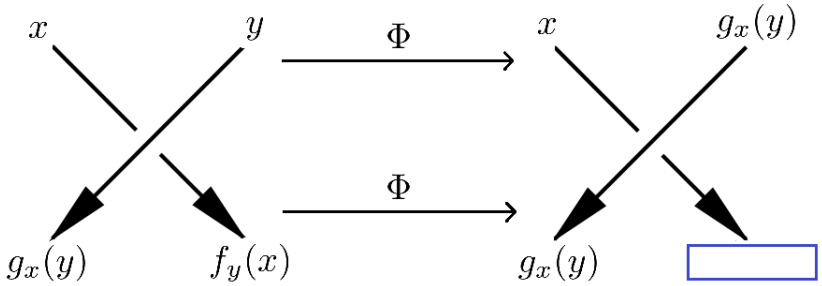


Figura 3.23: Construcción del rack asociado

Notar que $\triangleleft_\sigma$ depende de f_y y g_x, y por ello a $(X, \triangleleft_\sigma)$ lo llamamos el rack asociado de (X, σ). Dejamos como ejercicio corroborar que efectivamente $\triangleleft_\sigma$ cumple las propiedades de rack.

Ejemplo 3.10. Tomemos el biquandle de Wada, y encontremos su rack asociado. La biyección la podemos ver en la Figura 3.24, y queremos encontrar la operación $\triangleleft_\sigma$.

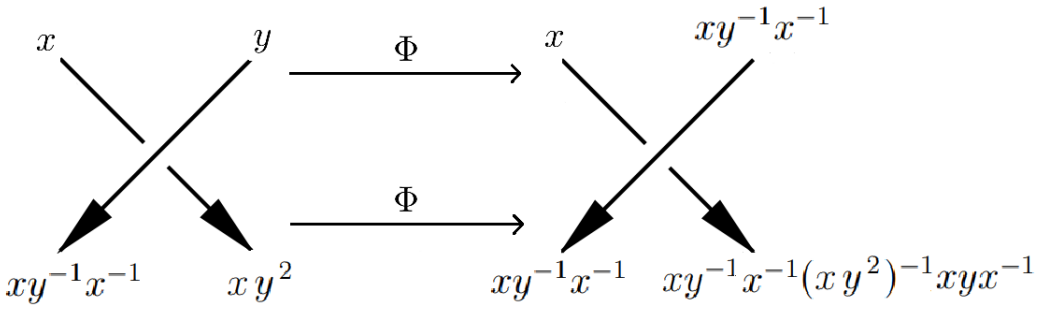


Figura 3.24: Rack asociado al biquandle de Wada

Tenemos la siguiente escritura:

$$xy^{-1}x^{-1}(xy^2)^{-1}xyx^{-1} = xy^{-1}x^{-1}y^{-1}x^{-1} = (xy^{-1}x^{-1})x^{-1}(xy^{-1}x^{-1})$$

Con lo cual podemos pensar a esta operación como $a \triangleleft_\sigma b = ba^{-1}b$.

3.4. Diagonalidad biyectiva

El concepto de la diagonalidad biyectiva, lo introdujo S. Nelson en [Nel14] como condición extra del birack, para poder colorear correctamente los links enmarcados. Luego, M. Farinati probó en [Far24] que para el caso finito, todo birack cumple esta propiedad, pasando por una descripción universal del birack a partir de las operaciones que lo componen. Desarrollaremos esa idea en esta sección.

No degeneración a izquierda

Sea $\sigma(x,y) = (g_x y, f_y x)$ una solución no degenerada a izquierda de la ecuación de Y-B, notamos

$$x * y := g_x y$$

La condición de no degeneración a izquierda dice que, fijado x, la fórmula $x * y$ depende biyectivamente de y. Notamos $\cdot$ la operación inversa, o sea,

$$x \cdot z = y \iff x * y = z$$

Recordemos que tenemos la operación del rack asociado $x \triangleleft g_x y = g_{g_x y}(f_y x)$, con lo cual

$$x \triangleleft (x * y) = g_{x*y}(f_y x) = (x * y) * (f_y x)$$

o equivalentemente

$$f_y x = (x * y) \cdot (x \triangleleft (x * y))$$

Con esto concluimos que

$$\sigma(x,y) = (g_x y, f_y x) = \Big(x * y, (x * y) \cdot \big(x \triangleleft (x * y)\big)\Big)$$

Teorema 3.5. Sea $\triangleleft$ operación binaria y sea $x \cdot -$ una biyección, que satisfacen:

1. La identidad dada por:

$$(y \cdot (x \triangleleft y)) \cdot (y \cdot z) = (x \cdot y) \cdot (x \cdot z) \qquad \forall x, y, z \in X$$

2. $x \cdot -$ es un morfismo con respecto a la operación $\triangleleft$, es decir:

$$x \cdot (y \triangleleft z) = (x \cdot y) \triangleleft (x \cdot z) \qquad \forall x, y, z \in X$$

3. $(X, \triangleleft)$ es autodistributiva:

$$((x \triangleleft y) \triangleleft z) = (x \triangleleft z) \triangleleft (y \triangleleft z) \qquad \forall x, y, z \in X$$

Entonces,

$$\sigma(x,y) := \Big(x * y, (x * y) \cdot \big(x \triangleleft (x * y)\big)\Big)$$

es una solución no degenerada a izquierda de la ecuación de Y-B, donde $x * - = (x \cdot -)^{-1}$.

Reciprocamente, si tenemos $\sigma(x,y) = (g_x y, f_y x)$ solución de la ecuación de Y-B con $x * - = g_x(-)$ una biyección y $x \cdot - = g_x^{-1}(-)$ su inversa; y notamos $x \triangleleft y := g_y(f_{g_x^{-1} y} x) = y * (f_{x \cdot y} x)$, entonces las operaciones $\cdot, *, \triangleleft$ cumplen las condiciones 1, 2 y 3 enunciadas arriba.

Demostración. Empezamos con $(X, \triangleleft, \cdot)$ donde $\cdot$ es una operación arbitraria que satisface la condición de no degeneración a izquierda, $- \triangleleft x$ y $x \cdot -$ son biyecciones de X; y notamos $x * -$ a la inversa de $x \cdot -$. Consideramos la función $\sigma : X \times X \to X \times X$ definida como

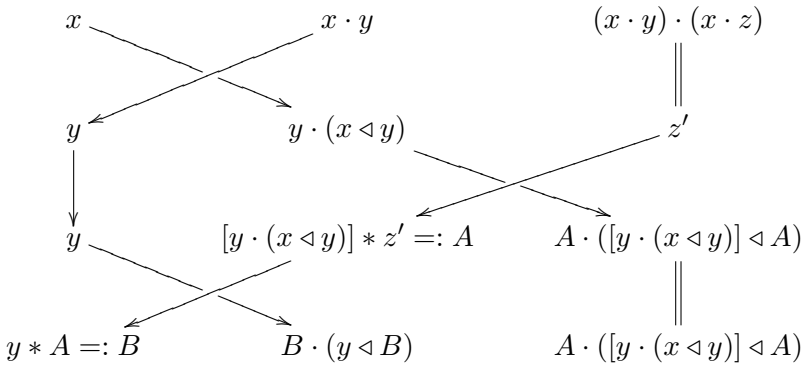
$$\sigma(x, y) = \Big(x * y \ , \ (x * y) \cdot (x \triangleleft (x * y)) \Big)$$

Vamos a probar que σ es solución de la ecuación de Y-B si y sólo si cumple 1, 2 y 3. En vez de chequear Y-B con (x, y, z), lo haremos con elementos de la forma

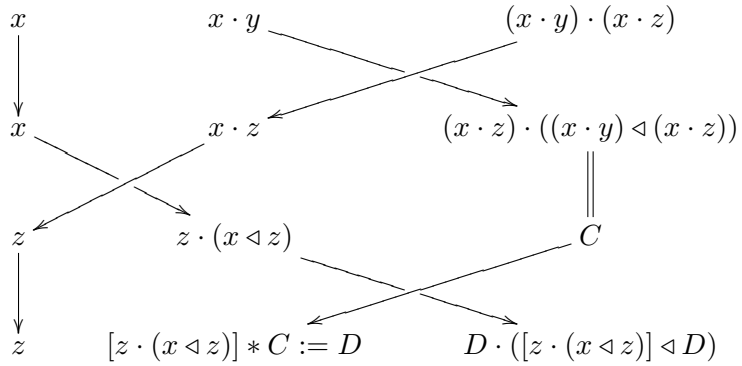
$$(x', y', z') := \big(x, x \cdot y, (x \cdot y) \cdot (x \cdot z) \big)$$

Y como la función $(x, y, z) \mapsto \big(x, x \cdot y, (x \cdot y) \cdot (x \cdot z) \big)$ es biyectiva, esto será equivalente.

Notemos LHS el resultado en $X \times X \times X$ de esta parte del diagrama:



Y notemos RHS el resultado de esta otra parte:



La primer condición (extremos izquierdos) es:

$$B = y * A = y * ([y \cdot (x \triangleleft y)] * z') \stackrel{?}{=} z$$

que es equivalente a decir

$$z' = (y \cdot (x \triangleleft y)) \cdot (y \cdot z)$$

Y como $z' = (x \cdot y) \cdot (x \cdot z)$, entonces tenemos la condición 1:

$$(x \cdot y) \cdot (x \cdot z) = (y \cdot (x \triangleleft y)) \cdot (y \cdot z)$$

Los términos del medio dan

$$B \cdot (y \triangleleft B) = [z \cdot (x \triangleleft z)] * C$$

Como $B = z$ y $C = (x \cdot z) \cdot ((x \cdot y) \triangleleft (x \cdot z))$ entonces

$$z \cdot (y \triangleleft z) = [z \cdot (x \triangleleft z)] * [(x \cdot z) \cdot ((x \cdot y) \triangleleft (x \cdot z))]$$

o equivalentemente

$$[z \cdot (x \triangleleft z)] \cdot [z \cdot (y \triangleleft z)] = (x \cdot z) \cdot ((x \cdot y) \triangleleft (x \cdot z))$$

Pero usando

$$(b \cdot a) \cdot (b \cdot c) = (a \cdot (b \triangleleft a)) \cdot (a \cdot c)$$

con $a = z$, $b = x$ y $c = (y \triangleleft z)$ tenemos que la segunda condición es equivalente a

$$(x \cdot z) \cdot (x \cdot (y \triangleleft z)) = (x \cdot z) \cdot ((x \cdot y) \triangleleft (x \cdot z))$$

y cancelando $(x \cdot z)$ obtenemos la identidad 2:

$$x \cdot (y \triangleleft z) = (x \cdot y) \triangleleft (x \cdot z)$$

Finalmente, la tercera condición (la del extremo derecho) es:

$$A \cdot ([y \cdot (x \triangleleft y)] \triangleleft A) = D \cdot ([z \cdot (x \triangleleft z)] \triangleleft D)$$

Recordando que

$$A = [y \cdot (x \triangleleft y)] * z' = [y \cdot (x \triangleleft y)] * [(x \cdot y) \cdot (x \cdot z)]$$

y usando la primer condición, podemos remplazar $(x \cdot y) \cdot (x \cdot z)$ para obtener

$$A = [y \cdot (x \triangleleft y)] * [(y \cdot (x \triangleleft y)) \cdot (y \cdot z)] = y \cdot z$$

Luego, la tercera condición de LHS

$$(y \cdot z) \cdot ([y \cdot (x \triangleleft y)] \triangleleft (y \cdot z)) = (y \cdot z) \cdot (y \cdot [(x \triangleleft y) \triangleleft z])$$

donde usamos que $y \cdot (-)$ es un morfismo para la operación $\triangleleft$. Para RHS tenemos que $D = [z \cdot (x \triangleleft z)] * C$, donde

$$C = (x \cdot z) \cdot ((x \cdot y) \triangleleft (x \cdot z)) = (x \cdot z) \cdot (x \cdot (y \triangleleft z)) = (z \cdot (x \triangleleft z)) \cdot (z \cdot (y \triangleleft z))$$

Luego $D = [z \cdot (x \triangleleft z)] * C = z \cdot (y \triangleleft z)$. Con lo cual,

$$RHS = D \cdot ([z \cdot (x \triangleleft z)] \triangleleft D)$$

$$= (z \cdot (y \triangleleft z)) \cdot ([z \cdot (x \triangleleft z)] \triangleleft (z \cdot (y \triangleleft z)))$$

$$= (z \cdot (y \triangleleft z)) \cdot (z \cdot [(x \triangleleft z) \triangleleft (y \triangleleft z)])$$

y usando nuevamente que

$$(b \cdot a) \cdot (b \cdot c) = (a \cdot (b \triangleleft a)) \cdot (a \cdot c)$$

para $a = z$, $b = y$ y $c = [(x \triangleleft z) \triangleleft (y \triangleleft z)]$ obtenemos

$$RHS = (y \cdot z) \cdot (y \cdot [(x \triangleleft z) \triangleleft (y \triangleleft z)])$$

Así que concluimos que $LHS = RHS$ si y sólo si

$$(y \cdot z) \cdot (y \cdot [(x \triangleleft y) \triangleleft z]) = (y \cdot z) \cdot (y \cdot [(x \triangleleft z) \triangleleft (y \triangleleft z)])$$

lo que es equivalente a pedir

$$(x \triangleleft y) \triangleleft z = (x \triangleleft z) \triangleleft (y \triangleleft z)$$

□

No degeneración a derecha

Recordemos que, si $(X, \triangleleft)$ un rack y $\cdot$ es una operación binaria tal que satisface que $(\forall x \in X)\ x \cdot -$ es un automorfismo de $(X, \triangleleft)$ y $(y \cdot (x \triangleleft y)) \cdot (y \cdot z) = (x \cdot y) \cdot (x \cdot z)\ \forall x, y, z \in X$, entonces

$$\sigma(x, y) := \Big(x * y, (x * y) \cdot (x \triangleleft (x * y)\Big)$$

es una solución no degenerada a izquierda de la ecuación de Y-B, donde $x * - = (x \cdot -)^{-1}$.

Pero esto no nos dice que la solución sea no degenerada a derecha.

Lema 3.2. Sea $\sigma : X \times X \to X \times X$ una solución biyectiva no degenerada a izquierda de la ecuación de Y-B, dada por $\sigma(x, y) = (g_x y, f_y x)$, entonces $(X, \triangleleft)$ es un rack, donde $\triangleleft$ es la operación descripta en el Teorema 3.5.

