



ugr | Universidad
de Granada

TRABAJO FIN DE GRADO

DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS

Entendiendo la teoría de nudos mediante la simulación y la informática gráfica.

Autora

Cristina Zuheros Montes.

Tutores

Alejandro J. León Salas.
Antonio Martínez López.



Facultad de
Ciencias



—
Granada, Diciembre de 2016

Entendiendo la teoría de nudos mediante la simulación y la informática gráfica.

Cristina Zuheros Montes.

Palabras clave: nudo, enlace, nudo trivial, invariante, trenza, trenza cerrada, palabra, cruce, handle, toolbox

Resumen

En este proyecto comenzamos analizando la teoría de nudos: explicamos qué se entiende por un nudo, vemos un breve recorrido por la historia de dicha teoría mostrando sus aplicaciones más destacadas, estudiamos la composición de nudos, analizamos una de las cuestiones que más nos interesan como es la equivalencia entre nudos, mostramos algunos invariantes de nudos, las notaciones más usuales de los mismos y concluimos relacionando dicha teoría con la teoría de grafos y la teoría de trenzas.

Como ya hemos comentado, uno de los problemas clave que tratamos en este proyecto es la equivalencia de nudos. Encontramos solución usando invariantes de nudos, pero esta solución no es completa en el sentido de que no siempre vamos a obtener respuesta usando invariantes. Por este motivo, entre otros que iremos viendo, vamos a orientar nuestro trabajo al estudio de las trenzas, donde una relación directa entre nudos y trenzas cerradas nos permitirá plantear el problema de la equivalencia de nudos como el problema de la equivalencia de trenzas.

Abordamos el estudio general de las trenzas con la definición formal de las mismas. Vemos que el conjunto de las n -trenzas tiene estructura de grupo no abeliano, introducimos la equivalencia de trenzas y de trenzas cerradas, analizamos algunos invariantes para trenzas y trenzas cerradas y estudiamos el problema de las palabras que nos permite determinar si dos trenzas dadas serán o no equivalentes.

El desarrollo práctico para determinar si dos trenzas son o no equivalentes conlleva más trabajo del que puede parecer a simple vista. El análisis de una gran cantidad de trenzas se convierte en un trabajo muy tedioso y repetitivo. Es por eso que interesa desarrollar software que nos permita trabajar con trenzas.

Por este motivo, finalizamos presentando toxtrén: se trata de un toolbox que hemos creado en Matlab para trabajar tanto con trenzas como con trenzas cerradas. Mostramos cómo podemos instalar toxtrén de una forma realmente simple. Además, presentamos los pseudo algoritmos más destacados para trenzas y trenzas cerradas.

Por último, presentamos una guía en la que mostramos los comandos necesarios para trabajar con toxtrén.

En conclusión, hemos estudiado dos teorías relevantes en la rama de la topología como son la teoría de nudos y la teoría de trenzas, hemos visto la relación que hay entre ambas y el impacto que tienen hoy en día en diversas áreas de conocimiento. Finalmente, hemos desarrollado toxtrén, toolbox en el que implementamos todos los conceptos y algoritmos que hemos analizado en la teoría de trenzas.

Understanding knot theory through simulation and computer graphics.

Cristina Zuheros Montes.

Keywords: knot, link, trivial knot, invariant, braid, closed braid, word, crossing, toolbox, handle.

Abstract

Within mathematical science we find a branch, known as topology, which is in charge of the study of the properties of those objects (known as topological spaces) that, by means of continuous transformations on the objects, remain invariant. Although an exact date of the origin of the topology is not known, it is usually established in 1735 with the resolution, by Leonhard Euler, of the problem of the seven bridges of Königsberg.

Within topology arises the theory of knots that emerged about 200 years ago. We can say that it is a fairly recent theory and the most interesting results have been discovered in the last decades. This is one of the reasons why knot theory is very striking for mathematical researchers as well as for biologists, physicists, chemists, sailors, survival experts and a large number of scientists who find in this theory a response to their approaches. The attractiveness of this theory is that, being relatively recent, it still has many open questions on which one can work.

At the beginning of this project, the theory of knots is explained from a mathematical point of view. We present a formal definition of knot and link. Straightaway, we visualize some examples of knots in the three-dimensional space and its projection.

Next, we see a brief but interesting tour on the history of this theory and we show some of its most outstanding applications. It may seem surprising that we are able to connect knot theory to DNA structure, but nowadays in many laboratories, it is essential to have an expert in knot theory in order to understand what has been the process that has been happened to obtain a final DNA structure.

Once we know the notion of knot and we have some motivation to work with them, we study the composition of knots. Also at this point, we see what is meant by a prime or compound knot, we define an oriented knot and we examine when a knot is invertible. We show some projections of prime knots having less than 8 crossings.

Next we are concerned with one of the most striking aspects of this theory: the equivalence of knots. We consider the basic movements, known as Reidemeister movements, that allow us to modify the projection of the knot without modifying the knot itself. We expose Reidemeister's theorem, which allows us to determine if two projections represent the same knot. However, this process is really complicated and we have no guarantee that this algorithm ends in reasonable finite time.

For this reason, we analyze some invariants for knots. We present a formal definition of the invariant concept and we study the number of components of a link, the crossing number of a knot, tricolorability, the unknotting number and Alexander's polynomial. In order to understand all these concepts, we apply the different invariants over some projections of knots.

Afterwards, we see some of the most common knots's notations. In particular, we analyze Dowker's notation, Gauss's notation, and Conway's notation. This last notation, by the specific way of projecting knots, has a special interest in the study of the structure of DNA.

Finally, we connect this theory to other great theories such as graph theory and braid theory. We have previously commented that topology arises after the resolution of the problem of the bridges of Königsberg. It is interesting to comment the fact that the resolution of this problem lead to the graph theory. Therefore, it is not surprising that knot theory can be related to graph theory. We prove how we can construct a graph from a knot and how we could construct a knot from a graph.

Concerning braid theory, we justify the relationship a bit more deeply. In a relatively simple way we can get a knot from a given braid. To obtain the braid representing a knot we use the Alexander's theorem.

As we have previously discussed, one of the key problems that we are dealing in this project is the equivalence of knots, but with the algorithm provided in the Reidemeister's theorem we can not always obtain an answer in an acceptable time. We found a solution that uses invariants of knots, but this solution is not complete in the sense that we will not always get response using invariants. For this reason, we focus on braid theory. Having a direct relationship between knots and closed braids, we can consider the problem of knot equivalence as the problem of the equivalence of closed braids. In addition, braid theory has direct application to cryptography and fluid mechanics, among other subjects.

Regarding braid theory, we begin showing a formal definition of braid, the strings of a braid and a closed braid. In addition, we display some examples of braids in three-dimensional space.

As we do in knot theory, we study the notion of braid equivalence. Now we only give an overview of equivalence based on the elementary movements. We show a really simple example so that you can intuitively have the idea in your mind. Subsequently, we study this concept in more detail but it is important to have an initial idea to be able to obtain a projection of a braid that is in accordance with the definition that we give, which will be the easiest projection to manipulate braids. In addition, we show the notation for braids that we use defining the positive and negative crossings. We define the trivial n-braid, the word of a braid and we see an example of braid with its pertinent notation.

Below we study one of the most important aspect to consider in braid theory: by providing the set of all braids equivalent to each other of the product of braids we find a non-abelian group structure. We define the product of braids and we check the properties of the non-abelian group with some examples ir order to make proofs as intuitive as possible.

Once we have defined the group structure of the braids, we examine the notion of equivalence of braids in a more formal way. We define the group of equivalent braids by equivalence relations or basic movements. Clearly if two braids are equivalent, the braid closures will also be equivalent. The problem that we find is that if two braids are not equivalent, their closures maybe be equivalent. That is, the closed braids could be equivalent even though the respective braids are not equivalent. We study this idea through the Markov's movements: conjugation and stabilization.

As we commented in the knot theory, this notion of equivalence of braids and Markov-equivalent braids is not feasible to put into practice because we would have to apply a number of possible movements that might not finish in a reasonable time. Therefore, we return to study some of the most essential invariants. In this case, we study the exponent and the permutation of a braid and Alexander's polynomial of a closed braid.

However, these invariants will not let us always to know if two braids are or are not equivalent to each other. The problem of finding some method to know if the words of braids are equivalent or not is known as the problem of words. In this project we study the method of Patrick Dehornoy to solve this problem. The problem of analyzing whether or not two braids are equivalent can be seen as the problem of determining whether a given braid is equivalent to the trivial braid. This idea is essential in Dehornoy's method: we can determine in a relatively simple way if a word of a given braid is equivalent to the trivial braid. At this point, we use also the notion of handle.

Analyzing braids by hand implies more work than it may seem to the naked eye. In addition, if we would like to analyze a large number of braids, it becomes a really tedious and repetitive work. Because of this, it is interesting to develop software that allows us to work with braids.

In this project, we end up presenting `toxtren`: this is a toolbox that we have implemented in Matlab to work with braids and closed braids. We apply an object-oriented design by creating a class for each type of object. We show how we can install `toxtren` in a really simple way.

In addition, we present the most outstanding pseudo algorithms for both classes. In particular, we present the Dehornoy algorithm for braids, the algorithm of equivalences for closed braids and braids and the algorithm that allows us to know if a braid or a closed braid given is equivalent to the trivial braid. It is important to remember that for closed braids we will not always be able to obtain a positive or negative answer for these two last algorithms because we do not have mathematical algorithms that allow it. It is an issue that remains open and captures the attention of many researchers.

Next, we present some algorithms that we have created to represent closed braids and braids in the three-dimensional space.

Finally, we present a guide in which we show the basics commands necessary to work with `toxtren`. We have allowed the user to introduce braids in different ways to make it as comfortable as possible.

In conclusion, we have studied two relevant theories in the branch of topology such as knot theory and braid theory, we have seen the relationship between both theories and the impact that they have nowadays in different areas of knowledge. Finally, we have developed tox tren, a toolbox in which we implement all this concepts and algorithms that we have analyzed in braid theory.

Yo, **Cristina Zuheros Montes**, alumna de la titulación Doble Grado en Ingeniería Informática y Matemáticas de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación** y de la **Facultad de Ciencias** de la **Universidad de Granada**, con DNI XXXXXXXXXX, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca de ambos centros para que pueda ser consultada por las personas que lo deseen.

Fdo: Cristina Zuheros Montes.

Granada a 13 de diciembre de 2016.

D.Alejandro J. León Salas, Profesor del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada.

D.Antonio Martínez López, Profesor del Departamento de Geometría y Topología de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Entendiendo la teoría de nudos mediante la simulación y la informática gráfica***, ha sido realizado bajo su supervisión por **Cristina Zuheros Montes**, y autoriza la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a 13 de diciembre de 2016.

Los directores:

Alejandro J. León Salas

Antonio Martínez López

Agradecimientos

Quiero mostrar mi agradecimiento a la Universidad de Granada y a sus profesores por la oportunidad que me han brindado para formarme simultáneamente en dos grandes áreas como son matemáticas e informática.

A mis tutores, Antonio Martínez y Alejandro León, por la ayuda, atención y apoyo constante que me han proporcionado desde el primer momento.

A mi familia y amigos por la comprensión y enseñanzas que me han proporcionado.

Índice general

1	Introducción	1
1.1	Contextualización.	1
1.2	Problema abordado.	2
1.3	Técnicas y áreas matemáticas e informáticas.	2
1.4	Contenido de la memoria.	2
1.5	Fuentes principales.	3
1.6	Objetivos.	3
2	Teoría de nudos.	5
2.1	Motivación y primeras definiciones.	5
2.2	Sobre su historia y aplicaciones.	6
2.3	Componiendo nudos.	8
2.4	Equivalencia de nudos: movimientos de Reidemeister.	11
2.5	Algunos invariantes.	14
2.5.1	Número de componentes:	14
2.5.2	Crossing number:	15
2.5.3	Tricolorabilidad:	15
2.5.4	Unknotting number:	19
2.5.5	Polinomio de Alexander:	20
2.6	Notación de nudos.	23
2.6.1	Notación de Dowker:	23
2.6.2	Notación de Gauss:	24
2.6.3	Notación de Conway:	25
2.7	Conexión con distintas teorías.	28
2.7.1	Teoría de grafos:	28
2.7.2	Teoría de trenzas:	29
3	Teoría de trenzas.	33
3.1	Fundamentos.	34
3.1.1	Equivalecia de trenzas:	34
3.1.2	Proyección de una trenza.	35
3.1.3	Notación de trenzas:	36
3.2	El grupo de las trenzas.	38
3.2.1	Estructura de grupo no abeliano:	38
3.2.2	Trenzas equivalentes:	44
3.2.3	Trenzas Markov-equivalentes:	47

ÍNDICE GENERAL

3.3	Algunos invariantes.	49
3.3.1	Exponente:	50
3.3.2	Permutación:	52
3.3.3	Polinomio de Alexander:	53
3.4	El problema de las palabras.	61
3.5	Conclusiones	67
4	Desarrollo informático.	69
4.1	Teoría de trenzas con Matlab.	70
4.2	Pseudoalgoritmos.	72
4.3	Usando toxtren	85
4.3.1	Clase trenza.	85
4.3.2	Clase trenza_cerrada.	89
5	Conclusiones y vías futuras	91
	Bibliografía	93

Índice de figuras

2.1	De izquierda a derecha: nudo trivial, nudo trébol, nudo de ocho.	5
2.2	Proyecciones del nudo trivial, nudo trébol, nudo de ocho.	6
2.3	Nudo trébol y nudo de ocho.	8
2.4	Composición de nudo trébol y nudo de ocho.	8
2.5	Nudo trébol y nudo trivial.	8
2.6	Composición de nudo trébol y nudo trivial.	9
2.7	Tabla de nudos primos.	10
2.8	Ambas orientaciones del nudo trébol.	10
2.9	Nudo factor J y K.	11
2.10	Las composiciones no son equivalentes.	11
2.11	Primer movimiento Reidemeister.	12
2.12	Segundo movimiento de Reidemeister.	12
2.13	Tercer movimiento de Reidemeister.	12
2.14	Equivalencia de dos proyecciones de nudos.	13
2.15	Enlace con dos componentes.	14
2.16	Crossing number del nudo trébol.	15
2.17	Tipo de cruce.	16
2.18	Demo R1	16
2.19	Demo R2	17
2.20	Demo R2	17
2.21	Demo R3	18
2.22	Dos nudos no equivalentes.	19
2.23	Unknotting number trébol.	19
2.24	Tipos de cruces.	20
2.25	Cambio cruces trébol	21
2.26	Cambio cruces	21
2.27	Árbol de resolución del nudo trébol.	23
2.28	Numeración de cruces-Dowker.	24
2.29	Numeración de cruces-Gauss.	24
2.30	Enredos	25
2.31	Tipos básicos de enredos.	26
2.32	Operaciones con enredos.	26
2.33	Enlace -2 3 2	26
2.34	Enlaces equivalentes	27
2.35	Ejemplos de grafos planos.	28
2.36	De proyección a grafo	28

ÍNDICE DE FIGURAS

2.37 De grafo a proyección.	29
2.38 Ejemplos de trenzas	30
2.39 Transformando la proyección de un cruce.	30
2.40 Arco de reducción.	31
2.41 Movimiento de reducción.	31
2.42 Movimiento de reducción.	32
2.43 Algoritmo Yamada-Vogel.	32
3.1 Ejemplos de trenzas	33
3.2 Trenzas equivalentes.	34
3.3 Movimiento elemental	35
3.4 Proyección de una trenza	36
3.5 Proyección de trenzas	36
3.6 Signo cruce.	37
3.7 Trenza $\sigma 3\sigma 2^{-1}\sigma 4$	37
3.8 Producto trenzas	38
3.9 Equivalencia producto	39
3.10 Asociatividad producto	40
3.11 No conmutatividad producto	41
3.12 Primer movimiento	42
3.13 Trenzas inversas	43
3.14 Primer movimiento.	44
3.15 Segundo movimiento.	44
3.16 Tercer movimiento.	45
3.17 Trenzas equivalentes.	45
3.18 Trenzas equivalentes.	46
3.19 Trenzas Markov-equivalentes.	47
3.20 Conjugación Markov.	48
3.21 Estabilización Markov.	48
3.22 Trenzas Markov-equivalentes.	49
3.23 Trenza $\sigma 3\sigma 2^{-1}\sigma 3^{-1}$	50
3.24 Trenzas no equivalentes.	51
3.25 Trenzas con mismo exponente.	51
3.26 Trenza $\sigma 3\sigma 2^{-1}\sigma 3^{-1}$	52
3.27 Trenzas no equivalentes.	53
3.28 Trenza no libremente reducida.	62
3.29 Main handle.	62
3.30 A handle.	63
3.31 Reducción local.	64
3.32 Trenzas equivalentes.	65
4.1 Estructura clases	71
4.2 Documentación Toxtren.	71
4.3 Funciones base	78
4.4 Giros cruce	84
4.5 Representación trenza Matlab.	87
4.6 Transformación Dehornoy Matlab.	88

ÍNDICE DE FIGURAS

ÍNDICE DE FIGURAS

Capítulo 1

Introducción

1.1 Contextualización.

Dentro de la ciencia matemática nos encontramos una rama, conocida como topología, que se encarga del estudio de las propiedades de aquellos cuerpos geométricos inalterables por deformaciones. Aunque no se conoce una fecha exacta del origen de la topología, se suele establecer en 1735 con la resolución, por parte de Leonhard Euler, del problema de los 7 puentes de Königsberg.

Dentro de la topología surge la teoría de nudos que sólo tiene algo más de unos 200 años. Se puede decir que es una teoría bastante reciente y cuyos resultados más interesantes se han ido descubriendo en las últimas décadas. Es por esto que la teoría de nudos es muy llamativa tanto para investigadores matemáticos como para biólogos, físicos, químicos, marineros, expertos en supervivencia y una gran cantidad de científicos que encuentra en esta teoría una respuesta a sus planteamientos. Lo atractivo de dicha teoría es que, al ser relativamente reciente, aún tiene muchas cuestiones abiertas en las que se puede trabajar.

En el siglo XIX, algunos científicos trataron de relacionar la teoría de nudos con la química, asociando cada nudo con un elemento de la naturaleza. De este modo se crearía una tabla de nudos que reemplazaría a la tabla periódica actual. Aunque posteriormente este planteamiento fue rechazado, sirvió como base para incitar el estudio de dicha teoría.

A día de hoy, uno de los campos más relevantes en el que encontramos utilidad práctica de la teoría de nudos es en el campo de la biología. En concreto, para analizar y justificar las modificaciones que se producen sobre la estructura del ADN estudiaremos la teoría de nudos.

Además, a lo largo de este proyecto veremos que se ha podido establecer una relación directa entre la teoría de nudos y la teoría de trenzas. Estudiaremos con detalle dicha relación y ambas teorías.

1.2 Problema abordado.

El problema que abordamos en este proyecto consiste en la realización de un estudio teórico sobre teoría de nudos y teoría de trenzas, así como la implementación de un toolbox en Matlab donde recogemos todos los aspectos matemáticos desarrollados sobre teoría de trenzas.

El estudio teórico de la teoría de nudos aborda especialmente la composición, equivalencia, invariantes y notación de nudos. Además, mostramos la relación entre teoría de nudos y teoría de trenzas. El estudio teórico de la teoría de trenzas muestra la estructura de grupo de las trenzas, algunos invariantes y la equivalencia de trenzas, centrándonos en el problema de las palabras.

Finalmente, se ha desarrollado un programa informático que implementa los aspectos vistos en teoría de trenzas. Dicho programa también nos permite trabajar con trenzas cerradas que serán equivalentes a nudos.

1.3 Técnicas y áreas matemáticas e informáticas.

Por una parte, las principales áreas matemáticas en las que nos hemos basado para realizar este proyecto son:

- Topología I.
- Taller de Geometría y Topología.
- Álgebra - Teoría de grupos.
- Geometría.

Por otra parte, las principales áreas y técnicas informáticas en las que nos hemos basado son:

- Fundamentos de programación.
- Informática gráfica.
- Programación y diseño orientado a objetos.

En este punto cabe comentar que todas las imágenes que mostramos a lo largo de este proyecto, salvo la figura 2.7 que referenciamos posteriormente, han sido creadas expresamente para este proyecto mediante Adobe Photoshop CS6 o directamente haciendo uso de toxtren.

1.4 Contenido de la memoria.

En el capítulo 2 recogemos los conceptos y resultados matemáticos referentes a teoría de nudos. En la sección 2.1 proporcionamos definiciones formales con las que trabajar con los nudos. En la sección 2.2 damos un recorrido por la historia de la teoría de nudos y vemos algunas de sus aplicaciones más destacadas. En la sección 2.3 estudiamos la composición de nudos. A continuación, en la sección 2.4 estudiamos la equivalencia de nudos mediante los movimientos de

Reidemeinter. En la sección 2.5 estudiamos algunos de los invariantes más destacados y útiles de nudos. En la sección 2.6 mostramos algunas notaciones de nudos y en la última sección vemos la conexión con la teoría de grafos y de trenzas.

En el capítulo 3 recogemos los conceptos y resultados matemáticos referentes a teoría de trenzas. En la sección 3.1 analizamos la equivalencia, proyección y notación de trenzas. En la sección 3.2 estudiamos la estructura de grupo de las trenzas. En la siguiente sección vemos algunos invariantes de trenzas. En la sección 3.4 estudiamos el problema de las palabras y finalmente mostramos algunas conclusiones generales.

En el capítulo 4 realizamos el desarrollo informático sobre teoría de trenzas. Mostramos pseudocódigo para algunos de los algoritmos que hemos implementado y mostramos un recorrido por toxtren, el toolbox que hemos creado en Matlab, tanto para trenzas como para trenzas cerradas.

Finalmente en el capítulo 5 mostramos las conclusiones y vías futuras.

1.5 Fuentes principales.

Las principales fuentes consultadas son [Adams, 1994], [Flapan, 2016], [Murasugi and Kurpita, 1999], [Dehornoy, 1997] y [Murasugi, 2007]. Sin embargo, hemos consultado muchas otras fuentes destacando las que mostramos en la página 93. A lo largo del proyecto iremos referenciando las fuentes mostrando un breve comentario sobre las mismas.

1.6 Objetivos.

En la propuesta inicial se propusieron los siguientes objetivos:

- Estudio de la teoría de nudos haciendo especial énfasis en las técnicas que, de modo práctico, permiten establecer la equivalencia o ausencia de ella entre dos nudos. Normalmente esta comprobación se realiza mediante un invariante de nudo, que suele ser una valor que se calcula a partir de distintas representaciones (o descripciones) del mismo nudo, y que debe coincidir en el caso de que correspondan al mismo nudo.
- Estudio de una API gráfica que permita desarrollar semiautomáticamente nudos en el espacio R^3 , y desarrollo del prototipo de aplicación para definición de nudos.
- Estudio de técnicas de animación para desarrollar sobre la API gráfica animaciones que permitan comprobar visualmente la equivalencia entre dos diagramas de nudo.

En el capítulo 2 se cubre el primer objetivo por completo. Además, aquí se podría decir que se ha añadido otro objetivo con tanto peso como este al estudiar teoría de trenzas.

El desarrollo informático cubre los objetivos dos y tres con el siguiente detalle: no trabajamos directamente con nudos sino con trenzas y trenzas cerradas, pero las trenzas cerradas se pueden considerar como nudos y los nudos como trenzas cerradas.

Capítulo 2

Teoría de nudos.

2.1 Motivación y primeras definiciones.

¿Alguna vez has visto una cuerda con nudos y te has preguntado si podrías deshacerlos sin necesidad de romper la cuerda? ¿Te has planteado si algo tan usual como los nudos pueden estar presentes en áreas esenciales para la vida? Es más, ¿cuál fue el motivo inicial para estudiar dicha teoría de nudos? Quizás te sorprendan las respuestas pero antes de descubrirlo, veamos que se entiende formalmente por un nudo.

Definición:

Un **nudo** es una curva cerrada en \mathbb{R}^3 que no tiene auto-intersecciones.
Veamos algunos ejemplos:

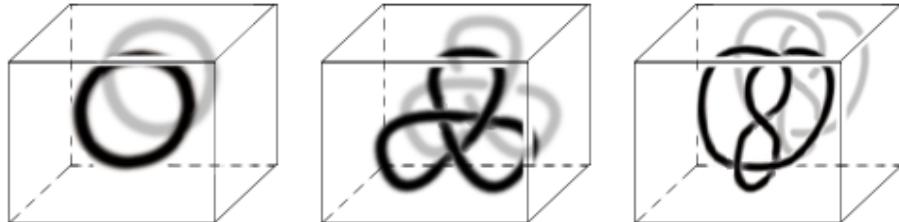


Figura 2.1: De izquierda a derecha: nudo trivial, nudo trébol, nudo de ocho.

Nos interesa saber cuándo dos nudos son equivalentes: se puede pasar de uno a otro mediante deformaciones. Veámoslo formalmente:

Se dice que dos nudos son equivalentes si existe un homeomorfismo de \mathbb{R}^3 que nos lleve de un nudo al otro.

Podemos representar un nudo en el plano visualizando su proyección. Como hay muchas formas de representar un mismo nudo, podremos tener diferentes proyecciones que representen al mismo nudo. Se pueden ver dos proyecciones distintas de un mismo nudo en la figura 2.16. Algunos ejemplos básicos de proyecciones son los siguientes:



Figura 2.2: Proyecciones del nudo trivial, nudo trébol, nudo de ocho.

Podemos ver en ellos una serie de cruces. En concreto en el nudo trébol tenemos 3 cruces y el nudo de ocho tenemos 4 cruces. El nudo trivial destaca porque puede tener proyecciones sin ningún cruce. No obstante, hay nudos con cruces que corresponden a proyecciones del nudo trivial.

Definición:

Un **enlace** es una o más curvas cerradas disjuntas en \mathbb{R}^3 . Cada una de sus curvas recibe el nombre de componente.

Por ejemplo, el siguiente enlace tiene dos componentes:



Por tanto, podemos ver un nudo como un caso particular de enlace en el que sólo tenemos una componente.

Una de las cuestiones más interesantes en la teoría de nudos es la siguiente:

¿Dada un nudo, o alguna proyección suya, podremos saber si se trata del nudo trivial?. A lo largo de este proyecto trataremos de dar respuesta, en parte, a dicha cuestión.

2.2 Sobre su historia y aplicaciones.

En el siglo XIX, ciertos físicos escoceses se preguntaban por la estructura de los átomos. Estos científicos tomaron como base la teoría de Descartes, que afirmaba que el éter era un fluido que ocupaba todo el espacio y transmitía la luz (éter lumínico), para desarrollar su modelo del átomo.

Aunque dichos físicos conocían la existencia de los elementos y que estaban formados por átomos, no conocían la propia estructura de los átomos.

Científicos como Peter Guthrie Tait y Willian Thomson llegaron a la teoría de que los átomos se concebían como vórtices, que podríamos ver como remolinos tubulares, en dicho fluido. Estos vórtices se encontraban anudados y en función del tipo de anudamiento darían lugar a un tipo de elemento u otro.

De este modo se plantearon que los diferentes nudos corresponderían a los diferentes elementos

de la naturaleza. De acuerdo con la teoría, si conociésemos todos los nudos posibles, crearíamos la tabla de elementos que reemplazaría la tabla periódica actual.

Para hacernos una idea más clara, para Willian Thomson el nudo trébol podría corresponder con el átomo de helio, el nudo de ocho con el átomo de oxígeno....

Numerosos científicos contribuyeron a dicha teoría intentando crear la tabla de nudos pero a finales de este mismo siglo, Michelson-Morley demostró que el éter lumínico no existía y por tanto la teoría de los átomos de vórtice fue descartada.

Tras este hecho, la teoría de nudos perdió su interés hasta que fue objeto de estudio en Topología a principios del siglo XX.

Posteriormente esta rama de la topología de baja dimensión destacó por su gran interés en áreas como:

1. Química: ya hemos visto que la teoría de nudos nace en este área.
2. Biología: se estudia la teoría de nudos en la estructura de ADN.

Conocemos como ácido desoxirribonucleico (ADN) a aquella molécula que se encuentra en el núcleo de nuestras células y, por tanto, que contiene nuestro código genético. Se trata de un elemento esencial para la vida. Es muy conocida su forma: se puede ver como dos cuerdas enrolladas formando un doble hélice.

Su forma de doble hélice puede encontrarse cerrada por los extremos de forma que nos encontraríamos con la propia forma de un nudo. Las ideas que veremos en este proyecto, junto con algunas más, se pueden aplicar a estas estructuras de ADN.

Además, esta estructura de ADN puede sufrir ciertas alteraciones producidas por la enzima topoisomerasa. Lo interesante es que estas alteraciones se corresponden con algunos de los movimientos que veremos para los nudos.

3. Criptografía [ISIK, 2005]: La fuerte relación que veremos entre la teoría de nudos y la teoría de trenzas, hace que podamos establecer cierta relación entre la teoría de nudos y la criptografía.

Haciendo uso de la criptografía tratamos de encontrar métodos para que el intercambio de información no sea comprensible por terceras personas.

Algunos métodos para encriptar dicha información están inspirados en la teoría de trenzas. En concreto el problema de la conjugación de trenzas, que trata de estudiar la igualdad de dos trenzas haciendo uso de una tercera trenza, nos permitirá encriptar la información de forma segura.

2.3 Componiendo nudos.

Supongamos que tenemos dos proyecciones J y K de nudos. Podemos definir un nuevo nudo a partir de ellos eliminando un arco de cada una de las proyecciones y conectando los 4 extremos finales de dos en dos mediante otros arcos de modo que no se añadan ni eliminen cruces. A este nudo resultante le llamaremos **suma conexa** o composición de los dos nudos y se denotará como $J \# K$. A los nudos originales J y K les llamaremos **nudos factores**.

Por ejemplo, consideremos como nudos factores el nudo trébol y el nudo de ocho.

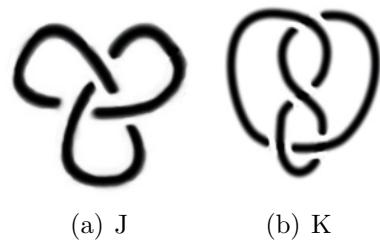


Figura 2.3: Nudo trébol y nudo de ocho.

Haciendo la suma conexa de ambos nudos obtenemos el nudo composición:



Figura 2.4: Composición de nudo trébol y nudo de ocho.

El nudo trivial es un elemento identidad para la suma conexa: si hacemos la composición de un nudo cualquiera J con el nudo trivial, vamos a obtener el propio nudo J. Por ejemplo, seguimos considerando J como el nudo trébol y el nudo trivial como K.

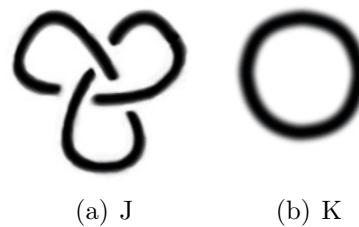


Figura 2.5: Nudo trébol y nudo trivial.

Su suma conexa nos seguiría dando el nudo factor J, es decir, el nudo trébol.

