Ville Lukkarila

A Mathematica-package for algebraic braid groups

Turku Centre for Computer Science

TUCS Technical Report
No 689, May 2005

A Mathematica-package for algebraic braid groups

Ville Lukkarila
vinilu@utu.fi

TUCS Technical Report
No 689, May 2005

Contents

Abstract

This technical report briefly describes the contents of the 'official' version of the Mathematica-package which was originally written during the preparation of author's thesis [31]. The thesis concentrated mainly on the treatment of the fundamental algebraic and algorithmic theory of the so-called Artin's braid group and on the recent years' attempts to use braids in public-key cryptographic protocols. This report is — in a way — an abstract of a part of the thesis [31].

Since there are many rigorous introductory texts for the theory of braids with heavy use of algebraic topology, this report does not repeat the exact mathematical definitions. Instead, this report contains simplified definitions and tries to offer a brief introduction for people with little background in topology and combinatorial group theory. However, some knowledge of basic algebra (groups, equivalence, etc.) is required.

Birman's classic book [6] is a good source for people hoping for a more rigorous treatment of the subject. However, the book [6] requires basic combinatorial group theory [32] and homotopy theory [26] as preliminary knowledge.

A continuously updated cryptographical bibliography on braid groups can be found on Helger Lipmaa's homepage at `http://www.cs.ut.ee/~helger/`. A widely used C++ library for braids can be found at [10].

Keywords: Braid group, Conjugacy problem, Cryptography, Mathematica, Word problem

TUCS Laboratory
Discrete Mathematics for Information Technology

1 Preliminaries

1.1 Definitions

1.1.1 The intuitive definition

A *geometric braid* can be defined as a finite family of parametrized continuous curves located between two (predefined) parallel planes in the space $\mathbb{R}^3$. The curves are not allowed to intersect and they start on a predefined set of distinct points, the *starting points*, on the first plane and end on a similar set of points, the *ending points*, on the second plane. The curves can either be defined only between the planes or to have constant values beyond the space between the planes. It can be assumed that the starting points and the ending points are indexed $1, 2, \ldots$. The curves can be called *strings*. A braid with n strings is called an *n-braid*.

The *multiplication* of two braids can be defined as attaching (and properly shrinking) a string of the first braid ending at the (ending) point i to the string of the second braid starting at the (starting) point i.

Two geometric braids are called *equivalent*, if the first one can be continuously deformed into the second one without any strings intersecting one another or intersecting the planes in any other points than the starting and ending points. This equivalence corresponds to the topological notion of isotopy.

Considering these equivalence classes, it can be shown that there is a multiplicative inverse for every braid and this construction gives us the *Artin braid group*, denoted by B_n. An element of an Artin braid group B_n, i.e. an equivalence class of geometric n-braids, is simply called a *braid*.

1.1.2 Topological definitions

There are two non-trivial topological definitions for the braid group. The first definition is that the braids correspond to homotopy classes of homeomorphisms of the n-punctured disc which fix the border of the disc and permute the puncture points [2]. The braid group is exactly the homotopy group (i.e. mapping class group) of the punctured sphere D_n with a fixed border.

The second definition gives the braid group B_n as the fundamental group (i.e. Poincaré group) of the set of sets $\{a_1, a_2, \ldots, a_n\} \subset \mathbb{R}^2$ with the base point $\{(1,0), (2,0), \ldots, (n,0)\}$ [20].

1.1.3 The algebraic braid group

It seems to be mathematical folklore that the free group of n generators is the fundamental group of n-punctured sphere. It can be proved that the braid group is isomorphic to a certain subgroup of the automorphisms of the free group [9, 13, 32]. Using that result it can be shown that the braid group B_n has the algebraic presentation

$$B_n = \left\langle \sigma_1, \ldots, \sigma_{n-1} \,\middle|\, \begin{array}{c} \sigma_i\sigma_j = \sigma_j\sigma_i, \text{ when } |i-j| \geq 2, \\ \sigma_i\sigma_{i+1}\sigma_i = \sigma_{i+1}\sigma_i\sigma_{i+1}, \text{ when } i = 1, \ldots, n-2. \end{array} \right\rangle . \quad (1.1)$$

The correspondence of braids as homotopy classes of surface homeomorphisms and the free group automorphisms is illustrated in figure 1.

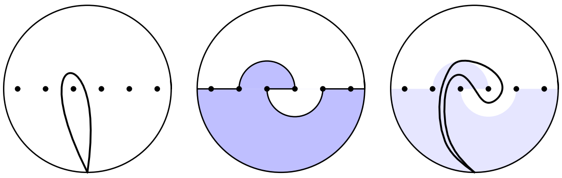

Figure 1: Braids acting on the free group

The generator σ_i can be considered as the action of intertwining strings number i and $i+1$. Therefore the defining relations can be understood as physical rules, determining how the strings can be moved, twisted, stretched, etc. without the strings intersecting each other. If the braids are represented as figures, braid diagrams, the generators are drawn as in figure 2.

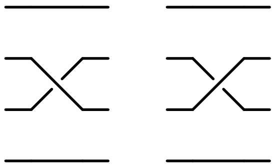

Figure 2: Generator σ_i and the inverse generator ${\sigma_i}^{-1}$.

Using the interpretation of figure 2, the half-twist Δ_5 (defined in eq. (1.2) on p. 4) of group B_5 can be depicted as in figure 3.

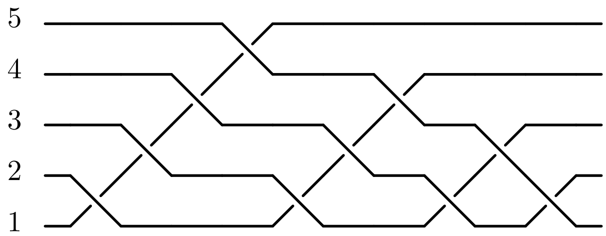

Figure 3: Half-twist Δ_5.

The first one of the defining relations in (1.1) is called the *far commutativity relation* and the second one is the *braid relation*. The defining relations can be represented graphically as in figures 4 and 5.