Demostración. Sean $x, y \in X$, notamos $\sigma^{-1}(y, y \cdot x) =: (a, b)$. Como $a * -$ es una biyección, podemos escribir $b = a \cdot c$ para algún c y obtenemos

$$(y, y \cdot x) = \sigma(a, b) = \sigma(a, a \cdot c) = (c, c \cdot (a \triangleleft c))$$

Entonces necesariamente $y = c$ y como $y \cdot -$ es biyectiva,

$$y \cdot (a \triangleleft y) = y \cdot x \Longrightarrow a \triangleleft y = x$$

con lo cual $- \triangleleft y$ es suryectiva. Pero también, si $a \triangleleft y = a' \triangleleft y$ entonces

$$\sigma(a, a \cdot y) = (y, y \cdot (a \triangleleft y)) = (y, y \cdot (a' \triangleleft y)) = \sigma(a', a' \cdot y)$$

y como σ es biyectiva, concluimos que $a = a'$. Es decir, $- \triangleleft y$ es también inyectiva. □

Proposición 3.2. Sea $(X, \cdot, \triangleleft)$ un conjunto con dos operaciones que satisfacen 1,2,3 del Teorema 3.5. Si $(X, \triangleleft)$ es un rack y la función $x \mapsto x \cdot x =: x^2$ es una biyección en X, entonces para cada y, la función $x \mapsto (x * y) \cdot (x \triangleleft (x * y))$ es una biyección en X.

O sea que la correspondiente solución de la ecuación de Y-B, es también no degenerada a derecha.

Demostración. Usando la condición 1 del Teorema 3.5

$$(b \cdot (a \triangleleft b)) \cdot (b \cdot c) = (a \cdot b) \cdot (a \cdot c)$$

para $a = x$, $b = x * y$ y $c = a \triangleleft b$, obtenemos de cada lado de la igualdad, que

$$\big(x \cdot (x * y)\big) \cdot \Big(x \cdot \big(x \triangleleft (x * y)\big)\Big) = \Big((x * y) \cdot \big(x \triangleleft (x * y)\big)\Big)^2$$
$$y \cdot (x^2 \triangleleft y) =$$

Observemos que $z \mapsto y \cdot (z \triangleleft y)$ es una biyección (con inversa $w \mapsto (y * w) \triangleleft^{-1} y$). Como asumimos $x \mapsto x^2$ biyectiva, se sigue que

$$x \mapsto y \cdot (x^2 \triangleleft y)$$

es biyectiva, y esto es suficiente para decir que la siguiente función es biyectiva

$$x \mapsto (x * y) \cdot (x \triangleleft (x * y))$$

□

Proposición 3.3. Sea $\sigma : X \times X \to X \times X$ una solución biyectiva y no degenerada a izquierda de la ecuación de Y-B. Supongamos además que es no degenerada a derecha, es decir que para todo y fijo, la función $x \mapsto (x * y) \cdot (x \triangleleft (x * y))$ es una biyección en X, con inversa $\circ$. O sea que

$$y \circ z = x \iff z = (x * y) \cdot (x \triangleleft (x * y))$$

Entonces

$$x^2 \circ \phi(x^2) = x$$

donde $\phi(x) = x \triangleleft x$. En particular, la función $x \mapsto x^2$ es injectiva, y si X es finito entonces también es bijectiva.

Demostración. Podemos usar que vale

$$y \circ z = x \iff z = (x * y) \cdot (x \triangleleft (x * y))$$

con $y = x^2$ y $z = \phi(x^2)$, dando como resultado

$$x^2 \circ \phi(x^2) = x \iff \phi(x^2) = (x * x^2) \cdot (x \triangleleft (x * x^2))$$

Claramente $x * x^2 = x * (x \cdot x) = x$, con lo cual

$$x^2 \circ \phi(x^2) = x \iff \phi(x^2) = x \cdot (x \triangleleft x)$$

Ahora usamos que vale la condición 2 del Teorema 3.5, para decir que

$$a \cdot \phi(x) = a \cdot (x \triangleleft x) = (a \triangleleft x) \cdot (a \triangleleft x) = \phi(a \cdot x) \qquad \forall a \in X$$

y luego obtenemos

$$x \cdot (x \triangleleft x) = x \cdot \phi(x) = \phi(x \cdot x) = \phi(x^2)$$

Lo que nos dice que el término del lado derecho se cumple, con lo cual tenemos que vale que $x^2 \circ \phi(x^2) = x$. Y esto implica que si $\overline{x}^2 = x^2$ entonces $\overline{x} = x$, es decir que $x \mapsto x^2$ es inyectiva. □

Resumiendo, tenemos estos dos resultados:

- Toda solución no degenerada a izquierda de la ecuación de Y-B es de la forma

$$\sigma(x, y) := \Big(x * y, (x * y) \cdot (x \triangleleft (x * y)\Big)$$

  donde las operaciones cumplen las condiciones del Teorema 3.5.

- Una solución no degenerada a izquierda de la ecuación de Y-B, con X finito, es también no degenerada a derecha si y solo si $x \mapsto x \cdot x$ es biyectiva.

Con esto, ya estamos listos para probar el resultado que comentamos en la sección anterior. Recordemos que un birack es diagonalmente biyectivo si $(S_1^{\pm 1} \circ \Delta)$ y $(S_2^{\pm 1} \circ \Delta)$ son biyectivas, donde S es la biyección lateral del birack definida por la ecuación 3.4 y $\Delta(x) = (x, x)$.

Teorema 3.6. Todo birack finito es diagonalmente biyectivo.

Demostración. Consideremos (X, σ) un birack finito. Luego,

$$(S_1 \circ \Delta)(x) = y \iff \sigma_1(x, y) = x$$

Sabemos que $\sigma_1(x, y) = x * y$, y como $x \mapsto x \cdot x$ es biyectiva, tenemos que

$$x * y = x \iff y = x \cdot x$$

Con lo cual, y depende biyectivamente de x. Es decir que $(S_1 \circ \Delta)(x) = y$ es una biyección. Analogámente, podemos ver que $(S_1^{-1} \circ \Delta)$ y $(S_2^{\pm 1} \circ \Delta)$ son biyectivas. □

3.5. 2-cociclos universales

Por más de que la función de partición sea un potente invariante, históricamente ha sido difícil hallar 2-cociclos. Estos generalmente provenían de la teoría de álgebras de Hopf trenzadas, como las construcciones de Yetter-Drinfeld que se pueden ver en [AG04], o

posteriormente en trabajos de cohomologías de quandles en [Car+04]. Buscando una manera de fabricarlos, M.Farinati y J.Galofre construyeron en [FG16] un grupo no conmutativo a partir de un birack, que codifica las relaciones necesarias para obtener 2-cociclos. Reproduciremos e implementaremos esta construcción en el caso abeliano.

Definición 3.17. Dado X un birack y A un grupo abeliano, una función $f : X \times X \to A$ se dice un **2-cociclo de biracks** si $\forall\, x_1, x_2, x_3 \in X$, se satisface que:

$$f(x_1,x_2)f(\sigma_2(x_1,x_2),x_3)f(\sigma_1(x_1,x_2),\sigma_1(\sigma_2(x_1,x_2),x_3)) = f(x_1,\sigma_1(x_2,x_3))f(\sigma_2(x_1,\sigma_1(x_2,x_3)),\sigma_2(x_2,x_3))f(x_2,x_3)$$

Fijemos un grupo abeliano A (escrito multiplicativamente) y un link enmarcado orientado K. Sean X un birack finito, $\mathcal{C}\colon R(K) \to X$ una coloración de K y $f\colon X \times X \to A$ un 2-cociclo de X con coeficientes en A.

Definición 3.18. En cada cruce χ como el que vemos en la figura 3.9 se define el **peso de Boltzmann** $BR\omega_f(\mathcal{C},\chi)$ (con respecto a la coloración $\mathcal{C}$, al 2-cociclo f y al cruce χ) como el elemento de A dado por la expresión

$$BR\omega_f(\mathcal{C},\chi) = \begin{cases} f(x,y) & \text{si el cruce es positivo} \\ f(\sigma_1^{-1}(x,y),\sigma_2^{-1}(x,y))^{-1} & \text{si el cruce es negativo} \end{cases}$$

Definición 3.19. La **función de partición** $BRZ_{X,f}(K)$ del link enmarcado orientado K (asociada al birack X y al 2-cociclo f) es la expresión

$$BRZ_{X,f}(K) = \sum_{\mathcal{C}} \prod_{\chi} R\omega_f(\mathcal{C},\chi), \tag{3.5}$$

donde el producto se toma sobre todos los cruces χ que tiene el diagrama del link enmarcado K y la suma se toma sobre todos las coloraciones $\mathcal{C}$ de K dadas por el birack X. La fórmula (3.5) define un elemento de $\mathbb{Z}[A]$, el anillo de grupo de A.

Teorema 3.7. La función de partición $BRZ_{X,f}$ es un invariante de links enmarcados orientados.

Demostración. Tenemos que ver que los pesos son invariantes por los movimientos $\mathcal{R}_1$', $\mathcal{R}_2$ y $\mathcal{R}_3$.

El caso de $\mathcal{R}_3$, cumple por las condiciones de 2-cociclo, sumadas a las propiedades del birack, al igual que cuando trabajamos con la función de partición usando quandles.

En el caso de $\mathcal{R}_2$, se cumple por definición de peso de Boltzann, ya que estos se anulan en los cruces consecutivos de distinto signo.

En cuanto a $\mathcal{R}_1$', sucede lo mismo, ya que las etiquetas puestas mediante la coloración, generan pesos en los dos cruces, que son inversos entre si. □

Definición 3.20. Definimos el **grupo universal de 2-cociclos del birack** (X, σ), como el grupo libre generado por $(x, y) \in X \times X$ cocientado por las relaciones:

$$(x,y)(\sigma_2(x,y),z)(\sigma_1(x,y),\sigma_1(\sigma_2(x,y),z)) = (x,\sigma_1(y,z))(\sigma_2(x,\sigma_1(y,z)),\sigma_2(y,z))(y,z)$$

Notamos a este grupo $U = U(X, \sigma)$.

Definición 3.21. Definimos el **grupo universal abeliano de 2-cociclos del birack** como el abelianizado del grupo anteriormente definido, es decir $U_{ab} = U/[U, U]$.

La siguiente proposición se sigue inmediatamente de las definiciones:

Proposición 3.4. Sea X un birack:

- Notemos $[x, y]$ a la clase de (x, y) en U_{ab}. Luego la función

$$\begin{aligned} \pi : X \times X &\to U_{ab} \\ (x,y) &\mapsto [x,y] \end{aligned}$$

  es un 2-cociclo de biracks.

- Sea H un grupo abeliano y $f : X \times X \to H$ un 2-cociclo, entonces existe un único homomorfismo de grupos $\overline{f} : U_{ab} \to H$ tal que $f = \overline{f} \circ \pi$

$$\begin{array}{ccc} X \times X & \xrightarrow{f} & H \\ {\scriptstyle \pi}\downarrow & \nearrow_{\exists!\ \overline{f}} & \\ U_{ab} & & \end{array}$$

En particular, dado un birack (X, σ), existen 2-cociclos no triviales si y sólo si U_{ab} es un grupo no trivial.

Ejemplo 3.11. Veamos un cálculo explícito del grupo U_{ab} para un ejemplo particular, el biquandle de Wada que toma de base al grupo abeliano $G = \mathbb{Z}/3\mathbb{Z}$ y la operación $\sigma(x, y) = (-y, x - y) = (2y, x + 2y)$ (que también es bialexander con $s = -1$, $t = 1$).

Las relaciones del grupo U_{ab} nos indican que $\forall x, y, z \in G$:

$$(x,y) + (x + 2y, z) + (2y, 2z) = (x, 2z) + (x + z, y + 2z) + (y, z)$$

Esto da un total de 27 ecuaciones que describirán al grupo abeliano generado por los $(a, b) \in X \times X$. Las mismas son:

$$
\begin{aligned}
Ec(0,0,0):&\ (0,0)+(0,0)+(0,0)=(0,0)+(0,0)+(0,0)\\
Ec(0,0,1):&\ (0,0)+(0,1)+(0,2)=(0,2)+(1,2)+(0,1)\\
Ec(0,0,2):&\ (0,0)+(0,2)+(0,1)=(0,1)+(2,1)+(0,2)\\
Ec(0,1,0):&\ (0,1)+(2,0)+(2,0)=(0,0)+(0,1)+(1,0)\\
Ec(0,1,1):&\ (0,1)+(2,1)+(2,2)=(0,2)+(1,0)+(1,1)\\
Ec(0,1,2):&\ (0,1)+(2,2)+(2,1)=(0,1)+(2,2)+(1,2)\\
Ec(0,2,0):&\ (0,2)+(1,0)+(1,0)=(0,0)+(0,2)+(2,0)\\
Ec(0,2,1):&\ (0,2)+(1,1)+(1,2)=(0,2)+(1,1)+(2,1)\\
Ec(0,2,2):&\ (0,2)+(1,2)+(1,1)=(0,1)+(2,0)+(2,2)\\
Ec(1,0,0):&\ (1,0)+(1,0)+(0,0)=(1,0)+(1,0)+(0,0)\\
Ec(1,0,1):&\ (1,0)+(1,1)+(0,2)=(1,2)+(2,2)+(0,1)\\
Ec(1,0,2):&\ (1,0)+(1,2)+(0,1)=(1,1)+(0,1)+(0,2)\\
Ec(1,1,0):&\ (1,1)+(0,0)+(2,0)=(1,0)+(1,1)+(1,0)\\
Ec(1,1,1):&\ (1,1)+(0,1)+(2,2)=(1,2)+(2,0)+(1,1)\\
Ec(1,1,2):&\ (1,1)+(0,2)+(2,1)=(1,1)+(0,2)+(1,2)\\
Ec(1,2,0):&\ (1,2)+(2,0)+(1,0)=(1,0)+(1,2)+(2,0)\\
Ec(1,2,1):&\ (1,2)+(2,1)+(1,2)=(1,2)+(2,1)+(2,1)\\
Ec(1,2,2):&\ (1,2)+(2,2)+(1,1)=(1,1)+(0,0)+(2,2)\\
Ec(2,0,0):&\ (2,0)+(2,0)+(0,0)=(2,0)+(2,0)+(0,0)\\
Ec(2,0,1):&\ (2,0)+(2,1)+(0,2)=(2,2)+(0,2)+(0,1)\\
Ec(2,0,2):&\ (2,0)+(2,2)+(0,1)=(2,1)+(1,1)+(0,2)\\
Ec(2,1,0):&\ (2,1)+(1,0)+(2,0)=(2,0)+(2,1)+(1,0)\\
Ec(2,1,1):&\ (2,1)+(1,1)+(2,2)=(2,2)+(0,0)+(1,1)\\
Ec(2,1,2):&\ (2,1)+(1,2)+(2,1)=(2,1)+(1,2)+(1,2)\\
Ec(2,2,0):&\ (2,2)+(0,0)+(1,0)=(2,0)+(2,2)+(2,0)\\
Ec(2,2,1):&\ (2,2)+(0,1)+(1,2)=(2,2)+(0,1)+(2,1)\\
Ec(2,2,2):&\ (2,2)+(0,2)+(1,1)=(2,1)+(1,0)+(2,2)
\end{aligned}
$$

Por supuesto, hay muchas cancelaciones y repeticiones de relaciones, pero finalmente las

que definen el cociente serán:

$$\begin{aligned}(0,0) &= (2,1)\\ (0,1) &= 2*(2,1)+(2,2)\\ (0,2)+(1,1) &= 2*(2,1)\\ (1,0) &= (2,1)\\ (1,2) &= (2,1)\\ (2,0) &= (2,1)\end{aligned}$$

Y con esto, podemos concluir que $U_{ab} \simeq \mathbb{Z} \oplus \mathbb{Z} \oplus \mathbb{Z}$.