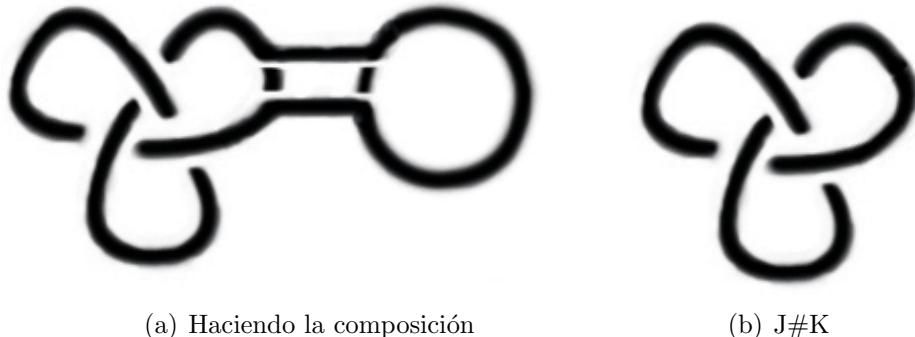


Figura 2.6: Composición de nudo trébol y nudo trivial.

Definición:

Diremos que un **nudo es primo** si no puede ser expresado como la suma conexa de dos nudos, a menos que uno de ellos sea el nudo trivial.

Definición:

Diremos que un **nudo es compuesto** si no es el nudo trivial ni es un nudo primo.

Por ejemplo, los nudos trébol y nudo de ocho de la figura 2.3 son nudos primos mientras que el nudo de la figura 2.4 es un nudo compuesto.

Hay una gran variedad de nudos primos. Cualquier nudo puede ser expresado singularmente como suma conexa de nudos primos. En la tabla 2.7, extraída de [Jkasd, 2008], podemos ver los diferentes nudos primos que tienen menos de 8 cruces.

Finalmente, es importante destacar el hecho de que la elección que hacemos de los arcos que eliminamos de cada uno de los nudos factores afecta al nudo composición. Por tanto, es posible construir dos nudos composición diferentes a partir del mismo par de nudos factores. Veamos esta idea con más detalle, para ello necesitamos:

Definición:

Un nudo orientado es un nudo al que se le ha asignado una orientación, es decir, es un nudo que dispone de una dirección de viaje sobre él mismo. Esta orientación se indica mediante flechas en la proyección.

Definición:

Un nudo es invertible si es equivalente a sí mismo con la orientación opuesta.

El problema de determinar si un nudo cualquiera es o no invertible no es para nada trivial.

Como ejemplo de nudo invertible nos podemos encontrar el nudo trébol, que vemos en la imagen 2.8.

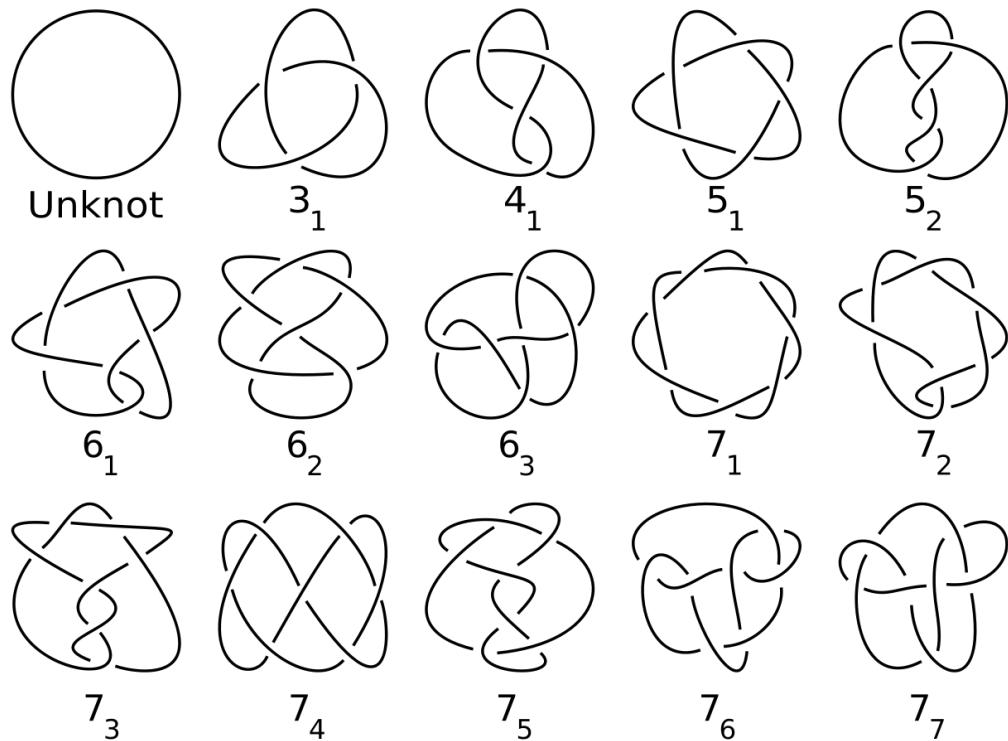


Figura 2.7: Tabla de nudos primos.

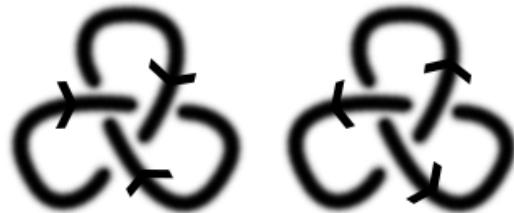


Figura 2.8: Ambas orientaciones del nudo trébol.

Sean los dos nudos factores J y K a los que se asignamos una orientación. Tendremos dos formas posibles de hacer la composición: conectar con las orientaciones emparejadas o no emparejadas.

Todas las composiciones de los nudos cuyas orientaciones emparejan al componer, darán el mismo nudo composición. Todas las composiciones de los nudos cuyas orientaciones no emparejan al componer, también darán el mismo nudo composición. Sin embargo, es posible que la composición de los nudos cuyas orientaciones emparejen no de lugar al mismo nudo que haciendo la composición de los nudos cuyas orientaciones no emparejen. Serán el mismo si uno de los nudos factores es invertible.

Veamos un caso en el que la composición de dos mismos factores, genera nudos diferentes. Consideraremos el nudo de la imagen 2.9.



Figura 2.9: Nudo factor J y K.

Si componemos el nudo consigo mismo conectando las orientaciones emparejadas y desemparejadas obtenemos nudos que no son equivalentes. Lo podemos comprobar visualmente en la Figura 2.10.



Figura 2.10: Las composiciones no son equivalentes.

2.4 Equivalencia de nudos: movimientos de Reidemeister.

Dos nudos K_1 y K_2 serán equivalentes ($K_1 \sim K_2$) si podemos distorsionar uno de ellos en el otro sin hacer ningún corte.

Para ver si dos proyecciones corresponden a nudos equivalentes, usaremos el concepto de isotopía plana. Más precisamente, definimos una **isotopía plana** de las proyecciones P_1 y P_2 de nudos como la aplicación continua $F : \mathbb{R}^2 \times [0, 1] \rightarrow \mathbb{R}^2$ tal que $F_0 = \text{identidad}$, $F_1(P_1) = P_2$ y F_t es un homeomorfismo $\forall t$.

Los movimientos de Reidemeister que vamos a ver a continuación nos permiten cambiar la proyección de un nudo de modo que se cambie la relación entre los cruces pero que no cambie el nudo al que representa la proyección.

Cada uno de estos movimientos es una isotopía:

Primer movimiento de Reidemeister - R1

En cualquier zona de la proyección nos permite añadir o eliminar un giro tal y como vemos en la figura 2.11.

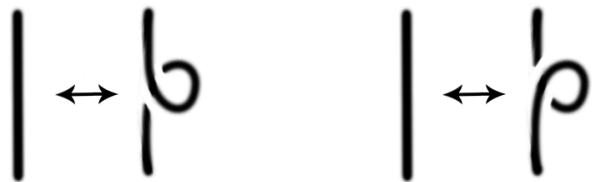


Figura 2.11: Primer movimiento Reidemeister.

Segundo movimiento de Reidemeister - R2.

Nos permite añadir o eliminar dos cruces del nudo como se ve en la figura 2.12.



Figura 2.12: Segundo movimiento de Reidemeister.

Tercer movimiento de Reidemeister - R3.

Nos permite deslizar una hebra del nudo de un lado de un cruce al otro lado del cruce. Veamos la figura 2.13 para aclarar la idea.



Figura 2.13: Tercer movimiento de Reidemeister.

Teorema 2.4.1. Teorema de Reidemeister [Murasugi, 2007]. Sean P_1 y P_2 las proyecciones que representan a dos nudos K_1 y K_2 , respectivamente. Entonces, $K_1 \sim K_2$ si, y solo si, P_1 y P_2 están conectados por una secuencia finita de movimientos de Reidemeister e isotopías planas.

Veamos un ejemplo en el que vemos la equivalencia de dos proyecciones, que en un primer momento podrían no parecernos equivalentes:

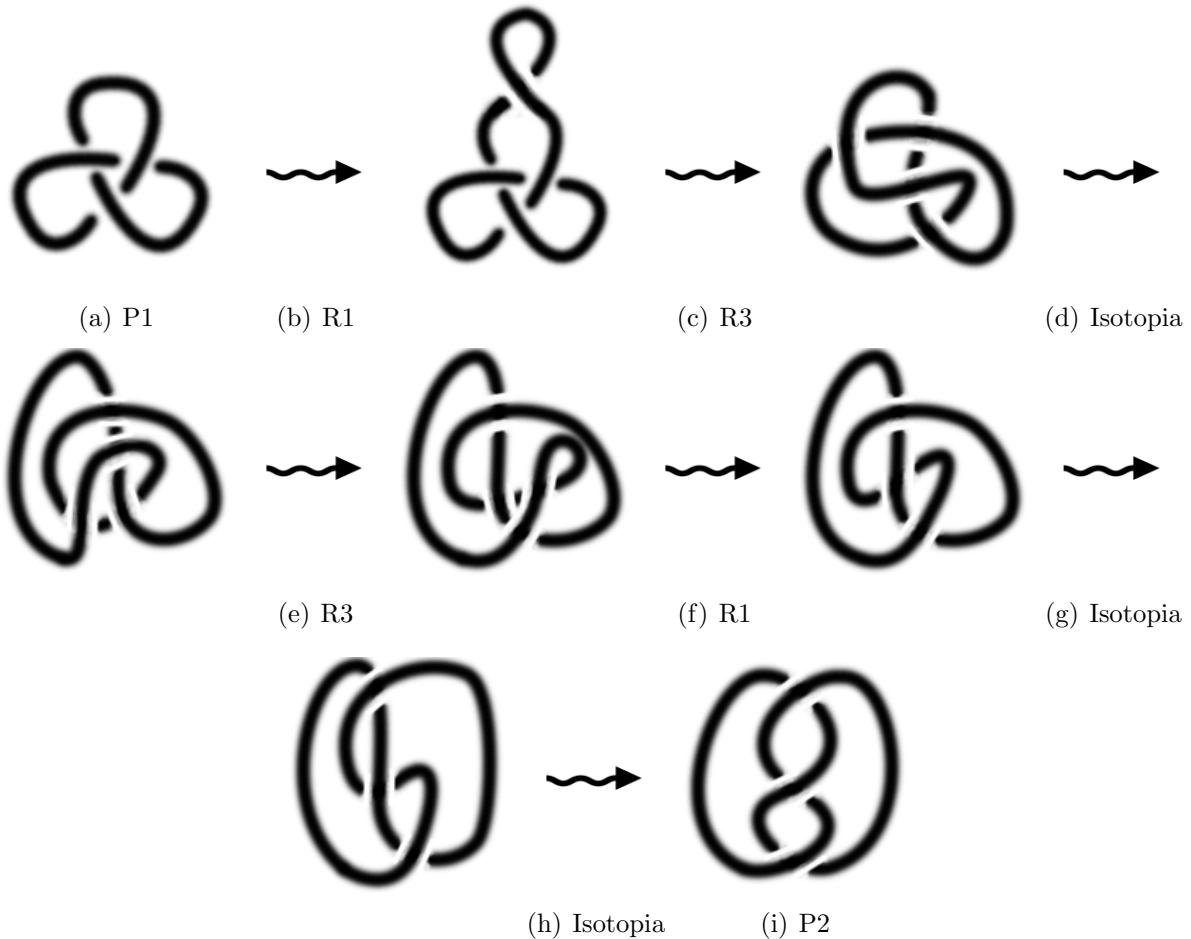


Figura 2.14: Equivalencia de dos proyecciones de nudos.

Gracias a dicho teorema podremos estudiar si dos proyecciones representan el mismo nudo. Para ello tendremos que encontrar una secuencia de movimientos de Reidemeister que nos lleve de una proyección a la otra. Sin embargo, este proceso puede no tener el número de movimientos intermedios limitado por lo que no tiene mucho sentido implementarlo.

Aunque este teorema no nos permita ver de una forma cómoda la equivalencia entre dos nudos en la práctica (por la fuerte complejidad) si que nos permite obtener una conclusión esencial:

Si una propiedad de un nudo no cambia al aplicarle cualquiera de estos tres movimientos de Reidemeister, entonces esta propiedad no va a cambiar por muchas deformaciones que se le hagan al nudo. En definitiva, si un nudo cumple cierta propiedad y otro nudo no la cumple, esos nudos no podrán ser equivalentes. Incidiremos en esta idea en la siguiente Sección 2.5.

2.5 Algunos invariantes.

Suponga que cierto día ha ido a trabajar con su portátil fuera de casa y se lo ha dejado olvidado. Cuando te das cuenta, vuelves a buscarlo pero ya no está. Pasan unos días y un amigo te comenta que se ha encontrado un portátil con x número de puertos, procesador y y modelo z .

Es claro que si alguna de esas características no fuese la de tu portátil, tendrías claro que no es el tuyo. Pero resulta que tu portátil tiene exactamente las mismas características. Piensas que tal vez sea tu portátil pero no tienes garantía de que realmente lo sea.

Algo similar, aplicado a nudos, es lo que vamos a tratar de ver en esta sección. ¿Dadas dos proyecciones, podremos decir que representan al mismo nudo? Vamos a tratar de estudiar ciertos invariantes sobre los nudos. Al igual que ocurría con el caso del portátil, si dos proyecciones tienen valores de un invariante distintos podremos decir que no representan al mismo nudo. Sin embargo, si ambos tienen valores de un invariante iguales no podremos saber si se trata del mismo nudo. Tendremos que estudiar otros invariantes de nudos.

¿Y si estudiamos una gran cantidad de invariantes de dos proyecciones y siempre toman los mismos valores? ¿Podremos garantizar que son equivalentes? La respuesta es clara, no. Tendremos que analizar en mayor detalle las proyecciones, pero puede ser que no lleguemos a obtener una conclusión definitiva.

Por tanto, el estudio de invariantes de los nudos nos va a permitir saber si dos nudos pueden ser (destacamos, que no quiere decir que lo sean) o no son equivalentes.

Antes de ver algunos de los invariantes de nudos más conocidos y útiles vamos a dar una definición formal de lo que se conoce como un invariante:

Definición:

Un **invariante** de un nudo (o de un enlace) es una propiedad que no cambia cuando el nudo sufre deformaciones en el espacio.

2.5.1 Número de componentes:

Es uno de los invariantes de enlaces más sencillos que nos podemos encontrar y que ya hemos comentado ligeramente en las secciones anteriores.

Cada componente de un enlace es una curva disjunta del mismo. En la Figura 2.15 vemos un enlace con dos componentes.



Figura 2.15: Enlace con dos componentes.

Al aplicarle a un enlace cualquier transformación no se le puede añadir ni eliminar ninguna componente. De modo que este invariante no nos va a permitir comparar nudos, pues todo nudo tiene una sola componente.

2.5.2 Crossing number:

Sea un nudo K . Su **crossing number** es el menor número de cruces que se encuentra en cualquier proyección del nudo. Se denota como $c(K)$.

Esto nos lleva a deducir el nudo K tendrá como mínimo $c(K)$ cruces por muchas transformaciones que se haga a sus proyecciones.

Aunque es un invariante sencillo de visualizar, no es fácil de obtener: puede ser que al estudiar muchas proyecciones de un nudo pensemos que su número de cruces es n , sin embargo pueden existir otras proyecciones del nudo que no conocemos con menor número de cruces. Por este motivo, no vamos a trabajar posteriormente con este invariante.

Para dejarlo más claro vamos a ver un ejemplo.

Consideramos la proyección de la primera figura 2.16. Podemos pensar que el número de cruces del nudo sería 4 (ver los 4 cruces de la figura a). Sin embargo, esta proyección es equivalente a la proyección que vemos en la figura b, que tiene como número de cruces el valor 3. Por tanto, el número de cruces del nudo, que es el nudo trébol, es 3.



¹For example, see 16. G. C. Smith, *Principles of Economics* (London, 1930), pp. 1-11.

2.5.3 Tricolorabilidad:

Para definir este invariante de enlaces, tenemos que conocer unos conceptos previos:
Entendemos por undercrossing y overcrossing a un recorrido en el cruce de la proyección de un nudo que nos lleva por encima o por debajo en el cruce.



Figura 2.17: Tipo de cruce.

Veamos la imagen 2.17 donde queda representado:

Definición:

Una **hebra** de una proyección de un enlace es una región de cuerda del enlace que va desde un undercrossing a otro undercrossing atravesando sólo overcrossings.

Definición:

Una proyección de un enlace es tricolorable si cada una de las hebras de la proyección puede ser coloreada con uno de tres colores diferentes de modo que en cada cruce o los 3 colores se junten o se junte un sólo color. En este caso diremos que el enlace es **tricolorable**.

Teorema 2.5.1. *La tricolorabilidad se perserva mediante movimientos de Reidemeister.*

Demostración. Supongamos que partimos de una proyección de un nudo que es tricolorable y vamos a aplicarle los movimientos de Reidemeister. Tendremos que ver que la proyección final es tricolorable. Podemos centrarnos en la zona de la proyección a la que se le aplica el movimiento. Veamos qué ocurre al aplicar cada movimiento:

R1: En este caso sólo se puede dar que tengamos un sólo color. Al aplicar el movimiento R1 seguiremos utilizando ese mismo color, de modo que la proyección modificada seguirá siendo tricolorable.



Figura 2.18: Demo R1

R2: Supongamos que el coloreado inicial se ha realizado con un sólo color. Al igual que con el movimiento R1, no tendríamos ninguna dificultad pues al aplicar el movimiento R2 seguiremos coloreando con el mismo color y la proyección seguirá siendo tricolorable.

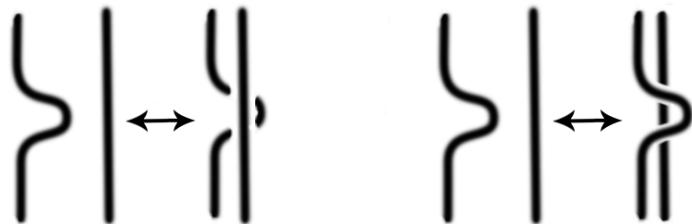


Figura 2.19: Demo R2

Ahora tenemos otra segunda opción de coloreado en la proyección inicial y es que en ambos cruces posibles usemos los 3 colores posibles. Se puede ver en la figura 2.20. Se observa que la proyección sigue siendo tricolorable.

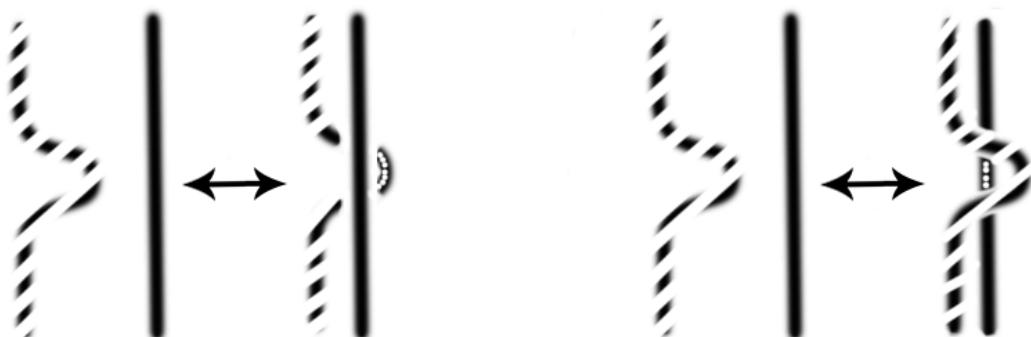


Figura 2.20: Demo R2

R3: Este movimiento tiene más complejidad pues tenemos 3 cruces posibles. Analizamos todos los casos posibles y vemos que nos podemos reducir a cinco situaciones. Veamos en la figura 2.21 cómo sería el cambio de color en cada caso para confirmar que las proyecciones resultantes al aplicar el movimiento siguen siendo tricolorables.

Concluimos que si tenemos una proyección tricolorable y aplicamos los movimientos de Reidemeister, la proyección resultante sigue siendo tricolorable.

Si la proyección de partida no fuese tricolorable, la proyección resultante tampoco podría ser tricolorable: supongamos que la proyección resultante es tricolorable, por el razonamiento que hemos seguido anteriormente, tendríamos que tener que la proyección inicial sería tricolorable llegando a contradicción.

□

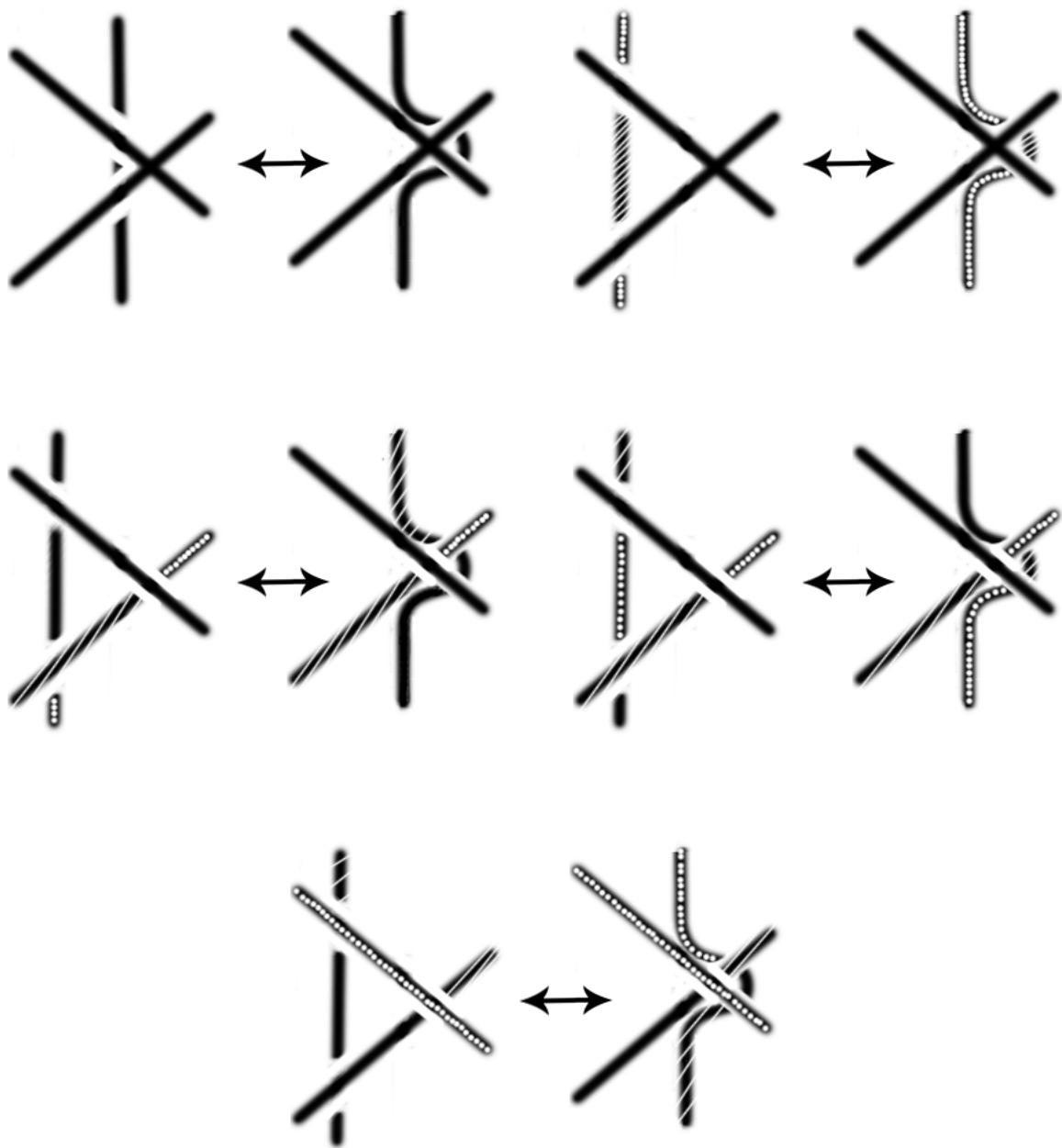


Figura 2.21: Demo R3

Haciendo uso de este invariante podemos demostrar que el nudo trivial y el nudo trébol no son equivalentes, es decir, son topológicamente distintos. Esto se debe a que el nudo trébol es tricolorable pero el nudo trivial no lo es. Podemos verlo en la imagen 2.22.

Este hecho confirma que existe al menos un nudo distinto del nudo trivial. Es más, todo nudo que sea tricolorable será distinto del nudo trivial.

Sin embargo, este invariante no es muy potente en el sentido de que sólo clasifica los enlaces

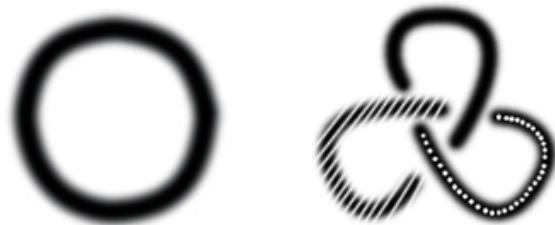


Figura 2.22: Dos nudos no equivalentes.

en tricolorables y no tricolorables y sólo podremos afirmar que dos proyecciones representan a diferentes nudos si una de ellas es tricolorable y la otra no lo es.

2.5.4 Unknotting number:

Supongamos que tenemos la proyección de un nudo no trivial. Modificar la proyección para que un undercrossing pase a ser un overcrossing, o viceversa, no es un movimiento válido pues estamos intersecando la cuerda. Sin embargo, es un movimiento interesante ya que nos llevará al nudo trivial.

Definición: Se define el **número de desanudamiento** de un nudo como el menor número de cambios en los cruces necesarios para desanudarlo, es decir, para llegar al nudo trivial. Se denota como $u(K)$.

Es claro que en un número finito de pasos conseguiremos obtener el número de desanudamiento: Recorremos el nudo con cierta orientación y al llegar a un cruce, si no hemos pasado anteriormente por él y es un undercrossing, lo convertimos en overcrossing.

Podemos ver, en la figura 2.23, que el número uno es el número de desanudamiento para el nudo trébol. El punto de partida lo indicamos con un punto grueso.



(a) Nudo de partida (b) Nudo modificado

Figura 2.23: Unknotting number trébol.

2.5.5 Polinomio de Alexander:

Anteriormente hemos visto algunos invariantes geométricos y numéricos que nos pueden ayudar en la tarea de comprobar si dos proyecciones representan al mismo nudo. Ahora vamos a ver un invariante polinómico que estudiaremos con más detalle pues será uno de los puntos fuertes que implementaremos para resolver, en parte, la cuestión.

Se trata de un polinomio, con variable t , para nudos orientados. En 1969, se probó que dicho polinomio puede calcularse computacionalmente haciendo uso de dos reglas:

- Regla 1:

El polinomio de Alexander del nudo trivial es el polinomio trivial equivalente a 1. Esta regla se representa como $\Delta(\bigcirc) = 1$

Para poder exponer la segunda regla tenemos que conocer antes las siguientes ideas.

Consideramos el enlace orientado de partida. Si nos centramos en uno de sus cruces, podemos crear tres nuevos enlaces orientados exactamente iguales al de partida variando únicamente en este cruce seleccionado. A cada uno de estos nuevos enlaces le establecemos uno de estos nuevos cruces:

1. L_+ : El cruce seleccionado se establece como positivo.
2. L_- : El cruce seleccionado se establece como negativo.
3. L_0 : El cruce seleccionado se elimina.

Podemos ver estos nuevos cruces en la figura 2.24.

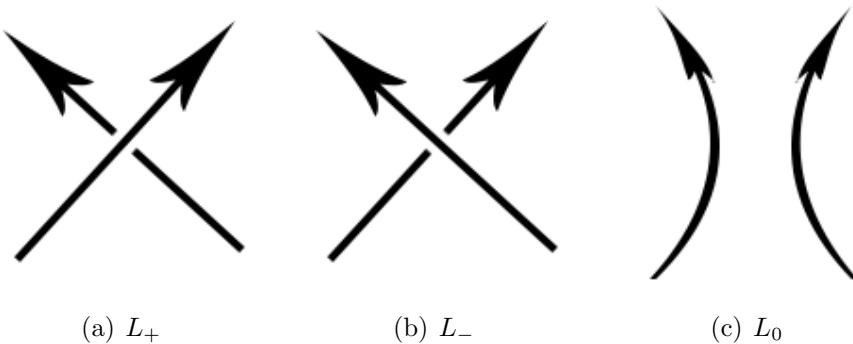


Figura 2.24: Tipos de cruces.

- Regla 2:

Esta regla se representa como $\Delta(L_+) - \Delta(L_-) + (t^{\frac{1}{2}} - t^{-\frac{1}{2}}) \Delta(L_0) = 0$

Veamos cómo se calcularía el polinomio de Alexander con el nudo trébol:

Como el nudo de partida tiene más de un cruce, tenemos que aplicarle la segunda regla. Consideraremos como cruce a cambiar el que nos encontramos más a la izquierda aunque podríamos considerar cualquier otro cruce. Generamos los tres nuevos enlaces haciendo el cambio en el cruce. Sus proyección se pueden ver en la figura 2.25. Las llamamos A, B y C por comodidad.



Figura 2.25: Cambio cruces trébol

Al aplicar la segunda regla tendremos:

$$\Delta(A) - \Delta(B) + (t^{\frac{1}{2}} - t^{-\frac{1}{2}}) \Delta(C) = 0. \quad (2.1)$$

Sabemos que $\Delta(B) = \Delta(\bigcirc) = 1$. Luego nos quedaría la ecuación:

$$\Delta(A) + (t^{\frac{1}{2}} - t^{-\frac{1}{2}}) \Delta(C) = 1. \quad (2.2)$$

Ahora necesitamos conocer el valor de $\Delta(C)$. Al igual que ocurría anteriormente, la proyección de este nuevo nudo tiene más de un cruce de modo que vamos a aplicarle la segunda regla. Esta vez seleccionamos el cruce superior como cruce que se modifica en los nuevos nudos. Obtenemos las proyecciones que vemos en la figura 2.26, a las que llamaremos D y E.

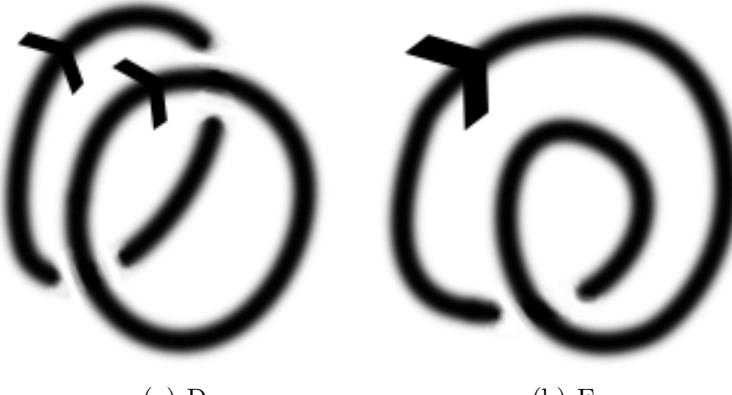


Figura 2.26: Cambio cruces

Al aplicar la segunda regla tendremos:

$$\Delta(C) - \Delta(D) + (t^{\frac{1}{2}} - t^{-\frac{1}{2}}) \Delta(E) = 0. \quad (2.3)$$

Sabemos que $\Delta(E) = \Delta(\bigcirc) = 1$. Además $\Delta(D) = \Delta(\bigcirc \bigcirc) = 0$. Luego nos quedaría la ecuación:

$$\Delta(C) = -(t^{\frac{1}{2}} - t^{-\frac{1}{2}}). \quad (2.4)$$

Volviendo a la ecuación (2) tendremos:

$$\Delta(A) = (t^{\frac{1}{2}} - t^{-\frac{1}{2}})^2 + 1 = t^{-1} - 1 + t. \quad (2.5)$$

Luego $t^{-1} - 1 + t$ es el polinomio de Alexander del nudo trébol.