1.1.4 Properties

The braid group is well-studied. For example:

- it is biautomatic (so the word problem is solvable in quadratic time with respect to the word length) [19],
- it is torsion-free (i.e. it has no non-trivial elements of finite order) [20, 17, 14, 16],
- it has a lattice structure [19],

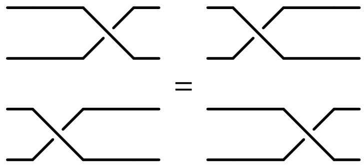

Figure 4: $\sigma_i\sigma_j = \sigma_j\sigma_i$, iff $|i-j| \geq 2$.

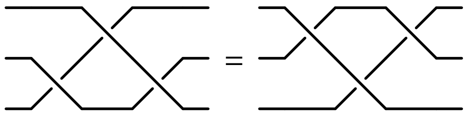

Figure 5: $\sigma_i\sigma_{i+1}\sigma_i = \sigma_{i+1}\sigma_i\sigma_{i+1}$.

- it is linear (i.e. isomorphic to some matrix group) [3, 5, 29]
- it has a total ordering, which is invariant under multiplication from the left [14, 15],
- recognizing a shortest word representative of a braid is a co-NP-complete problem [34] and
- braid theory is one approach to the problems of knot theory.

1.2 The word problem

When are two elements of the same semigroup equal? More specifically, if we are given two products of generators (or words over the set generator symbols) of the given (semi)group, are they equal? This problem, known as the *word problem* is known to be undecidable for groups (and hence also for semigroups in general). For a better account on the word problem, see [25].

The first solution to the word problem for braids was given by E. Artin himself [2]. For a given braid he defined a normal form known as the *combed braid*. His solution was exponential in terms of time. The second solution was given by Garside [23]. Garside's solution was to enumerate all the equivalent braids and represent them as numbers in base n. This approach was exponential as well.

The first polynomial-time solutions were given by Thurston [19] and Elrifai and Morton [18]. An even faster algorithm was given by Birman, Ko and Lee but for a different presentation of the braid group, known as the *band presentation* or *BKL-presentation* [7].

Different approaches for the word problem were used in articles [15, 22, 35]. The algorithm by Wiest seems to be quite efficient in finding a shortest representative for the given braid. This problem in general is known to be co-NP-complete [34].

1.2.1 Permutations

It is generally known that the symmetric group S_n is generated by transpositions $\tau_i = (i\ i+1)$ and it has defining relations

$$\begin{aligned} \tau_i\tau_j &= \tau_j\tau_i, && \text{when } |i-j| \geq 2, \\ \tau_i\tau_{i+1}\tau_i &= \tau_{i+1}\tau_i\tau_{i+1}, && \text{when } i = 1, \dots, n-2, \\ \tau_i^2 &= 1, && \text{when } i = 1, \dots, n-1. \end{aligned}$$

Hence, by the Homomorphic Principle, the symmetric group S_n is isomorphic to the quotient group $B_n / \langle {\sigma_i}^2 | i = 1, \dots, n-1 \rangle$ of the braid group B_n. The homomorphic image of a braid in the symmetric group is called an *induced permutation*. If the word length of the braid (or a word length of some of its equivalent representatives) equals the minimum word length of the induced permutation, the braid is called a *permutation braid*. If the braid contains no inverse generators, it is called a *positive* permutation braid. An equivalent definition for a permutation would be to require that every two strings cross at most once in the braid.

The word $W \in S_n$ is a *left factor* of the word $U \in S_n$, if there exists a word $V \in S_n$ so that $U = WV$ and $|U| = |W| + |V|$. This is also denoted by $W \leq_L^{S_n} U$. A *right factor* W can be defined in a similar way and it is denoted by $W \leq_R^{S_n} U$. These relations extend to the braid group almost directly.

Denote by B_n^+ the monoid of *positive braids* (braids that can be represented as words without inverse generators). We write $W \leq_L U$, if we have $U = WV$ for some braid word $V \in B_n^+$, and $W \leq_R U$, if $U = VW$ for some braid word $V \in B_n^+$. We also write $W \leq U$ if $U = V_1 W V_2$ for some positive braids $V_1, V_2 \in B_n^+$.

Given two braids U and V, the greatest element W, for which equations $W \leq_L U$ and $W \leq_L V$ hold, is called the *left meet*. It is denoted by $W = U \wedge_L V$. Similarly the *right meet* can be defined using relation $\leq_R$ and it is denoted by $U \wedge_R V$. The least element for which equations $U \leq_L W$ and $V \leq_L W$ hold, is called the *left join*. It is denoted by $U \vee_L V$. Again, the *right join* $U \vee_R V$ can be defined in a similar way. A meet is also known as the *greatest common divisor* and join is known as the *least common multiple*. If meet and join operations (according to some relation) are defined in the given set, then the set is called a *lattice* (or a *lattice-ordered set*).

With respect to these previous relations, the greatest element in the set of all positive permutation braids is the *half-twist* Δ_n, defined recursively by

$$\Delta_n = \sigma_1\sigma_2 \cdots \sigma_{n-1}\Delta_{n-1}. \tag{1.2}$$

The induced permutation Ω_n of Δ_n (defined by $\Omega_n : i \mapsto n - i$) is likewise the greatest element in the permutation group with respect to earlier relations. Hence it is quite obvious, that a natural way to handle braids is to consider them as a sequence of permutations.

1.2.2 The left canonical form

The most frequently mentioned normal form for a braid word W in terms of generators $\sigma_1, \dots, \sigma_{n-1}$ is the *left canonical form* [18] or *left greedy form* [19]

$$W = \Delta_n^m A_1 A_2 \cdots A_k, \tag{1.3}$$

where words A_i are permutation braids fulfilling the *left-weightedness* condition

$$\sigma_i \leq_L A_{i+1} \Longrightarrow \sigma_i \leq_R A_i. \tag{1.4}$$

If condition (1.4) holds for some braids A and B, then the product AB is *left-weighted*. For the given braid W we define *infimum* $\inf W = \max\left\{i \in \mathbb{Z} \middle| \Delta^i \leq W\right\}$ and *supremum* $\sup W = \min\left\{i \in \mathbb{Z} \middle| W \leq \Delta^i\right\}$. Integer value $\ell_{\text{Can}}(W) = \sup W - \inf W$ is known as the *canonical length* of braid W. With respect to eq. (1.3), it can be shown that $\ell_{\text{Can}}(W) = k$, $\inf W = m$ and $\sup W = m + k$ [18].