Proposición 3.5. U_{ab} es funtorial, es decir que si $\phi : (X, \sigma) \to (Y, \tau)$ es un morfismo de soluciones teóricas de la ecuación de Yang-Baxter, o sea que satisface que $(\phi \times \phi)\sigma(x, x') = \tau(\phi(x), \phi(x'))$, entonces ϕ induce un único homomorfismo de grupos $U_{ab}(X) \to U_{ab}(Y)$ que satisface $[x, x'] \mapsto [\phi(x), \phi(x')]$.

Demostración. Lo que hay que probar es que la asignación $(x, x') \mapsto (\phi(x), \phi(x'))$ es compatible con las relaciones definidas por $U_{ab}(X)$ y $U_{ab}(Y)$, respectivamente, y esto es evidente ya que $(\phi \times \phi) \circ \sigma = \tau \circ (\phi \times \phi)$. □

Observación 3.8. Dado X, siempre tenemos la proyección $\pi : X \times X \to U_{ab}$, con lo cual, cualquier 2-cociclo f que quisieramos usar para calcular la función de partición, va a estar factorizado por el universal. Por ello, basta solamente conocer las coloraciones y π para poder obtener la información que nos brindan las funciones de partición.

Entonces, si fijamos un link y el 2-cociclo dado por $\pi : X \times X \to U_{ab}$, podemos obtener los pesos de Boltzmann $BR\omega_\pi(\mathcal{C}, \chi)$, que serán elementos de U_{ab} escrito multiplicativamente. El problema notacional, es que estos elementos son clases del cociente, y nosotros queremos identificar fácilmente cuando dos de estos son el mismo, así luego los sumamos en $\mathbb{Z}[U_{ab}]$.

Observación 3.9. Recordemos que por el teorema de estructura de grupos abelianos, $U_{ab} \simeq \mathbb{Z} \oplus ... \oplus \mathbb{Z} \oplus T$ donde tenemos tantos sumandos de $\mathbb{Z}$ como el rango de U_{ab}, y T es la parte de torsión del grupo.

Si r es el rango, lo que queremos obtener con la función de partición, es un polinomio de Laurent en r variables, olvidándonos de la parte de torsión. Para construirlo necesitamos tener una forma de identificar los elementos del cociente U_{ab} con la parte libre de este isomorfismo.

Sea $X = \{x_1, \ldots, x_n\}$ el birack y sea G el grupo libre abeliano generado por

$(x, y) \in X \times X$. Supongamos que tenemos las relaciones del 2-cociclo dadas como:

$$\begin{aligned}(x_1, x_1) &= \sum_{c_k \in C} a^{c_k}_{(x_1,x_1)} c_k \\ (x_1, x_2) &= \sum_{c_k \in C} a^{c_k}_{(x_1,x_2)} c_k \\ &\dots \\ (x_n, x_{n-1}) &= \sum_{c_k \in C} a^{c_k}_{(x_n,x_{n-1})} c_k \\ (x_n, x_n) &= \sum_{c_k \in C} a^{c_k}_{(x_n,x_n)} c_k\end{aligned}$$

donde $C = \{c_1, \dots, c_r\} \subset X \times X$ es un conjunto minimal de generadores, producto de simplificar las relaciones de 2-cociclo, y $a^{c_k}_{(x_i,x_j)} \in \mathbb{Q}$ son los coeficientes de la escritura de (x_i, x_j) en C.

Observamos que en este caso de que $(x_i, x_j) \in C$ entonces $\exists\, m/(x_i, x_j) = c_m$.

Definición 3.22. Definimos el **2-cociclo universal abeliano** como $\vartheta : X \times X \to \mathbb{Q}^{\oplus r}$

$$\vartheta((x_i, x_j)) = (a^{c_1}_{(x_i,x_j)}, ..., a^{c_r}_{(x_i,x_j)})$$

O si lo queremos ver como la suma de antes, tenemos

$$\vartheta((x_i, x_j)) = \sum_{c_k \in C} a^{c_k}_{(x_i,x_j)} c_k$$

Definición 3.23. A la función de partición $BRZ_{X,\vartheta}(K)$, la llamaremos **función de partición universal**.

Para llegar a tener relaciones como las de antes, necesitamos poder despejar algunos elementos en función de otros. Así, podemos tomar C como los elementos que son generados por un solo término.

Para hacer esto creamos una matriz E donde cada fila representa una ecuación: las entradas son los elementos de X x X, con un valor igual a la cantidad de veces que aparecen en esa ecuación, con un signo positivo si están del lado izquierdo y negativo si están del lado derecho.

Luego, lo que haremos será triangular esa matriz de forma racional, para obtener el valor de un elemento, en función de los que generan. El problema con esto, es que podemos perder informacíon, por ejemplo al llevar

$$3 * (x_i, x_j) = 3 * (x_k, x_l) \quad \longrightarrow \quad (x_i, x_j) = (x_k, x_l)$$

donde evidentemente la primer ecuación no es equivalente a la segunda. Pero esto afecta solamente a la parte de torsión así que nos centraremos sólo en hallar los generadores de la parte libre, y dejaremos la torsión de lado.

Ejemplo 3.12. En el Ejemplo 3.11 calculamos las ecuaciones, y luego obtuvimos relaciones triangulando racionalmente, pero los coeficientes que obtuvimos fueron coeficientes enteros.

Si tomamos el birack $X = \{x_1, x_2, x_3\}$ dado por:

σ	x_1	x_2	x_3
x_1	(x_2, x_2)	(x_1, x_2)	(x_3, x_2)
x_2	(x_1, x_3)	(x_3, x_3)	(x_2, x_3)
x_3	(x_3, x_1)	(x_2, x_1)	(x_1, x_1)

entonces obtenemos las relaciones:

$$\begin{aligned}
(x_1, x_1) + (x_2, x_2) &= -6 * (x_3, x_2) + 4 * (x_3, x_3) \\
(x_1, x_2) &= -2 * (x_3, x_2) + (x_3, x_3) \\
(x_1, x_3) + \frac{1}{2} * (x_2, x_2) &= -4 * (x_3, x_2) + \frac{5}{2} * (x_3, x_3) \\
(x_2, x_1) - \frac{1}{2}(x_2, x_2) &= -1 * (x_3, x_2) + \frac{1}{2} * (x_3, x_3) \\
(x_2, x_3) &= -2 * (x_3, x_2) + (x_3, x_3) \\
(x_3, x_1) &= -2 * (x_3, x_2) + (x_3, x_3)
\end{aligned}$$

Si cocientamos al grupo libre abeliano generado por $X \times X$, por estas relaciones, lo que obtenemos es $\mathbb{Z} \oplus \mathbb{Z} \oplus \mathbb{Z} \oplus \mathbb{Z}/2\mathbb{Z}$, que no es lo que queremos, ya que podríamos haber perdido información. Lo que sí podemos concluir, es que $U_{ab}/T \simeq \mathbb{Z} \oplus \mathbb{Z} \oplus \mathbb{Z}$.

En este ejemplo $C = \{c_1 := (x_2, x_2), c_2 := (x_3, x_2), c_3 := (x_3, x_3)\}$. Luego:

ϑ	x_1	x_2	x_3
x_1	$-c_1 - 6c_2 + 4c_3$	$-2c_2 + c_3$	$-\frac{1}{2}c_1 - 4c_2 + \frac{5}{2}c_3$
x_2	$\frac{1}{2}c_1 - c_2 + \frac{1}{2}c_3$	c_1	$-2c_2 + c_3$
x_3	$-2c_2 + c_3$	c_2	c_3

son los valores del 2-cociclo universal abeliano, donde cada elemento pertenece a $\mathbb{Q} \oplus \mathbb{Q} \oplus \mathbb{Q}$.

Ejemplo 3.13. En el ejemplo 4.11 mostramos un caso donde los sumandos de la función de partición universal son elementos racionales.

CAPÍTULO 4

GAP

GAP quiere decir *Groups, Algorithms and Programming* [24]. Lo cual en español seria: Grupos, Algoritmos y Programación. GAP es un sistema algebraico del argot computacional (CAS), diseñado para situaciones donde se use teoría de grupos (o similares, como haremos nosotros). Es un lenguaje de programación, una biblioteca con miles de funciones que implementan algoritmos algebraicos y grandes librerías de datos de objetos algebraicos. Este software fue programado entre los años 1986 y 1997 para la cátedra matemática de una Universidad en Alemania, la Universidad Técnica de Aquisgrán.

4.1. Definiciones básicas

Definición 4.1. Una **orden** es una instrucción que se introduce como código, para que GAP la interprete. Toda orden debe terminar con el símbolo ; (punto y coma). Si una orden termina con ;; (doble punto y coma), no producirá texto en la pantalla. Los comentarios en GAP comienzan con el símbolo #. Si lo que queremos ingresar en la línea de comandos ocupa mucho lugar, el uso del símbolo \ aumenta la legibilidad.

Definición 4.2. Para GAP, un **objeto** es algo que puede asignarse a una variable. Un objeto puede ser un número, una cadena de carácteres, un cuerpo, un grupo, un elemento o un subgrupo de un grupo, un morfismo entre dos grupos, un anillo, una matriz, un espacio vectorial, etc. Asignar un objeto a una variable se hará mediante el operador := tal como muestra el ejemplo siguiente:

```
gap > p := 32;;
gap > p ;
32
gap > p = 32;
true
gap > p := p +1;;
```

gap > p ;

33

gap > p = 32;

false

Definición 4.3. Podemos representar expresiones lógicas, usando $>$, $<$, $=$, y la respuesta tendrá un valor de true o false, dependiendo de la veracidad de la expresión representada. Además tenemos la expresión x<>y , que tendrá un valor de true si las variables x e y son distintas, y false en caso contrario.

Definición 4.4. En GAP existen dos formas de definir funciones. La más sencilla es definir la función, mediante $x -> f(x)$ como vemos a continuación:

gap > cubo := x -> x ^ 3;

function(x) ... end;

gap > cubo(4);

64

Otra forma es definirla así:

gap > cubo := function(x)

> return x ^ 3;

> end;

function(x) ... end;

gap > cubo(4);

64

Definición 4.5. En GAP una **lista** es simplemente una sucesión ordenada de objetos, separados con comas, adentro de dos corchetes. La lista pueden tener objetos diferentes y además tener lugares vacíos.

gap > lista := [1 .. 10];

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

gap > pares := [2, 4, 6, 8, 10];

[2, 4, 6, 8, 10]

Podemos agregar un elemento con el comando **Add** o unirle una lista con el comando **Append**:

gap > Add(pares,12);

gap > pares;

[2, 4, 6, 8, 10, 12]

```
gap > Append(pares,[14, 16]);
gap > pares;
[ 2, 4, 6, 8, 10, 12, 14, 16 ]
```

Podemos obtener información sobre la longitud de la lista, obtener o redefinir el valor de una o varias coordenadas, u obtener la posición de algún elemento o de varios elementos en concreto:

```
gap > Length(pares);
8
gap > pares[3];
6
gap > pares[2] := 8;
gap > pares;
[ 2, 8, 6, 8, 10, 12, 14, 16 ]
gap > pares[5,6] := [11,13] ;;
gap > pares;
[ 2, 8, 6, 8, 11, 13, 14, 16 ]
gap > Position(pares, 13);
6
gap > Position(pares, 5);
fail
gap > Positions(pares, 8);
[ 2, 4 ]
```

Definición 4.6. Si se quiere hacer una copia de una lista, para poder alterarla sin alterar la lista original, se utilizará el comando **StructuralCopy**:

```
gap > a := [1 , 2 , 3 , 4];;
gap > b := a ;;
gap > c := StructuralCopy( a );;
gap > Add(a , 5);
gap > a ;
[ 1 , 2 , 3 , 4 , 5 ]
gap > b ;
[ 1 , 2 , 3 , 4 , 5 ]
gap > c ;
[ 1 , 2 , 3 , 4 ]
```

Definición 4.7. Para conocer la cantidad de elementos de una lista que cumplen determinada propiedad, usamos el comando **Number**. Este toma una lista y una función (que devuelva true o false) y como resultado obtenemos un número.

```
gap > Number( [1..20], IsPrime );
8
```

Definición 4.8. También podemos aplicarle funciones a cada lugar de la lista, como vemos en este ejemplo:

```
gap > cubo := x - > x ^ 3;;
gap > List([2..10] , cubo);
[ 8, 27, 64, 125, 216, 343, 512, 729, 1000 ]
```

Definición 4.9. El comando **Cartesian**, tiene como entrada una cantidad finita de listas, y genera una lista con listas de las posibles combinaciones ordenadas de elementos de cada una de las entradas:

```
gap > A := [1,2,3];; B := [6,7,8];;
gap > Cartesian(A,B);
[ [ 1, 6 ], [ 1, 7 ], [ 1, 8 ], [ 2, 6 ], [ 2, 7 ], [ 2, 8 ], [ 3, 6 ], [ 3, 7 ], [ 3, 8 ] ]
gap > Cartesian(A,B,A);
[ [ 1, 6, 1 ], [ 1, 6, 2 ], [ 1, 6, 3 ], [ 1, 7, 1 ], [ 1, 7, 2 ], [ 1, 7, 3 ], [ 1, 8, 1 ], [ 1, 8, 2 ],
[ 1, 8, 3 ], [ 2, 6, 1 ], [ 2, 6, 2 ], [ 2, 6, 3 ], [ 2, 7, 1 ], [ 2, 7, 2 ], [ 2, 7, 3 ], [ 2, 8, 1 ],
[ 2, 8, 2 ], [ 2, 8, 3 ], [ 3, 6, 1 ], [ 3, 6, 2 ], [ 3, 6, 3 ], [ 3, 7, 1 ], [ 3, 7, 2 ], [ 3, 7, 3 ],
[ 3, 8, 1 ], [ 3, 8, 2 ], [ 3, 8, 3 ] ]
```