Es importante destacar el hecho de que podemos tener un nudo no trivial que tenga como polinomio de Alexander el polinomio trivial equivalente a 1. Por este motivo, con dicho invariante no podemos distinguir cualquier nudo del nudo trivial.

¿Mediante estas dos reglas podremos obtener siempre el polinomio de Alexander en tiempo finito? La respuesta es afirmativa, veámoslo:

La idea que vamos a seguir en este proceso será reducir las proyecciones, a las que le queremos obtener el polinomio de Alexander, hasta llegar al nudo trivial. Reducir estas proyecciones quiere decir ir modificando sus cruces obteniendo L_+ , L_- y L_0 . Como veíamos en el invariante de la sección 2.5.4, a cualquier proyección le podemos aplicar una secuencia finita de cambios en los cruces de modo que resulte el nudo trivial. Por tanto, este procedimiento tendrá fin en una secuencia finita de pasos.

Al ir considerando los distintos nudos de la proyección, obtendremos dos nuevas proyecciones en cada paso. Con estas nuevas proyecciones podremos formar lo que se denomina **árbol de resolución**. En la figura 2.27 podemos ver el árbol de resolución del nudo trébol.

Definición:

Se define la **profundidad de un nudo** (o enlace) como el mínimo número de niveles que nos encontramos en los árboles de resolución del nudo.

Esta profundidad del nudo es un invariante que además nos permite medir la complejidad que supondrá el cálculo del polinomio de Alexander.

El árbol de resolución del nudo trébol que vemos en la figura 2.27 tiene una profundidad de 2 niveles (la proyección inicial no se cuenta como nivel). Es importante destacar que el único nudo que tiene profundidad de nudo igual a 0 es el nudo trivial.

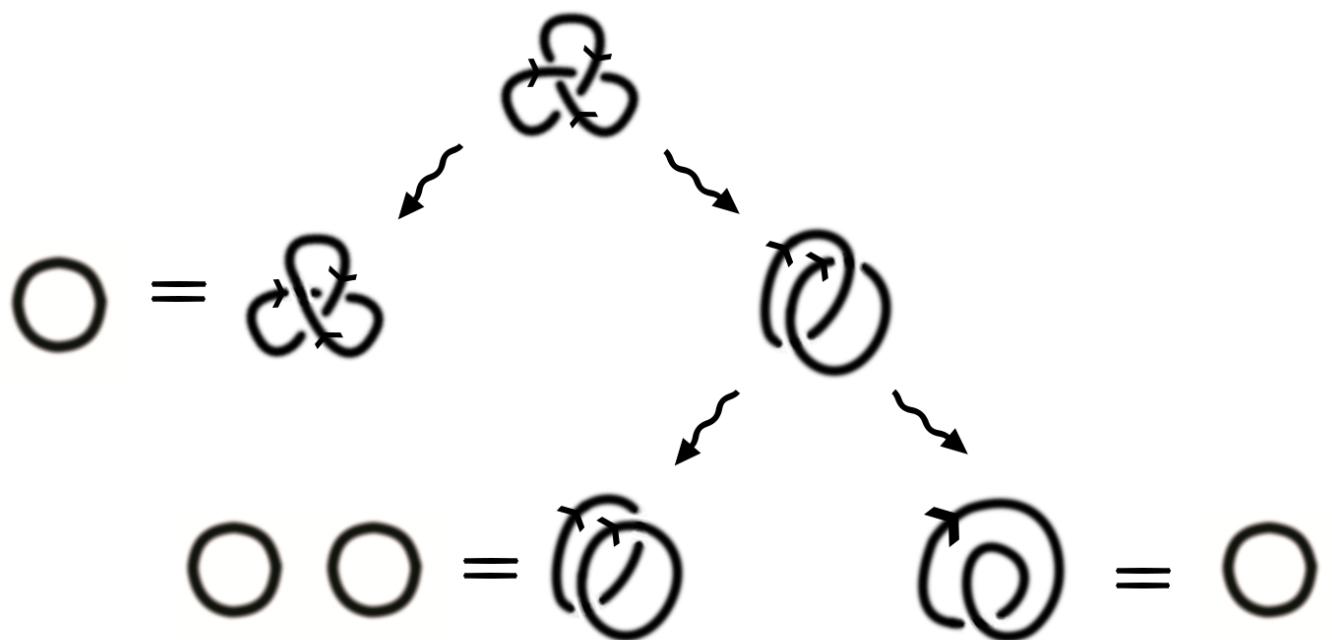


Figura 2.27: Árbol de resolución del nudo trébol.

2.6 Notación de nudos.

Ya sabemos qué son los nudos y algunas nociones esenciales sobre los mismos, pero aún no sabemos asociar una notación concreta a un nudo. En esta sección vamos a ver algunas de las notaciones más comunes de los nudos.

2.6.1 Notación de Dowker:

Se trata de una notación muy sencilla para describir la proyección de un nudo. La notación en sí es una secuencia de números enteros, veamos cómo se obtiene:

Consideraremos una orientación en la proyección de n cruces y asignamos el valor 1 al primer cruce que nos encontramos. Continuamos por la proyección y asignamos el valor 2 al siguiente cruce. Vamos repitiendo el proceso hasta pasar por cada cruce un par de veces (una vez por el undercrossing y otra por el overcrossing). Como resultado tendremos un número par y un número impar por cada cruce de la proyección.

Finalmente es necesario asignar los signos a cada uno de estos $2n$ números. Los números pares que correspondan a un overcrossing tendrán signo negativo. Veamos un ejemplo con el nudo trébol que podemos ver en la figura 2.28:

Tendríamos los pares $(1, -4)$, $(3, -6)$ y $(5, 2)$. La notación de Dowker sería $-4 \ -6 \ 2$ pues nos quedamos únicamente con los números pares en el orden que indican los números impares.

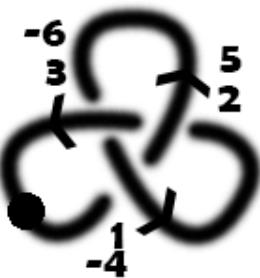


Figura 2.28: Numeración de cruces-Dowker.

2.6.2 Notación de Gauss:

La notación de Gauss es una notación parecida a la de Dowker. Consideremos de nuevo una orientación en la proyección de n cruces de un nudo.

En este caso cada vez que pasamos por un cruce, tendremos un solo número asignado. De este modo la secuencia de números de la notación se compone de $2n$ elementos con valores desde 1 hasta n , cada uno de ellos repetido dos veces.

Consideraremos una orientación en la proyección de n cruces y asignamos el valor 1 al primer cruce que nos encontramos. Realizamos el siguiente proceso: continuamos por la proyección hasta el siguiente cruce. Si ya hemos pasado por él, anotamos el número de cruce que tenga asociado. Si no hemos pasado anteriormente por él, asignamos el siguiente número de cruce.

Finalmente es necesario asignar los signos a cada uno de esos $2n$ números. Los números que representen uncrossings tendrán signo negativo. Veamos la notación de Gauss para el nudo trébol.

Si vamos haciendo el recorrido partiendo desde el punto grueso indicado tendríamos la secuencia



Figura 2.29: Numeración de cruces-Gauss.

de números: -1 2 -3 1 -2 3. Esta sería su notación de Gauss.

2.6.3 Notación de Conway:

Por último vamos a ver una notación que puede resultar algo más compleja pero que tiene gran uso e interés, sobre todo en la teoría del ADN.

Definición:

Un **enredo** o tangle de una proyección de un enlace es una región de la proyección rodeada por una bola de modo que las dos cuerdas enlazadas de la proyección tocan la bola exactamente en cuatro puntos. A estos puntos los denotaremos como NO, NE, SO y SE.

En la figura 2.30 vemos un enredo general y un ejemplo particular de un enredo.

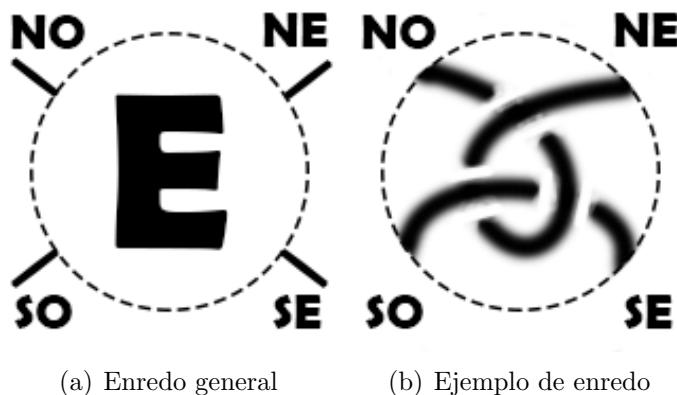


Figura 2.30: Enredos

Pero la idea es trabajar con enredos más sencillos como son los que vemos en la figura 2.31. La notación se corresponde con el número de cruces que tiene el enredo con signo positivo y negativo según se ve en las imágenes.

Haciendo uso de estos enredos básicos podemos construir nuevos enredos uniendo, respectivamente, los extremos NE y SE de un enredo con los extremos NO y SO del otro enredo. A esta operación se le conoce como suma de enredos.

Además, disponemos de una operación de multiplicación: reflejaremos el primer enredo y haremos la operación de suma.

A los enredos construidos con estas operaciones se les conoce con **enredos racionales**. Podemos ver un esquema básico estas dos operaciones en la imagen 2.32:

En la figura 2.33 podemos ver un ejemplo de un enlace más complejo:

A partir de estos enredos podemos construir proyecciones de nudos enlazando los puntos NO con NE y los puntos SO con SE. La notación del nudo corresponde con la notación que le damos a su enredo.

Nos interesa ver si dos proyecciones de nudos representan al mismo nudo, luego nos interesa ver si dos enredos son equivalentes. Veamos qué quiere decir que sean equivalentes:

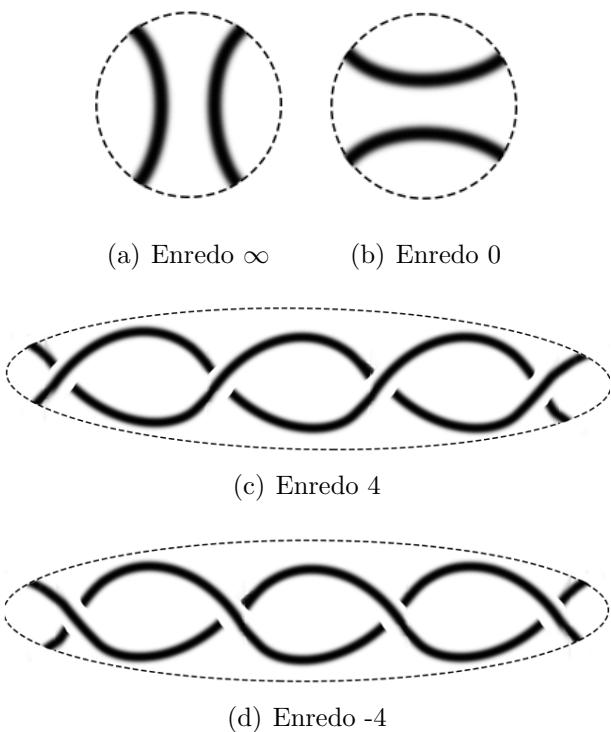


Figura 2.31: Tipos básicos de enredos.

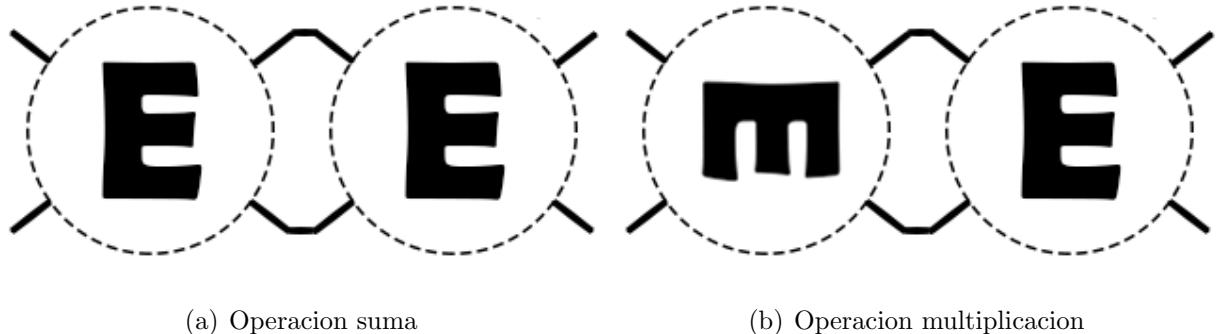


Figura 2.32: Operaciones con enredos.



Figura 2.33: Enlace -2 3 2

Definición:

Diremos que dos enredos son equivalentes si podemos pasar de un enredo al otro mediante los movimientos de Reidemeister, que vimos en la sección 2.4, manteniendo los cuatro extremos fijos en la bola imaginaria.

Ver si dos enredos son equivalentes por definición no es viable de modo que vamos a aplicar otro método: consiste en calcular la fracción continua asociada a cada enredo.

Definición:

Sea un enredo con notación $a_n \dots a_1 a_0$. Su **fracción continua** es una expresión del tipo:

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\dots a_n}}} \quad (2.6)$$

siendo $a_i \in \mathbb{Z}, \forall i = 0, 1, \dots, n$.

Teorema 2.6.1. *Dos enredos racionales son equivalentes si y solo si sus fracciones continuas toman el mismo valor.*

Veamos un ejemplo. Consideramos los enredos racionales $-2 \ 3 \ 2$ y $3 \ -2 \ 3$ que vemos en la figura 2.34.

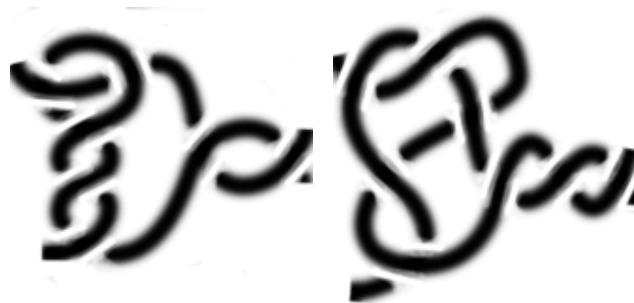


Figura 2.34: Enlaces equivalentes

A simple vista resulta difícil confirmar que sean equivalentes. Vamos a obtener sus fracciones continuas asociadas:

-2 3 2 tiene fracción continua

$$2 + \frac{1}{3 + \frac{1}{-2}} = \frac{12}{5} \quad (2.7)$$

3 - 2 3 tiene fracción continua

$$3 + \frac{1}{-2 + \frac{1}{3}} = \frac{12}{5} \quad (2.8)$$

Ambas fracciones continuas son iguales, luego los enredos -2 3 2 y 3 -2 3 son equivalentes.

2.7 Conexión con distintas teorías.

2.7.1 Teoría de grafos:

Definición:

Un **grafo** es un par (V, A) de conjuntos, junto con la aplicación

$$\gamma : A \rightarrow \{\{u, v\} / u, v \in V\}$$

Al conjunto de puntos V se le llama conjunto de vértices y al conjunto A le llamaremos conjunto de aristas.

Definición:

Un **grafo plano** G es un grafo que permanece en el plano.

Podemos ver varios ejemplos en la figura 2.35.

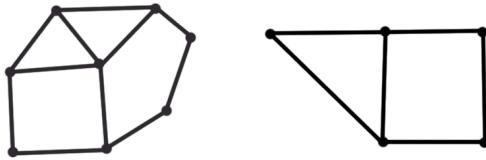


Figura 2.35: Ejemplos de grafos planos.

A partir de la proyección de un nudo (o de un enlace en general) podemos generar su grafo plano asociado. Para ello tendremos que realizar el siguiente proceso:

Sombreamos las regiones de la proyección que estén de modo que la región externa al nudo se quede sin sombrear y situamos un vértice en cada zona. Unimos los vértices con aristas que pasan por los cruces de la proyección. Ya tendríamos el grafo plano. Además, si el nudo tiene asignada una orientación, podremos asignarle el tipo de cruce (positivo o negativo) a cada arista. Podemos ver un ejemplo en la figura 2.36.

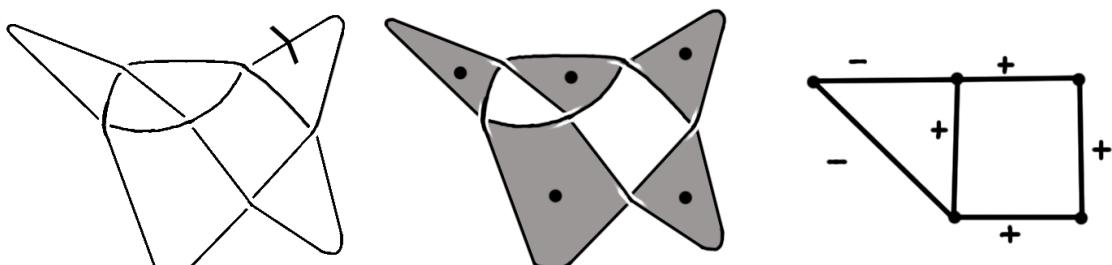


Figura 2.36: De proyección a grafo

Finalmente, para ver que los problemas de nudos se pueden ver como problemas de grafos y viceversa, vamos a ver el procedimiento inverso. Dado un grafo plano, podremos obtener la proyección del nudo asociado. Veamos cuál sería el procedimiento:

Partiendo del grafo plano con los signos asociados en cada vértice, marcamos cada una de las aristas. Uniremos cada una de estas marcas con aquellas marcas que estén en las aristas que conectan con los vértices de la arista que tiene la marca. A continuación, sombreadamos las zonas que contienen a cada vértice. Finalmente, establecemos los cruces conforme a los signos del grafo plano. Podemos ver un ejemplo en la figura 2.37.

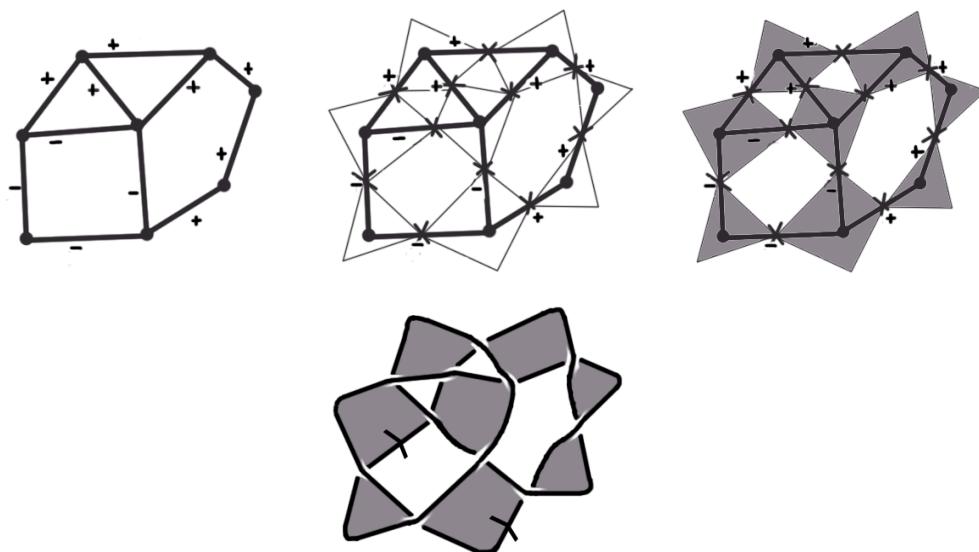


Figura 2.37: De grafo a proyección.

2.7.2 Teoría de trenzas:

En esta sección vamos a introducir la relación que hay entre teoría de nudos y teoría de trenzas, teoría que estudiaremos con mayor detalle en el próximo tema.

Vamos a ver la idea general de lo que se entiende por el término trenza y veremos una definición más precisa más adelante. Podemos pensar en una trenza como un conjunto de n cadenas que son atadas a un tope imaginario arriba y abajo. Podemos ver algunos ejemplos de trenzas en la figura 2.38.

A partir de una trenza, podemos obtener su nudo o enlace correspondiente. Simplemente tendremos que unir en orden los topes superiores de las cadenas con los inferiores. Esta trenza cerrada será el nudo al que representa la trenza. Podemos ver algunos ejemplos en la sección 3.2.3.



Figura 2.38: Ejemplos de trenzas

Para ver el proceso inverso haremos uso del siguiente teorema:

Teorema 2.7.1. Teorema de Alexander.

Todo nudo puede ser representado como una trenza cerrada.

Para ver la demostración de dicho teorema podemos inspirarnos en varias ideas [William Menasco, 2005], [Hoberg, 2011]. En este caso vamos a ver el algoritmo de Yamada-Vogel, que transforma un nudo en una trenza cerrada. Este proceso se puede realizar a cualquier nudo y el resultado es siempre una trenza cerrada.

Demostración. Dada nudo orientado K , realizaremos los siguientes pasos:

1. A partir de la proyección D del nudo K , vamos a obtener su imagen de Seifert S realizando el siguiente proceso:

Sabemos que en cada cruce de la proyección D nos encontramos dos hebras entrando al cruce y dos hebras salientes. Vamos a eliminar el cruce conectando cada una de las hebras que entran al cruce con las hebras adyacentes que salen del mismo. Podemos ver la idea en la figura 2.39.

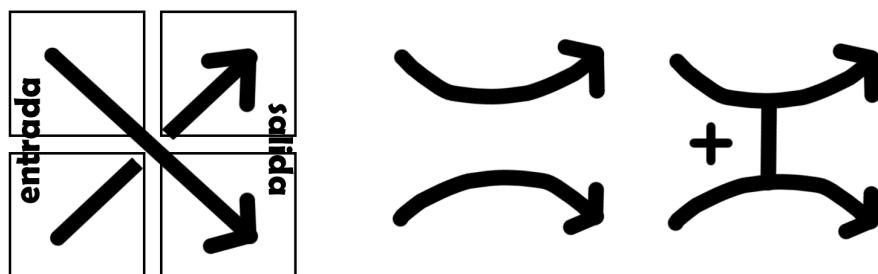


Figura 2.39: Transformando la proyección de un cruce.

Como resultado, vamos a obtener un conjunto de círculos en el plano a los que se les conoce como círculos de Seifert.

Para no perder el tipo de cruce, vamos a conservar la conexión de los cruces y asignaremos el símbolo + a los cruces positivos y el símbolo - a los cruces negativos. De este modo, conseguimos conservar toda la información sobre la proyección del nudo.

2. Antes de ver cómo realizar este segundo paso, necesitamos unos conceptos previos.

Sean dos círculos de Seifert C_1 y C_2 . Diremos que son incoherentes si no existe un arco que los conecte.

Definimos la altura de la proyección D ($h(D)$) como el número de pares de círculos de Seifert incoherentes.

Si $h(D)=0$, entonces D ya representa una trenza cerrada, finalizo.

Si $h(D)>0$, podemos encontrar un arco que une dos pares de círculos de Seifert incoherentes (se conoce como arco de reducción). Se puede de ver cómo representaremos un arco de reducción entre dos círculos de Seifert en la figura 2.40.

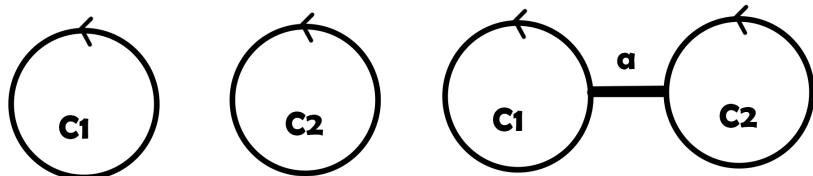


Figura 2.40: Arco de reducción.

Este arco de reducción nos sirve de guía para realizar un movimiento de reducción sobre ambos círculos de Seifert de modo que obtenemos la primera proyección de la figura 2.41. Con las siguientes imágenes de la figura vemos que efectivamente se conservan los círculos de Seifert originales pero hemos conseguido que ahora sean coherentes.

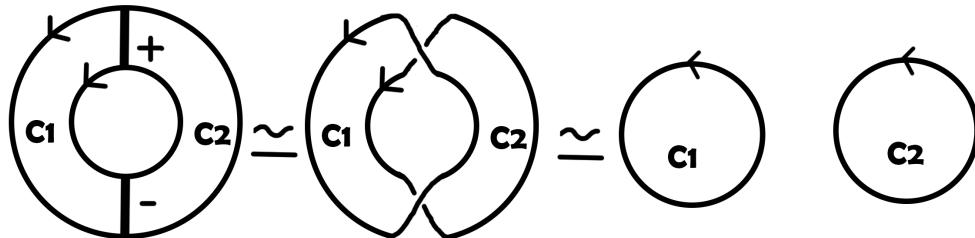


Figura 2.41: Movimiento de reducción.

3. Continuamos realizando movimientos de reducción hasta obtener $h(D)=0$.

Se puede demostrar [William Menasco, 2005] que dicho algoritmo finaliza haciendo uso del siguiente lema:

Supongamos que realizamos un movimiento de reducción al a proyección D , obteniendo la proyección D' . En ese caso, $h(D')=h(D)-1$.

□

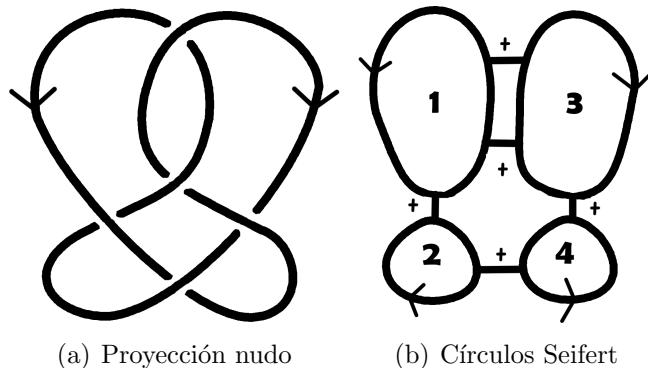


Figura 2.42: Movimiento de reducción.

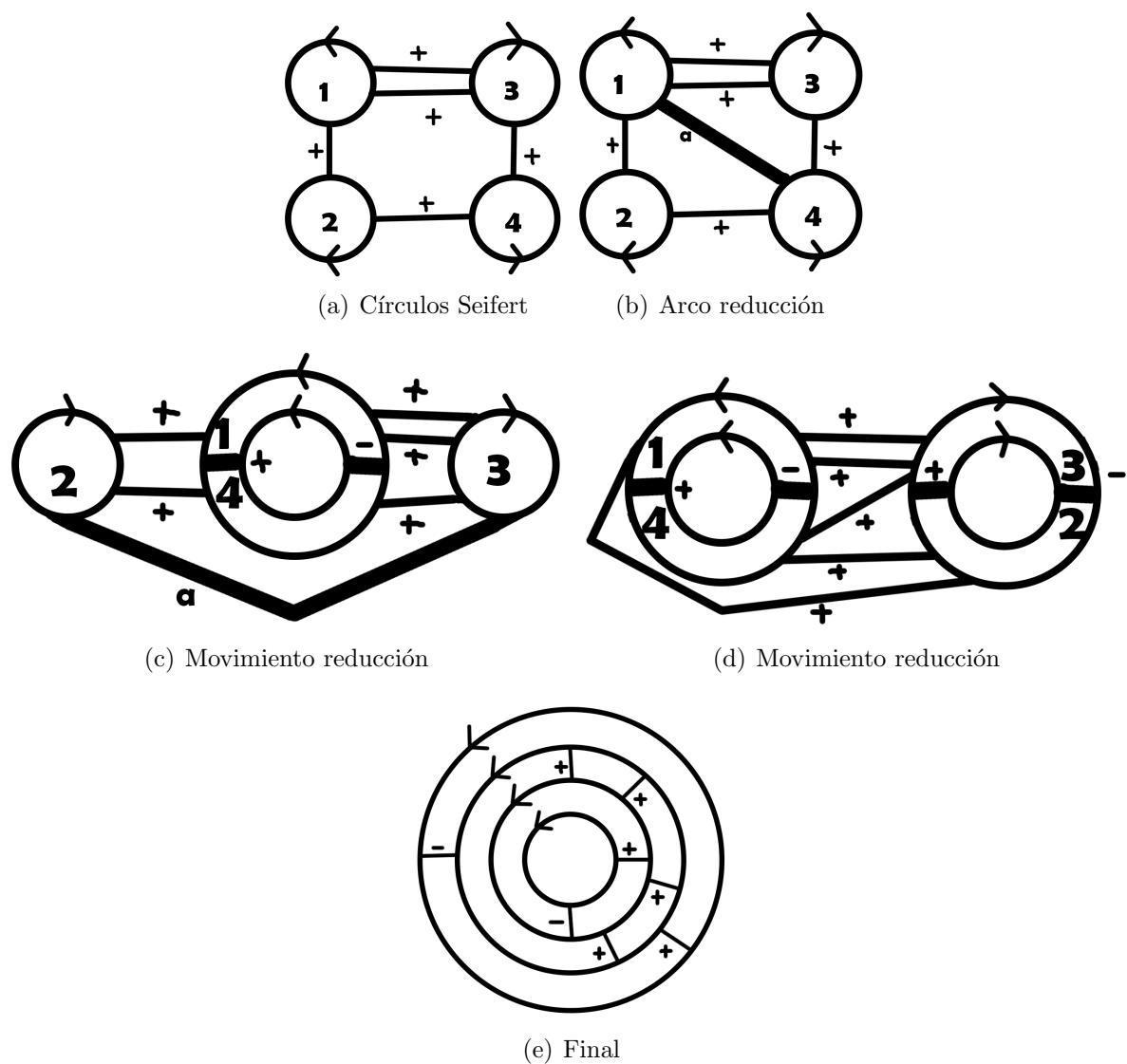


Figura 2.43: Algoritmo Yamada-Vogel.

En la figura 2.43 aplicamos el algoritmo a un nudo particular.

Capítulo 3

Teoría de trenzas.

En la sección 2.7 dimos una idea general de lo que se entiende por una trenza. Veamos su definición más formal:

Definición:

Consideremos el cubo $\mathbb{D} = \{(x, y, z) / 0 \leq x, y, z \leq 1\}$ y situamos A_i puntos en su cara superior y B_i puntos en la cara inferior, siendo $i \in \mathbb{N}$. Podemos considerar dichos puntos como

$$A_1 = \left(\frac{1}{2}, \frac{1}{n+1}, 1\right), A_2 = \left(\frac{1}{2}, \frac{2}{n+1}, 1\right), \dots, A_n = \left(\frac{1}{2}, \frac{n}{n+1}, 1\right),$$

$$B_1 = \left(\frac{1}{2}, \frac{1}{n+1}, 0\right), B_2 = \left(\frac{1}{2}, \frac{2}{n+1}, 0\right), \dots, B_n = \left(\frac{1}{2}, \frac{n}{n+1}, 0\right).$$

Unimos cada punto A_i con un cierto punto B_k , $i, k \in \mathbb{N}$, mediante arcos poligonales simples d_i de modo que:

1. d_1, d_2, \dots, d_n sean disjuntos.
2. Los arcos d_i no pueden conectar puntos A_i o B_i entre sí.
3. Al cortar por cualquier plano horizontal, cada arco d_i toca en un sólo punto al plano.