It can be shown that representation (1.3) is unique and can be constructed in quadratic time with respect to the word length and in almost linear time with respect to the number of generators [18, 19]. Using the *starting sets*

$$\mathbf{S}(W) = \left\{i \middle| W = \sigma_i W', W' \in B_n^+\right\}$$

and *finishing sets*

$$\mathbf{F}(W) = \left\{i \middle| W = W'\sigma_i, W' \in B_n^+\right\}$$

defined for positive braids W in [18], condition (1.4) can be expressed as $\mathbf{F}(A_i) \supseteq \mathbf{S}(A_{i+1})$. A nice property of left-weightedness is that $\mathbf{S}(A_1) = \mathbf{S}(A_1 A_2 \cdots)$ for any left-weighted sequence $A_1, A_2, \ldots$ of permutation braids.

1.2.3 The mixed canonical form and the right canonical form

There are also other normal forms for braid words. One of them is the *right canocical form* (or *right greedy form*)

$$W = A_1 A_2 \cdots A_k \Delta_n^m, \tag{1.5}$$

defined in the book [19]. In eq. (1.5) words A_i are again permutation braids, but now holding the *right-weightedness* condition $\mathbf{F}(A_i) \subseteq \mathbf{S}(A_{i+1})$.

Yet another normal form is the *mixed canonical form* [19] (or *Thurston normal form*)

$$W = U^{-1}V. \tag{1.6}$$

In mixed canonical form (1.6) both braids U and V are positive, in the left canonical form and they fulfill condition $\mathbf{S}(U) \cap \mathbf{S}(V) = \emptyset$. The mixed canonical form gives the shortest word representative of form $U^{-1}V$. However, this is not the shortest word representative in general.

1.3 The conjugacy problem

The *conjugacy problem* is stated generally as follows: *Given two elements x and y of a semigroup S, determine whether there exists an element u of S such that*

$$xu = uy \tag{1.7}$$

as elements of S. In the case of S being a group, equation (1.7) is usually rewritten in the form of $x = uyu^{-1}$ or $y = u^{-1}xu$. The search version or computational version of the conjugacy problem, *the conjugacy search problem*, asks to *find an* *element u, for which equation* (1.7) *holds when elements x and y are given.* This element u is called simply a *conjugator*.

The conjugacy problem is a generalization of the word problem. If the conjugacy problem could be solved, so could the word problem by choosing $u = 1_S$.

The conjugacy problem of braids is related to the notorious equivalence problem of knots and links. If two braids are conjugate, their closures are equivalent as links (or knots). Unfortunately, the converse does not hold. There are different definitions for equivalence depending on whether braids are considered only as elements of a group or as parts of links (and knots).

1.3.1 The Summit set

Garside was the first person to show that the conjugacy problem of braids is solvable [23, 6]. The solution is based on the algorithmic construction of a set, known as the *summit set*. The summit set contains all the conjugates with maximal infimum of the given braid. Garside showed that two braids are conjugates if and only if their summit sets are equivalent.

The nice property of the summit set (as with super- and ultra summit set) is that to generate all the elements of the set, only a small number of conjugators is needed. More specifically, all the elements of the super summit set can be found by conjugating existing elements with permutation braids. This result is known as the *convexity theorem* [23, 18, 7, 21, 24]. Furthermore, the task of finding just some element of the summit set can be solved with a quadratic amount of normal form conversions [8].

The maximal infimum within the conjugacy class (and summit set) is called *summit exponent* (according to Garside) or *summit infimum*.

1.3.2 The super summit set

The *super summit set* (defined by Elrifai and Morton [18] and later by Birman, Ko and Lee [7]) is a small subset of the summit set. The super summit set consists of those braids which have the minimal supremum among all the braids within the summit set. The minimal supremum within conjugacy class is called *summit supremum*.

It can be shown that the summit infimum and summit supremum can be simultaneously achieved within the conjugates, so the super summit set is the set of those conjugates that achieve both the summit infimum and the summit supremum. Again, it can be shown that two braids are conjugates to each other if and only if their super summit sets intersect (and hence are also equal).

It is conjectured that the size of the super summit set is polynomial with respect to the word length (in Artin's presentation or in band presentation) of a braid [8]. However, it has not been proved so far.

1.3.3 The ultra summit set

An even smaller subset of the summit set is the *ultra summit set* [24]. It contains exactly those elements W of the super summit set for which $\mathbf{c}^k(W) = W$

for some $k > 0$. Mapping $\mathbf{c}(\cdot)$ is the so-called *cycling* defined as $\mathbf{c} : W \mapsto (\tau^{-m}(A_1))^{-1} W \, \tau^{-m}(A_1)$ using the notation from equation (1.3).

The difference in size with respect to the super summit set is noticeable. For example, the super summit set of braid

$$\Delta_4^{-2} \, (\sigma_2\sigma_3\sigma_1) \, (\sigma_3\sigma_1) \, (\sigma_3\sigma_1) \, (\sigma_1\sigma_2\sigma_3\sigma_2) \, (\sigma_2\sigma_3\sigma_2\sigma_1) \, (\sigma_3) \, (\sigma_3\sigma_2)$$

contains 110 elements, but the ultra summit set contains only 14 elements. For more dramatic figures, please refer to section 5 of Gebhardt's article [24].

1.4 Linear representations

There are many linear representations for the braid group. However, in this report we consider only two of them, namely the Burau representation and the Lawrence-Krammer representation.