Definición 4.10. Un **registro** es un tipo de dato de fundamental importancia ya que permite empaquetar objetos dentro de una misma estructura. Luego, para usar estos objetos, los llamamos desde el registro desde donde están almacenados:

```
gap > numeros := rec( pares := [2,4,6] , impares := [1,3,5,7] );;
gap > numeros.pares;
[ 2, 4, 6 ]
gap > numeros.impares[2];
3
```

Definición 4.11. Un **bucle** es un código que repite una acción, a la cual le ponemos condiciones para controlar desde donde empieza y hasta donde llega. Una forma de hacerlo es con los comandos **for...do...od**. **for** establece que es lo qué estoy recorriendo; **do** ejecuta la acción que viene a continuación; y **od** lo corta finalmente.

```
gap > s := 0
gap > for i in [1..100] do
> s := s + i;
> od;
gap > s;
5050
```

Una idea similar, es usar **while...do...od**. Lo que hace **while** es ejecutar la acción, siempre y cuando se cumpla la condición que le sigue a continuación.

```
gap > s := 0;;
gap > k := 1;;
gap > while k <= 100 do
> s := s+k;
> k := k+1;
> od;
gap > s;
5050
```

O también podemos construir un bucle usando **repeat...until**:

```
gap > s := 0;;
gap > k := 1;;
gap > repeat
> s := s + k;
> k := k + 1;
> until k > 100;
gap > s;
5050
```

Definición 4.12. Para funciones en las que necesitemos almacenar algún dato durante el proceso, utilizamos lo que se conoce como **variables locales**. Éstas se definen como en el siguiente ejemplo:

```
gap > mcd := function(a,b)
> local c;
> while b <> 0 do
> c := b;
```

```
> b := a mod b;
> a := c;
> od;
> return c;
> end;
function( a, b ) ... end
gap > mcd(30,63);
3
```

Definición 4.13. Para dividir el accionar del código, usamos condicionales, que permiten contemplar varias posibilidades a la vez. **fi** se usa para abrir el condicional, y **elif** para seguir describiendo posibles casos. Estos dos van acompañados de **then** al final de su condición, para determinar como se actúa en esas situaciones. **else** se usa para englobar a los casos restantes, y finalmente se cierra la condición con **fi**. Veámoslo a continuación:

```
gap > signo := function(n)
> if n < 0 then
> return -1;
> elif n = 0 then
> return 0;
> else
> return 1;
> fi;
> end;
function(n) ... end
gap > signo(0); signo(-99); signo(11);
0
-1
1
```

Si queremos que la condición no genere ninguna acción, usamos el código **continue**. Como ejemplo, hagamos una función que sume hasta N, los pares solamente:

```
gap > sumapar := function(N)
> local k,i;
> k := 0;
> for i in [1..N] do
```

```
> if i mod 2 = 0 then
> k := k + i;
> else continue;
> fi;
> od;
> return k;
> end;
gap > sumapar(8);
20
```

4.2. Backtracking

En ésta sección, usaremos un paquete adicional para GAP llamado *YAGS* que significa *Yet Another Ghaph System* [Ced+19] . Este es un paquete que tiene un lenguaje muy útil a la hora de trabajar con grafos y lo que usaremos es un código que sirve para considerar subfamilias de grafos o aplicar coloraciones a estos.

Lo que queremos usar es un **backtracking**, una estrategia algorítmica que busca todas las posibles soluciones dado un conjunto de variables inicial y unas condiciones impuestas, para encontrar el resultado definido por el problema. Esta técnica se apoya en el uso de la recursividad para la búsqueda exhaustiva de todas las combinaciones posibles. Para esto, se genera un árbol combinatorio implícito que incluye todas las posibles soluciones, y el algoritmo lo que hace es recorrerlo en orden creciente (como en la figura 4.1), "podando" las ramas que no cumplan las condiciones impuestas, es decir, cortando su trayecto cuando la rama que se va a recorrer no cumple lo pedido. En caso de que la rama si lo cumple, la recorre hasta llegar al final, y almacena su resultado final, para luego volver al paso anterior y seguir con la siguiente rama.

Las ventajas de este algoritmo son:

- De existir una solución, la calcula.
- Es un esquema sencillo de implementar.
- Es adaptable a las características específicas de cada problema.

Aunque como deventaja tiene:

- Coste exponencial en la mayoría de los casos.
- Por término medio consume mucha memoria al tener que almacenar las llamadas recursivas.

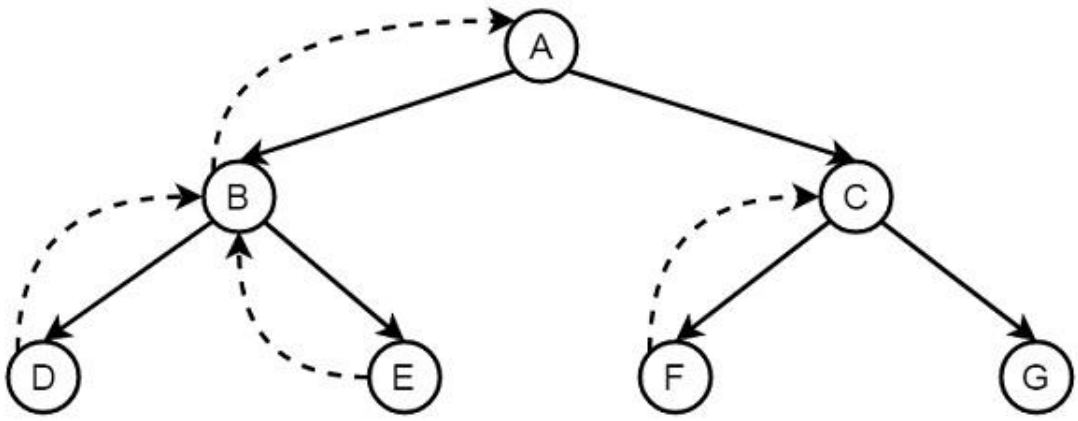


Figura 4.1: Recorrido del algoritmo de backtracking

Ejemplo 4.1. En la figura 4.2 podemos ver una aplicación del backtracking al problema de las N reinas:

Supongamos que tenemos un tablero de ajedrez de $N \times N$ y queremos encontrar todas las formas de ubicar N reinas en ese tablero sin que se ataquen mutuamente (recordemos que las reinas atacan en linea recta o en diagonales). Para ello, se ubica la primer reina en algún sitio (primera rama de elección), y luego se consideran donde ubicar la segunda, eliminando las posibilidades en donde se produzca un ataque con la primer reina (podando las subramas no deseadas). Repitiendo este proceso, llegaremos (de haber solución) al resultado que queremos.

Ejercicio 4.1. Invitamos al lector a realizar un programa para calcular la solución del problema de las N reinas, en GAP, utilizando las herramientas brindadas anteriormente y el siguiente código de backtracking.

Definición 4.14. Con el paquete YAGS, tenemos disponible el código

BacktrackBag(Opts,Check,Done,Extra) para almacenar las listas que cumplan esas propiedades que queremos.

La entrada de Opts, será la lista que contenga los elementos con los cuales llenaremos nuestra lista BacktrackBag.

Check(L,Extra), será una función que toma el elemento L del proceso de backtracking (y puede tomar también elementos externos de Extra), y nos devuelve true o false, dependiendo de si este proceso debe seguir o si detenemos esa rama del backtracking.

La entrada de Done, será la longitud de cada lista de nuestra BacktrackBag.

Finalmente en Extra tendremos la información adicional que queremos utilizar en la función check.

Ejemplo 4.2. Acá nuestra función check es ReturnTrue, o sea que no descarta ningún caso:

```
gap > BacktrackBag([0,1],ReturnTrue,3,[]);
```

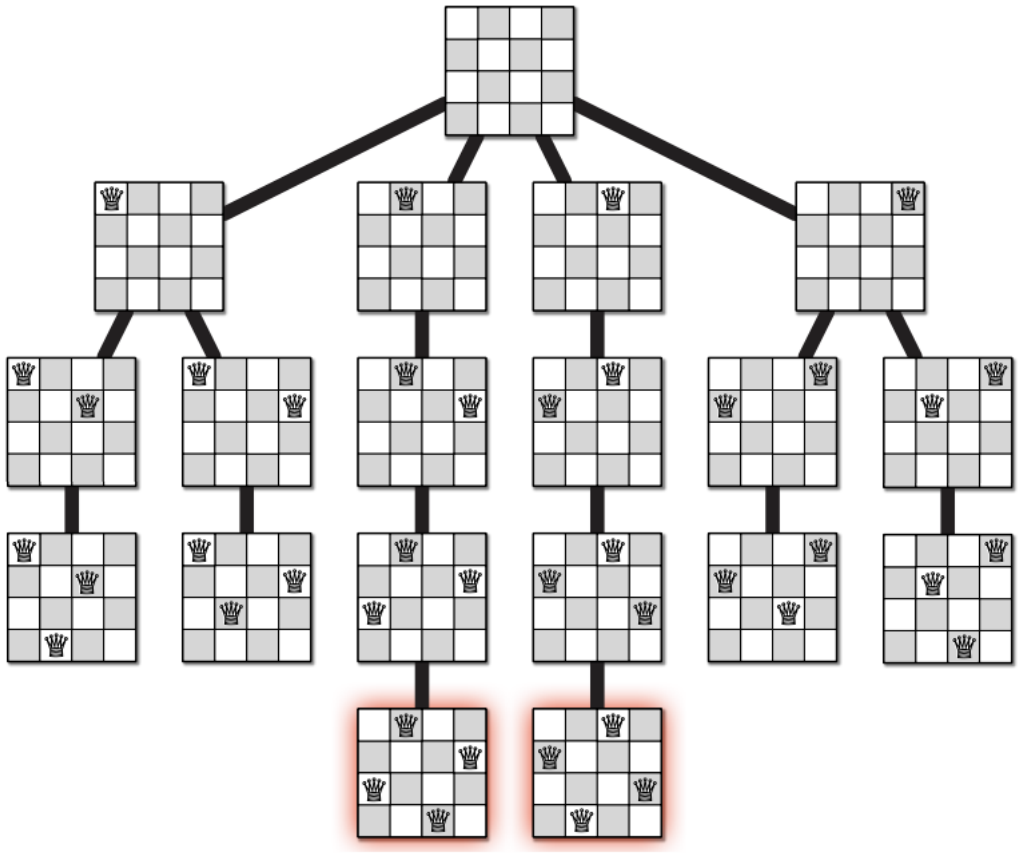


Figura 4.2: Problema de las N reinas para $N = 4$

[[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1], [1, 0, 0],
[1, 0, 1], [1, 1, 0], [1, 1, 1]]

Ejemplo 4.3. Un ejemplo más interesante es si planteamos una función que nos deje sólo los casos en los que vamos agregando cosas mayores que N y que sean pares. Vemos aquí, como en cada paso voy utilizando el elemento L parcial del backtracking, además de observar que no uso el elemento Extra. Si por ejemplo N = 3, tengo:

```
gap > N = 3;;
gap > parmayorN := function(L,Extra)
> local l,n;
> l := Size(L);
> n := L[l];
> if n > N then
> return n mod 2 = 0;
> else
> return false;
```

> fi;

> end;

>

gap > BacktrackBag([1,2,3,4,5,6],parmayorN,5,[]);

[[4, 4, 4, 4, 4], [4, 4, 4, 4, 6], [4, 4, 4, 6, 4], [4, 4, 4, 6, 6], [4, 4, 6, 4, 4], [4, 4, 6, 4, 6], [4, 4, 6, 6, 4], [4, 4, 6, 6, 6], [4, 6, 4, 4, 4], [4, 6, 4, 4, 6], [4, 6, 4, 6, 4], [4, 6, 4, 6, 6], [4, 6, 6, 4, 4], [4, 6, 6, 4, 6], [4, 6, 6, 6, 4], [4, 6, 6, 6, 6], [6, 4, 4, 4, 4], [6, 4, 4, 4, 6], [6, 4, 4, 6, 4], [6, 4, 4, 6, 6], [6, 4, 6, 4, 4], [6, 4, 6, 4, 6], [6, 4, 6, 6, 4], [6, 4, 6, 6, 6], [6, 6, 4, 4, 4], [6, 6, 4, 4, 6], [6, 6, 4, 6, 4], [6, 6, 4, 6, 6], [6, 6, 6, 4, 4], [6, 6, 6, 4, 6], [6, 6, 6, 6, 4], [6, 6, 6, 6, 6]]

4.3. Algoritmos de nudos

Estructuras algebraicas

Definición 4.15. Para definir la solución de la ecuación de Y-B (X, σ), consideramos el conjunto finito X y trabajamos con funciones de X a X, identificando los elementos $\{x_1, \ldots, x_k\}$ con $\{1, \ldots, k\}$. Usamos dos listas, donde los elementos serán esas funciones, llamadas **lfunc** y **rfunc**. Tomamos $\sigma(n, m) = (g_n(m), f_n(m))$, donde g_n es la función ubicada en el lugar n-ésimo de lfunc y f_m es la función ubicada en el lugar m-ésimo de rfunc, y a cada una la valuamos en los puntos m y n respectivamente.

Esto lo hacemos de la siguiente manera. Si X es un conjunto de n elementos, definimos la estructura como:

SolYB := rec(lfunc := [g_1 , g_2 , ... , g_n] , rfunc := [f_1 , f_2 , ... , f_n])

Donde cada g_i es [$g_i(1)$, ... , $g_i(n)$], o sea una lista de números, que podemos interpretar como función. Luego, definimos la función $\sigma(x, y)$ como:

gap > sigma := function(SolYB, x, y)

> return [bi.lfunc[x][y], bi.rfunc[y][x]];

> end;

Y esto lo que hace es tomar los elementos x e y; primero devuelve lo que vale la lista lfunc[x] en el lugar y, o sea que da el valor de $g_x(y)$; y luego devuelve lo que vale la lista rfunc[y] en el lugar x, o sea que me el valor de $f_y(x)$.

Para ver que sea un solución de la ecuación de Y-B, habría que chequear que para todo $n, m, l \in X$:

$$(Id \times \sigma)(\sigma \times Id)(Id \times \sigma)(n, m, l) = (\sigma \times Id)(Id \times \sigma)(\sigma \times Id)(n, m, l)$$

Esto dependerá solamente de las funciones que están en cada lista, con lo cual, cuando queremos trabajar con (X, σ), nos basta saber cómo son lfunc y rfunc.

Como nos interesa trabajar con biracks, chequearemos que vale Y-B para nuestros candidatos a biracks directamente. Pero primero veamos como traducir las condiciones de estos.