A cada uno de estos arcos poligonales d_i les llamaremos **cadenas** y al conjunto de las n -cadenas se le conoce como **trenza**. Podemos ver algunos ejemplos de trenzas en la figura 3.1.

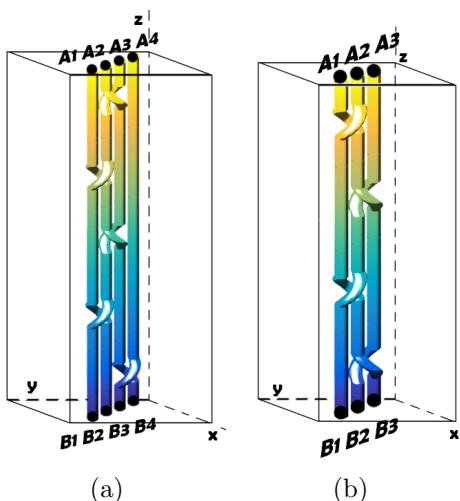


Figura 3.1: Ejemplos de trenzas

Anteriormente vimos que a cada trenza le corresponde un nudo o un enlace particular. Se obtendrá uniendo los extremos superiores con los extremos inferiores de las cadenas en el mismo orden. A este nudo se le conocerá como **trenza cerrada**.

Denotaremos como \mathcal{B}_n al conjunto de todas las trenzas de n cadenas.

3.1 Fundamentos.

3.1.1 Equivalencia de trenzas:

Intuitivamente diremos que dos trenzas son equivalentes si podemos deformar las cadenas de las trenzas de forma que ambas trenzas se vean iguales. Las trenzas de la figura 3.2 son equivalentes.

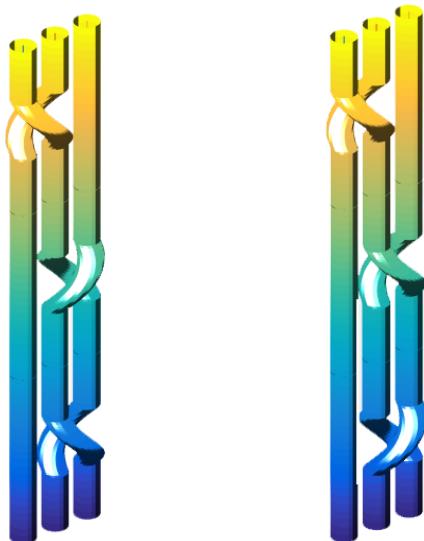


Figura 3.2: Trenzas equivalentes.

Definición:

Consideremos la cadena d de una trenza situada en el cubo $\mathbb{D} = \{(x, y, z) / 0 \leq x, y, z \leq 1\}$. Podemos quedarnos con una representación poligonal de las cadenas de la trenza.

Sea AB un segmento de dicha cadena y C un punto en el cubo de forma que el triángulo $\triangle ABC$ no corta a ninguna otra cadena de la trenza y sólo toca a la cadena d en el segmento AB . Supongamos además que los segmentos AC y CB cortan a cualquier plano horizontal del cubo en un sólo punto como mucho. Visualizamos estas condiciones en la primera imagen de la figura 3.3.

Bajo estas condiciones definimos un **movimiento elemental** como la operación Ω que intercambia el segmento AB por los segmentos $AC \cup CB$.

La operación inversa Ω^{-1} que intercambia los segmentos $AC \cup CB$, que formen parte de una cadena, por el segmento AB de forma que el triángulo $\triangle ABC$ no corte a ninguna otra cadena, también es considerada un movimiento elemental.

Podemos ver la representación de ambos movimientos en la figura 3.3.

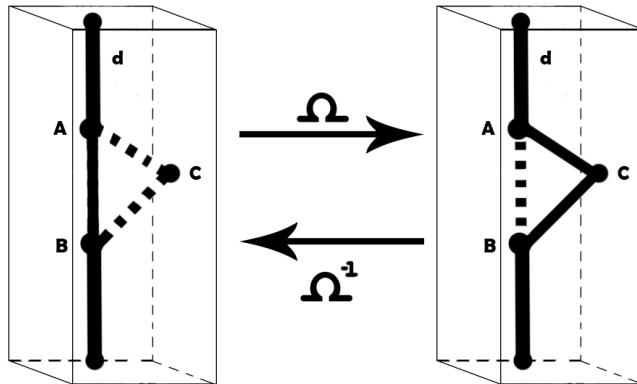


Figura 3.3: Movimiento elemental

Definición 2.1:

Sean dos trenzas β, β' . Diremos que son **equivalentes** ($\beta \sim \beta'$) si existe una cadena finita de trenzas $\beta = \beta_0, \beta_1, \dots, \beta_m = \beta'$ tal que cada par de trenzas $\beta_i, \beta_{i+1}, i = 0, \dots, m-1$, está relacionado por un movimiento elemental. A esta cadena de trenzas equivalentes la representaremos del siguiente modo:

$$\beta = \beta_0 \rightarrow \beta_1 \rightarrow \dots \rightarrow \beta_m = \beta'$$

Si dos trenzas β, β' no son equivalentes, lo denotaremos como $\beta \not\sim \beta'$.

Denotaremos como \mathbf{B}_n al conjunto de todas las trenzas de n cadenas no equivalentes entre sí. Es decir, $B_n = \mathcal{B}_n / \sim$.

3.1.2 Proyección de una trenza.

Consideremos una trenza situada en el cubo $\mathbb{D} = \{(x, y, z) / 0 \leq x, y, z \leq 1\}$. Podemos obtener la proyección de la trenza sobre el plano- yz haciendo la proyección $p(x, y, z) = p(0, y, z)$ de cada uno de los puntos de la misma. De este modo visualizaremos las cadenas como curvas poligonales simples sobre el plano- yz .

Haciendo uso de movimientos elementales, podemos obtener trenzas equivalentes a la inicial de modo que cualquier intersección que vemos en su proyección sea producida por sólo dos cadenas y en un sólo punto. Además podemos obtener trenzas equivalentes tales que las intersecciones de las cadenas se sitúen en distintos niveles. De modo que dada una trenza cualquiera, nos quedaremos con su trenza equivalente cuya proyección cumpla estas condiciones. Podemos ver un ejemplo de cómo obtener una proyección de una trenza en la figura 3.4.

En la figura 3.5 se pueden ver las proyecciones de las trenzas representadas en la figura 3.1.

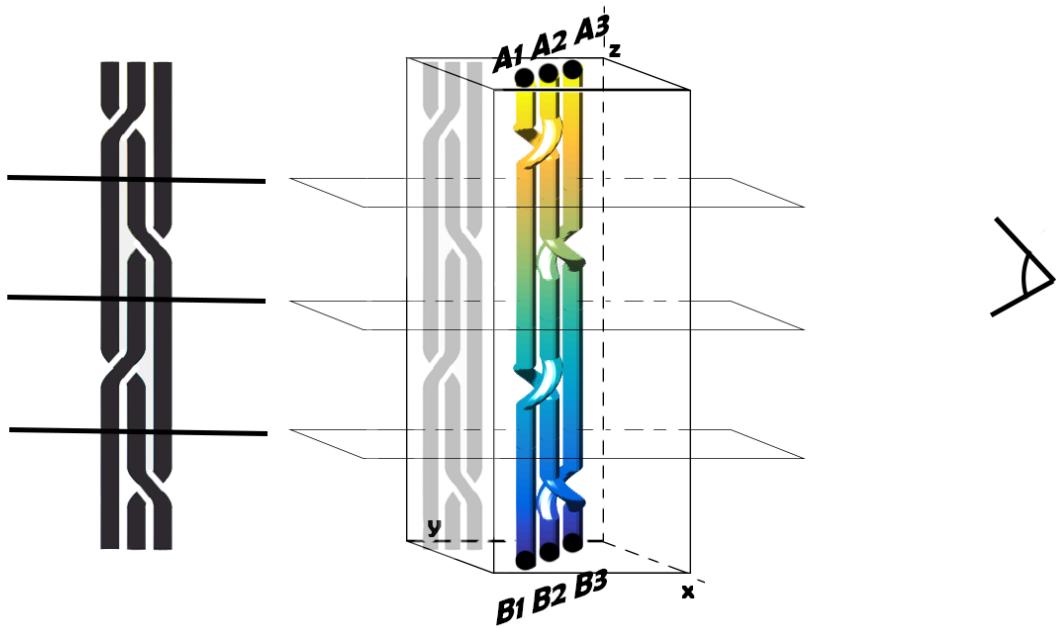


Figura 3.4: Proyección de una trenza



Figura 3.5: Proyección de trenzas

3.1.3 Notación de trenzas:

Para poder trabajar de forma cómoda con las trenzas vamos a darle la siguiente notación:

Supongamos que disponemos de una trenza y de su proyección. Recordemos que modificamos la trenza con movimientos elementales de modo que en cada plano horizontal que corta imaginariamente a la trenza sólo podemos tener dos cadenas que intercambien sus posiciones generando así la intersección que vemos a cierto nivel en la proyección. Vamos a quedarnos con estas secciones de cadenas que producen la intersección en la proyección.

Consideremos que estos segmentos de cadenas unen las posiciones i con la $i + 1$ y las posiciones $i + 1$ con la i . Al producir un intercambio de posiciones de estos segmentos se producirá un **cruce**, que se verá como la intersección de cadenas en la proyección de la trenza. Este cruce puede realizarse de dos formas:

- El segmento que parte de la posición i cruza por delante al segmento que inicialmente parte en la posición $i + 1$. En este caso el cruce se denota como σ_i^{-1} y se conoce como un cruce negativo.
- El segmento que parte de la posición i cruza por detrás al segmento que inicialmente parte en la posición $i + 1$. En este caso el cruce se denota como σ_i^{+1} o simplemente σ_i y se conoce como un cruce positivo.

Podemos verlo más claro en la figura 3.6.

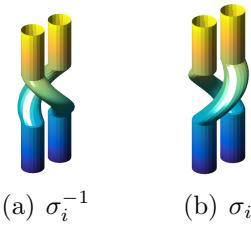


Figura 3.6: Signo cruce.

La **n-trenza trivial** se define como la n-trenza que no realiza ningún cruce. La denotaremos como 1_n .

Cualquier trenza no trivial constará de una serie de cruces. En cada plano horizontal podremos tener como mucho un cruce. Notaremos a la trenza con la secuencia de cruces que tenga, empezando por la parte superior de la trenza. A esta secuencia se le conoce como **palabra** que representa a la trenza. Podemos ver un ejemplo en la figura 3.7.

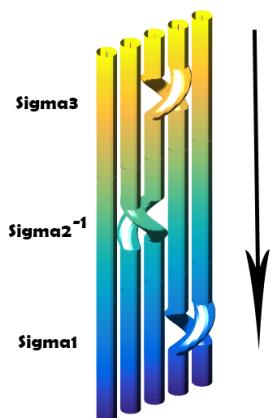


Figura 3.7: Trenza $\sigma_3\sigma_2^{-1}\sigma_4$.

3.2 El grupo de las trenzas.

Hemos visto una definición que nos permite saber cuándo dos trenzas son equivalentes, pero es demasiado general como para poder llevarla a la práctica. En esta sección vamos a hacer un estudio más profundo de la teoría de trenzas. Para ello tenemos que empezar viendo que el conjunto B_n , dotado del producto de trenzas que veremos a continuación, tiene estructura de grupo no abeliano.

3.2.1 Estructura de grupo no abeliano:

Definición 2.2:

Sean las trenzas $\beta, \beta' \in \mathcal{B}_n$. Definimos su **producto** $\beta\beta'$ como la n-trenza que se crea al unir los extremos finales de las cadenas de β con los extremos iniciales de las cadenas de β' .

En la figura 3.8 se puede ver un ejemplo del producto de dos trenzas: la trenza (c) es el producto de las trenzas (a) y (b).

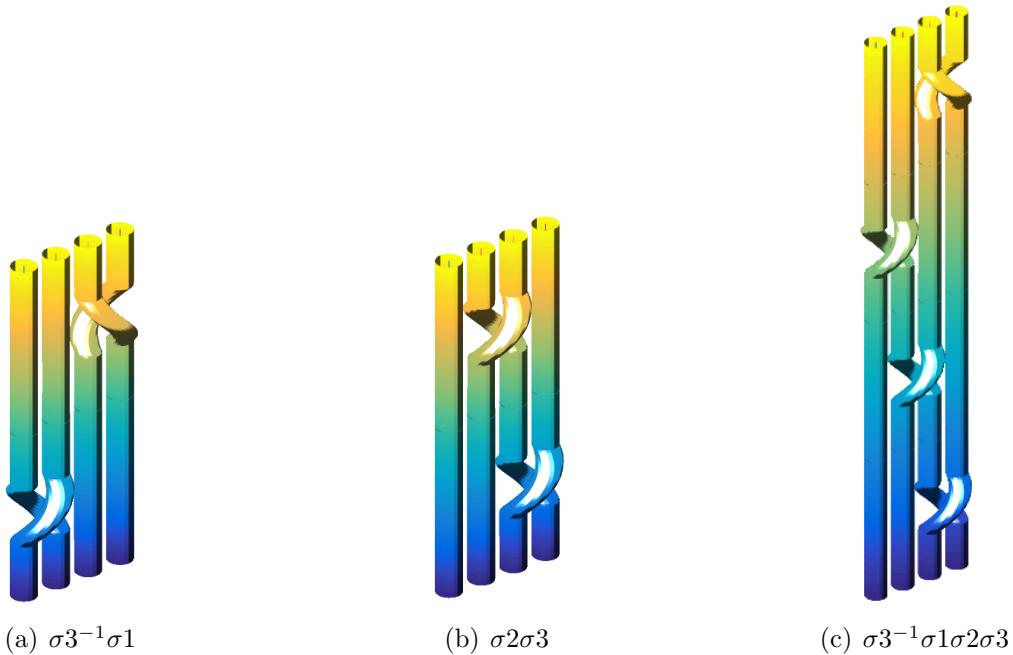


Figura 3.8: Producto trenzas

Proposición 3.2.1. Sean las trenzas $\beta_1, \beta_1', \beta_2, \beta_2' \in \mathcal{B}_n$ verificando las equivalencias $\beta_1 \sim \beta_1'$ y $\beta_2 \sim \beta_2'$. Entonces se verifica la equivalencia $\beta_1\beta_2 \sim \beta_1'\beta_2'$.

Demostración. Por hipótesis tenemos $\beta_1 \sim \beta_1'$ y $\beta_2 \sim \beta_2'$. Por la definición 3.1 sabemos que existen las secuencias de trenzas equivalentes tales que:

$$\beta_1 = \beta_{10} \rightarrow \beta_{11} \rightarrow \dots \rightarrow \beta_{1m} = \beta_1'$$

$$\beta_2 = \beta_{20} \rightarrow \beta_{21} \rightarrow \dots \rightarrow \beta_{2m} = \beta_2'$$

Mediante la primera igualdad tenemos

$$\beta_1\beta_2 = \beta_{10}\beta_2 \rightarrow \beta_{11}\beta_2 \rightarrow \dots \rightarrow \beta_{1m}\beta_2 = \beta_1'\beta_2 \text{ luego } \beta_1\beta_2 \sim \beta_1'\beta_2.$$

Mediante la segunda igualdad tenemos

$$\beta_1'\beta_2 = \beta_1'\beta_{20} \rightarrow \beta_1'\beta_{21} \rightarrow \dots \rightarrow \beta_1'\beta_{2m} = \beta_1'\beta_2' \text{ luego } \beta_1'\beta_2 \sim \beta_1'\beta_2'.$$

Por la transitividad de la equivalencia de trenzas se tiene que

$$\beta_1\beta_2 \sim \beta_1'\beta_2 \sim \beta_1'\beta_2'$$

□

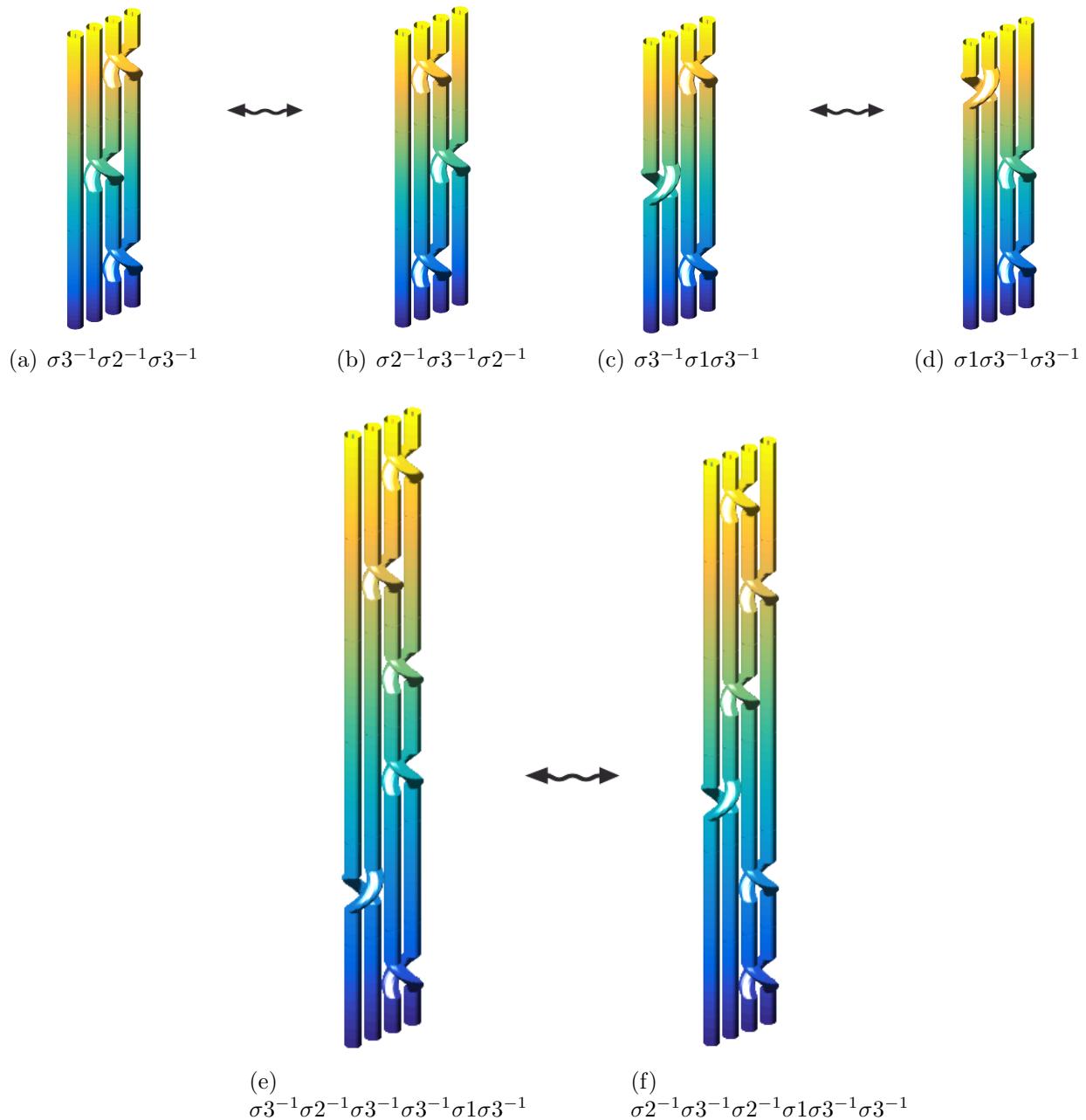


Figura 3.9: Equivalencia producto

Veamos el ejemplo de la figura 3.9: Las trenzas (a) y (b) son equivalentes luego la parte superior de las trenzas (e) y (f) es equivalente. Las trenzas (c) y (d) son equivalentes, luego la parte inferior de las trenzas (e) y (f) es equivalente. La unión de estas dos partes que mencionamos en las trenzas (e) y (f) no supone la pérdida de equivalencia.

Proposición 3.2.2. Producto de trenzas *asociativo*.

Sean las trenzas $\beta_1, \beta_2, \beta_3 \in \mathcal{B}_n$. Se verifica $(\beta_1\beta_2)\beta_3 \sim \beta_1(\beta_2\beta_3)$.

*Demuestra*ción. Por la definición 3.2 sabemos que el producto $(\beta_1\beta_2)\beta_3$ une los extremos finales β_1 con los extremos iniciales de β_2 y posteriormente une los extremos finales de β_2 con los extremos iniciales de β_3 .

Por otra parte el producto $\beta_1(\beta_2\beta_3)$ une los extremos finales β_2 con los extremos iniciales de β_3 y posteriormente une los extremos finales de β_1 con los extremos iniciales de β_2 .

En definitiva, es claro que las trenzas $(\beta_1\beta_2)\beta_3$ y $\beta_1(\beta_2\beta_3)$ son equivalentes. \square

En la figura 3.10 podemos ver que, efectivamente, el producto de las trenzas dadas $\beta_1, \beta_2, \beta_3$ es asociativo: las trenzas (e) y (g) son iguales.

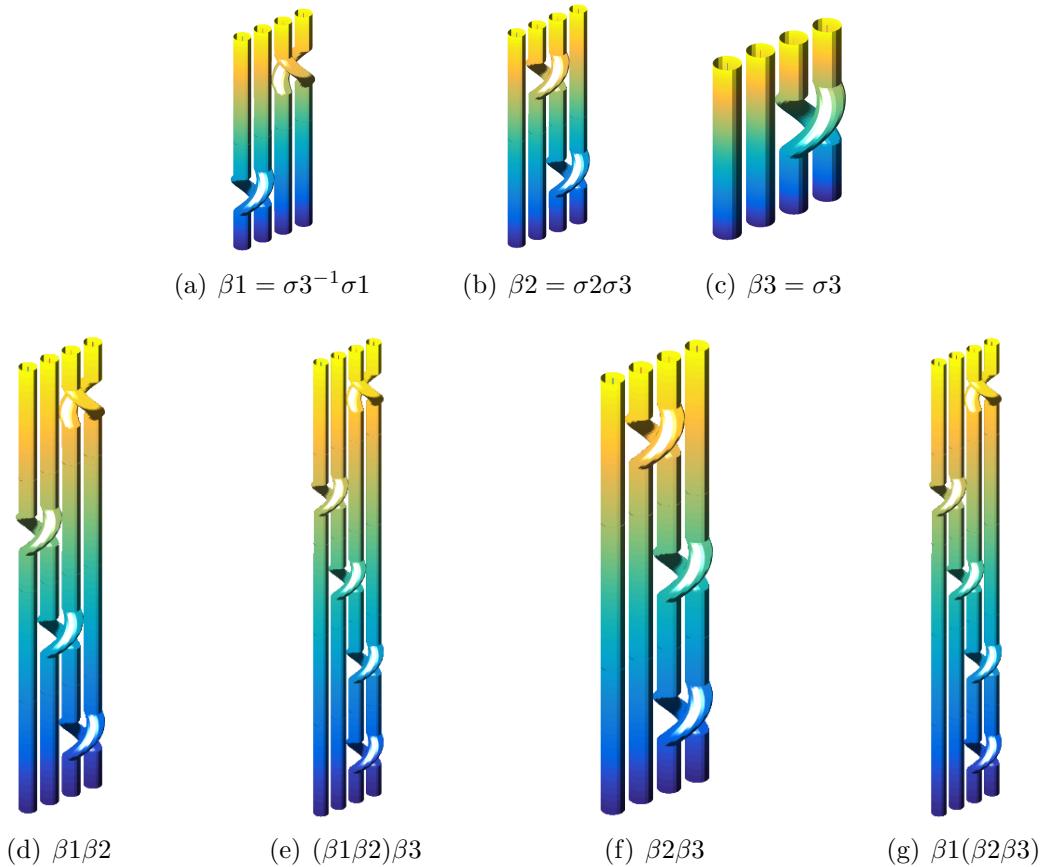


Figura 3.10: Asociatividad producto

Proposición 3.2.3. *Producto de trenzas no conmutativo.*

Sean las trenzas $\beta_1, \beta_2 \in \mathcal{B}_n$. No se tiene porqué verificar la equivalencia $\beta_1\beta_2 \sim \beta_2\beta_1$.

Demuestração. Por la definición 3.2 sabemos que:

El producto $\beta_1\beta_2$ une los extremos finales β_1 con los extremos iniciales de β_2 .

El producto $\beta_2\beta_1$ une los extremos finales β_2 con los extremos iniciales de β_1 .

Es claro que las trenzas resultantes no tienen porqué ser equivalentes. \square

Podemos ver un ejemplo de dos trenzas no conmutativas en la figura 3.11. Efectivamente las trenzas (c) y (e) no son iguales.

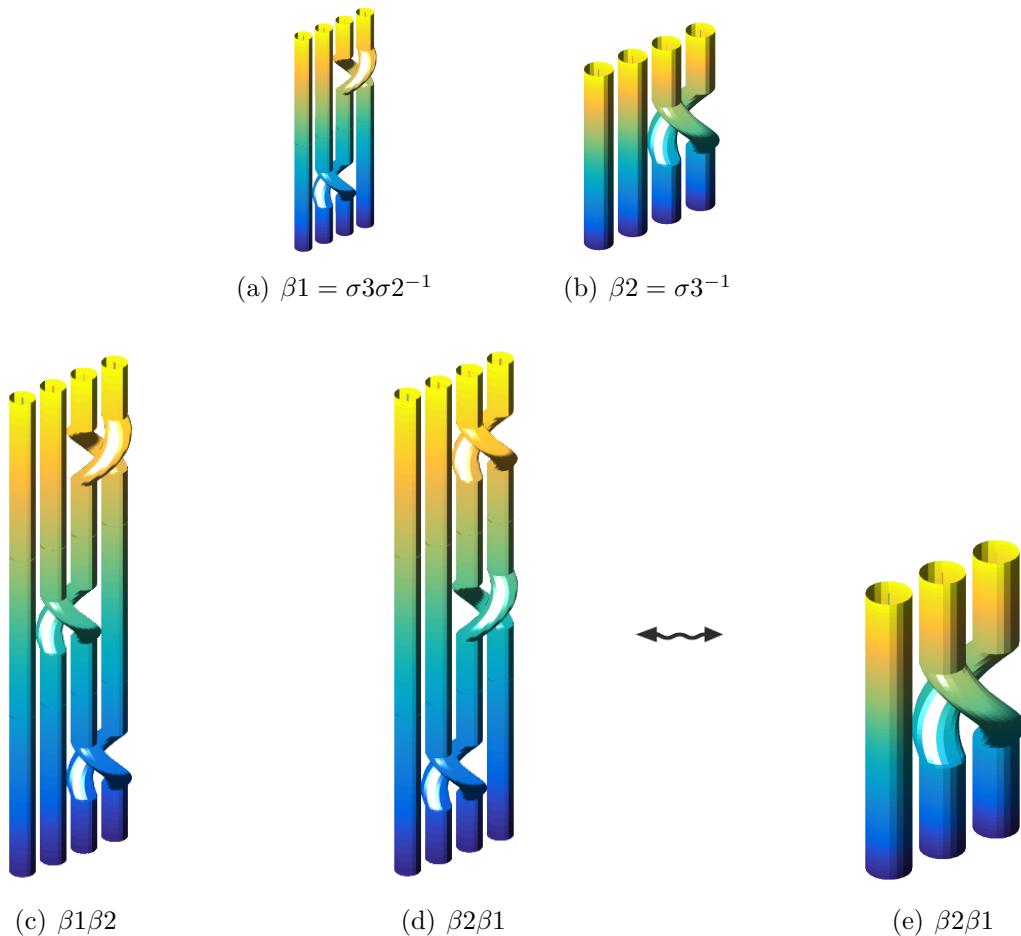


Figura 3.11: No comutatividad producto

Proposición 3.2.4. Elemento neutro.

Sea la trenza $\beta \in \mathcal{B}_n$. Se verifica:

$$\beta 1_n \sim \beta \sim 1_n \beta.$$

Demuestração. Es claro por la definición de n-trenza trivial 1_n (las cadenas de 1_n no tienen cruces luego no afectan a las cadenas de la trenza β). \square

Proposición 3.2.5. Elemento inverso.

Sea la trenza $\beta \in \mathcal{B}_n$. Existirá una trenza $\beta^{-1} \in \mathcal{B}_n$ verificando:

$$\beta\beta^{-1} \sim 1_n \sim \beta^{-1}\beta.$$

A esta trenza β^{-1} se le conoce como trenza inversa.

Demostración. Sea la trenza β a la que notamos como $\beta = \sigma_{i_1}^{\pm 1} \sigma_{i_2}^{\pm 1} \dots \sigma_{i_m}^{\pm 1}$

Construimos la trenza β' a la que notamos como $\beta' = \sigma_{i_m}^{\mp 1} \dots \sigma_{i_2}^{\mp 1} \sigma_{i_1}^{\mp 1}$. Veamos que esta trenza es la trenza inversa:

Por una parte, $\beta\beta' = \sigma_{i_1}^{\pm 1} \sigma_{i_2}^{\pm 1} \dots \sigma_{i_m}^{\pm 1} \sigma_{i_m}^{\mp 1} \dots \sigma_{i_2}^{\mp 1} \sigma_{i_1}^{\mp 1} = \sigma_{i_1}^{\pm 1} \sigma_{i_2}^{\pm 1} \dots \sigma_{i_{m-1}}^{\pm 1} \sigma_{i_{m-1}}^{\mp 1} \dots \sigma_{i_2}^{\mp 1} \sigma_{i_1}^{\mp 1} = 1_n$. Luego $\beta\beta' \sim 1_n$.

Por otra parte, $1_n = \sigma_{i_m}^{\mp 1} \sigma_{i_m}^{\pm 1} = \sigma_{i_m}^{\mp 1} \dots \sigma_{i_2}^{\mp 1} \sigma_{i_2}^{\pm 1} \dots \sigma_{i_m}^{\pm 1} = \sigma_{i_m}^{\mp 1} \dots \sigma_{i_2}^{\mp 1} \sigma_{i_1}^{\mp 1} \sigma_{i_1}^{\pm 1} \sigma_{i_2}^{\pm 1} \dots \sigma_{i_m}^{\pm 1} = \beta'\beta$. Luego $1_n \sim \beta\beta'$.

Por tanto $\beta' = \beta^{-1}$.

Nota: estas igualdades son ciertas porque $\sigma_i^{\mp 1} \sigma_i^{\pm 1} = 1_2$ Se puede ver claro en la figura 3.12.

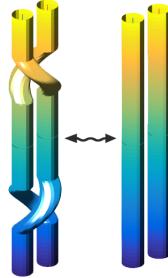


Figura 3.12: Primer movimiento

□

En la figura 3.13 podemos ver un ejemplo de una trenza (a) y su trenza inversa (b). En la secuencia de trenzas (c)-(f) vemos que efectivamente su producto genera la trenza trivial.

Teorema 3.2.1. *El conjunto B_n , dotado del producto de trenzas, es un grupo. El grupo se conoce como el grupo de n -trenzas o el **grupo de n -trenzas de Artin**.*

Demostración. Sea la trenza $\beta \in \mathcal{B}_n$, denotamos su clase de equivalencia como $[\beta] \in B_n$. Veamos que B_n es un grupo:

1. Por la proposición 3.2.1 sabemos que el producto de trenzas está bien definido: Sean $[\beta_1], [\beta_2] \in B_n$, se tiene que $[\beta_1][\beta_2] = [\beta_1\beta_2]$
2. Por la proposición 3.2.2 sabemos que el producto de trenzas es asociativo.