1.4.1 The Burau representation

The *Burau representation* is defined as mapping $\rho_{\mathrm{B}} : B_n \to GL_n(\mathbb{Z}[t, t^{-1}])$

$$\sigma_i \mapsto \begin{pmatrix} \mathbf{I}_{i-1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & 1-t & t & \mathbf{0} \\ \mathbf{0} & 1 & 0 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}_{n-i-1} \end{pmatrix}.$$

The Burau representation can be reduced to an $(n-1)$-representation $\rho_{\mathrm{rB}} : B_n \to GL_{n-1}(\mathbb{Z}[t, t^{-1}])$,

$$\sigma_i \mapsto \begin{pmatrix} \mathbf{I}_{i-2} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & 1 & 0 & 0 & \mathbf{0} \\ \mathbf{0} & t & -t & 1 & \mathbf{0} \\ \mathbf{0} & 0 & 0 & 1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}_{n-2-i} \end{pmatrix}$$

known as the *reduced Burau representation* [6].

The Burau representation is not *faithful* (i.e. the image of the group with respect to the mapping is not isomorphic to the group itself). It also seems that there is no deterministic algorithm which would return some preimage of the given Burau matrix. However, there are a few heuristic algorithms to calculate a preimage with empirically high success rate. Although the Burau representation is not faithful, it has a lower order than the Lawrence-Krammer representation. This makes it slightly more popular in the linear-algebraic cryptanalytic attacks against the braid group cryptosystems [27, 30]. The question whether the Burau representation is faithful for $n = 4$ is still without an answer. It is faihtful for $n = 3$ [6, 31] and unfaithful for values $n \geq 5$ [4, 6].

There is also the *colored Burau representation* [33] which is used in the AAFG1 key agreement protocol [1]. However, it will not be represented it here.

1.4.2 The Lawrence-Krammer representation

The Lawrence-Krammer representation (also known as the Lawrence-Krammer-Bigelow representation) is a special case of the Lawrence representation. The *Lawrence-Krammer representation* is defined as the linear mapping $\rho_{\mathrm{K}} : B_n \to GL_{n(n-1)/2}(V)$, where $V = \mathbb{Z}[t^{\pm 1}, q^{\pm 1}]$. If we assume that the module V has base $\{x_{i,j} \,|\, 1 \leq i < j \leq n\}$, $\rho_{\mathrm{K}}(\sigma_k)$ acts as follows [29]:

$$\rho_{\mathrm{K}}(\sigma_k)x_{i,j} = \begin{cases} tq^2 x_{k,k+1}, & i = k, j = k+1, \\ (1-q)x_{i,k} + qx_{i,k+1}, & j = k, i < k, \\ x_{i,k} + tq^{k-i+1}(q-1)x_{k,k+1}, & j = k+1, i < k, \\ tq(q-1)x_{k,k+1} + qx_{k+1,j}, & i = k, k+1 < j, \\ x_{k,j} + (1-q)x_{k+1,j}, & i = k+1, k+1 < j, \\ x_{i,j}, & i < j < k \text{ or } k+1 < i < j \text{ and} \\ x_{i,j} + tq^{k-i}(q-1)^2 x_{k,k+1}, & i < k < k+i < j. \end{cases}$$

The Lawrence-Krammer representation is known to be faithful and hence the braid group is linear [5, 29]. A cryptanalytic attack by Cheon and Jun [12] uses the Lawrence-Krammer representation to defeat both the old braid group public-key cryptosystem [28] and the new one [11].

2 The package `AlgebraicBraids`

2.1 About the package

A large part of the algorithms contained in this package can be found from articles [18, 21, 24, 11]. Some functions are natural implementations of the results given in book [19].

The braids are handled in five different forms in the package. These representation forms are a word, left canonical form, mixed canonical form, right canonical form, and a specifically created list structure for Mathematica.

Words are simply represented as lists of integers, where a positive integer i corresponds to generator symbol σ_i and a negative integer $-i$ corresponds the inverse generator ${\sigma_i}^{-1}$. Left canonical form for a braid is represented as a list $\{i, A_1, A_2, ...A_l\}$, where i is the infimum and A_k are permutations representing the canonical factors. Mixed canonical form $U^{-}V$ is represented as a two-element list $\{U, V\}$ of corresponding expressions for the left canonical forms. Right canonical form is represented as a list $\{A_1, A_2, ..., A_l, i\}$, where A_k are again the canonical factors and i is the infimum. A more general way to express an arbitrary braid is to represent it as a three-element expression `ArtinBraid[`n`,`t`,`r`]`. Here element n is the braid index (number of the strings), element r is an expression representing the braid in some form, and element t tells which (normal) form is used for expression r.

All the algorithms are implemented mainly for the left canonical form only. Instead of implementing same algorithms for all the normal forms, the package contains conversion methods between different normal forms. Some algorithms are not implemented for all the normal forms to emphasize the increase of the time complexity caused by non-optimal normal form. For example, no algorithm `FinishingSetLCF` is contained in the package, because it is inconvenient to search the finishing set for a given braid in left canonical form, since the braid should be converted to the right canonical form first.

The functions operating with different normal forms or groups are distinguished by the suffixes in their names. For example, `LeftMeetLCF` calculates the left meet for the given sequence for braids in left canonical form, whereas `LeftMeetPermutation` does the same for the permutations of the `Combinatorica`-package. Likewise functions `ProductLCF`, `ProductPermutation` and `ProductBraid` calculate the product for left canonical forms, permutations and `ArtinBraid`-structures, respectively.

For brevity, in the following subsections similar functions are gathered under the same headline. If there are functions `FunctionLCF`, `FunctionMCF` and `FunctionRCF`, their descriptions are under headline `Function[L,M,R]CF`.

In this report and package the multiplication and the mapping direction of permutations are defined to be from right to left. That is, the permutations act as mappings from the left. This convention is different from the one in the article of Elrifai and Mortin [18] but the same as in the works of Thurston [19] and Krammer [29]. The braid strings are numbered from the left, but the strings are colored form the right [33].

The package described in this report can be found at the publication database of Turku Center for Computer Science at

`www.tucs.fi`

along with this report.