Definición 4.16. Para definir un birack (X, σ), tomamos las listas de funciones de una solución de Y-B, y vemos que condiciones le debemos pedir para obtener la no degeneración:

- Esta condición nos dice que para todo $n, l \in X$, existe un único $m \in X$ tal que $g_n(m) = l$. Es decir que todas las funciones de lfunc son permutaciones.
- Esta condición nos dice que para todo $n, l \in X$, existe un único $m \in X$ tal que $f_n(m) = l$. Es decir que todas las funciones de rfunc son permutaciones.

Entonces, en cada lista los elementos serán permutaciones de n elementos, y por ello renombramos a lfunc y rfunct como **lperms** y **rperms**, respectivamente.

La siguiente función chequea si se cumplen las condiciones de birack:

```
gap > checkYB := function(bi)
> local x,y,z, N,V;
> V := true;
> N := Size(bi.rperms);
> for x in [1..N] do;
> for y in [1..N] do;
> for z in [1..N] do;
> if [sigma(bi,sigma(bi,x,y)[1],sigma(bi,sigma(bi,x,y)[2],z)[1])[1],
> sigma(bi,sigma(bi,x,y)[1],sigma(bi,sigma(bi,x,y)[2],z)[1])[2],
> sigma(bi,sigma(bi,x,y)[2],z)[2]]
> =
> [sigma(bi,x,sigma(bi,y,z)[1])[1],
> sigma(bi,sigma(bi,x,sigma(bi,y,z)[1])[2],sigma(bi,y,z)[2])[1],
> sigma(bi,sigma(bi,x,sigma(bi,y,z)[1])[2],sigma(bi,y,z)[2])[2]] then
> continue;
> else
> V := false;
> z := N;x := N;y := N;
> fi;
> od;od;od;
> return V;
> end;
```

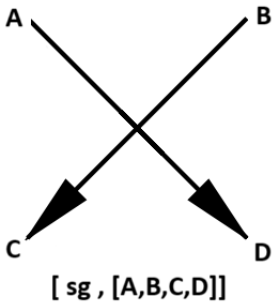


Figura 4.3: PD-code de un cruce

Código de Links

¿Cómo llevamos los links a algo algebraico? Hay varias maneras. La que usamos en este trabajo, es una llamada **PD-Code**, que codifica el link en base a la información de cada cruce.

Primero, tomamos un punto base de una componente del link, y etiquetamos el semiarco en el que se encuentra con el número 1. Siguiendo la orientación, lo recorremos hasta llegar al primer cruce y luego, etiquetamos el semiarco que sigue al primero a través del cruce, con el número 2. Y así continuamos etiquetando hasta llegar al nuevamente al semiarco inicial.

En caso de tener un link, etiquetamos la primer componente, y luego tomamos un nuevo punto base para la nueva componente, repitiendo el proceso anterior.

Ahora codifiquemos los cruces:

En cada cruce tomamos una lista con dos lugares. En el primero tenemos el signo del cruce, y en el segundo, una lista de cuatro elementos, donde el primero corresponde al semiarco superior izquierdo entrante, el segundo al semiarco superior derecho entrante, el tercero al semiarco inferior izquierdo saliente, y el cuarto al semiarco inferior derecho saliente.

Esto se puede ver en la figura 4.3.

Para obtener finalmente el PD-code, creamos una lista, donde en cada lugar tengamos un cruce, ordenados como van apareciendo a lo largo del etiquetado del link.

Ejemplo 4.4. En la figura 4.4 se puede ver el etiquetado del nudo 4_1:

Finalmente el PD-code de este diagrama es:

k := [[1,[4,1,2,5]],[-1,[7,2,3,8]],[-1,[3,6,7,4]],[1,[8,5,6,1]]];

Ejercicio 4.2. Construir una función que tenga de entrada un PD-code de un link L y que devuelva el link m(L).

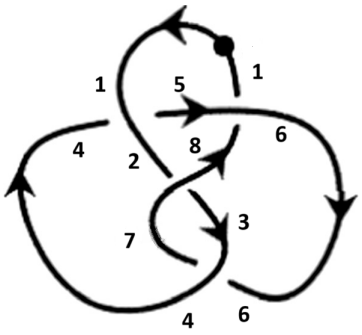


Figura 4.4: Etiquetado del nudo 4_1

Ejercicio 4.3. Construir una función que tenga de entrada un PD-code de un link L y que devuelva el link r(L).

Ejercicio 4.4. Construir una función que tenga de entrada al par (K, n) donde K es el PD-code de un nudo , n es un entero, y devuelva el PD-code del nudo K luego de hacerle n twists.

Ejercicio 4.5. (Más difícil) Construir una función que tenga de entrada al par (L, N) donde L es el PD-code de un link de r componentes, $N = (n_1, ..., n_r)$ es una lista de r enteros, y devuelva el PD-code del link L luego de hacerle n_i twist a la i-ésima componente.

Coloreos

La idea del algoritmo será armar una lista que contenga las coloraciones del nudo, dónde cada coloración será una lista de C elementos (uno por cada semiarco), con elementos de nuestro birack X. Si reemplazamos los C semiarcos ordenados de nuestro diagrama con los elementos de una coloración, lugar a lugar, lo que obtenemos es un nudo etiquetado con elementos de X que cumple las relaciones de coloreo.

Estas coloraciones las armaremos mediante backtracking, asignándole un elemento de X al primer semiarco del diagrama y arrancando así un árbol de posibles listas de coloraciones. Para el segundo semiarco tenemos $|X|$ ramas posibles para seguir e igualmente para la tercera elección, y así seguimos. Lo que iremos haciendo paso a paso, es ir podando las ramas que me generen alguna relación de coloreo incompatible en el diagrama, y las que sobreviven son las coloraciones que cumplen las relaciones de coloreo, y éstas se guardan en mi lista de coloraciones.

Dado un PD-code de un link, quieremos obtener el número de semiarcos de esa escritura. Para ello tenemos la siguiente función:

gap > Cantarcos := function(k)

> local c,i;

```
> c := 1;
> for i in [1..Size(k)] do
> if c < k[i][2][1] then
> c := k[i][2][1];
> elif c < k[i][2][2] then
> c := k[i][2][2];
> else continue;
> fi;
> od;
> return c;
> end;
function( k ) ... end
```

Luego, nos enfrentamos a un problema de notación, ya que tenemos numerados los semiarcos del diagrama, del 1 al C; y tenemos numerados los elementos de X, de 1 a N. Para poder cambiar los nombres de los arcos diferenciando cuando son elementos de X y cuando todavía no les asignamos un elemento, lo que hacemos es mover el nudo N lugares, y así, si el diagrama tiene un semiarco con etiqueta menor que N+1, sabremos que ese semiarco ya está etiquetado con un elemento de X. El link original seguirá siendo k, mientras que nuestro link desplazado se llamará mk.

```
gap > Move := function(bi,k)
> local ck,i,S,N;
> N := Size(bi.lperms[1]);
> S := Size(k);
> ck := StructuralCopy(k);
> for i in [1..S] do
> ck[i][2] := List (ck[i][2] , x -> x+N);
> od;
> return ck;
> end;
function( bi, k ) ... end
```

La siguiente función toma el diagrama, un arco (del N+1 al N+C) y un elemento de X. Lo que hace es buscar en cada cruce del diagrama, las posiciones en las que aparece ese arco. Luego reemplaza ese arco en cada una de esas posiciones, por la etiqueta del elemento de X.

```
gap > Etiquetado := function(k,r,x)
> local P, R,i,S;
> S := Size(k);
> for i in [1..S] do
> if Position(k[i][2],r) <> fail then
> P := Positions(k[i][2],r);
> R := List([1..Size(P)], i -> x);
> k[i][2]P := R;
> else
> continue;
> fi;
> od;
> return k;
> end;
function( k, r, x ) ... end
```

La siguiente función toma un PD-code movido por N, y chequea si sus arcos etiquetados cumplen las relaciones de coloreo, devolviendo true (si se cumple todo) o false (si al menos un cruce falla). Lo hace cruce a cruce, identificando cuántos arcos etiquetados tiene y cuál es el signo de ese cruce. En caso de tener 4 arcos etiquetados, chequea que dos sean sigma de los otros (dependiendo el signo). En caso de que sean tres, identifica la posición de los tres y en cada caso chequea que esos tres cumplan la relación, ya que como vimos en el capítulo anterior, dados dos arcos, tenemos determinados los otros dos. En caso de que tengan menos de tres arcos etiquetados, lo considera como verdadero para seguir con el backtracking. Luego de considerar cada cruce, toma la lista de respuestas en cada uno, y si hay algún false, devuelve false el chequeo.

```
gap > checkX := function(bi,mk)
> local l,i,S,N;
> N := Size(bi.lperms[1]);
> S := Size(mk);
> l := [];;
> for i in [1..S] do
> if Number(mk[i][2], n -> n < N+1)=4 then
> if (mk[i][1])=1 then
> l[i] := sigma(bi, mk[i][2][1] , mk[i][2][2] )=[ mk[i][2][3] , mk[i][2][4] ];
```

```
> else
> l[i] := sigma(bi, mk[i][2][3] , mk[i][2][4] )=[ mk[i][2][1] , mk[i][2][2] ];
> fi;
> elif Number(mk[i][2], n -> n < N+1)=3 then
> if (mk[i][1])=1 then
> if mk[i][2][1] > N then
> l[i] := mk[i][2][3] = sigma(bi, Position(bi.rperms[mk[i][2][2]] , mk[i][2][4] ),
mk[i][2][2] )[1];
> elif mk[i][2][2] > N then
> l[i] := mk[i][2][4] = sigma(bi, (mk[i][2][1]), Position(bi.lperms[ mk[i][2][1] ],
mk[i][2][3] ) )[2];
> elif mk[i][2][3] > N then
> l[i] := mk[i][2][4] = sigma(bi, mk[i][2][1],mk[i][2][2])[2];
> else
> l[i] := mk[i][2][3] = sigma(bi, mk[i][2][1],mk[i][2][2])[1];
> fi;
> else
> if mk[i][2][1] > N then
> l[i] := mk[i][2][2] = sigma(bi, mk[i][2][3] , mk[i][2][4] )[2];
> elif mk[i][2][2] > N then
> l[i] := mk[i][2][1] = sigma(bi, mk[i][2][3] , mk[i][2][4] )[1];
> elif mk[i][2][3] > N then
> l[i] := mk[i][2][1] = sigma(bi, Position(bi.rperms[ mk[i][2][4] ], mk[i][2][2] ),
mk[i][2][4] )[1];
> else
> l[i] := mk[i][2][2] = sigma(bi, mk[i][2][3], Position(bi.lperms[ mk[i][2][3] ],
mk[i][2][1] ) )[2];
> fi;
> fi;
> else
> l[i] := true;
> fi;
> od;
> if false in l then
```

```
> return false;
> else
> return true;
> fi;
> end;
function( mk, bi ) ... end
```

A continuación tenemos la función chequeo del backtracking, que toma nuestra lista de backtrack parcial L y nuestro Extra que es k, el diagrama del nudo. Lo que hace es generar una copia cmk de mk que se pueda modificar, para poder aplicarle el etiquetado según L, lugar a lugar. Finalmente aplica a ck la función checkX para ver si su coloración parcial por L es correcta. En base a eso devuelve true o false.

```
gap > check := function(L,Extra)
> local cmk,i,S,N;
> N := Extra[1];
> S := Size(Extra[2]);
> cmk := StructuralCopy(Extra[2]);
> for i in [1..Size(L)] do
> cmk := Etiquetado(cmk,N+i,L[i]);
> od;
> return checkX(Extra[3],cmk);
> cmk := Extra[2];
> end;
function( L, mk ) ... end
```

Nuestro paso final es hacer la función con backtracking que devuelva la lista de coloreos, dándole como entrada el nudo y el birack.

```
gap > colorlist := function(bi,k)
> local C,mk, N, ext ;
> N := Size(bi.lperms[1]);
> C := Cantarcos(k);
> mk := Move(bi,k);
> ext := [N,mk,bi];
> return BacktrackBag([1..N],check,C,ext);
```

> end;

function(bi, k) ... end

Ejemplo 4.5. Si consideramos el trébol trenzado k visto en el ejemplo 2.4 , tenemos que su escritura en PD-code es:

k := [[1, [1,4,5,2]] , [1,[5,2,3,6]] , [1,[3,6,1,4]]]

En ese ejemplo, habiamos considerado el quandle de Alexander con $X = \mathbb{Z}_2[t]/(t^2+t+1)$. Ahora, consideramos el bialexander con $r = 1$ y $t = t$ (ahora como elemento en X). Su código en permutaciones será el siguiente:

bi := rec(lperms := [[1,2,3,4] , [1,2,3,4] , [1,2,3,4] , [1,2,3,4]] ,

rperms := [[1,3,4,2] , [4,2,1,3] , [2,4,3,1] , [3,1,2,4]])

Observamos que mirar las coloraciones del diagrama por el quandle, es lo mismo que mirar las coloraciones del diagrama por este birack, dada la estructura que le dimos. Finalmente:

gap > A := colorlist(bi,k))

[[1, 1, 1, 1, 1, 1], [1, 2, 2, 3, 3, 1], [1, 3, 3, 4, 4, 1],

[1, 4, 4, 2, 2, 1], [2, 1, 1, 4, 4, 2], [2, 2, 2, 2, 2, 2],

[2, 3, 3, 1, 1, 2], [2, 4, 4, 3, 3, 2], [3, 1, 1, 2, 2, 3],

[3, 2, 2, 4, 4, 3], [3, 3, 3, 3, 3, 3], [3, 4, 4, 1, 1, 3],

[4, 1, 1, 3, 3, 4], [4, 2, 2, 1, 1, 4], [4, 3, 3, 2, 2, 4],

[4, 4, 4, 4, 4, 4]]

gap > Length(A);

16

Mismo resultado que obtuvimos en ese mismo ejemplo, coloreando a mano.