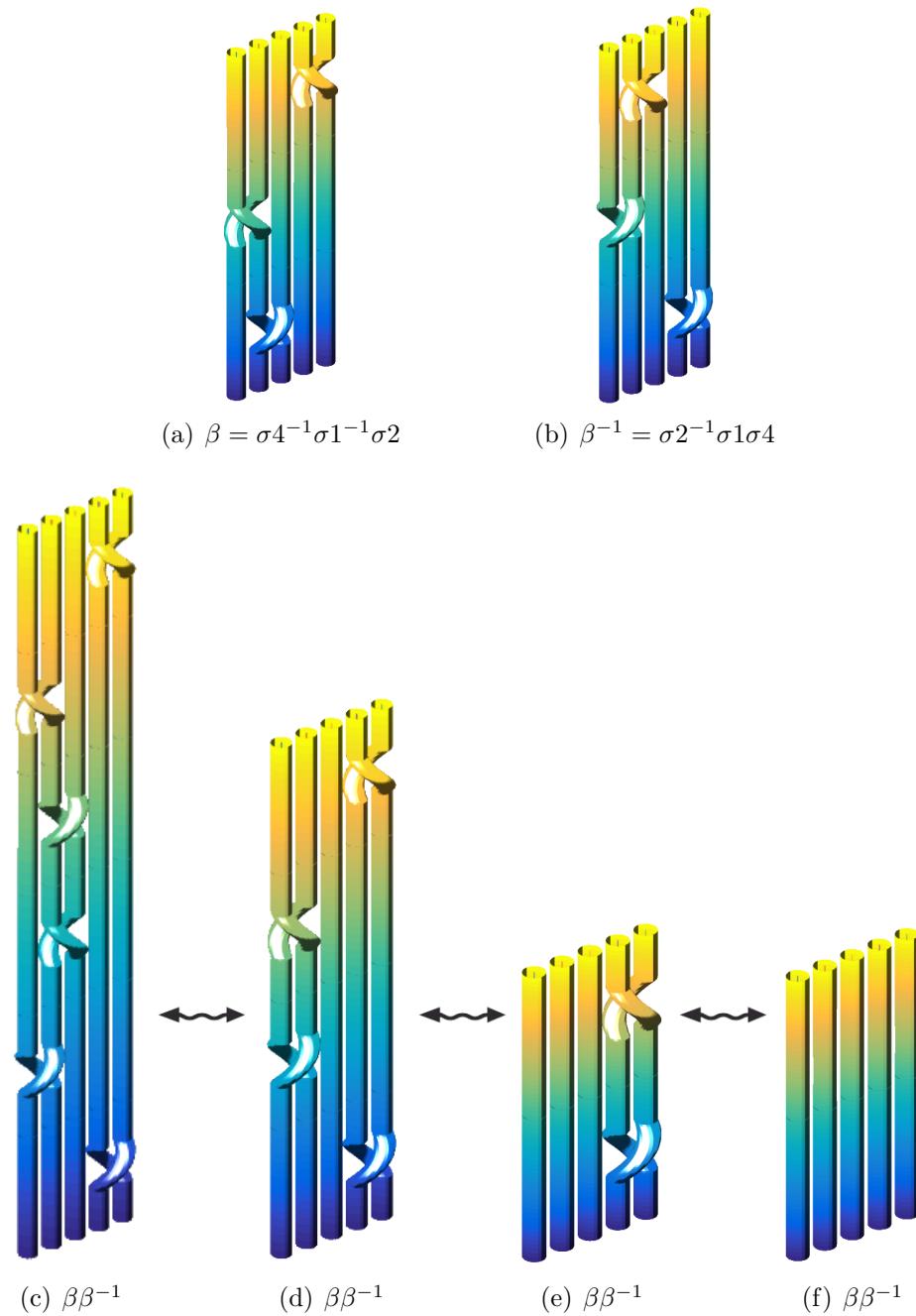


Figura 3.13: Trenzas inversas

3. Por la proposición 3.2.4 sabemos que la n -trenza trivial es el elemento identidad para el producto de trenzas.
4. Por la proposición 3.2.5 sabemos que el elemento inverso de $[\beta] \in B_n$ es $[\beta^{-1}]$, luego $[\beta]^{-1} = [\beta^{-1}]$.

□

3.2.2 Trenzas equivalentes:

Hemos definido \mathcal{B}_n como el conjunto de todas las n-trenzas, siendo estas n-trenzas representadas por palabras. En concreto si tenemos la trenza $\beta \in \mathcal{B}_n$, que consta de m-cruces, la podremos representar como:

$$\beta = \sigma_{i_1}^{\pm 1} \sigma_{i_2}^{\pm 1} \dots \sigma_{i_m}^{\pm 1}, \quad 1 \leq i_1, i_2, \dots, i_m \leq n-1.$$

Además, hemos visto que el conjunto $B_n = \mathcal{B}_n / \sim$, de todas las n-trenzas no equivalentes entre sí, tiene estructura de grupo al dotarle del producto de trenzas.

En esta sección vamos a ver cuáles son los movimientos que nos permitirán estudiar la equivalencia de dos n-trenzas y posteriormente vamos a reflejar estos movimientos como el conjunto de relaciones que representan al conjunto B_n .

Para analizar cuándo dos trenzas son equivalentes tendremos que ver cuándo las palabras que representan a dichas trenzas son equivalentes. Dos palabras serán equivalentes si y sólo si podemos pasar de una palabra a otra mediante un secuencia de estos tres movimientos:

1. Primer movimiento - M1:

Podemos añadir o eliminar $\sigma_i \sigma_i^{-1}$ o $\sigma_i^{-1} \sigma_i$ en cualquier palabra. Es claro que la palabra inicial y la palabra final representan a la misma trenza. Podemos verlo en la figura 3.14.

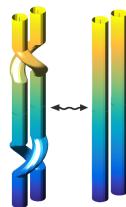


Figura 3.14: Primer movimiento.

2. Segundo movimiento - M2:

Las palabras $\sigma_i \sigma_{i+1} \sigma_i$ y $\sigma_{i+1} \sigma_i \sigma_{i+1}$ son equivalentes (o bien con cruces negativos). Se puede ver en la figura 3.15.

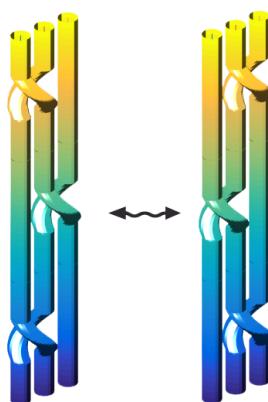


Figura 3.15: Segundo movimiento.

3. Tercer movimiento - M3:

Las palabras $\sigma_i\sigma_j$ y $\sigma_j\sigma_i$ son equivalentes si se verifica que $|i - j| > 1$ (o bien con cruces negativos). Se ve claro en la figura 3.16.

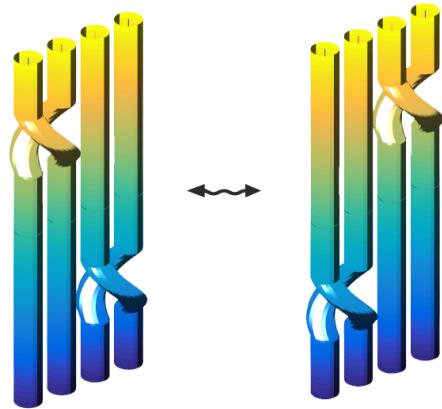


Figura 3.16: Tercer movimiento.

En la figura 3.17 podemos ver los movimientos que nos demuestran que dos trenzas dadas más complejas son equivalentes.

Partimos de la trenza $\sigma_3^{-1}\sigma_1\sigma_2^{-1}\sigma_3^{-1}$ y aplicamos M3 a los dos primeros cruces obteniendo la trenza $\sigma_1\sigma_3^{-1}\sigma_2^{-1}\sigma_3^{-1}$ que vemos en la segunda imagen.

Por último, aplicamos el movimiento M2 a los tres últimos cruces obteniendo la trenza $\sigma_1\sigma_2^{-1}\sigma_3^{-1}\sigma_2^{-1}$ que vemos en la última imagen.

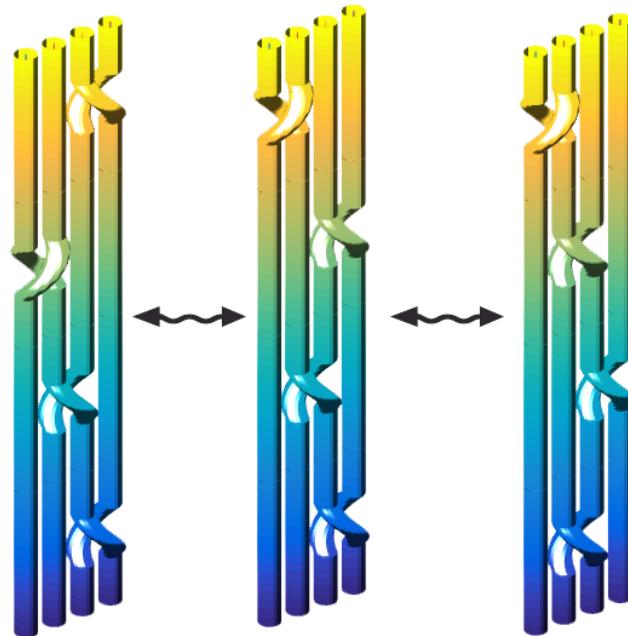


Figura 3.17: Trenzas equivalentes.

Teorema 3.2.2. Defino las relaciones:

1. $\sigma_{i+1}\sigma_i\sigma_{i+1} = \sigma_i\sigma_{i+1}\sigma_i$ siendo $i = 2, \dots, n-2$
2. $\sigma_i\sigma_j = \sigma_j\sigma_i$ siendo $1 \leq i < j \leq n-1$, $j-i \geq 2$

El grupo B_n tiene la siguiente representación:

$$B_n = \langle \sigma_1, \sigma_2, \dots, \sigma_{n-1} / \text{las relaciones 1 y 2 se verifican} \rangle$$

A partir de estas relaciones de equivalencia base podremos construir nuevas relaciones de equivalencia. Vamos a mostrar algunas relaciones de equivalencia que usaremos posteriormente:

1. $\sigma_{i+1}\sigma_i\sigma_{i+1}^{-1} = \sigma_i^{-1}\sigma_{i+1}\sigma_i$ siendo $i = 2, \dots, n-2$
2. $\sigma_{i+1}\sigma_i^{-1}\sigma_{i+1}^{-1} = \sigma_i^{-1}\sigma_{i+1}^{-1}\sigma_i$ siendo $i = 2, \dots, n-2$

Demostración:

1. $\sigma_{i+1}\sigma_i\sigma_{i+1}^{-1} = \sigma_i^{-1}\sigma_i\sigma_{i+1}\sigma_i\sigma_{i+1}^{-1} = \sigma_i^{-1}\sigma_{i+1}\sigma_i\sigma_{i+1}\sigma_{i+1}^{-1} = \sigma_i^{-1}\sigma_{i+1}\sigma_i$
2. $\sigma_{i+1}\sigma_i^{-1}\sigma_{i+1}^{-1} = \sigma_{i+1}\sigma_i^{-1}\sigma_{i+1}^{-1}\sigma_i^{-1}\sigma_i = \sigma_{i+1}\sigma_{i+1}^{-1}\sigma_i^{-1}\sigma_{i+1}^{-1}\sigma_i = \sigma_i^{-1}\sigma_{i+1}^{-1}\sigma_i$

En la figura 3.18 podemos visualizar las equivalencias.



Figura 3.18: Trenzas equivalentes.

3.2.3 Trenzas Markov-equivalentes:

Es claro que si tenemos dos trenzas que son equivalentes, sus trenzas cerradas nos darán nudos que serán equivalentes. Pero... ¿puede darse el caso de tener dos trenzas no sean equivalentes y que sus trenzas cerradas sí lo sean? Veamos, mediante el ejemplo de la figura 3.19 que sí es posible: las trenzas de partida no son equivalentes (tenemos distinto número de cadenas), sin embargo sus trenzas cerradas sí lo son. Lo demostraremos en la figura 3.22.

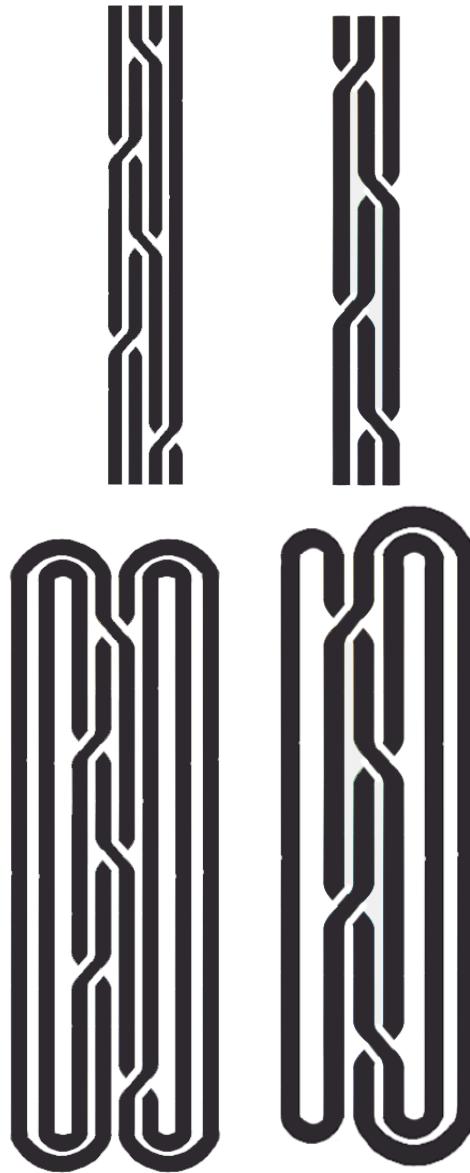


Figura 3.19: Trenzas Markov-equivalentes.

Definición:

Se dice que dos trenzas son **Markov-equivalentes** si sus cierres producen el mismo enlace. En este caso tendremos que los nudos representados por las trenzas son equivalentes.

Teorema 3.2.3. Teorema de Markov.

Dos trenzas son Markov-equivalentes si y solo si podemos pasar de una trenza a otra mediante una secuencia de las tres operaciones que hemos visto en la subsección 3.2.2 y los movimientos de Markov.

Veamos los movimientos de Markov:

1. Conjugación - Mv1:

Las palabras β y $\sigma_i\beta\sigma_i^{-1}$ (o bien $\sigma_i^{-1}\beta\sigma_i$) generan trenzas cerradas equivalentes. Se puede ver en la figura 3.20.

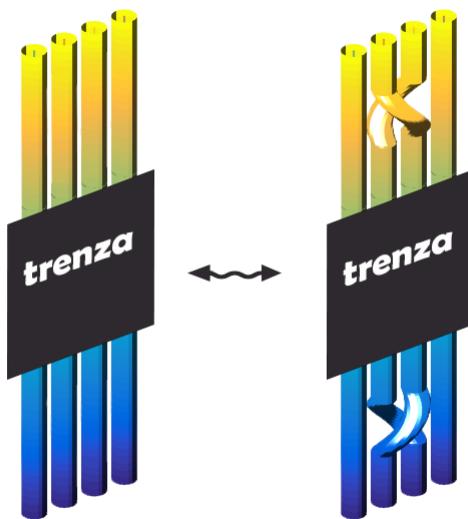


Figura 3.20: Conjugación Markov.

2. Estabilización - Mv2:

Esta operación nos permite modificar el número de cadenas de las trenzas. Sea la palabra β de n cadenas. Esta palabra genera una trenza cerrada equivalente a $\beta\sigma_n$ o bien $\sigma_n\beta$. Se puede ver en la figura 3.21.

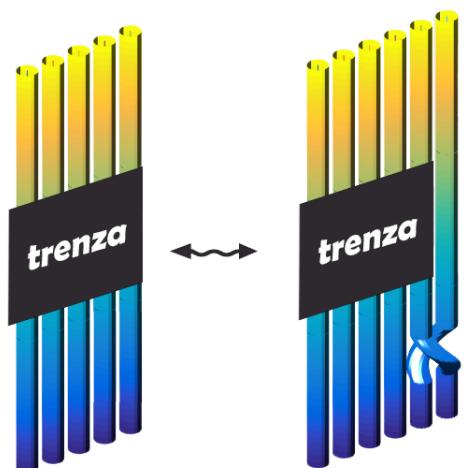


Figura 3.21: Estabilización Markov.

Podemos ver un ejemplo de dos trenzas Markov-equivalentes en la figura 3.22.

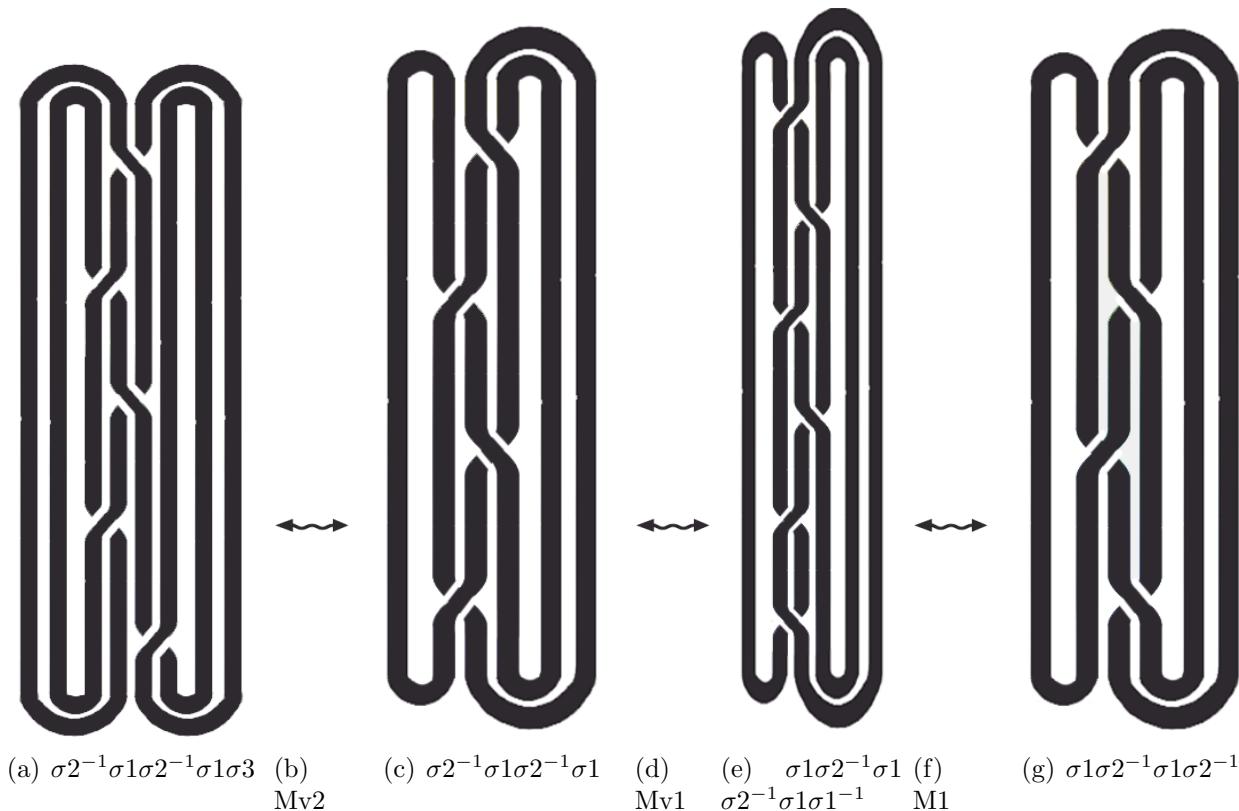


Figura 3.22: Trenzas Markov-equivalentes.

3.3 Algunos invariantes.

Ya conocemos cuáles son los movimientos básicos que nos permiten determinar si dos trenzas dadas son equivalentes. Pero al igual que ocurría con los nudos, llevar esta idea a la práctica no es factible.

En esta sección vamos a ver algunos invariantes de trenzas que nos permitirán determinar de forma relativamente rápida si dos trenzas no son equivalentes.

Definición:

Un **invariante** de una trenza es una propiedad que no cambia cuando la trenza sufre deformaciones en el espacio.

Vamos a ver algunos invariantes de trenzas que usaremos posteriormente en la práctica.

3.3.1 Exponente:

Definición:

Sea $\beta \in B_n$ representada como $\beta = \sigma_{i_1}^{e_1} \sigma_{i_2}^{e_2} \dots \sigma_{i_m}^{e_m}$ donde $e_i \in \{-1, 1\}$, $1 \leq i_1, i_2, \dots, i_m \leq n-1$. Llamamos **exponente** de β al entero $\sum_{i=1}^m e_i$. Se denotará como $\exp(\beta)$.

El exponente de la trenza $\beta = \sigma 3 \sigma 2^{-1} \sigma 3^{-1}$ de la figura 3.23 es $+1 - 1 - 1 = -1$.



Figura 3.23: Trenza $\sigma 3 \sigma 2^{-1} \sigma 3^{-1}$

Proposición 3.3.1. *El exponente de una trenza $\beta \in B_n$ es un invariante.*

Demuestra. Veamos que dadas las trenzas $\beta_1, \beta_2 \in B_n$ tales que $\beta_1 \sim \beta_2$, se verifica que $\exp(\beta_1) = \exp(\beta_2)$.

Al ser $\beta_1 \sim \beta_2$, por la definición 3.1, sabemos que existe la secuencia de trenzas equivalentes tal que:

$$\beta_1 = \beta_{10} \rightarrow \beta_{11} \rightarrow \dots \rightarrow \beta_{1m} = \beta_2$$

donde cada par de trenzas $\beta_j, \beta_{j+1}, j = 0, \dots, m-1$, está relacionado por un movimiento elemental. Estos movimientos elementales se pueden reducir a las relaciones las 1 y 2 que definen al grupo B_n . Por tanto, para verificar que el exponente de cada par β_j, β_{j+1} es igual, tenemos que ver que estas relaciones de igualdad tienen el mismo exponente:

1. $\sigma_{i+1} \sigma_i \sigma_{i+1} = \sigma_i \sigma_{i+1} \sigma_i$ siendo $i = 2, \dots, n-2$:

$$\exp(\sigma_{i+1} \sigma_i \sigma_{i+1}) = \exp(\sigma_i \sigma_{i+1} \sigma_i).$$

2. $\sigma_i \sigma_j = \sigma_j \sigma_i$ siendo $1 \leq i < j \leq n-1$, $j-i \geq 2$:

$$\exp(\sigma_i \sigma_j) = \exp(\sigma_j \sigma_i).$$

□

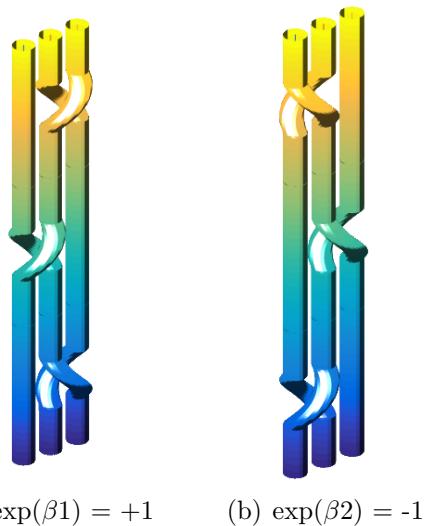
(a) $\exp(\beta_1) = +1$ (b) $\exp(\beta_2) = -1$

Figura 3.24: Trenzas no equivalentes.

Haciendo uso de este invariante podremos ver muy fácilmente si dos trenzas dadas tienen distintos exponente y por tanto no son equivalentes. Podemos ver un ejemplo de dos trenzas no equivalentes en la figura 3.24. La trenza $\beta_1 = \sigma_2\sigma_1\sigma_2^{-1}$ tiene exponente $+1$, mientras que la trenza $\beta_2 = \sigma_2^{-1}\sigma_1^{-1}\sigma_2$ tiene exponente -1 .

Si tuviesen igual exponente tendríamos que estudiar otros invariantes para ver si las trenzas son iguales. Por ejemplo, en la figura 3.25 vemos que las trenzas $\beta_1 = \sigma_2\sigma_2\sigma_3^{-1}$ y $\beta_2 = \sigma_3^{-1}\sigma_2\sigma_1$ tienen el mismo exponente ($+1$) luego no podremos saber si son o no equivalentes. Al estudiar el invariante de la siguiente sección veremos que no lo son.

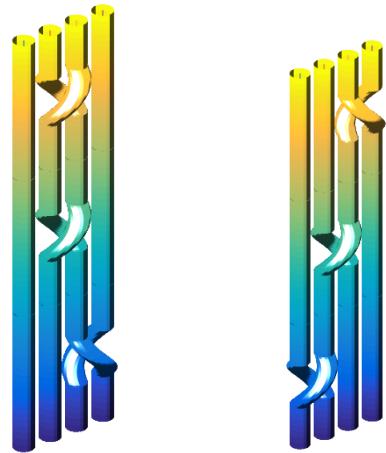
(a) $\exp(\beta_1) = +1$ (b) $\exp(\beta_2) = +1$

Figura 3.25: Trenzas con mismo exponente.

3.3.2 Permutación:

Definición:

Sea $\beta \in B_n$. Consideramos la cadena i que conecta el punto inicial A_i con el punto final B_{j_i} . Llamaremos **permutación** asociada a la trenza a la matriz:

$$\pi(\beta) = \begin{bmatrix} 1 & 2 & .. & n \\ j_1 & j_2 & .. & j_n \end{bmatrix}$$

Por simplicidad podremos denotar la permutación de la trenza como el vector $j_1 j_2 \dots j_n$.

La permutación de la trenza $\beta = \sigma 3 \sigma 2^{-1} \sigma 3^{-1}$ de la figura 3.26 es

$$\pi(\beta) = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 3 & 2 \end{bmatrix}$$

o bien directamente la permutación: 1 4 3 2.



Figura 3.26: Trenza $\sigma 3 \sigma 2^{-1} \sigma 3^{-1}$

Proposición 3.3.2. *La permutación de una trenza $\beta \in B_n$ es un invariante.*

Demuestra. Veamos que dadas las trenzas $\beta_1, \beta_2 \in B_n$ tales que $\beta_1 \sim \beta_2$, se verifica que $\pi(\beta_1) = \pi(\beta_2)$.

Por la definición 3.1, existirá la secuencia de trenzas equivalentes tal que:

$$\beta_1 = \beta_{10} \rightarrow \beta_{11} \rightarrow \dots \rightarrow \beta_{1m} = \beta_2$$

donde cada par de trenzas $\beta_j, \beta_{j+1}, j = 0, \dots, m-1$, está relacionado por un movimiento elemental. Los movimientos elementales, por definición, no permiten intercambiar los extremos a los que están sujetas las cuerdas de las trenzas. Por este motivo las permutaciones tienen que ser iguales.

□

Haciendo uso de este invariante podemos ver un ejemplo de dos trenzas que no son equivalentes en la figura 3.27: la trenza $\beta_1 = \sigma_2\sigma_2\sigma_3^{-1}$ tiene permutación 1 2 4 3 mientras que la trenza $\beta_2 = \sigma_3^{-1}\sigma_2\sigma_1$ tiene permutación 2 3 4 1.

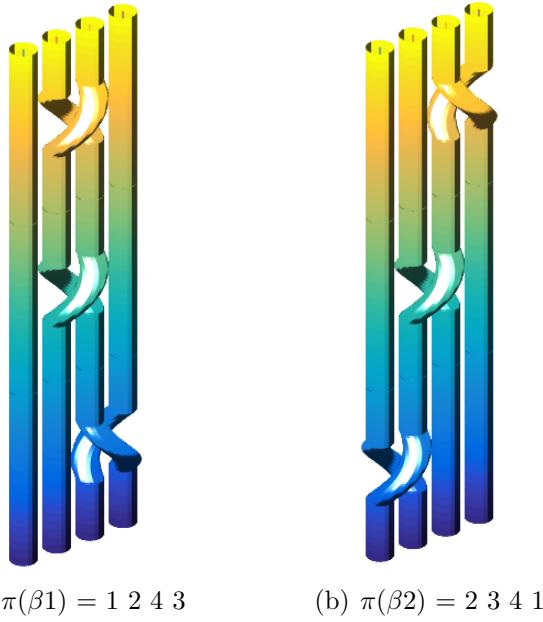


Figura 3.27: Trenzas no equivalentes.

3.3.3 Polinomio de Alexander:

En la sección 2.5.5 estudiamos el polinomio de Alexander como un invariante para nudos. Por el teorema 3.2.3, podremos hacer uso de las trenzas como una base para obtener los invariantes de nudos, en concreto, para estudiar este invariante que implementaremos posteriormente.

Para poder obtener el polinomio de Alexander de una trenza, tendremos que ver unos conceptos previos, [Murasugi and Kurpita, 1999]:

Definición:

Sea la trenza $\beta = \sigma_{i_1}^{e_1} \sigma_{i_2}^{e_2} \dots \sigma_{i_m}^{e_m}$ donde $e_i \in \{-1, 1\}$, $1 \leq i_1, i_2, \dots, i_m \leq n-1$. Podemos definir el homomorfismo

$$\phi_n : B_n \rightarrow M(n, \mathbb{Z}[t, t^{-1}])$$

$$\phi_n(\sigma_i) = \begin{bmatrix} I_{i-1} & & & \\ & 1-t & t & \\ & 1 & 0 & \\ & & & I_{n-i-1} \end{bmatrix}$$

donde I_i representa a la matriz ixi identidad y los espacios en blanco representan matrices nulas. Llamaremos a esta representación como **representación de Burau**.

En consecuencia tenemos

$$\phi_n(\sigma_i^{-1}) = \phi_n(\sigma_i)^{-1} = \begin{bmatrix} I_{i-1} & & & \\ & 0 & 1 & \\ & t^{-1} & 1-t^{-1} & \\ & & & I_{n-i-1} \end{bmatrix}$$

Veamos la matriz de Burau de la trenza $\beta \in B_4$ representada como $\beta = \sigma_2\sigma_3^{-1}$.

Matriz Burau de σ_2 :

$$\phi_4(\sigma_2) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1-t & t & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matriz Burau de σ_3^{-1} :

$$\phi_4(\sigma_3^{-1}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & t^{-1} & 1-t^{-1} \end{bmatrix}$$

Por tanto, la matriz Burau de $\beta = \sigma_2\sigma_3^{-1}$ es:

$$\phi_4(\sigma_2\sigma_3^{-1}) = \phi_4(\sigma_2)\phi_4(\sigma_3^{-1}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1-t & 0 & t \\ 0 & 1 & 0 & 0 \\ 0 & 0 & t^{-1} & 1-t^{-1} \end{bmatrix}$$

Teorema 3.3.1. *Sea la trenza $\beta \in B_n$ con representación de Burau $\phi_n(\beta)$. Supongamos que su trenza cerrada genera el nudo K .*

Entonces $\exists k \in \mathbb{Z}$ tal que el polinomio $(\pm t^k)\det[\phi_n(\beta) - I_n]_{1,1}$ es un invariante de K . A este polinomio, conocido como polinomio de Alexander, se le denota como $\Delta_k(t)$.

Nota: Sea la matriz cuadrada A de dimensión n y A' la matriz de dimensión $n-1$ que se obtiene al suprimir en la matriz A la fila i y la columna i . Se tiene $\det[A]_{i,i} = \det[A']$.