2.2 The word problem

2.2.1 Basic word operations

BraidAsWordQ `BraidAsWordQ[`W`]` returns `True`, if expression W can represent a braid word. Otherwise it returns `False`.

PositiveBraidAsWordQ `PositiveBraidAsWordQ[`W`]` outputs `True`, if expression W can represent a positive braid word. Otherwise it returns `False`.

FreelyReducedWord `FreelyReducedWord[`W`]` returns a freely reduced word equivalent to word W.

Delta `Delta[`n`]` returns element Δ_n as a word.

WordToPermutation `WordToPermutation[`W`,`n`]` converts the given word W to a permutation in the set S_n according to the mapping $i \mapsto (|i|\ |i|+1)$, where i is an integer representing a generator.

WordTo[L,M,R]CF `WordToLCF[`W`,`n`]` converts the given braid word W into its left canonical representation in group B_n. `WordToMCF[`W`,`n`]` and `WordToRCF[`W`,`n`]` do the same for mixed and right canonical forms, respectively.

ProductWord `ProductWord[`W_1`,`W_2`,...]` returns the concatenation of words W_1,W_2,....

InverseWord `InverseWord[`W`]` returns the formal inverse word W^{-1} of the given word W.

ReverseWord `ReverseWord[`W`]` returns the word reverse W^{R} of the given word W.

NegationWord Given a word W, `NegationWord[`W`]` returns the word negation (word $(W^{\mathrm{R}})^{-1}$).

HalfTwistWord Given a braid word W, `HalfTwistWord[`W`,`n`]` returns element $\tau(W) = \Delta_n^{-1} W \Delta_n$.

InducedPermutationWord This function outputs the same result as `WordToPermutation`.

2.2.2 Permutation operations

Transposition `Transposition[`i`,`n`]` returns transposition $(i\ i+1) \in S_{\mathsf{n}}$.

Omega `Omega[`n`]` returns the induced permutation of the half-twist Δ_n.

PermutationToWord `PermutationToWord[`P`]` return a word representation for the given permutation P in terms of generator cycles $(i\ i+1)$.

PermutationTo[L,M,R]CF Assuming that the given permutation P represents a positive permutation braid, `PermutationToLCF[`P`]` converts it to a left canonical form.

HalfTwistPermutation `HalfTwistPermutation[`P`]` conjugates the given permutation P by element Ω_n.

ProductPermutation `ProductPermutation[`P_1`,`P_2`,...]` returns the product of permutations P_1,P_2,....

SolveConjugatorPermutation If permutations P and Q are conjugates, `SolveConjugatorPermutation[`P`,`Q`]` returns the permutation X for which equation $Q = X^{-1}PX$ holds. This is only an implementation of the "first year studies"-algorithm.

FinishingSetPermutation Assuming that the given permutation P represents a positive permutation braid, `FinishingSetPermutation[P]` returns the finishing set for permutation braid P.

StartingSetPermutation Assuming that the given permutation P represents a positive permutation braid, `StartingSetPermutation[P]` returns the starting set for permutation braid P.

LeftWeightedQ Assuming that the given permutations P_1,P_2,... represent positive permutation braids, `LeftWeightedQ[`P_1`,`P_2`,...]` returns `True` iff the given sequence is left-weighted.

RightWeightedQ Assuming that the given permutations P_1,P_2,... represent positive permutation braids, `RightWeightedQ[`P_1`,`P_2`,...]` returns `True` iff the given sequence is right-weighted.

PermutationLength `PermutationLength[`P`]` returns the minimal length for the given permutation P with respect to generators $(i\ i+1)$, $i = 1, \ldots, n-1$. This equals the cardinality of the minimal relation whose transitive closure equals the given permutation [19].

PermutationBraidQ `PermutationBraidQ[`B`]` returns `True`, iff the given word B represents a permutation braid, that is, the word length of word B equals the word length of the induced permutation.

HalfPermutationPairsQ Given a set of integer pairs considered as relation, `HalfPermutationPairsQ[`S`]` returns `True` iff the transitive closure of S is a half-permutation [29].

PermutationPairsQ Given a set of integer pairs considered as relation, `PermutationPairsQ[`S`]` returns `True` iff the transitive closure of S is a permutation [19, 29].

MaximalPermutation `MaximalPermutation[`H`]` returns the maximal permutation (considered as a relation) contained within the given half-permutation H [29].

2.2.3 Braid operations

BraidIn[L,M,R]CFQ `BraidInLCFQ[`B`]` returns `True` iff B is a left canonical form for some braid. `BraidInMCFQ[`B`]` and `BraidInRCFQ[`B`]` do the same for mixed and right canonical forms.

PositiveBraidIn[L,M,R]CFQ `PositiveBraidInLCFQ[`B`]` returns `True` iff B is a left canonical form for some positive braid.

[L,M,R]CFToWord `LCFToWord[`B`,`n`]` converts the given left canonical form to a word over the generators of B_n. `MCFToWord[`B`,`n`]` and `RCFToWord[`B`,`n`]` do the same for mixed and right canonical form.

[L,M,R]CFTo[L,M,R]CF These functions convert one representation form to another.

Infimum[L,R]CF `InfimumLCF[`B`]` returns the infimum of braid B.

Supremum[L,R]CF `SupremumLCF[`B`]` returns the supremum of braid B.

CanonicalLength[L,R]CF `CanonicalLengthLCF[`B`]` returns the canonical length of braid B.

LeftmostFactorLCF Assuming that braid $W = A_1 A_2 \cdots A_k \in B_n^+$ is written in its left canonical form `LeftmostFactorLCF[`B`,`n`]` returns factor A_1.

RightmostFactorRCF Assuming that braid $W = A_1 A_2 \cdots A_k \in B_n^+$ is written in its right canonical form `RightmostFactorRCF[`B`,`n`]` returns factor A_k.

StartingSetLCF `StartingLCF[`B`,`n`]` returns the starting set of braid $W \in B_n^+$.

FinishingSetRCF `FinishingRCF[`B`,`n`]` returns the finishing set of braid $W \in B_n^+$.

Product[L,M,R]CF `ProductLCF[`W_1`,`W_2`,...]` returns the product of braids W_1,W_2,....