Ejemplo 4.6. Veamos que efectivamente el grupo fundamental no distingue links. Recordemos los links de la observación 1.4, ellos tienen los mismos grupos fundamentales. Los escribimos con PD-code:

L1 := [[-1 , [10,1,2,7]] , [-1 , [2,7,8,3]] , [1 , [6,3,4,1]] , [-1 , [9,4,5,10]] , [-1 , [5,8,9,6]]]

L2 := [[-1 , [9,1,2,10]] , [-1 , [2,8,9,3]] , [-1 , [3,13,14,4]] , [-1 , [14,4,1,5]] , [1 , [10,5,6,11]] ,[1 , [6,11,12,7]] , [1 , [12,7,8,13]]]

Y luego, coloreándolos con el bialexander del ejemplo anterior, tengo:

gap > colorlist(bi,L1);

[[1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 2, 2, 2, 3, 3, 4, 1, 4, 4], [1, 3, 3, 3, 4, 4, 2, 1, 2, 2],

[1, 4, 4, 4, 2, 2, 3, 1, 3, 3], [2, 1, 1, 1, 4, 4, 3, 2, 3, 3], [2, 2, 2, 2, 2, 2, 2, 2, 2, 2],

[2, 3, 3, 3, 1, 1, 4, 2, 4, 4], [2, 4, 4, 4, 3, 3, 1, 2, 1, 1], [3, 1, 1, 1, 2, 2, 4, 3, 4, 4],

[3, 2, 2, 2, 4, 4, 1, 3, 1, 1], [3, 3, 3, 3, 3, 3, 3, 3, 3, 3], [3, 4, 4, 4, 1, 1, 2, 3, 2, 2],

[4, 1, 1, 1, 3, 3, 2, 4, 2, 2], [4, 2, 2, 2, 1, 1, 3, 4, 3, 3], [4, 3, 3, 3, 2, 2, 1, 4, 1, 1],

[4, 4, 4, 4, 4, 4, 4, 4, 4, 4]]

gap > colorlist(bi,L2);

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

[3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3], [4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]]

Y como sus cantidades de coloreos son diferentes, concluyo que son links diferentes.

Ejemplo 4.7. Coloreando, podemos construir una función que nos indique la cantidad de componentes del link. Para ello, usamos al biquandle Flip, dado por:

bi := rec(lperms := [[1,2] , [1,2]] , rperms := [[1,2] , [1,2]])

La regla de coloreo por el Flip, colorea con el mismo elemento, a los semiarcos diagonales, es decir, que colorea igual al semiarco entrante y a su semiarco saliente opuesto (que es su continuación en el diagrama). Luego, las coloraciones por el Flip, serán elegir un elemento para cada componente conexa, es decir, que tendremos 2^r coloraciones, donde $r = \#$componentes conexas del link.

gap > CompConexas := function(k)

> local bi, N ;

> bi := rec(lperms := [[1,2] , [1,2]] , rperms := [[1,2] , [1,2]]);

> N := 2;

> return LogInt(Size(colorlist(bi,k)), 2);

> end;

function(k) ... end

Por ejemplo, si lo aplicamos al enlace de Hopf y a los anillos de Borromeo, obtenemos:

gap > HopfLink := [[-1 , [4,1,2,3]] , [1 , [2,3,4,1]]];;

gap > CompConexas(HopfLink);

2

gap > Borromeo := [[1 , [8,1,2,5]] , [-1 , [11,2,3,12]] , [-1 , [3,6,7,4]] ,

> [1 , [4,9,10,1]] , [1 , [12,5,6,9]] , [-1 , [7,10,11,8]]];;

gap > CompConexas(Borromeo);

3

Función de partición

Para contruir la función de partición, necesitamos tener un 2-cociclo y un link. La construcción de 2-cociclos para que cumplan las condiciones necesarias, la abordaremos en la siguiente sección.

Por el momento, supongamos que tenemos una función $f : X \times X \to A$ que es 2-cociclo de biracks con A un grupo abeliano, que por conveniencia, escribimos en términos de permutaciones. Veamos como obtener su función de partición.

La siguiente función tiene como entrada un nudo y una lista de coloreos, y lo que hace es devolver un lista de nudos coloreados, uno por cada coloreo de la lista.

```
gap > Cols := function(bi,k,colorlist)
> local Col, mk, cmk, i, j ,N ;
> N := Size(bi.lperms[1]);
> Col := [ ];;
> mk := Move(bi,k);
> for i in [1..Size(colorlist)] do
> cmk := StructuralCopy(mk);
> for j in [1..Size(colorlist[1])] do
> cmk := Etiquetado(cmk,N+j, colorlist[i][j]);
> od;
> Add(Col, cmk);
> od;
> return Col;
> end;
function( bi , k , colorlist ) ... end
```

La siguiente función tiene como entrada un nudo, una función que sea 2-cociclo de biracks, una lista de coloreos, y lo que devuelve es una lista de pares. El segundo elemento de cada par, es un elemento del grupo al que llega el 2-cociclo, y el primer elemento es la cantidad de veces que aparece en la sumatoria. Es decir, que en la función están realizadas las productorias y sumatorias de la función de partición:

```
gap > Statesum := function(bi,k,f,colorlist)
> local S,col,bolt,ss,i,j ;
> S := Size(k);
> ss := [ ];
```

```
> col := Cols(bi,k,colorlist);
> bolt := [ ];
> for i in [1..Size(colorlist)] do
> if col[i][1][1] = 1 then
> bolt[i] := f(col[i][1][2][1], col[i][1][2][2]) ;
> else
> bolt[i] := f(col[i][1][2][3], col[i][1][2][4]) ;
> fi;
> od;
> for i in [1..Size(colorlist)] do
> for j in [2..S] do
> if col[i][j][1] = 1 then
> bolt[i] := bolt[i] * f(col[i][j][2][1], col[i][j][2][2]);
> else
> bolt[i] := bolt[i] * f(col[i][j][2][3], col[i][j][2][4]);
> fi;
> od;
> od;
> for i in [1..Size(colorlist)] do
> Add(ss , [ Size(Positions(bolt, bolt[i])), bolt[i] ] );;
> od;
> ss := Set(ss);;
> return ss;
> end;
function( bi , k , f , colorlist ) ... end
```

Ejemplo 4.8. Recordemos el bialexander $X = \mathbb{Z}_2[t]/(t^2 + t + 1)$ dado por $r = 1$ y $t = t$, expresado en código como:

```
bi := rec( lperms := [ [1,2,3,4] , [1,2,3,4] , [1,2,3,4] , [1,2,3,4] ] ,
rperms := [ [1,3,4,2] , [4,2,1,3] , [2,4,3,1] , [3,1,2,4] ] )
```

Tenemos que la siguiente función $f : X \times X \to A$ es un 2-cociclo, con $A = \langle \tau \rangle$ el cíclico de orden 2, dada por:

$$f(x, y) = \begin{cases} 1 & \text{si } x = y \text{ o } x = 1 \text{ o } y = 1, \\ \tau & \text{en otro caso.} \end{cases}$$

Y su código es:

```
gap > cocicle := function(x,y)
> if x=2 or y=2 or x=y then
> return ( );
> else return (1,2);
> fi;
> end;
function( x , y ) ... end
```

Para el nudo trébol del Ejemplo 2.4, el 2-cociclo f y sus coloreos dados por el bialexander, tengo:

```
gap > Statesum(bi,k,cocicle,colorlist);
[ [ 4, () ], [ 12, (1,2) ] ]
```

Lo que interpretamos como $BRZ_{X,f}(k) = 4 + 12\tau \in \mathbb{Z}[A]$

2-Cociclo universal abeliano

Lo que construimos primero, son dos funciones que generen los términos que hay de cada lado de la igualdad de nuestra ecuación, escrita como lista de pares de elementos.

```
gap > Eq1 := function(bi, x, y, z)
> local L ;
> L := [ [x,y],[sigma(bi,x,y)[2],z], [sigma(bi,x,y)[1], sigma(bi,sigma(bi,x,y)[2],z)[1]]
];
> return L;
> end;
function( bi, x, y, z ) ... end
> Eq2 := function(bi, x, y, z)
> local L ;
> L := [ [x, sigma(bi,y,z)[1]], [sigma(bi,x,sigma(bi,y,z)[1])[2], sigma(bi,y,z)[2]],
[y,z] ];
> return L;
> end;
function( bi, x, y, z ) ... end
```

El siguiente paso es considerar estas dos listas, y en caso de tener un mismo término de ambos lados, lo eliminamos. A continuación procedemos a generar una lista de NxN, donde en cada lugar (i,j) está la cantidad de veces que aparece (i,j) en la ecuación, con signo positivo si aparece del lado izquierdo, y signo negativo si aparece del otro lado.

```
gap > EqList := function(bi,x,y,z)
> local C,L,Leq,i,N;
> N := Size(bi.lperms[1]);
> C := Cartesian( [1..N] , [1..N] );
> L := [ Eq1(bi,x,y,z) , Eq2(bi,x,y,z) ];
> Leq := [ ];
> for i in [1..N*N] do
> while C[i] in L[1] and C[i] in L[2] do
> Remove(L[1], Position(L[1] ,C[i] ) );
> Remove(L[2], Position(L[2] , C[i] ) );
> od;
> od;
> for i in [1..N*N] do
> if C[i] in L[1] then
> Leq[i] := Size(Positions(L[1] , C[i]);
> elif C[i] in L[2] then
> Leq[i] := -Size(Positions(L[2] , C[i]));
> else
> Leq[i] := 0;
> fi;
> od;
> return Leq;
> end; function( x, y, z ) ... end
```

Ahora que podemos construir una lista que codifique sencillamente a la ecuación, lo siguiente es ir armando una lista que registre todas las ecuaciones posibles para (x,y,z) y elimine las repetidas, para obtener finalmente una matriz con filas distintas que podamos triangular racionalmente y así quedarnos con las relaciones buscadas.

```
gap > RelX := function(bi)
> local L, i, j, k, N ;
> N := Size(bi.lperms[1]);
> L := [ ];
> for i in [1..N] do
> for j in [1..N] do
```

```
> for k in [1..N] do
> Add(L, EqList(i,j,k) );
> od;
> od;
> od;
> L := Set(L);
> TriangulizeMat(L);
> L := L[1..RankMat(L)];
> return L;
> end;
function( bi ) ... end
```

La siguiente función será el 2-cociclo universal. Toma un birack, y en base a la matriz triangulada de relaciones, genera una lista de NxN elementos, donde en el lugar (i-1)*N+j (que es correspondiente al par (i,j)-esimo), tiene una lista de los coeficientes $(a^{c_r}_{(x_i,x_j)})_r$.

```
gap > vartheta := function(bi)
> local M, N, R, P, V, W, L, i, j, t ;
> N := Size(bi.lperms[1]) ;
> L := [ ] ;
> R := RelX(bi) ;
> V := Size(R);
> P := N*N - V;
> W := Position(R[V],1);
> M := List([1..(N*N)] , i -> List([1..(P)], i -> 0 ) ) ;
> t := 0;
> for i in [1..W] do
> if R[i-t][i] = 1 then
> continue ;
> else
> Add(L, i) ;
> t := t+1 ;
> fi;
> od;
> Append(L, [(W+1)..(N*N)] ) ;
```

```
> t := 0 ;
> for j in [1..(N*N)] do
> if j in L then
> M[j][Position(L, j)] := 1 ;
> t := t+1 ;
> else
> M[j] := -1*R[j-t]L ;
> fi;
> od;
> return M;
> end;
function( bi ) ... end
```

Finalmente, la función de partición con el 2-cociclo universal abeliano, se conseguirá asignándole el correspondiente peso de Boltzmann de ϑ para cada cruce, y luego sumando entre todos los coloreos. Como resultado, obtenemos una lista con dos términos: el segundo será una lista con los valores de los coeficientes de los generadores del grupo universal abeliano sin torsión, y el primero será la cantidad de veces que estos coeficientes aparecen al recorrer todos los coloreos.

```
gap > Universalstatesum := function(bi,k)
> local S,col,bolt,ss,i,j,N,M,C;
> S := Size(k);
> N := Size(bi.lperms[1]);
> M := vartheta(bi) ;
> C := colorlist(bi,k) ;
> ss := [ ];
> col := Cols(bi,k,C);
> bolt := [ ];
> for i in [1..Size(C)] do
> if col[i][1][1] = 1 then
> bolt[i] := M[ (col[i][1][2][1] - 1)*N + col[i][1][2][2] ] ;
> else
> bolt[i] := M[ (col[i][1][2][3] - 1)*N + col[i][1][2][4] ] ;
> fi;
> od;
```

```
> for i in [1..Size(C)] do
> for j in [2..S] do
> if col[i][j][1] = 1 then
> bolt[i] := bolt[i] + M[ (col[i][j][2][1] - 1)*N + col[i][j][2][2] ] ;
> else
> bolt[i] := bolt[i] + M[ (col[i][j][2][3] - 1)*N + col[i][j][2][4] ] ;
> fi;
> od;
> od;
> for i in [1..Size(C)] do
> Add(ss , [ Size(Positions(bolt, bolt[i])), bolt[i] ] );;
> od;
> ss := Set(ss);;
> return ss;
> end;
function( bi, k ) ... end
```

Ejemplo 4.9. En este ejemplo, vamos a probar que los nudos primos de 5, 6 y 7 cruces, son todos no equivalentes. Para ello, vamos a usar los PD-codes elaborados a partir de la Figura 4.5, donde empezamos en el punto rojo del nudo, etiquetando con 1, y seguimos la orientación hacía donde está el número 2:

```
5_1 := [ [ -1 , [ 6,1,2,7 ] ] , [ -1 , [ 2,7,8,3 ] ] , [ -1 , [ 8,3,4,9 ] ] , [ -1 , [ 4,9,10,5 ] ] , [
-1 , [ 10,5,6,1 ] ] ];

5_2 := [ [ -1 , [ 1,6,7,2 ] ] , [ -1 , [ 7,2,3,8 ] ] , [ -1 , [ 3,10,1,4 ] ] , [ -1 , [ 9,4,5,10 ] ] , [
-1 , [ 5,8,9,6 ] ] ];

6_1 := [ [ -1 , [ 1,9,8,2 ] ] , [ -1 , [ 2,7,8,3 ] ] , [ -1 , [ 3,12,1,4 ] ] , [ -1 , [ 11,4,5,12 ] ] ,
[ -1 , [ 5,10,11,6 ] ] , [ -1 , [ 9,6,7,10 ] ] ];

6_2 := [ [ 1 , [ 1,6,7,2 ] ] , [ -1 , [ 2,11,12,3 ] ] , [ -1 , [ 8,3,4,9 ] ] , [ -1 , [ 4,9,10,5 ] ] ,
[ -1 , [ 10,5,6,11 ] ] , [ 1 , [ 7,12,1,8 ] ] ];

6_3 := [ [ -1 , [ 1,10,11,2 ] ] , [ 1 , [ 2,5,6,3 ] ] , [ 1 , [ 8,3,4,9 ] ] , [ 1 , [ 4,9,10,5 ] ] , [
-1 , [ 11,6,7,12 ] ] , [ -1 , [ 7,12,1,8 ] ] ];

7_1 := [ [ -1 , [ 8,1,2,9 ] ] , [ -1 , [ 2,9,10,3 ] ] , [ -1 , [ 10,3,4,11 ] ] , [ -1 , [ 4,11,12,5 ]
] , [ -1 , [ 12,5,6,13 ] ] , [ -1 , [ 6,13,14,7 ] ] , [ -1 , [ 14,7,8,1 ] ] ];

7_2 := [ [ -1 , [ 1,12,13,2 ] ] , [ -1 , [ 11,2,3,12 ] ] , [ -1 , [ 3,10,11,4 ] ] , [ -1 , [ 9,4,5,10
] ] , [ -1 , [ 5,8,9,6 ] ] , [ -1 , [ 13,6,7,14 ] ] , [ -1 , [ 7,14,11,8 ] ] ];
```