Gracias a este teorema podemos afirmar que si dos trenzas $\beta_1 \in B_n, \beta_2 \in B_m$ cerradas generan el mismo nudo entonces se verificará la igualdad:

$$\det[\phi_n(\beta_1) - I_n]_{1,1} = \pm t^k \det[\phi_m(\beta_2) - I_m]_{1,1}.$$

Para ver la demostración necesitamos un lema previo y los siguientes conceptos:

Sea $\phi_n(\beta) = M = \|\alpha_{ij}\|$ $1 \leq i, j \leq n$, donde $\beta \in B_n$. Definimos las siguientes matrices de dimensión nxn:

$$S = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 & 1 \\ 0 & 1 & 1 & \dots & 1 & 1 \\ 0 & 0 & 1 & \dots & 1 & 1 \\ \dots & \dots & \dots & \dots & 1 & 1 \\ 0 & 0 & 0 & \dots & 1 & 1 \\ 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix} S^{-1} = \begin{bmatrix} 1 & -1 & 0 & \dots & 0 & 0 \\ 0 & 1 & -1 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & 1 & -1 \\ 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix}$$

Podemos considerar el producto de matrices $S^{-1}MS$

$$\begin{aligned} MS &= \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \dots & a_{2,n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{n,1} & a_{n,2} & a_{n,3} & \dots & a_{n,n} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 0 & 1 & 1 & \dots & 1 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \\ &= \begin{bmatrix} a_{1,1} & a_{1,1} + a_{1,2} & \dots & a_{1,1} + a_{1,2} + \dots + a_{1,n} = 1 \\ a_{2,1} & a_{2,1} + a_{2,2} & \dots & a_{2,1} + a_{2,2} + \dots + a_{2,n} = 1 \\ \dots & \dots & \dots & \dots \\ a_{n,1} & a_{n,1} + a_{n,2} & \dots & a_{n,1} + a_{n,2} + \dots + a_{n,n} = 1 \end{bmatrix} \end{aligned}$$

Luego la matriz $S^{-1}MS$ tendrá la siguiente forma:

$$\begin{aligned} S^{-1}MS &= \begin{bmatrix} 1 & -1 & 0 & \dots & 0 \\ 0 & 1 & -1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \begin{bmatrix} a_{1,1} & a_{1,1} + a_{1,2} & \dots & a_{1,1} + a_{1,2} + \dots + a_{1,n} = 1 \\ a_{2,1} & a_{2,1} + a_{2,2} & \dots & a_{2,1} + a_{2,2} + \dots + a_{2,n} = 1 \\ \dots & \dots & \dots & \dots \\ a_{n,1} & a_{n,1} + a_{n,2} & \dots & a_{n,1} + a_{n,2} + \dots + a_{n,n} = 1 \end{bmatrix} \\ &= \begin{bmatrix} a_{1,1} - a_{2,1} & a_{1,1} + a_{1,2} - a_{2,1} - a_{2,2} & \dots & 1 - 1 = 0 \\ a_{2,1} - a_{3,1} & a_{2,1} + a_{2,2} - a_{3,1} - a_{3,2} & \dots & 1 - 1 = 0 \\ \dots & \dots & \dots & \dots \\ a_{n,1} & a_{n,1} + a_{n,2} & \dots & 1 \end{bmatrix} \end{aligned}$$

De un modo más simple, vamos a denotar a dicha matriz del siguiente modo:

$$S^{-1}MS = \left[\begin{array}{c|c} \Lambda(t) & 0 \\ \hline * \dots * & 1 \end{array} \right]$$

Nota: Es claro que $\det[S^{-1}(M - I_n)S]_{n,n} = \det(\Lambda(t) - I_{n-1})$.

Lema 3.3.1. Se verifica la siguiente igualdad:

$$\det(\Lambda(t) - I_{n-1}) = (1+t+\dots+t^{n-1}) \det[M - I_n]_{1,1}$$

*Demuestra*ción. Consideramos la matriz de dimensión $(n-1) \times (n-1)$:

$$\Lambda(t) - I_{n-1} =$$

$$\begin{bmatrix} a_{1,1} - a_{2,1} - 1 & a_{1,1} + a_{1,2} - a_{2,1} - a_{2,2} & \dots & a_{1,1} + \dots + a_{1,n-1} - a_{2,1} - \dots - a_{2,n-1} \\ a_{2,1} - a_{3,1} & a_{2,1} + a_{2,2} - a_{3,1} - a_{3,2} - 1 & \dots & a_{2,1} + \dots + a_{2,n-1} - a_{3,1} - \dots - a_{3,n-1} \\ \dots & \dots & \dots & \dots \\ a_{n-1,1} - a_{n,1} & a_{n-1,1} + a_{n-1,2} - a_{n,1} - a_{n,2} & \dots & a_{n-1,1} + \dots + a_{n-1,n-1} - a_{n,1} - \dots - a_{n,n-1} \end{bmatrix}$$

Es claro que $\det(\Lambda(t) - I_{n-1}) = \det(S^T(\Lambda(t) - I_{n-1})S^{-1})$ donde S^T es la matriz traspuesta de dimensión $(n-1) \times (n-1)$ de la matriz S . Por tanto, vamos a trabajar con la matriz $(S^T(\Lambda(t) - I_{n-1})S^{-1})$:

$$\begin{aligned} (\Lambda(t) - I_{n-1})S^{-1} &= (\Lambda(t) - I_{n-1}) \begin{bmatrix} 1 & -1 & 0 & \dots & 0 \\ 0 & 1 & -1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} = \\ &= \begin{bmatrix} a_{1,1} - a_{2,1} - 1 & a_{1,2} - a_{2,2} + 1 & \dots & a_{1,n-1} - a_{2,n-1} \\ a_{2,1} - a_{3,1} & a_{2,2} - a_{3,2} - 1 & \dots & a_{2,n-1} - a_{3,n-1} \\ \dots & \dots & \dots & \dots \\ a_{n-1,1} - a_{n,1} & a_{n-1,2} - a_{n,2} & \dots & a_{n-1,n-1} - a_{n,n-1} \end{bmatrix} \end{aligned}$$

Luego:

$$\begin{aligned} S^T(\Lambda(t) - I_{n-1})S^{-1} &= \\ &= \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & 1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix} \begin{bmatrix} a_{1,1} - a_{2,1} - 1 & a_{1,2} - a_{2,2} + 1 & \dots & a_{1,n-1} - a_{2,n-1} \\ a_{2,1} - a_{3,1} & a_{2,2} - a_{3,2} - 1 & \dots & a_{2,n-1} - a_{3,n-1} \\ \dots & \dots & \dots & \dots \\ a_{n-1,1} - a_{n,1} & a_{n-1,2} - a_{n,2} & \dots & a_{n-1,n-1} - a_{n,n-1} \end{bmatrix} = \\ &= \begin{bmatrix} a_{1,1} - a_{2,1} - 1 & a_{1,2} - a_{2,2} + 1 & \dots & a_{1,n-1} - a_{2,n-1} \\ a_{1,1} - a_{3,1} - 1 & a_{1,2} - a_{3,2} & \dots & a_{1,n-1} - a_{3,n-1} \\ \dots & \dots & \dots & \dots \\ a_{1,1} - a_{n,1} - 1 & a_{1,2} - a_{n,2} & \dots & a_{1,n-1} - a_{n,n-1} \end{bmatrix} = \begin{bmatrix} A_1 - A_2 \\ A_1 - A_3 \\ \dots \\ A_1 - A_n \end{bmatrix} \end{aligned}$$

Donde los vectores A_i $1 \leq i \leq n$ son las filas de la matriz $[M - I_n]_{0,n}$, es decir, tienen la siguiente forma:

$$\begin{aligned} A_1 &= [a_{1,1} - 1, a_{1,2}, \dots, a_{1,n-1}] \\ A_2 &= [a_{2,1}, a_{2,2} - 1, \dots, a_{2,n-1}] \\ &\quad \dots \\ A_n &= [a_{n,1}, a_{n,2}, \dots, a_{n,n-1}] \end{aligned}$$

Por tanto $\det(\Lambda(t) - I_{n-1}) = \det(S^T(\Lambda(t) - I_{n-1})S^{-1}) =$

$$\begin{aligned} &= \det \begin{bmatrix} A_1 - A_2 \\ A_1 - A_3 \\ \dots \\ A_1 - A_n \end{bmatrix} = \\ &= \det \begin{bmatrix} A_1 \\ A_1 - A_3 \\ \dots \\ A_1 - A_n \end{bmatrix} + \det \begin{bmatrix} -A_2 \\ A_1 - A_3 \\ \dots \\ A_1 - A_n \end{bmatrix} = \det \begin{bmatrix} A_1 \\ -A_3 \\ \dots \\ -A_n \end{bmatrix} + \det \begin{bmatrix} -A_2 \\ A_1 - A_3 \\ \dots \\ A_1 - A_n \end{bmatrix} = \dots = \\ &= \det \begin{bmatrix} A_1 \\ -A_3 \\ \dots \\ -A_n \end{bmatrix} + \det \begin{bmatrix} -A_2 \\ A_1 \\ \dots \\ -A_n \end{bmatrix} + \dots + \det \begin{bmatrix} -A_2 \\ -A_3 \\ \dots \\ -A_{k+2} \\ A_1 \\ -A_n \end{bmatrix} + \dots + \det \begin{bmatrix} -A_2 \\ -A_3 \\ \dots \\ -A_{n-1} \\ A_1 \\ -A_n \end{bmatrix} + \det \begin{bmatrix} -A_2 \\ -A_3 \\ \dots \\ -A_n \end{bmatrix} \end{aligned}$$

Para obtener el valor de estos determinantes vamos a hacer uso de la siguiente igualdad:

$$\det[M - I_n]_{p,q} = (-1)^{p-q} t^{p-1} \det[M - I_n]_{1,1} \quad 1 \leq p, q \leq n$$

De modo que tomando $p = k + 1$ y $q = n$, se tiene:

$$\det \begin{bmatrix} -A_2 \\ -A_3 \\ \dots \\ -A_k \\ A_1 \\ -A_{k+2} \\ \dots \\ -A_n \end{bmatrix} = (-1)^{n-k-1} \det[M - I_n]_{k+1,n} = t^k \det[M - I_n]_{1,1}$$

Por tanto, $\det(\Lambda(t) - I_{n-1}) = (1+t+\dots+t^{n-1}) \det[M - I_n]_{1,1}$. □

Corolario 3.3.1. *Se verifica la siguiente igualdad:*

$$\det[S^{-1}(M - I_n)S]_{n,n} = \det(\Lambda(t) - I_{n-1}) = (1+t+\dots+t^{n-1}) \det[M - I_n]_{1,1}$$

Ya estamos en condiciones de demostrar el teorema 3.3.1:

Demostración. Por el teorema 3.2.3 sabemos que es suficiente con probar las siguientes igualdades, donde $\gamma, \beta \in B_n$:

- Movimiento elemental M1:
 $\det[\phi_n(+\sigma_i - \sigma_i) - I_n]_{1,1} = \det[\phi_n(-\sigma_i + \sigma_i) - I_n]_{1,1}$, siendo $i < n$
- Movimiento elemental M2:
 $\det[\phi_n(\sigma_i \sigma_{i+1} \sigma_i) - I_n]_{1,1} = \det[\phi_n(\sigma_{i+1} \sigma_i \sigma_{i+1}) - I_n]_{1,1}$, siendo $i + 1 < n$.
- Movimiento elemental M3:
 $\det[\phi_n(\sigma_i \sigma_j) - I_n]_{1,1} = \det[\phi_n(\sigma_j \sigma_i) - I_n]_{1,1}$, siendo $i < n, |i - j| > 1$
- Verificando conjugación Mv1: $\det[\phi_n(\gamma \beta \gamma^{-1}) - I_n]_{1,1} = \det[\phi_n(\beta) - I_n]_{1,1}$
- Verificando estabilización Mv2: $\det[\phi_{n+1}(\beta \sigma_n) - I_{n+1}]_{1,1} = \det[\phi_n(\beta) - I_n]_{1,1}$

Para demostrar las igualdades referentes a los movimientos elementales basta con probar las siguientes igualdades:

- Para el movimiento elemental M1:
 $\phi_n(+\sigma_i - \sigma_i) = \phi_n(-\sigma_i + \sigma_i)$, siendo $i < n$.
 Esta igualdad es clara pues por definición $-\sigma_i$ es la matriz inversa de σ_i luego $\phi_n(+\sigma_i - \sigma_i) = \phi_n(+\sigma_i)\phi_n(-\sigma_i) = \phi_n(-\sigma_i)\phi_n(+\sigma_i) = \phi_n(-\sigma_i + \sigma_i)$
- Para el movimiento elemental M2:
 $\phi_n(\sigma_i \sigma_{i+1} \sigma_i) = \phi_n(\sigma_{i+1} \sigma_i \sigma_{i+1})$, siendo $i + 1 < n$.
 Sin pérdida de generalidad podemos suponer $i = 1, n = 3$ de modo que:

$$\begin{aligned} \phi_3(\sigma_1 \sigma_2 \sigma_1) &= \begin{bmatrix} 1-t & t & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1-t & t \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1-t & t & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \\ &= \begin{bmatrix} 1-t & t & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1-t & t & 0 \\ 1-t & 0 & t \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} (1-t)^2 + (1-t)t = 1-t & (1-t)t & t^2 \\ 1-t & t & 0 \\ 1 & 0 & 0 \end{bmatrix} = \\ &= \begin{bmatrix} 1-t & (1-t)t & t^2 \\ 1-t & t & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1-t & t \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1-t & (1-t)t & t^2 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1-t & t \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1-t & t & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1-t & t \\ 0 & 1 & 0 \end{bmatrix} = \phi_3(\sigma_2 \sigma_1 \sigma_2) \end{aligned}$$

- Para el movimiento elemental M3:
 $\phi_n(\sigma_i \sigma_j) = \phi_n(\sigma_j \sigma_i)$, siendo $i < n, |i - j| > 1$
 Sin pérdida de generalidad podemos suponer $i + 1 < j$ de modo que:

$$\phi_n(\sigma_i) \phi_n(\sigma_j) = \begin{bmatrix} I_{i-1} & & & & \\ & 1-t & t & & \\ & & 1 & 0 & \\ & & & I_{j-i-2} & \\ & & & & 1-t & t \\ & & & & & 1 & 0 \\ & & & & & & I_{n-j-1} \end{bmatrix} = \phi_n(\sigma_j) \phi_n(\sigma_i)$$

Veamos ahora que se verifican las igualdades referentes a los movimientos de Markov.

- Veamos que se verifica la igualdad para el primer movimiento de Markov, es decir, veamos que se verifica $\det[\phi_n(\gamma\beta\gamma^{-1}) - I_n]_{1,1} = \det[\phi_n(\beta) - I_n]_{1,1}$:

Consideramos las matrices S y S^{-1} que definimos anteriormente y definimos los productos de matrices

$$S^{-1}\phi_n(\gamma)S = \left[\begin{array}{c|c} \Lambda(\gamma) & 0 \\ \hline *...* & 1 \end{array} \right]; S^{-1}\phi_n(\beta)S = \left[\begin{array}{c|c} \Lambda(\beta) & 0 \\ \hline *...* & 1 \end{array} \right]; S^{-1}\phi_n(\gamma^{-1})S = \left[\begin{array}{c|c} \Lambda(\gamma)^{-1} & 0 \\ \hline *...* & 1 \end{array} \right]$$

De modo que

$$S^{-1}\phi_n(\gamma\beta\gamma^{-1})S = \left[\begin{array}{c|c} \Lambda(\gamma)\Lambda(\beta)\Lambda(\gamma)^{-1} & 0 \\ \hline *.....* & 1 \end{array} \right]$$

Por el corolario 3.3.1 sabemos que

$$(1 + t + \dots + t^{n-1})\det[\phi_n(\gamma\beta\gamma^{-1}) - I_n]_{1,1} = \det[S^{-1}(\phi_n(\gamma\beta\gamma^{-1}))S - I_n]_{n,n}$$

Desarrollando este segundo término tenemos:

$$\begin{aligned} \det[S^{-1}(\phi_n(\gamma\beta\gamma^{-1}))S - I_n]_{n,n} &= \det[\Lambda(\gamma)\Lambda(\beta)\Lambda(\gamma)^{-1} - I_{n-1}] = \\ &\det[\Lambda(\gamma)(\Lambda(\beta) - I_{n-1})\Lambda(\gamma)^{-1}] = \det[\Lambda(\beta) - I_{n-1}] \end{aligned}$$

Aplicando de nuevo el corolario 3.3.1 tenemos la siguiente igualdad

$$\det[\Lambda(\beta) - I_{n-1}] = (1 + t + \dots + t^{n-1})\det[\phi_n(\beta) - I_n]_{1,1}$$

Luego obtenemos la igualdad:

$$(1 + t + \dots + t^{n-1})\det[\phi_n(\gamma\beta\gamma^{-1}) - I_n]_{1,1} = (1 + t + \dots + t^{n-1})\det[\phi_n(\beta) - I_n]_{1,1}$$

y podemos concluir $\det[\phi_n(\gamma\beta\gamma^{-1}) - I_n]_{1,1} = \det[\phi_n(\beta) - I_n]_{1,1}$

- Por último vamos a ver que se verifica la igualdad para el segundo movimiento de Markov, es decir, veamos que se verifica $\det[\phi_{n+1}(\beta\sigma_n) - I_{n+1}]_{1,1} = \det[\phi_n(\beta) - I_n]_{1,1}$.

Llamemos $M = \phi_n(\beta)$ de modo que

$$\phi_{n+1}(\beta) = \left[\begin{array}{c|c} M & 0 \\ \hline 0 & 1 \end{array} \right]$$

Por otra parte sabemos que

$$\phi_{n+1}(\sigma_n) = \left[\begin{array}{ccc} I_{n-1} & & \\ & 1-t & t \\ & 1 & 0 \end{array} \right]$$

Por tanto,

$$\phi_{n+1}(\beta\sigma_n) = \begin{bmatrix} & a_{1,n}(1-t) & a_{1,n}t \\ M_{n-1,n-1} & \cdots & \cdots \\ a_{n,1} & \cdots & a_{n,n-1} & a_{n,n}(1-t) & a_{n,n}t \\ & a_{n,n-1} & a_{n,n}(1-t) & 1 & 0 \end{bmatrix}$$

De este modo se tiene

$$\begin{aligned} \det[\phi_{n+1}(\beta\sigma_n) - I_{n+1}]_{n,n} &= \det \left[\begin{array}{c|c} M_{n-1,n-1} - I_{n-1} & \\ \hline & -1 \end{array} \right] \\ &= -\det[M_{n-1,n-1} - I_{n-1}] = -\det[\phi_n(\beta) - I_n]_{n,n} \end{aligned}$$

Obteniendo la igualdad

$$\det[\phi_{n+1}(\beta\sigma_n) - I_{n+1}]_{1,1} = \det[\phi_n(\beta) - I_n]_{1,1}$$

□

De este modo, el polinomio de Alexander de la trenza $\beta 1 = \sigma 1$ se obtendría del siguiente modo:

Por definición tenemos que

$$\phi_2(\beta 1) = \begin{bmatrix} 1-t & t \\ 1 & 0 \end{bmatrix}$$

luego

$$\phi_2(\beta 1) - I_2 = \begin{bmatrix} -t & t \\ 1 & -1 \end{bmatrix}$$

Finalmente tenemos

$$\det(\phi_2(\beta 1) - I_2)_{1,1} = \det([-1]) = -1.$$

Veamos ahora el polinomio de Alexander de una trenza más compleja, en concreto de la trenza $\beta 2 = \sigma 2 \sigma 3^{-1}$. Ya sabemos que

$$\phi_4(\beta 2) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1-t & 0 & t \\ 0 & 1 & 0 & 0 \\ 0 & 0 & t^{-1} & 1-t^{-1} \end{bmatrix}$$

luego

$$\phi_4(\beta 2) - I_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -t & 0 & t \\ 0 & 1 & -1 & 0 \\ 0 & 0 & t^{-1} & t^{-1} \end{bmatrix}.$$

Finalmente tenemos

$$\det(\phi_4(\beta_2) - I_n)_{1,1} = \det\begin{bmatrix} -t & 0 & t \\ 1 & -1 & 0 \\ 0 & t^{-1} & t^{-1} \end{bmatrix} = -1 + 1 = 0.$$

Por tanto tenemos que las trenzas β_1 y β_2 no pueden generar nudos equivalentes pues sus polinomios de Alexander son distintos.

3.4 El problema de las palabras.

En esta sección vamos a tratar de ver si dos trenzas dadas son equivalentes entre sí sin hacer uso de invariantes [Dehornoy, 1997], [D. Garber, 2008]. Vamos a apoyarnos en la idea que vimos en la sección 3.2:

Consideramos el grupo B_n con la representación que vimos en el teorema 3.2.2. El problema de ver si dos trenzas dadas son equivalentes se puede ver como el problema de las palabras del grupo B_n .

Problema de las palabras del grupo de las trenzas:

Dadas dos palabras de trenzas $\beta_1, \beta_2 \in B_n$ tratamos de encontrar algún método que nos permita confirmar si son o no equivalentes.

Ver si dos palabras (de trenzas) dadas $\beta_1, \beta_2 \in B_n$ son equivalentes se puede ver cómo el problema de ver si la palabra $\beta_1\beta_2^{-1}$ es equivalente a la palabra vacía (trenza trivial). Por tanto el problema de las palabras se puede reducir a distinguir si una palabra es equivalente o no a la palabra vacía.

En este proyecto vamos a ver el método de Patrick Dehornoy, [Dehornoy, 1997], que nos permite reducir las palabras de trenzas dadas hasta una forma específica que nos permitirá ver si la palabra es igual a la cadena vacía. Para verlo con más detalle necesitamos ver algunas ideas previas.

Definición:

Diremos que una palabra es **libremente reducida** si no contiene secuencias de la forma $\sigma_i^{-1}\sigma_i$ ni $\sigma_i\sigma_i^{-1}$.

Por ejemplo, la palabra $\sigma_2\sigma_1^{-1}\sigma_1\sigma_3^{-1}$ que representa a la trenza de la figura 3.28 no es libremente reducida pues contiene la palabra $\sigma_1^{-1}\sigma_1$.

Definición:

Sea la palabra $\beta \in B_n$. Llamaremos **generador principal** de β al generador de β de menor índice.



Figura 3.28: Trenza no libremente reducida.

Por ejemplo, la palabra $\sigma_2\sigma_1^{-1}\sigma_1\sigma_3^{-1}$ tiene como generador principal el 1, mientras que la palabra $\sigma_2\sigma_4^{-1}$ tiene como generador principal el número 2.

Definición: Diremos que una palabra $\beta \in B_n$ es **reducida** si se cumple alguna de estas condiciones:

1. β es la palabra cadena vacía.
2. El generador principal de β se presenta sólo positiva o negativamente.

Por ejemplo, la palabra $\sigma_2\sigma_1^{-1}\sigma_3^{-1}\sigma_1$ no es reducida pues el generador principal (1) se encuentra en un cruce negativo y en un cruce positivo. La palabra $\sigma_2\sigma_1\sigma_3^{-1}\sigma_1$ si sería reducida pues el generador principal (1) se presenta sólo como cruces positivos.

Si tenemos una palabra no reducida con generador principal σ_i , podemos considerar ocurrencias de la palabra de la forma $\sigma_i^\pm \dots \sigma_i^\mp$ que no contengan cruces σ_i , es decir, entre los cruces de signo opuesto del generador σ_i sólo encontramos generadores σ_k , $k > i$. Podemos visualizar la idea en la figura 3.29. En este caso diremos que la cadena $i+1$ forma a **handle**.

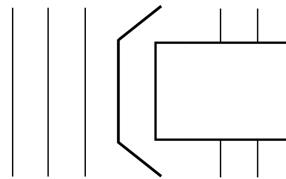


Figura 3.29: Main handle.

Proposición 3.4.1. *Una palabra reducida no vacía no puede ser equivalente a la cadena vacía.*

Por tanto, si encontramos un método para obtener la palabra reducida de una palabra, podremos resolver el problema de las palabras:

Si la palabra β_1 reducida es equivalente a la palabra β_2 , entonces β_2 es equivalente a la cadena vacía si y sólo si β_1 es la cadena vacía.

Proposición 3.4.2. *Cualquier palabra admite una palabra reducida.*

Para ver una demostración de esta proposición, vamos ir viendo el método que realizaremos para transformar una palabra cualquiera en su palabra reducida.

Con este método tratamos de eliminar los handles de la trenza, pero para no entrar en bucles infinitos en el algoritmo, vamos a tener que considerar handles generados por cualquier generador y no sólo por el generador principal.

Definición:

Un σ_j -handle de una palabra es una sub-palabra de la forma $\sigma_j^\pm v \sigma_j^\mp$ donde la sub-palabra v sólo contiene generadores $\sigma_k^{(-1)}$ con $k < j-1$ o $k > j$. Si el generador σ_j es el generador principal de la palabra, a dicho σ_j -handle se conocerá como **main handle**.

Podemos ver un σ_j -handle en la figura 3.30.

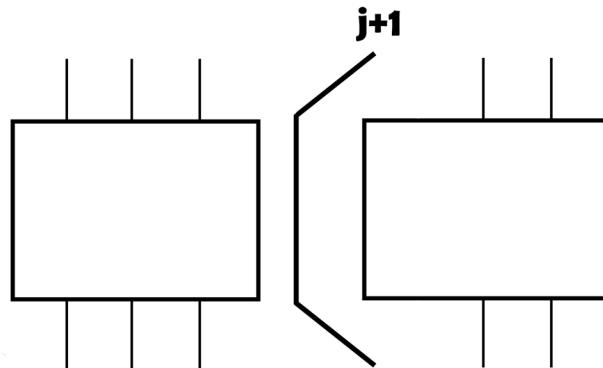


Figura 3.30: A handle.

Supongamos que tenemos la palabra $\sigma 3 \sigma 1^{-1} \sigma 2 \sigma 4^{-1} \sigma 2^{-1} \sigma 4^{-1} \sigma 1$.

La sub-palabra $\sigma 1^{-1} \sigma 2 \sigma 4^{-1} \sigma 2^{-1} \sigma 4^{-1} \sigma 1$ es un main handle mientras que $\sigma 2 \sigma 4^{-1} \sigma 2^{-1}$ es un $\sigma 2$ -handle.

Definición:

Sea $\sigma_j^e v \sigma_j^{-e}$ un σ_j -handle de una palabra β , donde $e \in \{-1, 1\}$. En particular, podemos denotar dicho σ_j -handle como la siguiente sub-palabra:

$$\sigma_j^e v_0 \sigma_{j+1}^{d_1} v_1 \dots v_{m-1} \sigma_{j+1}^{d_m} v_m \sigma_j^{-e}$$

donde v_0, \dots, v_m no contienen generadores $\sigma_k^{(-1)}$ con $j-1 \leq k \leq j+1$, $d_i \in \{-1, 1\}$

Podemos aplicarle una **reducción local** quedando la sub-palabra equivalente:

$$v_0 \sigma_{j+1}^{-e} \sigma_j^{d_1} \sigma_{j+1}^e v_1 \dots v_{m-1} \sigma_{j+1}^{-e} \sigma_j^{d_m} \sigma_{j+1}^e v_m$$

En definitiva hemos aplicado el homomorfismo $\phi_{j,e}$ definido como:

$$\begin{aligned} \sigma_j^{\pm 1} &\rightarrow \epsilon \\ \sigma_{j+1}^{\pm 1} &\rightarrow \sigma_{j+1}^{-e} \sigma_j^{\pm 1} \sigma_{j+1}^e \\ \sigma_k^{\pm 1} &\rightarrow \sigma_k^{\pm 1}, \text{ siendo } k \neq j, j+1. \end{aligned}$$

Podemos ver esta transformación en la figura 3.31.

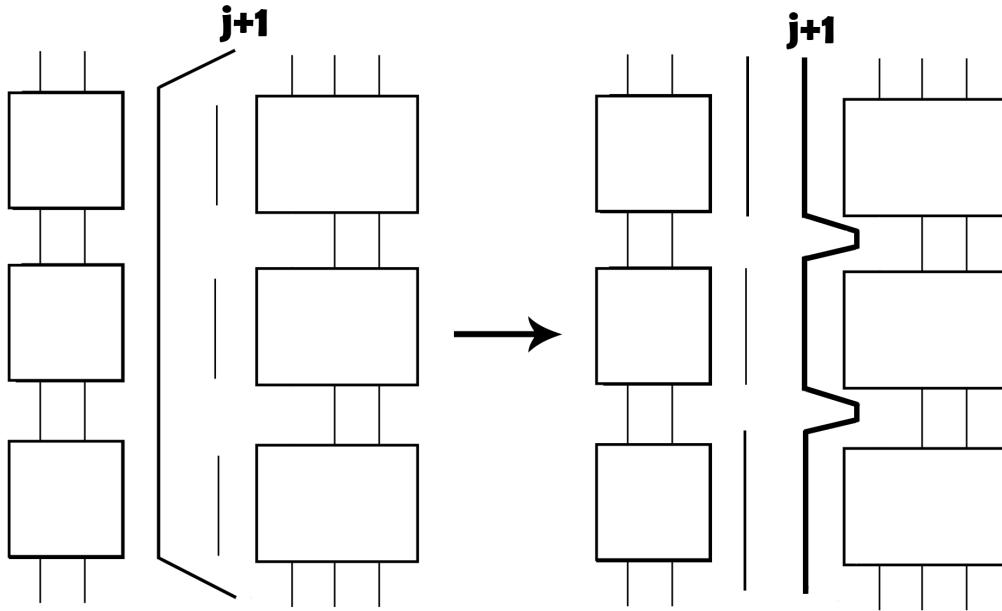


Figura 3.31: Reducción local.

Es importante destacar que antes de realizar una reducción local a una palabra, tendremos que asegurarnos de que dicha palabra sea libremente reducida para evitar una mayor complejidad.

Sea la palabra $\sigma_2\sigma_1^{-1}\sigma_2\sigma_3\sigma_1$. Podemos aplicar una reducción local al main handle $\sigma_1^{-1}\sigma_2\sigma_3\sigma_1$ quedando la palabra $\sigma_2\sigma_1\sigma_2^{-1}\sigma_3$. Esta sería su palabra reducida.

Pero antes de hacer una reducción local a un σ_j -handle, tendremos que asegurarnos que no tenemos ningún σ_{j+1} -handle en el σ_j -handle. En el caso de tenerlo, tendríamos que aplicar la reducción local a este σ_{j+1} -handle y posteriormente aplicar la reducción local al σ_j -handle. De este modo podremos evitar ciclos infinitos. Veámoslo con un ejemplo:

Supongamos que queremos obtener la palabra reducida de la palabra $\sigma_1\sigma_2\sigma_3\sigma_2^{-1}\sigma_1^{-1}$. El generador principal es 1 y tenemos un main handle (que, en este caso, es toda la palabra en sí misma). Aplicamos una reducción local a este main handle quedando:

$\sigma_2^{-1}\sigma_1\sigma_2\sigma_3\sigma_2^{-1}\sigma_1^{-1}\sigma_2$. Vemos que volvemos a tener un main handle y entraremos en bucle infinito. Veamos cómo aplicaríamos la solución que hemos comentado:

Supongamos de nuevo que queremos obtener la palabra reducida de la palabra $\sigma_1\sigma_2\sigma_3\sigma_2^{-1}\sigma_1^{-1}$. Tenemos un main handle pero dicho main handle consta de una sub-palabra σ_2 -handle. Aplicamos una reducción local a este σ_2 -handle quedando:

$\sigma_1\sigma_3^{-1}\sigma_2\sigma_3\sigma_1^{-1}$. Ahora sí podemos aplicar una reducción local al main handle quedando la palabra reducida:

$\sigma_3^{-1}\sigma_2^{-1}\sigma_1\sigma_2\sigma_3$.