Inverse[L,M,R]CF `ProductLCF[`B`]` returns the inverse W^{-1} of braid B.

Reverse[L,M,R]CF `ReverseLCF[`B`]` returns the word reverse B^{R} of the given braid.

Negation[L,M,R]CF `ReverseLCF[`B`]` returns the negation $B^{\mathrm{R}^{-1}}$ of the given braid.

HalfTwist[L,M,R]CF `HalfTwistLCF[`B`]` returns element $\tau(B) = \Delta^{-1} B \Delta$.

InducedPermutation[L,M,R]CF `InducedPermutationLCF[`B`]` returns the induced permutation of braid B.

2.2.4 Lattice operations

The algorithms for computing meet and join can be found in Epstein's book [19] and article [11]. The meet and join for braids are computed using the mixed canonical form as described by Thurston [19].

IsLeftFactorPermutation `IsLeftFactorPermutation[`P`,`Q`]` returns `True` iff $P \geq_L^{S_n} Q$.

IsRightFactorPermutation `IsRightFactorPermutation[`P`,`Q`]` returns `True` iff $P \geq_R^{S_n} Q$.

IsLeftFactor[L,M,R]CF `IsLeftFactorLCF[`A`,`B`]` returns `True` iff $A \geq_L B$.

IsRightFactor[L,M,R]CF `IsRightFactorLCF[`A`,`B`]` returns `True` iff $A \geq_R B$.

LeftMeetPermutation `LeftMeetPermutation[`P_1`,`P_2`,...]` returns the left meet $\bigwedge_L^{S_n}{}_{i=1,\ldots} P_i$.

RightMeetPermutation `RightMeetPermutation[`P_1`,`P_2`,...]` returns the right meet $\bigwedge_R^{S_n}{}_{i=1,\ldots} P_i$.

LeftJoinPermutation `LeftJoinPermutation[`P_1`,`P_2`,...]` returns the left join $\bigvee_L^{S_n}{}_{i=1,\ldots} P_i$.

RightJoinPermutation `RightJoinPermutation[`P_1`,`P_2`,...]` returns the right join $\bigvee_R^{S_n}{}_{i=1,\ldots} P_i$.

LeftMeet[L,M,R]CF `LeftMeetLCF[`B_1`,`B_2`,...]` returns the left meet $\bigwedge_{L\,i=1,\ldots} B_i$.

RightMeet[L,M,R]CF `RightMeetLCF[`B_1`,`B_2`,...]` returns the right meet $\bigwedge_{R\,i=1,\ldots} B_i$.

LeftJoin[L,M,R]CF `LeftJoinLCF[`B_1`,`B_2`,...]` returns the left join $\bigvee_{L\,i=1,\ldots} B_i$.

RightJoin[L,M,R]CF `RightJoinLCF[`B_1`,`B_2`,...]` returns the right join $\bigvee_{R\,i=1,\ldots} B_i$.

2.3 The conjugacy problem

The algorithm for computing the super summit set is due to Franco and Gonzalez-Meneses [21]. For debugging purposes the ”trivial” version of the algorithm (i.e. the original method of Elrifai and Morton using all permutation braids as conjugators) is contained in the private part of the package. Also the methods for computing elements r_x and ρ_x [21] are in the private context.

2.3.1 The super summit set

SummitForm[L,M,R]CFQ `SummitFormLCFQ[`B`]` returns `True` iff braid B belongs to its own summit set.

FindSummitForm[L,M,R]CF `FindSummitFormLCF[`B`]` returns some element in the summit set of braid B.

SuperSummitForm[L,M,R]CFQ `SuperSummitFormLCFQ[`B`]` returns `True` iff braid B belongs to its own super summit set.

FindSuperSummitForm[L,M,R]CF `FindSuperSummitFormLCF[`B`]` returns some element in the super summit set of braid B.

SuperSummitSet[L,M,R]CF `SuperSummitSetLCF[`B`]` returns the super summit set of braid B.

2.3.2 The ultra summit set

The ultra summit set is computed according to Gebhardt [24]. The subalgorithms for elements u_i, $\pi_y(s)$ and $p_x(s)$ and sets C_y and $F_x(u)$ [24] are located in the private context.

UltraSummitForm[L,M,R]CFQ `UltraSummitFormLCFQ[`B`]` returns `True` iff braid B belongs to its own ultra summit set.

FindUltraSummitForm[L,M,R]CF `FindUltraSummitFormLCF[`B`]` returns some element in the ultra summit set of braid B.

UltraSummitSet[L,M,R]CF `UltraSummitSetLCF[`B`]` returns the ultra summit set of braid B.

2.4 Linear representations

2.4.1 The Burau representation

The Burau representation is defined as in Birman’s book [6]. A heuristic algorithm for the preimage of a Burau matrix was written following the the article of Hughes [27]. The colored Burau matrices were defined following the article of Morton [33]. A variant of the colored Burau representation was used in the key extractor algorithm of the AAFG1 key generation algorithm [1].

Burau `Burau[`W`,`n`,`t`]` takes a braid word on n strings and outputs the corresponding $n \times n$ Burau matrix with variable t.

InvertBurauH `InvertBurauH[`M`,`t`,`l`]` tries to find heuristically a preimage for the given Burau matrix M with variable t. If an unreduced braid word with word length less than l cannot be found as preimage, the method returns `$Failed`.

ReducedBurau `ReducedBurau[`W`,`n`,`t`]` takes a braid word W on n strings and outputs the corresponding $(n-1) \times (n-1)$ reduced Burau matrix with variable t.

ColoredBurau `ColoredBurau[`W`,`n`,`t`]` takes a braid word W on n strings and outputs the $(n-1) \times (n-1)$ colored Burau matrix with variable $t[[i]]$ corresponding the ith string. The output follows the definition of Morton [33].

2.4.2 The Lawrence-Krammer representation

The preimage algorithm given by Cheon and Jun [12] contained a few errors and ambiquities, so the following algorithm was given in [31]:

ALGORITHM 2.1 ($\rho_{\mathrm{K}}{}^{-1}(M)$).