7_3 := [[1 , [8,1,2,9]] , [1 , [2,9,10,3]] , [1 , [10,3,4,11]] , [1 , [4,11,12,5]] , [1 , [14,5,6,1]] , [1 , [6,13,14,7]] , [1 , [12,7,8,13]]];

7_4 := [[1 , [1,10,11,2]] , [1 , [9,2,3,10]] , [1 , [3,8,9,4]] , [1 , [11,4,5,12]] , [1 , [5,14,1,6]] , [1 , [13,6,7,14]] , [1 , [7,12,13,8]]];

7_5 := [[-1 , [8,1,2,9]] , [-1 , [2,13,14,3]] , [-1 , [12,3,4,13]] , [-1 , [4,9,10,5]] , [-1 , [10,5,6,11]] , [-1 , [6,11,12,7]] , [-1 , [14,7,8,1]]];

7_6 := [[1 , [1,12,13,2]] , [1 , [11,2,3,12]] , [-1 , [8,3,4,9]] , [-1 , [4,9,10,5]] , [-1 , [14,5,6,1]] , [-1 , [6,13,14,7]] , [-1 , [10,7,8,11]]];

7_7 := [[1 , [1,8,9,2]] , [1 , [7,2,3,8]] , [1 , [3,12,13,4]] , [1 , [11,4,5,12]] , [-1 , [14,5,6,1]] , [-1 , [6,9,10,7]] , [-1 , [10,13,14,11]]];

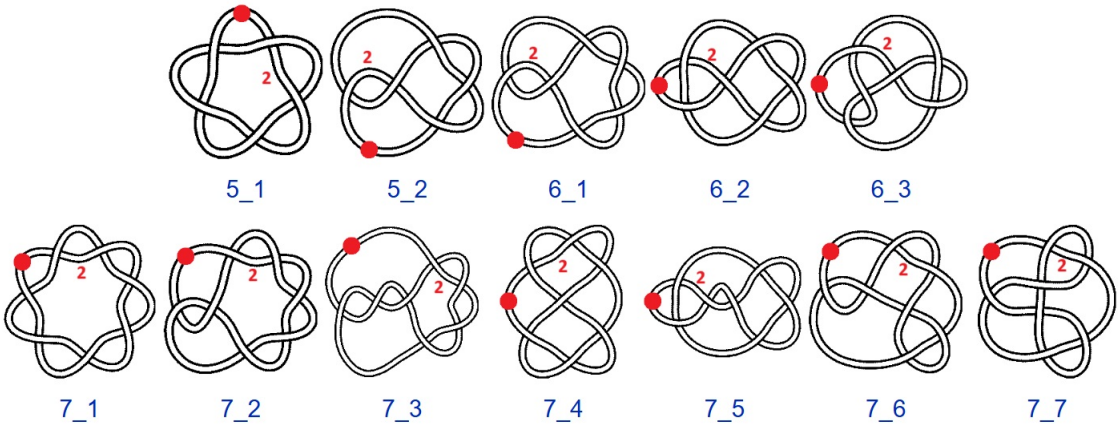


Figura 4.5: Nudos primos de 5, 6 y 7 cruces

Y los biracks serán los siguientes:

- Los biquandles de Wada no triviales hasta orden 7:

WadaZ3 := rec(lperms := [[1,2,3],[1,2,3],[1,2,3]],rperms := [[1,2,3],[3,1,2],[2,3,1]]);
WadaZ4 := rec(lperms := [[1,4,3,2],[1,4,3,2],[1,4,3,2],[1,4,3,2]],
rperms := [[1,2,3,4],[3,4,1,2],[1,2,3,4],[3,4,1,2]]);
WadaZ5 := rec(lperms := [[1,5,4,3,2],[1,5,4,3,2],[1,5,4,3,2],[1,5,4,3,2],[1,5,4,3,2]],
rperms := [[1,2,3,4,5],[3,4,5,1,2],[5,1,2,3,4],[2,3,4,5,1],[4,5,1,2,3]]);
WadaZ6 := rec(lperms := [[1,6,5,4,3,2],[1,6,5,4,3,2],[1,6,5,4,3,2],[1,6,5,4,3,2],
[1,6,5,4,3,2],[1,6,5,4,3,2]],rperms := [[1,2,3,4,5,6],[3,4,5,6,1,2],[5,6,1,2,3,4],
[1,2,3,4,5,6],[3,4,5,6,1,2],[5,6,1,2,3,4]]);
WadaS3 := rec(lperms := [[1,3,2,4,5,6],[1,3,2,5,6,4],[1,3,2,6,4,5],[1,2,3,4,6,5],
[1,2,3,6,5,4],[1,2,3,5,4,6]],rperms := [[1,2,3,4,5,6],[3,1,2,6,4,5],[2,3,1,5,6,4],
[1,2,3,4,5,6],[1,2,3,4,5,6],[1,2,3,4,5,6]]);

WadaZ7 := rec(lperms := [[1,7,6,5,4,3,2],[1,7,6,5,4,3,2],[1,7,6,5,4,3,2],[1,7,6,5,4,3,2],
[1,7,6,5,4,3,2],[1,7,6,5,4,3,2],[1,7,6,5,4,3,2]] , rperms := [[1,2,3,4,5,6,7],[3,4,5,6,7,1,2],
[5,6,7,1,2,3,4],[7,1,2,3,4,5,6],[2,3,4,5,6,7,1],[4,5,6,7,1,2,3],[6,7,1,2,3,4,5]]);

- Birack que no son biquandles:

bi_1 := rec(lperms := [[1,2,3],[1,3,2],[1,3,2]],rperms := [[1,3,2],[1,2,3],[1,2,3]]);
bi_2 := rec(lperms := [[1,3,4,2],[1,4,2,3],[1,4,2,3],[1,4,2,3]],
rperms := [[1,2,3,4],[1,4,2,3],[1,4,2,3],[1,4,2,3]]);

Comenzemos analizando la Tabla 4.1. Algo que observamos rápidamente, es que aunque haya biracks que me den la misma cantidad de coloreos para todos los nudos (como el WadaZ4 por ejemplo), los coeficientes de sus funciones de partición universales los distinguen según su cantidad de cruces. Así que enfoquémosnos en distinguir aquellos que tienen la misma cantidad de cruces.

Los biracks de Wada distinguen 6_1 de 6_2 y 6_3, pero no distingue a estos dos últimos. A 7_4 y 7_5 los distingo entre sí y los distingo del resto mediante coloreos de WadaZ3. También distingo a 7_7 con WadaZ6 o WadaS3.

En esta, observamos que hay nudos que podemos distinguir con coloreos, como los nudos 7_4 y 7_5 con el WadaZ3. También distingo al 7_7 de los demás con coloreos por WadaZ3 o WadaS3.

Y no podemos obtener más información con esta tabla, así que observamos la tabla 4.2. Con bi_1 puedo distinguir a 6_2 de 6_3. También distingo a 7_2 de los demás con bi_1.

Para finalizar, $Col_{WadaZ7}(7\_1) = 49 \neq 7 = Col_{WadaZ7}(7\_3)$, con lo cual probamos que todos los nudos primos de la figura 4.5 son no equivalentes.

Ejemplo 4.10. Ahora, probemos que podemos distinguir entre nudos enmarcados. Para ello, consideramos varios Hopf links donde una de las componentes tiene un twist, como se ve en la figura 4.6.

Para ello vemos la tabla 4.3, donde observamos algo que ya sabiamos: los coloreos por biquandles no distinguen los twist en los links. Pero con la función de partición universal, podemos distinguir si tiene o no un twist. Lo que no distingue es si ese twist es positivo o negativo, así que necesitamos biracks que no sean biquandles.

Con bi_1 podemos distinguir a HLF1 de HLF_1 y a HLF3 de HLF_3. Finalmente con bi_2 podemos identificar a HLF2 de HLF_2.

En la tabla 4.4 podemos ver que bi_1 distingue a todos salvo a HLF2 de HLF_2.

Ejemplo 4.11. Para terminar, podemos observar que es posible obtener funciones de partición con valores racionales, como es el caso para:

bi_3 := rec(lperms := [[3,1,2,4],[3,1,2,4],[3,1,2,4],[3,1,2,4]],
rperms := [[3,2,1,4],[2,1,3,4],[1,3,2,4],[2,3,1,4]])

K$\diagdown$ X	WadaZ3	WadaZ4	WadaZ5	WadaZ6	WadaS3
5_1	c^5	$2c_4^5 + 2c_2^5$	$25c_2^5$	$3c_6^5 + 3c_2^5$	$3c_8^5 + 3c_2^5$
5_2	c^5	$2c_4^5 + 2c_2^5$	$5c_2^5$	$3c_6^5 + 3c_2^5$	$3c_8^5 + 3c_2^5$
6_1	c^6	$c_4^5c_6 + c_4^7c_6^{-1} + 2c_4^6$	c_2^6	$c_6^6 + c_2^6$	$c_2^6 + 3c_8^6$
6_2	$3c^6$	$2c_4^6 + 2c_2^6$	$5c_2^6$	$3c_6^6 + 3c_2^6$	$3c_8^6 + 3c_2^6$
6_3	$3c^6$	$2c_4^6 + 2c_2^6$	$5c_2^6$	$3c_6^6 + 3c_2^6$	$3c_8^6 + 3c_2^6$
7_1	c^7	$2c_4^7 + 2c_2^7$	$5c_2^7$	$3c_6^7 + 3c_2^7$	$3c_8^7 + 3c_2^7$
7_2	c^7	$2c_4^7 + 2c_2^7$	$5c_2^7$	$3c_6^7 + 3c_2^7$	$3c_8^7 + 3c_2^7$
7_3	c^7	$2c_4^7 + 2c_2^7$	$5c_2^7$	$3c_6^7 + 3c_2^7$	$3c_8^7 + 3c_2^7$
7_4	$3c^7$	$2c_4^7 + 2c_2^7$	$25c_2^7$	$9c_6^7 + 9c_2^7$	$9c_8^7 + 9c_2^7$
7_5	$9c^7$	$2c_4^7 + 2c_2^7$	$5c_2^7$	$3c_6^7 + 3c_2^7$	$3c_8^7 + 3c_2^7$
7_6	c^7	$2c_4^7 + 2c_2^7$	$5c_2^7$	$3c_6^7 + 3c_2^7$	$3c_8^7 + 3c_2^7$
7_7	c^7	$2c_4^7 + 2c_2^7$	$5c_2^7$	$2c_1^{-2}c_6^7c_8 + 2c_1^{-2}c_2^9c_3^{-2}c_5^{-3}c_6c_7^3c_8 + 2c_6^9c_8^{-2} + 2c_2^9c_3^{-2} + 2c_1^2c_6^5 + 2c_1^2c_2^3c_3^4c_5^3c_6^{-1}c_7^{-3}c_8^{-1} + 3c_6^7 + 3c_2^7$	$2c_1^{-2}c_2^9 + 2c_5^{-4}c_6^4c_7^{-2}c_8^9 + 2c_5^2c_6^{-2}c_7^{-2}c_8^9 + 2c_5^2c_6^{-2}c_7^4c_8^3 + 2c_2^9c_3^{-2} + 2c_1^2c_2^3c_3^2 + 3c_8^7 + 3c_2^7$

Cuadro 4.1: Funciones de partición universales con biracks de Wada

K$\diagdown$ X	bi_1	bi_2
5_1	c_1^5	c_1^5
5_2	c_1^5	c_1^5
6_1	$c_1^6 + 2c_4^3c_5^3$	c_1^6
6_2	$c_1^6 + 2c_4^3c_5^3$	c_1^6
6_3	$c_1^6 + c_4^4c_5^2 + c_4^2c_5^4$	$c_4^2c_5^6c_6^{-2} + c_4^4c_5^2 + c_4^6c_5^4c_6^{-4} + c_1^6$
7_1	c_1^7	c_1^7
7_2	$c_1^7 + c_4^4c_5^3 + c_4^3c_5^4$	$c_3^3c_5^4 + c_3^4c_5^3 + c_1^7$
7_3	c_1^7	c_1^7
7_4	c_1^7	c_1^7
7_5	c_1^7	c_1^7
7_6	c_1^7	$c_4^2c_5^6c_6^{-1} + c_4^6c_5^2c_6^{-1} + c_4^6c_5^6c_6^{-5} + c_1^7$
7_7	c_1^7	c_1^7

Cuadro 4.2: Más funciones de partición universales

Aplicando la función de partición universal, a los nudos 5_1, 6_1 y 7_1, obtenemos:

gap > Universalstatesum(bi_3,5_1);

[[1, [0, 0, 0, 0, 0, 5]], [3, [5, 0, 10/3, 10/3, -20/3, 0]]]