Definición:

Diremos que un σ_j -handle está permitido si no contiene ningún σ_{j+1} -handle.

Por ejemplo el main handle $\sigma_1\sigma_2\sigma_3\sigma_2^{-1}\sigma_1^{-1}$ no está permitido porque incluye al σ_2 -handle $\sigma_2\sigma_3\sigma_2^{-1}$. Este 2-handle sí estaría permitido porque no incluye ningún σ_3 -handle.

Si un σ_j -handle no está permitido, le aplicaremos reducciones locales, como hemos hecho en el ejemplo, hasta que pase a estar permitido.

Definición:

Diremos que una palabra β' es deducida de una palabra β usando una **reducción de handle** si β' se obtiene a partir de una reducción local de un σ_j -handle permitido de β .

Teorema 3.4.1. *Cualquier palabra puede ser reducida por una secuencia finita de reducciones de handle hasta llegar a su palabra reducida.*

Para poner en uso todas estas ideas vamos a ver un ejemplo en el que demostraremos que dos palabras dadas son equivalentes:

Supongamos que tenemos las palabras $\beta_1 = \sigma_2\sigma_3^{-1}\sigma_1\sigma_2^{-1}\sigma_3$ y $\beta_2 = \sigma_1\sigma_2\sigma_3^{-1}\sigma_1\sigma_2^{-1}$ que representan a las trenzas de la figura 3.32.

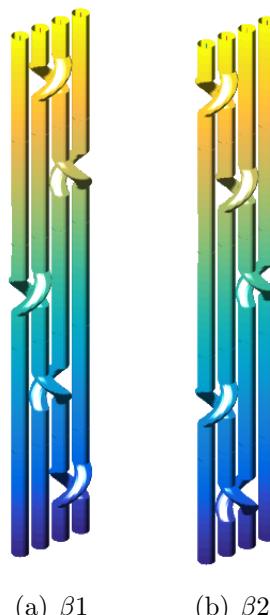


Figura 3.32: Trenzas equivalentes.

A simple vista es difícil saber si las palabras son o no equivalentes. Para ver que las palabras son equivalentes ($\beta_1 \sim \beta_2$), tenemos que ver que se verifica $\beta_1\beta_2^{-1} \sim \epsilon$, donde ϵ representa a la cadena vacía.

En primer lugar vamos a considerar la palabra con la que vamos a trabajar:
 $\beta_1\beta_2^{-1} = \sigma_2\sigma_3^{-1}\sigma_1\sigma_2^{-1}\sigma_3\sigma_2\sigma_1^{-1}\sigma_3\sigma_2^{-1}\sigma_1^{-1}$.

Nos encontramos el main handle $\sigma_1\sigma_2^{-1}\sigma_3\sigma_2\sigma_1^{-1}$. Vamos a aplicarle una reducción local, pero vemos que contiene al σ_2 -handle $\sigma_2^{-1}\sigma_3\sigma_2$. Por tanto aplicamos una reducción local a este σ_2 -handle:

$$\sigma_2^{-1}\sigma_3\sigma_2 \rightarrow \sigma_3\sigma_2\sigma_3^{-1}, \text{ obteniendo la palabra:}$$

$$\beta_1\beta_2^{-1} = \sigma_2\sigma_3^{-1}\sigma_1\sigma_3\sigma_2\sigma_3^{-1}\sigma_1^{-1}\sigma_3\sigma_2^{-1}\sigma_1^{-1}.$$

Nos encontramos el main handle permitido $\sigma_1\sigma_3\sigma_2\sigma_3^{-1}\sigma_1^{-1}$. Le aplicamos una reducción local:

$$\sigma_1\sigma_3\sigma_2\sigma_3^{-1}\sigma_1^{-1} \rightarrow \sigma_3\sigma_2^{-1}\sigma_1\sigma_2\sigma_3^{-1}, \text{ obteniendo la palabra:}$$

$$\beta_1\beta_2^{-1} = \sigma_2\sigma_3^{-1}\sigma_3\sigma_2^{-1}\sigma_1\sigma_2\sigma_3^{-1}\sigma_3\sigma_2^{-1}\sigma_1^{-1}.$$

Esta palabra no es libremente reducida porque nos encontramos un par de veces la sub-palabra $\sigma_3^{-1}\sigma_3$. Las eliminamos y obtenemos la palabra:

$$\beta_1\beta_2^{-1} = \sigma_2\sigma_2^{-1}\sigma_1\sigma_2\sigma_2^{-1}\sigma_1^{-1}.$$

De nuevo nos encontramos con una palabra que no es libremente reducida porque nos encontramos un par de veces la sub-palabra $\sigma_2\sigma_2^{-1}$. Las eliminamos y obtenemos la palabra:

$$\beta_1\beta_2^{-1} = \sigma_1\sigma_1^{-1}.$$

Otra vez nos encontramos con una palabra no libremente reducida. Eliminamos la sub-palabra $\sigma_1\sigma_1^{-1}$ quedando la cadena vacía, es decir:

$$\beta_1\beta_2^{-1} = \epsilon.$$

Concluimos que ambas palabras son equivalentes, luego las trenzas a las que representan son equivalentes. Sus trenzas cerradas serán equivalentes y por tanto los nudos a los que se está representando son también equivalentes.

3.5 Conclusiones

Para estudiar si dos nudos dados son o no equivalentes, tratamos de estudiar si las palabras que representan a las trenzas cerradas que generan dichos nudos son equivalentes.

El método que acabamos de ver (al que nos referiremos como método de Dehornoy) nos permite confirmar si dos palabras dadas representan a una misma trenza, y por tanto los nudos que generan sus trenzas cerradas serán equivalentes.

El problema es que podemos tener dos palabras de trenzas que no sean equivalentes, pero puede ser que sus trenzas cerradas sí que sean equivalentes dando lugar a nudos equivalentes. En este caso, podremos estudiar el polinomio de Alexander para comprobar si las dos palabras dadas generan trenzas cerradas que no son equivalentes o que sí podrían serlo (que no quiere decir que lo sean).

Por tanto, nos queda una serie de palabras que generan trenzas cerradas que no sabremos si son equivalentes entre sí. Se trata de un problema que sigue abierto.

El uso de invariantes como el exponente o la permutación de una palabra nos van a permitir determinar si dos palabras dadas representan a trenzas que no son equivalentes de una forma más rápida que usando método de Dehornoy, pero con el uso de estos invariantes no podremos confirmar que dos palabras sean equivalentes.

Capítulo 4

Desarrollo informático.

El objeto principal de estudio de este proyecto es la teoría de nudos. Sin embargo, hemos visto que esta teoría está estrechamente relacionada con la teoría de trenzas.

En este capítulo realizaremos el desarrollo informático sobre teoría de trenzas y trabajaremos con trenzas cerradas, que se pueden ver como si fuesen nudos. El motivo que nos ha llevado a trabajar con teoría de trenzas en lugar de con teoría de nudos se puede resumir en el hecho de que la teoría de trenzas proporciona ciertos atributos con los que podemos desarrollar la teoría de forma más cómoda que en la teoría de nudos.

Antes de comenzar a realizar dicho desarrollo hemos investigado y hemos encontrado varios software que permiten trabajar computacionalmente con trenzas y nudos. Algunos de lo más destacados son los siguientes:

- braidlab [Thiffeault and Budisic, 2016]: se trata de un paquete Matlab que permite analizar datos usando trenzas.
- knotilus [Flint and Rankin, 2003]: nos permite trabajar con nudos, visualizándolos y obteniendo sus notaciones de Dowker y Gauss entre otras funcionalidades.
- braid program [Fenn, 2016]: nos permite trabajar con trenzas. Lo más interesante de este programa es que nos permite obtener, a partir del algoritmo de Vogel, la trenza que genera cierto nudo, aunque no nos permite su visualización.

Hemos implementado nuestro propio programa libre y gratuito porque estos programas ya existentes no proporcionan unas visualizaciones agradables para el usuario además de que la documentación de algunos de ellos no está muy completa y no resultan intuitivos de usar. Además, la implementación que hemos realizado ha ido pasando por distintas versiones hasta conseguir un diseño lo más simple posible, de modo que es fácil añadir nuevas funcionalidades e incluso mejoras.

Un aspecto importante a destacar es el cambio de notación que vamos a hacer sobre las trenzas para que al usuario le resulte mucho más cómodo trabajar:

- A los cruces negativos de una trenza σ_i^{-1} les denotaremos como $-\sigma_i$.
- A los cruces positivos de una trenza σ_i^1 , o simplemente σ_i , les denotaremos como $+\sigma_i$

Por ejemplo, la trenza $\beta = \sigma_4^{-1}\sigma_1^{-1}\sigma_2$ la denotaremos como $\beta = -\sigma_4 - \sigma_1 + \sigma_2$.

4.1 Teoría de trenzas con Matlab.

Hemos implementado bajo Matlab R2015a todos los aspectos matemáticos que desarrollamos en el capítulo 3. Hemos creado métodos básicos para poder trabajar correctamente con el grupo de las trenzas. Además, hemos implementado un algoritmo muy básico para ver si una trenza dada es pura o no: diremos que una trenza es pura si su permutación se corresponde con la permutación identidad. También cabe comentar que sobre trenzas cerradas hemos implementado la notación de Dowker que vimos en el capítulo 2 sobre teoría de nudos.

Para recopilar todo este conjunto de herramientas que nos permiten trabajar con trenzas hemos creado **toxtren**. Se trata de un toolbox, una librería de funciones Matlab, asociado al estudio de trenzas y trenzas cerradas [Matlab, 2016a].

Para hacer uso de toxtren, el usuario simplemente tiene que disponer de Matlab e instalar toxtren.mltbx: seleccionamos como nuevo directorio de trabajo la carpeta que contiene el archivo .mltbx, hacemos click derecho sobre él e instalamos.

Para realizar el control de versiones de toxtren hemos utilizado git.

A la hora de implementar toxtren hemos seguido un diseño orientado a objetos ya que las trenzas se pueden representar de una forma cómoda haciendo uso de objetos y clases. En concreto, hemos construido dos clases:

- Clase trenza: mediante esta clase podremos crear cualquier trenza, visualizarla en 3D, obtener los invariantes que hemos explicado en el capítulo 3, estudiar su trivialidad, realizar operaciones básicas con distintas trenzas....
- Clase trenza_cerrada: la clase trenza_cerrada hereda de la clase trenza pues una trenza cerrada es una trenza en la que conectamos los extremos de las cadenas. Al hacer esta unión sobre los extremos de las cadenas podemos obtener varios enlaces: si únicamente tenemos un enlace, dicha trenza cerrada será equivalente a un nudo.

En dicha clase heredamos los métodos que hemos comentado para la clase trenza y además estudiamos la notación Dowker de la trenza cerrada, el polinomio de Alexander, podemos realizar los movimientos de Markov...

Podemos ver un esquema de la herencia de las clases con sus atributos y métodos en la figura 4.1.

Para generar la documentación de Toxtren [Matlab, 2016b], hemos creado ficheros de ayuda HTML a partir de distintos script Matlab que contienen la documentación para ambas clases y ejemplos de uso para una mejor comprensión. Además, hemos creado los ficheros info.xml y helptoc.xml para identificar los ficheros HTML en la creación del toolbox y mostrar el panel de contenido en el buscador de ayuda.

De este modo, podemos obtener la documentación de toxtren tras su instalación seleccionando, en el buscador de ayuda, Toxtren Toolbox, que se encontrará en la sección Supplemental Software. También podemos hacer uso de help en la ventana de comandos.

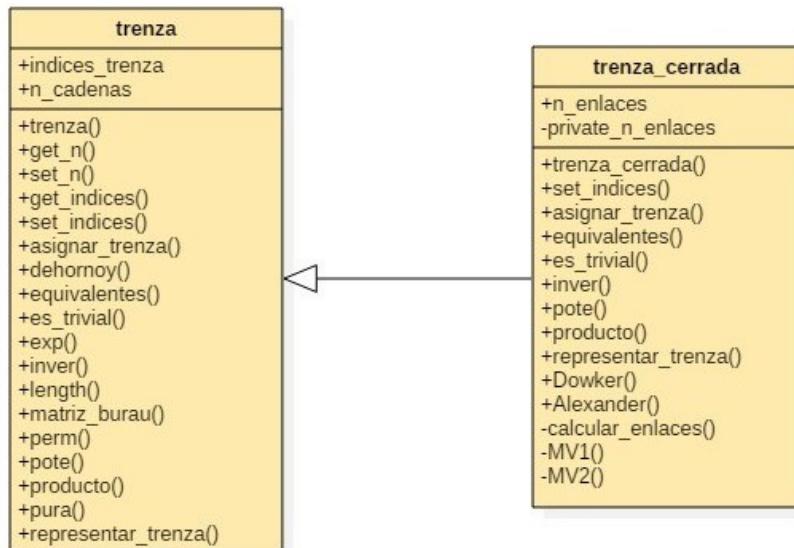


Figura 4.1: Estructura clases

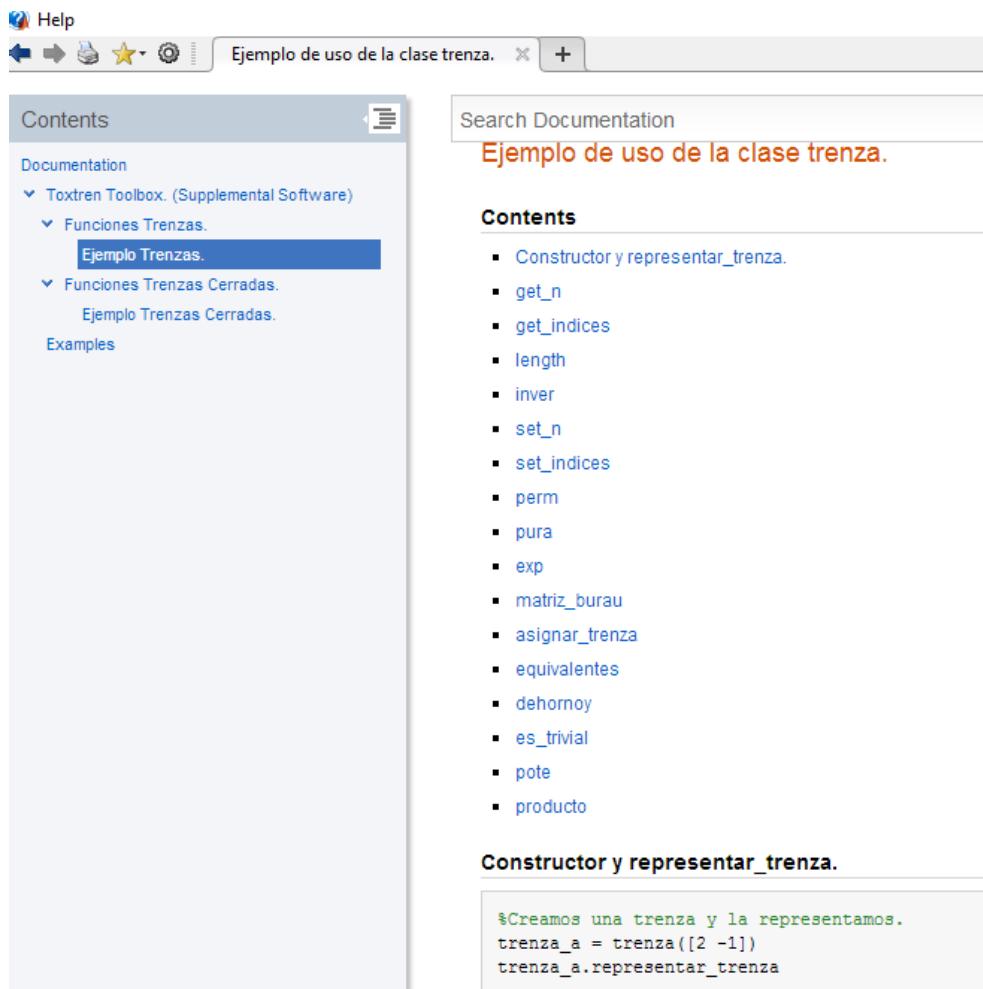


Figura 4.2: Documentación Toxtren.

4.2 Pseudoalgoritmos.

En esta sección vamos a ver algunos pseudoalgoritmos referentes a los algoritmos que hemos implementado. Nos vamos a centrar en aquellos que tienen mayor complejidad e interés, sin entrar en gran detalle. Es importante destacar que estos algoritmos no están establecidos como tales, sino que los hemos implementando intentando reflejar los conceptos matemáticos que hemos ido viendo.

En concreto, vamos a ver los pseudoalgoritmos para el algoritmo de Dehornoy (que descomponemos en varios algoritmos auxiliares), el algoritmo que nos permite comprobar si dos trenzas dadas (o sus cierres) son o no equivalentes entre sí y el algoritmo que nos permite comprobar si una trenza dada (o su cierre) es equivalente a la trenza trivial.

Algoritmo Dehornoy para trenzas.

Vamos a ir viendo los algoritmos auxiliares y concluiremos con el propio algoritmo de Dehornoy. Recordemos que este algoritmo se explica con detalle en la sección 3.4.

Creamos el método Simplifica mediante el cuál eliminamos ocurrencias de tipo $\sigma_i^e \sigma_i^{-e}$, siendo $e \in \{-1,1\}$, de una palabra que representa a cierta trenza.

Algoritmo 4.2.1. Algoritmo Simplifica(*índices_braid*)

ENTRADA: *índices_braid* (*cadena de enteros que representa los cruces de una trenza*)

SALIDA: *braid_aux* (*cadena auxiliar para mejor representación visual*)

nueva_braid (*cadena de enteros que representa los cruces de la trenza tras eliminar dos cruces consecutivos opuestos*)

encontrado (*bool para indicar si se produce simplificación*)

- 1 *Recorro los cruces de índices_braid*
- 2 *Si hay dos cruces seguidos con signos opuestos, creo una copia de índices_braid y elimino dichos cruces.*

Haciendo uso del siguiente algoritmo podremos encontrar las posiciones que delimitan un σ_{minimo} -handle.

Algoritmo 4.2.2. Algoritmo encuentra_handle(*índices_braid*, *mínimo*)

ENTRADA: *índices_braid* (*cadena de enteros que representa los cruces de una trenza*)
mínimo (*generador de handle a encontrar*)

SALIDA: *pos1* (*posición inicial del handle encontrado*)

pos2 (*posición final del handle encontrado*)

- 1 *Recorro los cruces de índices_braid*
- 2 *Si hay un cruce generado por el elemento mínimo, me quedo con esa posición como pos1 y su signo. Si no finalizo.*
- 3 *Si encuentro un cruce generado por el elemento mínimo con signo opuesto, me quedo con esa posición como pos2.*

Haciendo uso del algoritmo `reduccion_base` aplicamos una reducción local a la trenza representada por `indices_braid` sobre el $\sigma_{mínimo}$ -handle que está situado entre las posiciones `pos1` y `pos2`. Es importante destacar el hecho de que este $\sigma_{mínimo}$ -handle no contiene $\sigma_{mínimo+1}$ -handle. Este detalle lo controlamos en el algoritmo Dehornoy que veremos posteriormente.

Algoritmo 4.2.3. *Algoritmo reduccion_base*(índices_braid, mínimo, pos1, pos2)

ENTRADA: `índices_braid` (cadena de enteros que representa los cruces de una trenza)

`mínimo` (generador de handle)

`pos1` (posición inicial del handle)

`pos2` (posición final del handle)

SALIDA: `braid_aux2` (cadena auxiliar para mejor representación visual)

`nuevo` (cadena de enteros que representa los cruces de la trenza tras aplicar la reducción local al $\sigma_{mínimo}$ -handle entre `pos1` y `pos2`)

`simplificado2` (bool auxiliar para mejor representación visual)

- 1 Creo vector_auxiliar
- 2 Recorro los cruces de `índices_braid` desde `pos1` hasta `pos2`
- 3 Si hay un cruce generado por el elemento (`mínimo+1`) añado a `vector_auxiliar` los 3 cruces correspondientes del algoritmo. Si no, añado el mismo cruce.
- 4 Creo vector_nuevo con los elementos desde inicio de `índices_braid` hasta `pos1`, `vector_auxiliar`, y los elementos desde `pos2` hasta final de `índices_braid`.
- 5 Si `índices_braid` y `vector_auxiliar` tienen distinto tamaño, asigno a `braid_aux2` una cadena con ciertos ceros para mejor visualización `n`.

Estos algoritmos auxiliares no se proporcionan para uso directo al usuario ya que el realmente interesante es el algoritmo de Dehornoy, pero sí podrían ser usados. Veamos entonces el algoritmo de Dehornoy.

Algoritmo 4.2.4. *Algoritmo dehornoy*(br, N_cortes, Radio, representar)

ENTRADA: `br` (trenza)

`N_cortes` (número de cortes de las cadenas de la trenza)

`Radio` (radio de las cadenas de la trenza)

`representar` (bool para representar las equivalencias de la trenza)

SALIDA: `es_trivial` (bool que nos indica si la trenza dada es o no trivial)

`trenza_final` (cadena de enteros que representa a la trenza reducida equivalente a `br`)

- 1 Si `número_argumentos=1` -> `N_cortes=20`, `Radio=0.5`, `representar=1`.
- 2 `índices_braid` = cadena de enteros que representa a la trenza
- 3 Si `br` tiene cadenas a la derecha triviales, las eliminamos visualmente.
- 4 Mientras queden handles en la trenza dada...
- 5 Obtenemos la palabra libremente reducida de `índices_braid`.
- 6 Si no se produce reducción...
- 7 `mínimo` = generador principal de `índices_braid`.

```

8      [pos1, pos2]=encuentra_handle(índices_braid, mínimo)
9      Si pos1 y pos2 son posiciones válidas.....
10     Busco primer subhandle a realizar en el handle.
11     Actualizo pos1 y pos2
12     Aplico reduccion_base a dicho subhandle.
13     Creo matriz con secuencia de palabras generadas en el proceso
14     Si representar, muestro las trenzas usando dicha matriz.

```

Finalmente cabe comentar que el algoritmo de Dehornoy se podrá aplicar sobre trenzas cerradas. El proceso que se realizará internamente será exactamente el mismo que para trenzas, ya que el cerrar la trenza no aporta información nueva para el algoritmo de Dehornoy. Por este mismo motivo, a la hora de realizar la visualización del algoritmo de Dehornoy sobre una trenza cerrada, veremos las transformaciones sobre la trenza y no sobre la trenza cerrada.

Algoritmo de equivalencia para trenzas.

Para ver si dos trenzas dadas son equivalentes o no entre sí, vamos a implementar algunos de los invariantes que vimos para trenzas. En concreto, vamos a estudiar los invariantes exponente y permutación. Si con estos invariantes no conseguimos analizar la equivalencia de las trenzas, vamos a aplicar el algoritmo de Dehornoy que hemos visto anteriormente.

Algoritmo 4.2.5. *Algoritmo equivalentes(br1,br2,explicación)*

ENTRADA: br1 (trenza1)

br2 (trenza2)

explicación (bool para mostrar mensajes explicativos)

SALIDA: equi (bool para indicar si br1 y br2 son o no equivalentes)

```

1  Si número_argumentos=2 -> explicación=0.
2  Obtengo exponente de ambas trenzas.
3  Si son distintos -> No son equivalentes. Finalizo
4  Obtengo permutación de ambas trenzas.
5  Si son distintas -> No son equivalentes. Finalizo
6  Genero trenza_auxiliar = br1inver(br2).
7  Aplico Dehornoy a trenza_auxiliar y obtengo final_braid.
8  Si final_braid no es vacía -> No son equivalentes.
9  Si no -> Si explicacion=1 -> represento secuencia de trenzas de
dehornoy.

```

Algoritmo de equivalencia para trenzas cerradas.

Para analizar la equivalencia de dos trenzas cerradas, vamos a apoyarnos en el algoritmo de equivalencia de las trenzas sin cerrar que hemos visto anteriormente. Además haremos uso de varios algoritmos auxiliares. En concreto, hemos implementado los movimientos de Markov que vimos en la sección 3.2.3.

Sabemos que el movimiento 1 de Markov puede eliminar o añadir cruces. Siempre intentaremos eliminar cruces, mientras que para añadir cruces se tiene que solicitar explícitamente. Lo hacemos así para que las trenzas no aumenten en número de cruces salvo que no quede más opción.

Algoritmo 4.2.6. Algoritmo MV1(*br_c*, *completo*)

ENTRADA: *br_c* (*trenza cerrada*)

completo (*bool* para indicar si se desean añadir cruces)

SALIDA: *a2* (*trenza cerrada* tras aplicar *Movimiento1* eliminando de *Markov*.)

- 1 Si *número_argumentos*=1 -> *completo*=0.
- 2 Si el primer cruce de *br_c* es opuesto al último cruce de *br_c* -> *a2* = *br_c* sin dichos cruces.
- 3 Si no -> Si *completo*...
- 4 Obtengo cruce de mayor índice (*índice m*).
- 5 Añado cruces opuestos de índice *m* a principio y final de *br_c*.

Algoritmo 4.2.7. Algoritmo MV2(*br_c*)

ENTRADA: *br_c* (*trenza cerrada*)

SALIDA: *a3* (*trenza cerrada* tras aplicar *Movimiento2* de *Markov*.)

- 1 Mientras *br_c* cambie de cruces...
- 2 Si *br_c* tiene un solo cruce -> *a3*=*trenza cerrada trivial*.
Finalizo.
- 3 Obtengo cruce de mayor índice (*índice m*).
- 4 Si se encuentra solo una vez en *br_c*
 Si a izquierda o derecha del cruce no tenemos cruces con índice *m-1* -> *a3*=*br_c* sin dicho cruce.

Para implementar este algoritmo de equivalencia entre trenzas cerradas hemos analizado en primer lugar los invariantes básicos de ambas trenzas. Si no obtenemos una respuesta de equivalencia o no equivalencia, pasamos a ver el polinomio de Alexander de ambas trenzas cerradas. Una vez analizados los invariantes que hemos estudiado, vamos a ir realizando transformaciones de equivalencia sobre la trenza cerrada generada por el producto de la trenza inicial y la inversa de la segunda trenza.

Ya sabemos que ir haciendo transformaciones de equivalencia sobre una trenza cerrada para ver si es o no equivalente a la trenza cerrada trivial puede conllevar a una serie indefinida de movimientos.

La idea que hemos llevado a cabo consiste en aplicar el Movimiento 1 de Markov, el algoritmo de Dehornoy a la trenza cerrada que obtenemos, el Movimiento 2 de Markov y de nuevo el algoritmo de Dehornoy. Si en alguna de estas transformaciones conseguimos saber si la trenza cerrada es o no equivalente a la trivial, finalizamos el proceso. En caso contrario, realizaremos el mismo proceso un número delimitado de veces.

Algoritmo 4.2.8. Algoritmo equivalentes(*br1,br2,explicación*)*ENTRADA:* *br1* (*trenza cerrada 1*) *br2* (*trenza cerrada 2*) *explicación* (*bool para mostrar mensajes explicativos*)*SALIDA:* *equi* (*bool para indicar si br1 y br2 son o no equivalentes*)

```

1   Si número_argumentos=2 -> explicación=0.
2   Si el número de enlaces de br1 y de br2 son distintos -> No son
      equivalentes. Finalizo
3   Si equivalentes@trenza(br1,br2) son equivalentes -> sus cierres son
      equivalentes.
4   Si no ->
5       Obengo polinomio de Alexander de ambas trenzas.
6       Si son iguales salvo signo -> No sabemos si son o no
          equivalentes. equi <- 2.
7       Sino -> No son equivalentes.
8   Si equi=2
9       Creo trenza cerrada br = br1inver(br2)
10  Mientras número_iteraciones < límite(establecido a 3)
11      a1 = Aplico MV1 a br.
12      [es_trivial2,a2] = Aplico dehorno a trenza cerrada a1.
13      Si es_trivial2 -> Finalizo.
14      a3 = Aplico MV2 a trenza cerrada a2.
15      [es_trivial4,a4] = Aplico dehorno a trenza cerrada a3.
16      Si es_trivial4 -> Finalizo.
17      Si no ha cambiado br, a4 = Aplico MV1 completo a br.

```

Algoritmo para ver trivialidad de trenza.

Tanto para ver la trivialidad de una trenza como la trivialidad de una trenza cerrada vamos a usar la misma idea que será hacer uso de los algoritmos de equivalencia entre la trenza dada y la trenza trivial. Veámoslos:

Algoritmo 4.2.9. Algoritmo es_trivial(*br,explicación*)*ENTRADA:* *br* (*trenza*) *explicación* (*bool para mostrar mensajes explicativos*)*SALIDA:* *equi* (*bool para indicar si br es o no equivalente a la trenza trivial*)

```

1   Si número_argumentos=1 -> explicación=0.
2   Creo br2 = trenza trivial.
3   Aplico algoritmo equivalentes para br y br2.

```

Algoritmo para ver trivialidad de trenza cerrada.**Algoritmo 4.2.10. Algoritmo es_trivial(*br_c,explicación*)***ENTRADA:* *br_c* (*trenza cerrada*) *explicación* (*bool para mostrar mensajes explicativos*)*SALIDA:* *equi* (*bool para indicar si br_c es o no equivalente a la trenza cerrada trivial*)

```

1   Si número_argumentos=1 -> explicación=0.
2   Creo br2_c = trenza cerrada trivial.
3   Aplico algoritmo equivalentes para br_c y br2_c.

```

Una vez hemos visto los algoritmos más relevantes enfocados a los aspectos matemáticos que hemos ido desarrollando a lo largo de este proyecto, vamos a centrarnos en el algoritmo de representación de trenzas y trenzas cerradas que hemos creado. Cabe comentar que estos algoritmos han ido pasando por distintas versiones hasta obtener una versión con la que poder representar las transiciones de trenzas para el algoritmo de Dehornoy de la forma más elegante e intuitiva posible. Veamos la versión final:

Algoritmo para representar una trenza.

Vamos a ir viendo algunos algoritmos auxiliares que hemos creado y concluiremos con el algoritmo de representación de una trenza.

En primer lugar, creamos un método que nos permita representar un tubo de ciertas dimensiones (radio y número de cortes) con centro una función dada por las coordenadas x, y, z. Estos tubos formarán las cadenas de la trenza, pero ahora mismo sería un tubo sobre una función cualquiera, veámoslo:

Algoritmo 4.2.11. Algoritmo tubep(x,y,z,N,R)

ENTRADA: x (coordenada x de la función)

y (coordenada y de la función)

z (coordenada z de la función)

N (número de cortes que tendrá el tubo (cadena de la trenza))

R (Radio del tubo (cadena de la trenza))

- 1 *Obtenemos matrices con puntos circulares con centro la función de entrada.*
- 2 *Creamos una superficie a partir de dichas matrices.*
- 3 *Leemos imagen de textura.*
- 4 *Aplicamos la imagen a la superficie.*
- 5 *Establecemos puntos de iluminación.*

A continuación, vamos a ver el algoritmo que hemos creado para obtener la forma que presenta una sección de una cadena de una trenza al producirse un cruce. Con este algoritmo únicamente obtenemos las coordenadas de la función que tiene dicha forma. Hemos decidido darle esta parametrización a la función para que visualmente quede lo mejor posible.