Input:	`Krammer's matrix` $\rho_{\mathrm{K}}(W) \in GL_m\left(\mathbb{Z}[t^{\pm 1}, q^{\pm 1}]\right)$
Output:	`Preimage of` $\rho_{\mathrm{K}}(W)$ `in left canonical form.`

$d_t \leftarrow$ order of variable t in matrix $\rho_{\mathrm{K}}(W)$
$\rho_{\mathrm{K}}(W) \leftarrow \rho_{\mathrm{K}}(\Delta)^{-d_t}\, \rho_{\mathrm{K}}(W)$
$\ell \leftarrow$ degree of variable t in matrix $\rho_{\mathrm{K}}(W)$ -1
for $k = 1, \ldots, \ell$
  $\mathbf{v} \leftarrow \rho_{\mathrm{K}}(W) \cdot (1)_{1\times m}$
  $A \leftarrow \{(i,j) \,|\, (\mathbf{v}, x_{i,j}) \in t\mathbb{R}[t]\}$
  $A_k \leftarrow \mathrm{GB}(A)$
  $\rho_{\mathrm{K}}(W) \leftarrow \rho_{\mathrm{K}}(A_k)^{-1}\, \rho_{\mathrm{K}}(W)$
return $\Delta^{d_t} A_1 \cdots A_\ell$

The article [12] lacked the proof of the algorithm. However, lemmas 3.2, 4.3, 4.5 and theorem 6.2 of Krammer's article [29] give the needed argument. The notation used above follows Krammer's article.

`Krammer` `Krammer[`W`,`n`,`t`,`q`]` returns the Lawrence-Krammer matrix for the given braid word W with n strings. The head of argument t should be `Symbol` and q should fulfill condition $0 < q < 1$ if q is numeric.

`InvertKrammer` Given a Lawrence-Krammer matrix M, `InvertKrammer[`M`,`t`,`q`]` returns the preimage of M. Argument t must be a symbol and q should fulfill condition $0 < q < 1$ if q is numeric.

Methods for handling half-permutations and the set A [29, 12] are in the private context of the package.

2.5 Specific Mathematica-functions

2.5.1 Plotting braid diagrams

`BraidDiagramPlot` `BraidDiagramPlot[`W`,`n`,`c`,`*opts*`]` draws the braid diagram of braid word W with n strings. Color $c[[i]]$ is used for the string $ip[[i]]$ where ip is the inverse permutation of the induced permutation of W. In other words, the strings are colored in the order of their end points. Expression `opts` is passed as options to Graphics-function after removing options for `BraidDiagramPlot`. `BraidDiagramPlot` returns a `Graphics`-object.

`BraidDiagramPlot3D` `BraidDiagramPlot3D[`W`,`n`,`c`,`r`,`*opts*`]` draws a three-dimensional braid diagram of braid word W with n tube-like strings with radius r. Coloring and options are handled like with `BraidDiagramPlot`. `BraidDiagramPlot3D` returns a `Graphics3D`-object.

There are optional arguments for both `BraidDiagramPlot` and `BraidDiagramPlot3D` that have not been documented here. To access those explanations, use `Options`-command and `::usage`-prefix.

2.5.2 Using the ArtinBraid-head

In the package expressions with head ArtinBraid and three elements are used to express an arbitrary braid object. The form of a valid braid object is ArtinBraid[n,t,r], where n is the braid index (number of strings), r is a representation of the braid (word, mixed canonical sequence etc.) and element t tells what type of element r is.

The benefits of using ArtinBraid-objects are purely cosmetic. Functions Format and TeXForm have been overridden for head ArtinBraid, so Mathematica outputs clear braid word expressions instead of just lists. Also, Mathematica operations ArtinBraid[...]*ArtinBraid[...] and ArtinBraid[...]^i_Integer have been defined in the obvious way.

ArtinBraidQ ArtinBraidQ[obj] returns True iff the given object obj with property Head[obj]==ArtinBraid is really a valid braid object.

PositiveArtinBraidQ This method returns True, iff the given braid is positive.

ConstructBraidFromWord ConstructBraidFromWord[W,n] constructs an ArtinBraid-object from word W with braid index n.;

ConstructBraidFrom[L,M,R]CF ConstructBraidFromLCF[f,n] constructs an ArtinBraid-object from left canonical form f with braid index n.

ToWord,ToLCF,ToMCF,ToRCF ToWord[B] converts an ArtinBraid-expression B into a word. Functions ToLCF,ToMCF and ToRCF do the same for left, mixed and right canonical forms.

ProductBraid ProductBraid[B_1,B_2,...] returns the product of expressions $B_1, \ldots$.

InverseBraid, ReverseBraid and **NegationBraid** ProductBraid[B] returns the inverse braid of ArtinBraid-expression B. ReverseBraid[B] returns the word reverse braid of expression B. NegationBraid[B] returns the negation of expression B.

InducedPermutation and **HalfTwist** InducedPermutation[B] returns the induced permutation and HalfTwist[B] returns element $\tau(B)$ of the input braid expression B.

LeftmostFactor and **RightmostFactor** LeftmostFactor[B] returns the leftmost factor and RightmostFactor[B] returns the rightmost factor of the ArtinBraid-expression B.

StartingSet and **FinishingSet** StartingSet[B] returns the starting set and FinishingSet[B] returns the finishing set of ArtinBraid-expression B.

LeftMeet, RightMeet, LeftJoin and **RightJoin** Commands LeftMeet[B_1,B_2,...], RightMeet[B_1,B_2,...], LeftJoin[B_1,B_2,...] and RightJoin[B_1,B_2,...] return the meets with respect to orders $\leq_L$ and $\leq_R$ and joins with respect to orders $\leq_L$ and $\leq_R$, respectively.

Infimum, Supremum and **CanonicalLength** Infimum[B], Supremum[B] and CanonicalLength[B] return infimum, supremum and canonical length, respectively, for the given braid B.

SuperSummitSet and **UltraSummitSet** SuperSummitSet[B] returns the super summit set and UltraSummitSet[B] returns the ultra summit set of the given braid B.