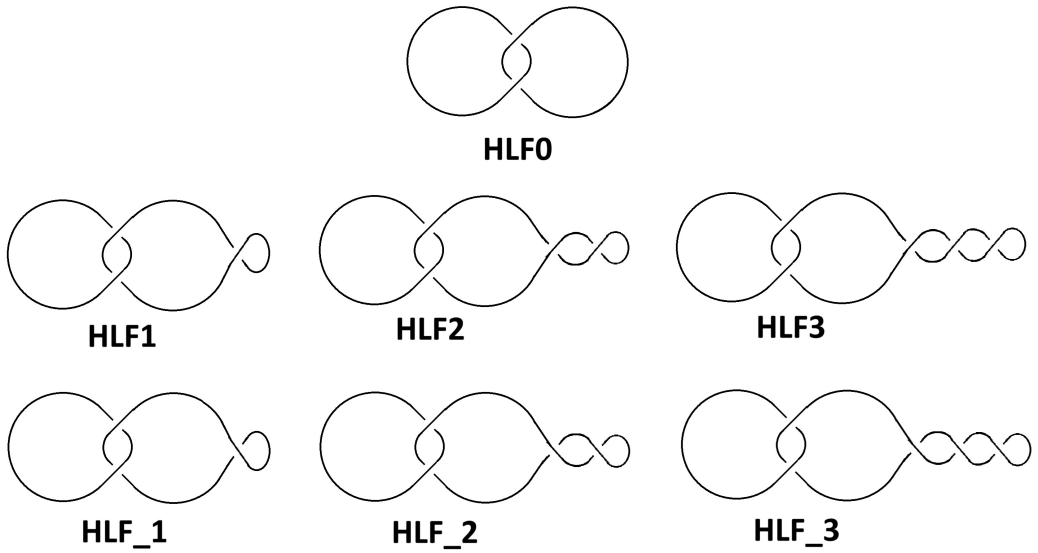


Figura 4.6: Hopf links twisteados

K╲ X	WadaZ6	WadaS3
HLF0	$3c_6^2 + 3c_2^2 + 6c_4c_5c_6c_8^{-1}$	$3c_8^2 + c_2^2 + 18c_4c_5^2c_6^{-1}c_7c_8^{-1}$
HLF1	$3c_6^3 + 3c_4c_5c_6^2c_8^{-1} + 3c_2c_4c_5c_6^1c_8^{-1} + 3c_2^3$	$3c_8^2 + c_2^2 + 9c_4c_5^2c_6^{-1}c_7 + 9c_2c_4c_5^2c_6^{-1}c_7c_8^{-1}$
HLF2	$3c_6^4 + 3c_4c_5c_6^3c_8^{-1} + 3c_2^3c_4c_5c_6^1c_8^{-1} + 3c_2^4$	$3c_8^3 + c_2^3 + 9c_4c_5^2c_6^{-1}c_7c_8 + 9c_2^2c_4c_5^2c_6^{-1}c_7c_8^{-1}$
HLF3	$3c_6^5 + 3c_4c_5c_6^4c_8^{-1} + 3c_2^3c_4c_5c_6^1c_8^{-1} + 3c_2^5$	$3c_8^5 + c_2^5 + 9c_4c_5^2c_6^{-1}c_7c_8^2 + 9c_2^3c_4c_5^2c_6^{-1}c_7c_8^{-1}$
HLF_1	$3c_6^3 + 3c_4c_5c_6^2c_8^{-1} + 3c_2c_4c_5c_6^1c_8^{-1} + 3c_2^3$	$3c_8^2 + c_2^2 + 9c_4c_5^2c_6^{-1}c_7 + 9c_2c_4c_5^2c_6^{-1}c_7c_8^{-1}$
HLF_2	$3c_6^4 + 3c_4c_5c_6^3c_8^{-1} + 3c_2^3c_4c_5c_6^1c_8^{-1} + 3c_2^4$	$3c_8^3 + c_2^3 + 9c_4c_5^2c_6^{-1}c_7c_8 + 9c_2^2c_4c_5^2c_6^{-1}c_7c_8^{-1}$
HLF_3	$3c_6^3 + 3c_4c_5c_6^2c_8^{-1} + 3c_2c_4c_5c_6^1c_8^{-1} + 3c_2^3$	$3c_8^5 + c_2^5 + 9c_4c_5^2c_6^{-1}c_7c_8^2 + 9c_2^3c_4c_5^2c_6^{-1}c_7c_8^{-1}$

Cuadro 4.3: Identificando los twist con biquandles

K╲ X	bi_1	bi_2
HLF0	c_1^2	c_1^2
HLF1	$c_2c_3c_5 + c_2c_3c_4^2c_5^{-1} + c_1^3$	$c_2c_3c_5 + c_2c_3c_4^2c_5^{-1} + c_2c_3c_4^2c_5c_6^{-2} + c_1^3$
HLF2	c_1^4	c_1^4
HLF3	$c_2c_3c_4c_5 + c_2c_3c_4^3 + c_1^5$	c_1^5
HLF_1	$c_1^3 + 2c_2c_3c_4$	c_1^3
HLF_2	c_1^4	$c_1^4 + 3c_2c_3c_4^2c_5c_6^{-1}$
HLF_3	$c_1^5 + 2c_2c_3c_4^2c_5$	c_1^5

Cuadro 4.4: Identificando los twist con biracks que no son biquandles

gap > Universalstatesum(bi_3 ,6_1);

[[1, [0, 0, 0, 0, 0, 6]], [3, [6, 0, 4, 4, -8, 0]]]

```
gap > Universalstatesum(bi_3 ,7_1);
[ [ 1, [ 0, 0, 0, 0, 0, 7 ] ], [ 3, [ 7, 0, 14/3, 14/3, -28/3, 0 ] ] ]
```

Anexo

Códigos para copiar y pegar

Código de coloreos

```
sigma := function(bi, x, y)
return [bi.lperms[x][y], bi.rperms[y][x]];
end;
Cantarcos := function(k)
local c,i;
c := 1;
for i in [1..Size(k)] do
if c < k[i][2][1] then
c := k[i][2][1];
elif c < k[i][2][2] then
c := k[i][2][2];
else continue;
fi;
od;
return c;
end;
Move := function(bi,k)
local ck,i,S,N;
N := Size(bi.lperms[1]);
S := Size(k);
ck := StructuralCopy(k);
for i in [1..S] do
ck[i][2] := List (ck[i][2] , x -> x+N);
```

```
od;
return ck;
end;
Etiquetado := function(k,r,x)
local P, R,i,S;
S := Size(k);
for i in [1..S] do
if Position(k[i][2],r) <> fail then
P := Positions(k[i][2],r);
R := List([1..Size(P)], i -> x);
k[i][2]P := R;
else
continue;
fi;
od;
return k;
end;
checkX := function(bi,mk)
local l,i,S,N;
N := Size(bi.lperms[1]);
S := Size(mk);
l := [];;
for i in [1..S] do
if Number(mk[i][2], n -> n < N+1)=4 then
if (mk[i][1])=1 then
l[i] := sigma(bi, mk[i][2][1] , mk[i][2][2] )=[ mk[i][2][3] , mk[i][2][4] ];
else
l[i] := sigma(bi, mk[i][2][3] , mk[i][2][4] )=[ mk[i][2][1] , mk[i][2][2] ];
fi;
elif Number(mk[i][2], n -> n < N+1)=3 then
if (mk[i][1])=1 then
if mk[i][2][1]> N then
```

```
l[i] := mk[i][2][3] = sigma(bi, Position(bi.rperms[mk[i][2][2]] , mk[i][2][4] ),
mk[i][2][2] )[1];
elif mk[i][2][2]> N then
l[i] := mk[i][2][4] = sigma(bi, (mk[i][2][1]), Position(bi.lperms[ mk[i][2][1] ],
mk[i][2][3] ) )[2];
elif mk[i][2][3]> N then
l[i] := mk[i][2][4] = sigma(bi, mk[i][2][1],mk[i][2][2])[2];
else
l[i] := mk[i][2][3] = sigma(bi, mk[i][2][1],mk[i][2][2])[1];
fi;
else
if mk[i][2][1]> N then
l[i] := mk[i][2][2] = sigma(bi, mk[i][2][3] , mk[i][2][4] )[2];
elif mk[i][2][2]> N then
l[i] := mk[i][2][1] = sigma(bi, mk[i][2][3] , mk[i][2][4] )[1];
elif mk[i][2][3]> N then
l[i] := mk[i][2][1] = sigma(bi, Position(bi.rperms[ mk[i][2][4] ], mk[i][2][2] ),
mk[i][2][4] )[1];
else
l[i] := mk[i][2][2] = sigma(bi, mk[i][2][3], Position(bi.lperms[ mk[i][2][3] ], mk[i][2][1]
) )[2];
fi;
fi;
else
l[i] := true;
fi;
od;
if false in l then
return false;
else
return true;
fi;
end;
check := function(L,Extra)
```

```
local cmk,i,S,N;
N := Extra[1];
S := Size(Extra[2]);
cmk := StructuralCopy(Extra[2]);
for i in [1..Size(L)] do
cmk := Etiquetado(cmk,N+i,L[i]);
od;
return checkX(Extra[3],cmk);
cmk := Extra[2];
end;
colorlist := function(bi,k)
local C,mk, N, ext ;
N := Size(bi.lperms[1]);
C := Cantarcos(k);
mk := Move(bi,k);
ext := [N,mk,bi];
return BacktrackBag([1..N],check,C,ext);
end;
```

Función de partición para un 2-cociclo de permutaciones

```
Cols := function(k,colorlist)
local Col, mk, cmk, i, j ;
Col := [ ];;
mk := Move(k);
for i in [1..Size(colorlist)] do
cmk := StructuralCopy(mk);
for j in [1..Size(colorlist[1])] do
cmk := Etiquetado(cmk,N+j, colorlist[i][j]);
od;
Add(Col, cmk);
od;
return Col;
end;
```

```
Statesum := function(k,f,colorlist)
local S,col,bolt,ss,i,j ;
S := Size(k);
ss := [ ];
col := Cols(k,colorlist);
bolt := [ ];
for i in [1..Size(colorlist)] do
if col[i][1][1] = 1 then
bolt[i] := f(col[i][1][2][1], col[i][1][2][2]) ;
else
bolt[i] := f(col[i][1][2][3], col[i][1][2][4]) ;
fi;
od;
for i in [1..Size(colorlist)] do
for j in [2..S] do
if col[i][j][1] = 1 then
bolt[i] := bolt[i] * f(col[i][j][2][1], col[i][j][2][2]);
else
bolt[i] := bolt[i] * f(col[i][j][2][3], col[i][j][2][4]);
fi;
od;
od;
for i in [1..Size(colorlist)] do
Add(ss , [ Size(Positions(bolt, bolt[i])), bolt[i] ] );;
od;
ss := Set(ss);;
return ss;
end;
```

Función de partición universal

```
Cols := function(bi,k,colorlist)
local Col, mk, cmk, i, j ,N ;
N := Size(bi.lperms[1]);
```

```
Col := [ ];;
mk := Move(bi,k);
for i in [1..Size(colorlist)] do
cmk := StructuralCopy(mk);
for j in [1..Size(colorlist[1])] do
cmk := Etiquetado(cmk,N+j, colorlist[i][j]);
od;
Add(Col, cmk);
od;
return Col;
end;
Eq1 := function(bi,x, y, z)
local L ;
L := [ [x,y] , [sigma(bi,x,y)[2] , z] , [sigma(bi,x,y)[1] , sigma(bi, sigma(bi,x,y)[2]
, z)[1] ] ];
return L;
end;
Eq2 := function(bi,x, y, z)
local L ;
L := [ [x ,sigma(bi ,y ,z)[1] ] , [sigma(bi ,x ,sigma(bi ,y ,z)[1]) [2] ,sigma(bi ,y
,z)[2] ] , [y ,z] ];
return L;
end;
EqList := function(bi,x,y,z)
local C,L,Leq,i,N;
N := Size(bi.lperms[1]);
C := Cartesian( [1..N] , [1..N] );
L := [ Eq1(bi,x,y,z) , Eq2(bi,x,y,z) ];
Leq := [ ];
for i in [1..N*N] do
while C[i] in L[1] and C[i] in L[2] do
Remove(L[1], Position(L[1] ,C[i] ) );
Remove(L[2], Position(L[2] , C[i] ) );
```

```
od;
od;
for i in [1..N*N] do
if C[i] in L[1] then
Leq[i] := Size(Positions(L[1] , C[i]));
elif C[i] in L[2] then
Leq[i] := -Size(Positions(L[2] , C[i]));
else
Leq[i] := 0;
fi;
od;
return Leq;
end;
RelX := function(bi)
local L, i, j, k, N ;
N := Size(bi.lperms[1]);
L := [ ];
for i in [1..N] do
for j in [1..N] do
for k in [1..N] do
Add(L, EqList(bi,i,j,k) );
od;
od;
od;
L := Set(L);
TriangulizeMat(L);
L := L[1..RankMat(L)];
return L;
end;
vartheta := function(bi)
local M, N, R, P, V, W, L, i, j, t ;
N := Size(bi.lperms[1]) ;
L := [ ] ;
```

```
R := RelX(bi) ;
V := Size(R);
P := N*N - V;
W := Position(R[V],1);
M := List([1..(N*N)] , i -> List([1..(P)], i -> 0 ) ) ;
t := 0;
for i in [1..W] do
if R[i-t][i] = 1 then
continue ;
else
Add(L, i) ;
t := t+1 ;
fi;
od;
Append(L, [(W+1)..(N*N)] ) ;
t := 0 ;
for j in [1..(N*N)] do
if j in L then
M[j][Position(L, j)] := 1 ;
t := t+1 ;
else
M[j] := -1*R[j-t]L ;
fi;
od;
return M;
end;
Universalstatesum := function(bi,k)
local S,col,bolt,ss,i,j,N,M,C;
S := Size(k);
N := Size(bi.lperms[1]);
M := vartheta(bi) ;
C := colorlist(bi,k) ;
ss := [ ];
```

```
col := Cols(bi,k,C);
bolt := [ ];
for i in [1..Size(C)] do
if col[i][1][1] = 1 then
bolt[i] := M[ (col[i][1][2][1] - 1)*N + col[i][1][2][2] ] ;
else
bolt[i] := M[ (col[i][1][2][3] - 1)*N + col[i][1][2][4] ] ;
fi;
od;
for i in [1..Size(C)] do
for j in [2..S] do
if col[i][j][1] = 1 then
bolt[i] := bolt[i] + M[ (col[i][j][2][1] - 1)*N + col[i][j][2][2] ] ;
else
bolt[i] := bolt[i] + M[ (col[i][j][2][3] - 1)*N + col[i][j][2][4] ] ;
fi;
od;
od;
for i in [1..Size(C)] do
Add(ss , [ Size(Positions(bolt, bolt[i])), bolt[i] ] );;
od;
ss := Set(ss);;
return ss;
end;
```

Bibliografía

[24] *GAP – Groups, Algorithms, and Programming, Version 4.13.1*. The GAP Group. 2024. URL: %7Bhttps://www.gap-system.org%7D.

[AG04] N. Andruskiewitsch y M. Graña. *From racks to pointed Hopf algebras*. Adv. Math178, 2004.

[BZH14] G. Burde, H. Zieschang y M. Heusener. *Knots*. Vol. 5. De Gruyter Studies in Mathematics. 2014.

[Car+04] J.S. Carter et al. *Cocycle knot invariants from quandle modules and generalized quandle homology*. Osaka J. Math, 2004.

[Ced+19] C. Cedillo et al. *YAGS - Yet Another Graph System, Version 0.0.5*. http://xamanek.izt.uam.mx/yags/. 2019.

[ESS99] P. Etingof, T. Schedler y A. Soloviev. *Set-theoretical solutions to the quantum Yang-Baxter equation*. Duke Math, 1999.

[Far24] M. Farinati. *Biracks: a notational proposal and applications*. 2024. URL: https://arxiv.org/pdf/2407.07650.

[FG16] M. Farinati y J. García Galofre. *Link and knot invariants from non-abelian Yang-Baxter 2-cocycles*. Vol. 25. Journal of Knot Theory and Its Ramifications. 2016.

[Liv94] C. Livingston. *Knot Theory*. The Carus Mathematical Monographs, Volumen Twenty Four. The Mathematical Association of America, 1994.

[Nel14] S. Nelson. *Link invariants from finite biracks*. Banach Center Publications, 2014.

[Rol04] D. Rolfsen. *Knots and Links*. American Mathematical Society, 2004.