Algoritmo 4.2.12. Algoritmo giro_base()

SALIDA: x (vector de coordenadas x de una función para generar una de las dos partes de un cruce)

y (vector de coordenadas y de una función para generar una de las dos partes de un cruce)

z (vector de coordenadas z de una función para generar una de las dos partes de un cruce)

- 1 *$x \leftarrow$ vector con valores nulos de 0 a π , coseno de π a 2π , valor 2 de 2π a 3π .*
- 2 *$y \leftarrow$ vector con valores nudos de 0 a π , seno de π a 2π , nulos de 2π a 3π .*
- 3 *$z \leftarrow$ vector con valores de 0 a 3π con frecuencia 0.1.*

De un modo similar creamos otra función para obtener la función que nos permitirá crear secciones de cadenas de las trenzas que no se encuentren cruzadas.

Algoritmo 4.2.13. *Algoritmo cilindro_base()*

SALIDA: x (vector de coordenadas x de una función para generar cilindro)

y (vector de coordenadas y de una función para generar cilindro)

z (vector de coordenadas z de una función para generar cilindro)

```
1      z <- vector con valores de 0 a 3pi con frecuencia 0.1  
2      x,y <- vectores nulos de longitud igual a la de z.
```

Para entender mejor estos tres algoritmos que hemos visto vamos a visualizar algunas secciones de cadenas con las que los explicaremos:

El algoritmo giro_base genera la función azul que vemos dentro del tubo en la imagen de la izquierda en la figura 4.3 , aunque el algoritmo en sí no representa visualmente la función.

El algoritmo cilindro_base genera la función azul que vemos dentro del tubo en la imagen derecha.

El algoritmo tubep genera y representa los tubos alrededor de las funciones indicadas.

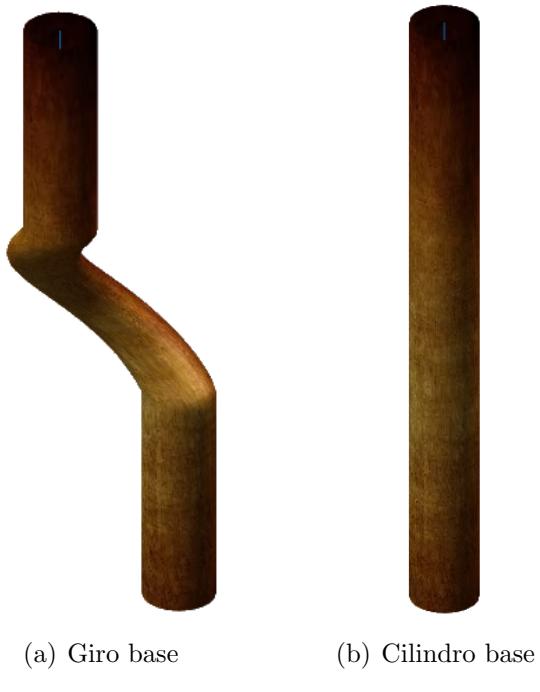


Figura 4.3: Funciones base

Ya tenemos la estructura base que podrá tener cada sección de una cadena de una trenza. Tenemos que ser capaces de obtener las funciones que representan a las cadenas de forma completa, y no sólo por secciones. Esto lo conseguimos mediante el siguiente algoritmo:

Algoritmo 4.2.14. Algoritmo param_cadenas(*indices_braid*, *n*)

ENTRADA: *indices_braid* (cadena de enteros que representa los cruces de una trenza)
n (número de cadenas de la trenza)

SALIDA: *matriz_x* (matriz que contiene las coordenadas *x* de las funciones que representan a las cadenas de la trenza)

matriz_y (matriz que contiene las coordenadas *y* de las funciones que representan a las cadenas de la trenza)

matriz_z (matriz que contiene las coordenadas *z* de las funciones que representan a las cadenas de la trenza)

- 1 Aplico *giro_base*: obtengo vectores con coordenadas *x*, *y*, *z* del giro que representa una de las dos partes del cruce de una trenza.
- 2 Aplico *cilindro_base*: obtengo vectores con coordenadas *x*, *y*, *z* de una función lineal que representa una cadena sin giro.
- 3 Si la trenza no tiene cruces...
- 4 anulamos las matrices *x*, *y*, *z*
- 5 Para cada cadena de la trenza (será la cadena actual)...
- 6 Para cada índice de *índices_braid*...
 - 7 Si $|índice| = 0$ (recordemos que es posible porque lo usamos para visualizaciones mejores) -> creo vectores con coordenadas *x*, *y*, *z* que representan a un *cilindro_base* con la traslación correspondiente.
 - 8 Si $|índice| = \text{cadena actual}$ -> creo vectores con coordenadas *x*, *y*, *z* que representan a un *giro_base* con la traslación y giro correspondiente.
 - 9 Si $|índice| = \text{cadena actual} - 1$ -> creo vectores con coordenadas *x*, *y*, *z* que representan a un *giro_base* con la traslación correspondiente.
 - 10 Si no -> creo vectores con coordenadas *x*, *y*, *z* que representan a un *cilindro_base* con la traslación correspondiente.
 - 11 Genero las matrices *x*, *y*, *z* como concatenación de dichos vectores.
 - 12 Si la trenza tiene cadenas a la derecha sin cruces...
 - 13 Añado para cada cadena una fila a cada matriz *x*, *y*, *z* con cilindros base con las traslaciones correspondientes.

Finalmente, vemos el algoritmo para representar una trenza cualquiera.

Algoritmo 4.2.15. Algoritmo representar_trenza(*br*, *N_cortes*, *Radio*)

ENTRADA: *br* (trenza)

N_cortes (número de cortes que tendrá el tubo (cadena de la trenza))

R (Radio del tubo (cadena de la trenza))

- 1 Si *número_argumentos*=1 -> *N_cortes*=20, *Radio*=0.5
- 2 Aplico *param_cadenas*: obtengo 3 matrices con coordenadas *x*, *y*, *z* de las funciones que representan las cadenas de la trenza.
- 3 Para cada cadena de la trenza (cada fila de las matrices)...
 - 4 Represento la función de la cadena (hago un plot de la cadena a partir de la fila correspondiente de la matriz *x*, *y*, *z*)
 - 5 Aplico *tubep* a dicha cadena con *N_cortes* y *Radio*.

Algoritmo para representar una trenza cerrada.

En esta última sección vamos a ver el algoritmo que hemos creado para representar una trenza cerrada cualquiera. Al igual que en la sección anterior, mostramos un pseudocódigo bastante específico para que se entienda bien pues puede resultar complejo a simple vista.

Algoritmo 4.2.16. *Algoritmo representar_trenza(br_c, N_cortes, Radio)*

ENTRADA: *br_c* (*trenza cerrada*)

N_cortes (*número de cortes que tendrá el tubo (cadena de la trenza)*)

R (*Radio del tubo (cadena de la trenza)*)

```

1   Si número_argumentos=1 -> N_cortes=20, Radio=0.5.
2   Aplico representar_trenza@trenza.
3   Creo una semicircunferencia en el plano x-z.
4   Aplico cilindro_base: obtengo vector con coordenadas x, y, z de una
      función lineal que representa una cadena sin giro.
5   Para cada cadena de la trenza...
6       Represento la semicircunferencia para los cierres superiores de
          las cadenas con los escalados y traslaciones correspondientes.
7   Aplico tubep.
8   Represento las cadenas para hacer los cierres a partir de
      cilindro_base.
9   Aplico tubep.
10  Represento la semicircunferencia para los cierres inferiores de
      las cadenas con los escalados y traslaciones correspondientes.
11  Aplico tubep.
```

Algoritmo para representar una trenza (versión antigua).

A continuación, vamos a mostrar la versión inicial que realizamos para representar trenzas. Antes de entrar en detalle vamos a tratar de mostrar una idea general sobre el algoritmo y argumentaremos por qué descartamos el uso de esta versión, creando la nueva versión que acabamos de ver.

En la versión que hemos explicado y que se encuentra implementada en tox tren, representamos las trenzas considerando cada una de las cadenas de la misma por separado pero sin cortes en ellas.

En la versión que vamos a ver a continuación, versión inicial y de la que ya no hacemos uso, representamos las trenzas mediante cortes de las cadenas que producen los cruces. Es decir, al considerar un cruce de la trenza representamos todas las secciones de cadenas que se encuentran en el mismo intervalo de planos que el cruce. Las funciones que representan a las cadenas están definidas por partes y tenemos un gran número de funciones para cada cadena, en concreto tendremos tantas funciones como cruces. Si consideramos el número de funciones por cadena y el número de cadenas de la trenza, vemos que esta versión es más complicada de manejar que en la versión actual.

Sin embargo, el motivo principal para desechar esta versión inicial es el siguiente: al aplicar el algoritmo de Dehornoy y tratar de representar los movimientos de las cadenas, es notablemente más elegante trabajar con la cadena como una función completa y no con funciones por partes que, por la manera de construir el algoritmo, están dispersas. No obstante, vamos a ver cómo implementamos dicha versión:

Al igual que hacíamos anteriormente, creamos un método que nos permita representar un tubo de ciertas dimensiones (radio y número de cortes) con centro una función dada por las coordenadas x , y , z . En este caso no hemos aplicado textura a las trenzas:

Algoritmo 4.2.17. *Algoritmo tubep(x,y,z,N,R)*

ENTRADA: x (coordenada x de la función)

y (coordenada y de la función)

z (coordenada z de la función)

N (número de cortes que tendrá el tubo (cadena de la trenza))

R (Radio del tubo (cadena de la trenza))

- 1 *Obtenemos matrices con puntos circulares con centro la función de entrada.*
- 2 *Creamos una superficie a partir de dichas matrices.*
- 3 *Establecemos puntos de iluminación.*

A continuación, vemos la función que nos permite crear y visualizar un cilindro a una altura determinada por los valores de inicio y fin. Dichos valores tienen que diferir en un valor de 3π . Además, representamos el cilindro en su posición correcta en base al eje x pues hacemos una traslación del cilindro hasta que se encuentre en el número de cadena indicado.

Algoritmo 4.2.18. *Algoritmo cilindro_braid()*

ENTRADA: $numero_cadena$ (número de cadena en la que representa el cilindro)

$inicio$ (plano z inicial en el que iniciamos el cilindro)

fin (plano z final en el que finaliza el cilindro)

N_cortes (número de cortes que tendrá el tubo (cadena de la trenza))

$Radio$ (Radio del tubo (cadena de la trenza))

- 1 *$z \leftarrow$ vector con valores de $inicio$ a fin con frecuencia 0.1*
- 2 *$y \leftarrow$ vectores nulos de longitud igual a la de z .*
- 3 *$x \leftarrow$ vectores de valor $numero_cadena$ de longitud igual a la de z .*
- 4 *Represento la función cilindro a partir de x , y , z .*
- 5 *Aplico tubep.*

Para representar los cruces de las trenzas creamos cuatro funciones auxiliares para contemplar los 4 casos que tenemos en un cruce: cadena de izquierda a derecha (de derecha a izquierda) cruzando por delante y cruzando por detrás. Este es otro aspecto clave por el que no nos parece un algoritmo sofisticado, pues se produce cierta repetición de código. Sin embargo, no deja de ser un algoritmo totalmente válido y rápido, aunque menos que el actual.

Algoritmo 4.2.19. Algoritmo braid_neg()

ENTRADA: numero_cadena (número de cadena en la que representa el cilindro)
 inicio (plano z inicial en el que iniciamos el cilindro)
 fin (plano z final en el que finaliza el cilindro)
 N_cortes (número de cortes que tendrá el tubo (cadena de la trenza))
 Radio (Radio del tubo (cadena de la trenza))

- 1 Aplico giro_braid_neg con signo positivo.
- 2 Aplico giro_braid_neg con signo negativo.

Con el siguiente algoritmo obtenemos los posibles giros que podemos tener en un cruce dependiendo del signo.

Algoritmo 4.2.20. Algoritmo giro_braid_neg()

ENTRADA: numero_cadena (número de cadena en la que representa el cilindro)
 signo (signo indicando si el giro va de izquierda a derecha o viceversa)
 inicio (plano z inicial en el que iniciamos el cilindro)
 fin (plano z final en el que finaliza el cilindro)
 N_cortes (número de cortes que tendrá el tubo (cadena de la trenza))
 Radio (Radio del tubo (cadena de la trenza))

- 1 *x <- vector con valores numero_cadena o siguiente cadena(en función de inicio, fin y signo) desde inicio hasta inicio+pi , coseno con la traslación correspondiente desde inicio+pi a inicio+2pi , valores con la cadena no tomada al inicio del vector de inicio +2pi a final.*
- 2 *y <- vector con valores nudos de inicio hasta inicio+pi , seno con la traslación correspondiente desde inicio+pi a inicio+2pi , nulos de inicio+2pi a final.*
- 3 *z <- vector con valores de inicio a fin con frecuencia 0.1.*
- 4 *Represento la función cilindro a partir de x, y, z.*
- 5 *Aplico tubep.*

Implementamos algoritmos similares para los otros dos tipos de giros posibles en un cruce.

Algoritmo 4.2.21. Algoritmo braid_pos()

ENTRADA: numero_cadena (número de cadena en la que representa el cilindro)
 inicio (plano z inicial en el que iniciamos el cilindro)
 fin (plano z final en el que finaliza el cilindro)
 N_cortes (número de cortes que tendrá el tubo (cadena de la trenza))
 Radio (Radio del tubo (cadena de la trenza))

- 1 Aplico giro_braid_pos con signo positivo.
- 2 Aplico giro_braid_pos con signo negativo.

Algoritmo 4.2.22. Algoritmo giro_braid_pos()

ENTRADA: numero_cadena (número de cadena en la que representa el cilindro)

signo (signo indicando si el giro va de izquierda a derecha o viceversa)
inicio (plano z inicial en el que iniciamos el cilindro)

fin (plano z final en el que finaliza el cilindro)

N_cortes (número de cortes que tendrá el tubo (cadena de la trenza))

Radio (Radio del tubo (cadena de la trenza))

```

1  x <- vector con valores numero_cadena o siguiente cadena(en función
   de inicio, fin y signo) desde inicio hasta inicio+pi , coseno
   con la traslación correspondiente desde inicio+pi a inicio+2pi ,
   valores con la cadena no tomada al inicio del vector de inicio
   +2pi a final.
2  y <- vector con valores nudos de inicio hasta inicio+pi, -seno con
   la traslación correspondiente desde inicio+pi a inicio+2pi,
   nulos de inicio+2pi a final.
3  z <- vector con valores de inicio a fin con frecuencia 0.1.
4  Represento la función cilindro a partir de x, y, z.
5  Aplico tubeP.

```

Para entender mejor estos últimos cuatro algoritmos que hemos mostrado, vamos a visualizar las salidas que se generan con algunos ejemplos. En concreto vamos a mostrar los giros que se producen al indicar inicio=0 y final=-3π (que sería el primer intervalo de planos que se mostraría en cualquier trenza). Podemos ver los cuatro casos posibles en la figura 4.4.

Finalmente mostramos el algoritmo que nos permite representar una trenza cualquiera. Otro de los motivos por los que hemos decidido usar la versión de la sección anterior reside en el hecho de que en esta versión las trenzas no podrán tener cadenas finales (a la derecha) si no intervienen en ningún cruce.

Algoritmo 4.2.23. Algoritmo representar_braid()

ENTRADA: indices_braid (cadena de enteros que representa los cruces de una trenza)

N_cortes (número de cortes que tendrá el tubo (cadena de la trenza))

Radio (Radio del tubo (cadena de la trenza))

```

1  Para cada cruce de indices_braid...
2    Desde 1 hasta el valor del cruce...
3      Aplicar cilindro_braid con traslación correspondiente.
4    Si el valor del cruce es positivo
5      Aplicar braid_neg con traslación correspondiente.
6    Si no
7      Aplicar braid_pos con traslación correspondiente.
8    Desde el valor del cruce hasta cadena final...
9      Aplicar cilindro_braid con traslación correspondiente.

```

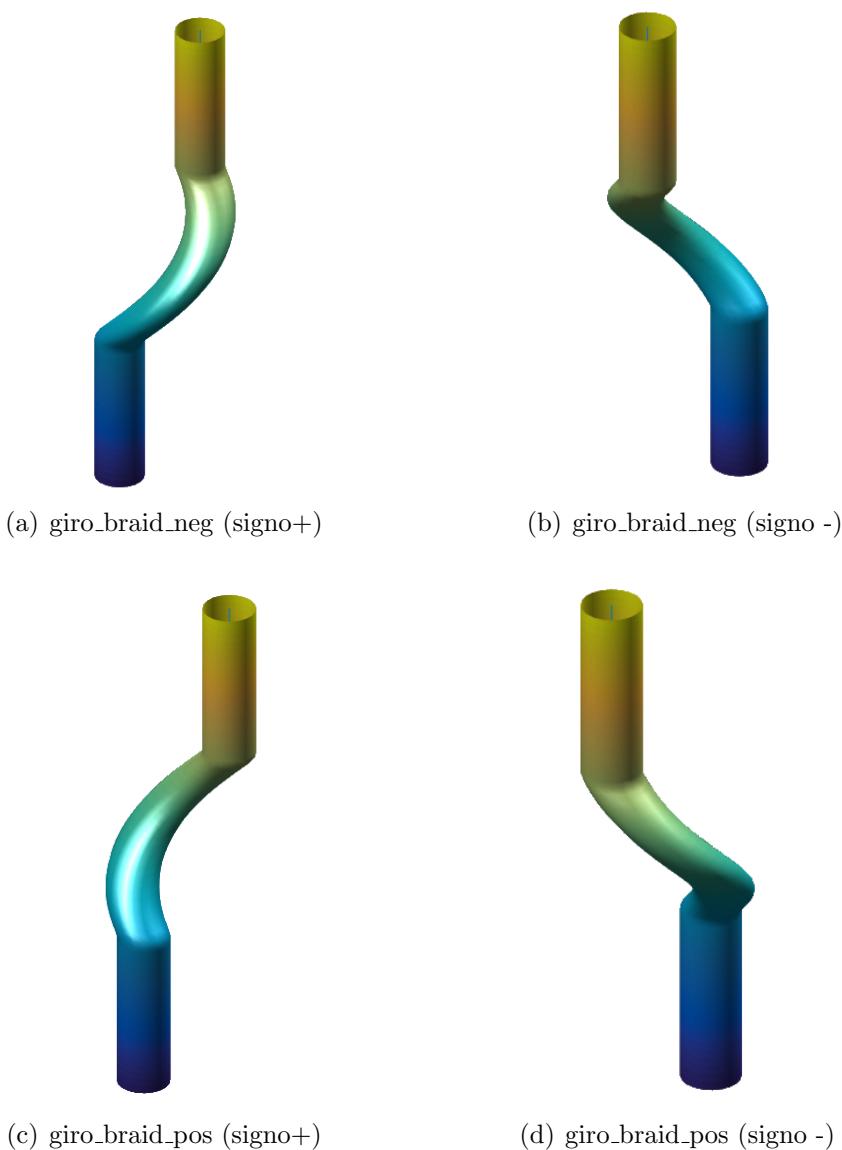


Figura 4.4: Giros cruce

4.3 Usando tox tren

En esta sección vamos a hacer un recorrido por el toolbox tox tren, de modo que el usuario podrá ver fácilmente cómo trabajar con el mismo. Haremos un repaso sobre los métodos esenciales y más importantes para el manejo de trenzas y trenzas cerradas.

4.3.1 Clase trenza.

Constructor.

En primer lugar vamos a ver el constructor para dicha clase. Podemos crear una trenza indicando los cruces de la trenza y el número de cadenas de la misma. Por ejemplo, creamos la trenza $\sigma_1\sigma_2^{-1}\sigma_1^{-1}$ con 5 cadenas del siguiente modo:

```
>> a=trenza([1 -2 -1],5)

a = trenza with properties:
indices_trenza: [1 -2 -1]
n_cadenas: 5
```

Si no indicamos el número de cadenas de la trenza, se establecerá al número mínimo de cadenas que tendrá que tener.

```
>> a=trenza([1 -2 -1])

a = trenza with properties:
indices_trenza: [1 -2 -1]
n_cadenas: 3
```

Además para ambos casos podemos indicar la trenza mediante una cadena tal y como hemos comentado al inicio del capítulo:

```
>> b=trenza('+s3-s2',7)

b = trenza with properties:
indices_trenza: [3 -2]
n_cadenas: 7
```

Del mismo modo que antes, si no indicamos el número de cadenas de la trenza se establecerá como el número mínimo que ha de tener.

```
>> b=trenza('+s3-s2')

b = trenza with properties:
indices_trenza: [3 -2]
n_cadenas: 4
```

Operaciones básicas.

Obtenemos la trenza potencia de una trenza.

```
>> pote(a,3)
ans = trenza with properties:
indices_trenza: [1 -2 -1 1 -2 -1 1 -2 -1]
n_cadenas: 3
```

Podemos obtener la trenza producto de un par de trenzas.

```
>> c=producto(a,b)
c = trenza with properties:
indices_trenza: [1 -2 -1 3 -2]
n_cadenas: 4
```

Obtenemos la trenza inversa de una trenza haciendo uso de la función inver:

```
>> c.inver
ans = trenza with properties:
indices_trenza: [2 -3 1 2 -1]
n_cadenas: 4
```

Podemos obtener el número de cruces de una trenza del siguiente modo:

```
>> c.length
ans = 5
```

Get/Set.

Podemos obtener los cruces y el número de cadenas de una trenza con los siguientes get.

```
>> a.get_indices
ans = 1      -2      -1

>> a.get_n
ans = 3
```

Cambiamos el número de cadenas de una trenza del siguiente modo:

```
>> set_n(a,6)
>> a
a = trenza with properties:
indices_trenza: [1 -2 -1]
n_cadenas: 6
```

Finalmente cambiamos los cruces de la trenza:

```
>> set_indices(a,[4 -3])
>> a
a = trenza with properties:
indices_trenza: [4 -3]
n_cadenas: 5
```

Si no indicamos un tercer parámetro que representa el número de cadenas de la trenza modificada, se establecerá al número de cadenas por defecto.

Invariantes básicos.

Para ver los invariantes básicos vamos a trabajar con la trenza que hemos creado anteriormente:

```
>> a
      a = trenza with properties:
      indices_trenza: [4 -3]
      n_cadenas: 5
```

Obtenemos el invariante exponente mediante el método exp:

```
>> a.exp
      ans = 0
```

Obtenemos la permutación de la trenza:

```
>> a.perm
      ans = 1      2      4      5      3
```

A partir de esta permutación podemos ver si la trenza es pura. Al obtener respuesta negativa, sabemos que la trenza no es pura.

```
>> a.pura
      ans = 0
```

Otros métodos destacados.

Podemos ver la representación 3D de una trenza del siguiente modo:

```
>> a.representar_trenza
```



Figura 4.5: Representación trenza Matlab.

Para aplicar el algoritmo de Dehornoy sobre una trenza y ver su representación, vamos a crear una nueva trenza más compleja y haremos uso del siguiente comando:

```
>> a=trenza([1 -2 2 2 -1])

    a = trenza with properties:
        indices_trenza: [1 -2 2 2 -1]
        n_cadenas: 3
>> [e_trivial,final]=a.dehornoy
    e_trivial = 0
    final = -2      1      2
```

Podemos ver el principio y final del proceso en la figura 4.6



Figura 4.6: Transformación Dehornoy Matlab.

Creamos una trenza a partir de los cruces que hemos obtenido al aplicar el algoritmo de Dehornoy a la trenza a:

```
>> f=trenza(final)

    f = trenza with properties:
        indices_trenza: [-2 1 2]
        n_cadenas: 3
```

Obviamente al aplicar el algoritmo de equivalencia entre las trenzas a y f tendremos que obtener respuesta afirmativa:

```
>> equivalentes(a,f)
    ans = 1
```

Finalmente podemos comprobar si una trenza dada es o no trivial. En este caso la trenza a no es equivalente a la trenza trivial.

```
>> a.es_trivial
    ans = 0
```

4.3.2 Clase trenza_cerrada.

Constructor.

Podemos obtener una trenza cerrada a partir de un vector que contiene los cruces o bien indicando la cadena de cruces, al igual que hacíamos con las trenzas. Del mismo modo, podemos indicar el número de cadenas que tendrá la cadena o establecerlo al número mínimo por defecto. Vamos a ver un ejemplo entre estas cuatro opciones posibles:

```
>> m=trenza_cerrada([3 -1 2],5)

m = trenza_cerrada with properties:
n_enlaces: 1
indices_trenza: [3 -1 2]
n_cadenas: 5
```

Además, podemos crear una trenza cerrada a partir de una trenza dada. Vamos a crear la trenza cerrada a_c a partir de la trenza a que creamos anteriormente:

```
>> a

a = trenza with properties:
indices_trenza: [1 -2 2 2 -1]
n_cadenas: 3

>> a_c=trenza_cerrada(a)

a_c = trenza_cerrada with properties:
n_enlaces: 2
indices_trenza: [1 -2 2 2 -1]
n_cadenas: 3
```

Los métodos get/set y los relativos a operaciones e invariantes básicos se llaman del mismo modo que hemos visto para trenzas. Por este motivo, para trenzas cerradas vamos a mostrar solamente la sección de otros métodos destacados.

Otros métodos destacados.

Podemos obtener la notación de Dowker de una trenza cerrada de un sólo enlace con el siguiente método:

```
>> m.Dowker
ans = 6      -4      2
>> a_c.Dowker
Se trata de un enlace.
```

Obtenemos el polinomio de Alexander de una trenza cerrada:

```
>> aux=trenza_cerrada([ 1 2 3])

aux = trenza_cerrada with properties:
n_enlaces: 1
indices_trenza: [1 2 3]
n_cadenas: 4

>> aux.Alexander
ans = -1
```

Podemos ver la representación 3D de una trenza cerrada haciendo uso de representar_trenza. Aunque para el caso de trenzas no lo comentamos, a dicho método le podemos para el número de cortes que tendrán las cadenas y el radio de las mismas.

Por ejemplo, visualizamos la trenza cerrada m con un radio de 0.4 y 30 cortes por cadena del siguiente modo:

```
>> representar_trenza(m,30,0.4)
```



Figura 4.7: Representación trenza cerrada Matlab.

Podemos ver si dos trenzas cerradas son equivalentes. Además, podemos pedir mensajes explicativos que nos muestren porqué las trenzas cerradas son o no equivalentes. Este detalle también es posible realizarlo para trenzas, simplemente tenemos que indicarlo con un nuevo parámetro.

```
>> equivalentes(a_c,m,1)
    Las trenzas cerradas no son equivalentes porque tienen
        distinto numero de enlaces.
ans = 0
```

Finalmente podemos ver si una trenza cerrada es equivalente a la trivial, y mostrar o no mensajes de salida explicativos.

```
>> es_trivial(m,true)
    La trenza dada no es equivalente a la trenza trivial. Su
        exponente no es nulo.
    Pero es posible que la trenza cerrada si sea equivalente a la
        trivial.
    Comparando los polinomios de Alexander obtenemos que...
    La trenza cerrada no es equivalente a la trivial.
ans = 0
```

Capítulo 5

Conclusiones y vías futuras

En este proyecto hemos mostrado las bases matemáticas de la teoría de nudos y teoría de trenzas, quedando una gran cantidad de aspectos por tratar pues son teorías bastante ricas. Sin embargo, sirve como una fuerte base de conocimiento que permitirá el estudio de las teorías a posibles investigadores. Tras el proyecto podemos concluir que disponemos de un amplio número de campos en los que ambas teorías pueden ser aplicadas.

Los objetivos propuestos inicialmente han sido cubiertos en su totalidad con el matriz de que el desarrollo informático lo hemos realizado en base a teoría de trenzas y no a teoría de nudos, pero ésto nos ha permitido realizar el estudio de ambas teorías matemáticas.

Por otra parte, existen diversas vías futuras a partir del trabajo realizado.

- **Possible mejora de la visualización del movimiento de trenzas equivalentes en el algoritmo de Dehornoy:**

Hemos ido mejorando la visualización de las trenzas hasta conseguir que el movimiento de las mismas sea lo menos brusco posible para que el usuario vea que los movimientos son completamente válidos. Sin embargo, algunos casos pueden resultar algo confusos de visualizar porque intervienen numerosas cadenas. Además, se podría modificar la velocidad de los movimientos en función de su dificultad visual para hacerlo aún más agradable para el usuario.

- **Implementar el algoritmo de Yamada-Vogel para demostrar el teorema de Alexander:**

Hemos visto la demostración de este algoritmo desde un punto formal matemático pero sería interesante implementarlo y obtener una representación visual de los movimientos que se van produciendo. Además, con dicho algoritmo obtendríamos una trenza cerrada que representa un nudo concreto y viceversa.

- **Creación de una interfaz de usuario en Matlab.**

Sería interesante desarrollar una interfaz de usuario que englobe todas las funcionalidades de tox tren para aquellos usuarios que prefieren trabajar en un entorno visual sencillo en lugar de trabajar con la ventana de comandos.

- **Profundización en la relación entre teoría de nudos y ADN.**

Al conocer el tipo de nudos que nos encontramos en las estructuras iniciales y finales de ADN, nos interesaría deducir la acción de la encima que actúa sobre la estructura.

Otro aspecto interesante que podríamos realizar aquí sería el siguiente: tras analizar una base de datos real que contenga la estructuras iniciales y finales de ADN y obtener la acción de las encimas que han actuado, podremos construir modelos que nos permitan predecir la estructura final de una estructura inicial de ADN.

Bibliografía

- [Adams, 1994] Adams, C. C. (1994). *The knot book: An elementary introduction to the mathematical theory of knots.* W.H. Freeman and Company.
- [D. Garber, 2008] D. Garber, S. Kaplan, M. T. (2008). A new algorithm for solving the word problem in braid groups. *Department of Mathematics and Computer Sciences Bar-Ilan University.*
- [Dehornoy, 1997] Dehornoy, P. (1997). A fast method for comparing braids. *Advances in Mathematics*, 125(2):200–235.
- [Fenn, 2016] Fenn, R. (2016). Braid programme. <http://www.layer8.co.uk/maths/braids/>.
- [Flapan, 2016] Flapan, E. (2016). Knots, molecules, and the universe: An introduction to topology. *American Mathematical Society.*
- [Flint and Rankin, 2003] Flint, O. and Rankin, S. (2003). Knotilus. <http://knotilus.math.uwo.ca/>.
- [Hoberg, 2011] Hoberg, R. (2011). knots and braids-paper.
- [ISIK, 2005] ISIK, U. (2005). Computational problems in the braid group with applications to cryptography.
- [Jkasd, 2008] Jkasd (2008). A table of prime knots up to seven crossings. https://en.wikipedia.org/wiki/File:Knot_table.svg#/media/File:Knot_table.svg.
- [Matlab, 2016a] Matlab (1994-2016a). Create and share toolboxes. http://es.mathworks.com/help/matlab/matlab_prog/create-and-share-custom-matlab-toolboxes.html#buf2ahi-3.
- [Matlab, 2016b] Matlab (1994-2016b). Display custom documentation. https://es.mathworks.com/help/matlab/matlab_prog/display-custom-documentation.html.
- [Murasugi, 2007] Murasugi, K. (2007). *Knot theory and its applications.* Springer Science & Business Media.
- [Murasugi and Kurpita, 1999] Murasugi, K. and Kurpita, B. I. (1999). A study of braids. mathematics and its applications. *Springer Science+Business Media Dordrecht*, 484.
- [Thiffeault and Budišić, 2016] Thiffeault, J.-L. and Budišić, M. (2013–2016). Braidlab: A software package for braids and loops. Version 3.2.
- [William Menasco, 2005] William Menasco, M. T. (2005). Handbook of knot theory.