2.6 Some examples

The inverse braid

An inverse braid can be calculated as follows:

```
W = {1, 2, 1, -3, -3, 1};
lcf = WordToLCF[W, 5];
InverseLCF[lcf]
B = ConstructBraidFromLCF[lcf, 5];
B^(-1)
```

The output is

```
{-2, {5, 4, 3, 1, 2}, {3, 2, 4, 5, 1}, {2, 1, 3, 4, 5}, {2, 3, 4, 1, 5}}
```

$\Delta_5^{-2}\,(\sigma_2\,\sigma_1\,\sigma_3\,\sigma_2\,\sigma_1\,\sigma_4\,\sigma_3\,\sigma_2\,\sigma_1)\,(\sigma_1\,\sigma_2\,\sigma_3\,\sigma_4\,\sigma_1)\,(\sigma_1)\,(\sigma_1\,\sigma_2\,\sigma_3)$

The ultra summit set

Assume that we wish to calculate the left canonical form and the ultra summit set of braid $\sigma_1{\sigma_3}^{-1}{\sigma_3}^{-1}\sigma_2\sigma_3{\sigma_4}^{-1} \in B_5^+$. Then we type the following:

```
W = {1, -3, -3, 2, 3, -4};
lcf = WordToLCF[w, 5]
UltraSummitSetLCF[lcf]
```

After that we get the following output:

```
{-2, {5, 3, 4, 2, 1}, {1, 5, 4, 2, 3}, {3, 4, 5, 1, 2}}

{{-1, {1, 5, 4, 2, 3}, {3, 4, 1, 5, 2}}, {-1, {3, 4, 2, 1, 5}, {4, 1, 5, 2, 3}},
 {-1, {3, 4, 2, 5, 1}, {3, 1, 4, 5, 2}}, {-1, {3, 5, 1, 4, 2}, {5, 1, 2, 3, 4}},
 {-1, {4, 1, 5, 3, 2}, {1, 4, 5, 2, 3}}, {-1, {4, 2, 5, 1, 3}, {2, 3, 4, 5, 1}},
 {-1, {4, 2, 5, 3, 1}, {1, 5, 2, 3, 4}}, {-1, {4, 3, 1, 5, 2}, {3, 4, 1, 2, 5}},
 {-1, {5, 1, 4, 2, 3}, {4, 1, 2, 5, 3}}, {-1, {5, 3, 1, 4, 2}, {2, 3, 4, 1, 5}}}
```

Output contains first the left canonical form of the given braid word W as a sequence of an integer representing the infimum and three permutations representing the remaining non-maximal permutation braid factors. Likewise the resulting set from `UltraSummitSetLCF`-function contains braids as sequences of an integer and two permutations.

Using the properties of Mathematica and typing

```
A = ConstructBraidFromLCF[lcf, 5]
ToMCF[A]
ToRCF[A]
UltraSummitSet[A]
```

we get the following output:

$\Delta_5^{-2}\,(\sigma_1\,\sigma_2\,\sigma_1\,\sigma_3\,\sigma_2\,\sigma_4\,\sigma_3\,\sigma_2\,\sigma_1)\,(\sigma_3\,\sigma_2\,\sigma_4\,\sigma_3\,\sigma_2)\,(\sigma_2\,\sigma_1\,\sigma_3\,\sigma_2\,\sigma_4\,\sigma_3)$

$((\sigma_1\,\sigma_2\,\sigma_3\,\sigma_4\,\sigma_3)\,(\sigma_3))^{-1}\,(\sigma_2\,\sigma_1\,\sigma_3\,\sigma_2\,\sigma_4\,\sigma_3)$

$(\sigma_1\,\sigma_2\,\sigma_3)\,(\sigma_1\,\sigma_2\,\sigma_3\,\sigma_2\,\sigma_1\,\sigma_4\,\sigma_3\,\sigma_2)\,(\sigma_1\,\sigma_2\,\sigma_1\,\sigma_3\,\sigma_2\,\sigma_4\,\sigma_3\,\sigma_2\,\sigma_1)\,\Delta_5^{-2}$

$\{\Delta_5^{-1}\,(\sigma_3\,\sigma_2\,\sigma_4\,\sigma_3\,\sigma_2)\,(\sigma_2\,\sigma_3\,\sigma_1\,\sigma_4\,\sigma_2),\ \Delta_5^{-1}\,(\sigma_1\,\sigma_2\,\sigma_1\,\sigma_3\,\sigma_2)\,(\sigma_3\,\sigma_2\,\sigma_4\,\sigma_3\,\sigma_1),$
$\Delta_5^{-1}\,(\sigma_1\,\sigma_2\,\sigma_3\,\sigma_1\,\sigma_4\,\sigma_2)\,(\sigma_2\,\sigma_3\,\sigma_4\,\sigma_1),\ \Delta_5^{-1}\,(\sigma_2\,\sigma_3\,\sigma_1\,\sigma_4\,\sigma_3\,\sigma_2)\,(\sigma_4\,\sigma_3\,\sigma_2\,\sigma_1),$
$\Delta_5^{-1}\,(\sigma_2\,\sigma_3\,\sigma_2\,\sigma_4\,\sigma_3\,\sigma_1)\,(\sigma_3\,\sigma_2\,\sigma_4\,\sigma_3),\ \Delta_5^{-1}\,(\sigma_1\,\sigma_3\,\sigma_2\,\sigma_4\,\sigma_3\,\sigma_1)\,(\sigma_1\,\sigma_2\,\sigma_3\,\sigma_4),$
$\Delta_5^{-1}\,(\sigma_1\,\sigma_2\,\sigma_3\,\sigma_2\,\sigma_4\,\sigma_3\,\sigma_1)\,(\sigma_4\,\sigma_3\,\sigma_2),\ \Delta_5^{-1}\,(\sigma_2\,\sigma_3\,\sigma_1\,\sigma_4\,\sigma_2\,\sigma_1)\,(\sigma_2\,\sigma_1\,\sigma_3\,\sigma_2),$
$\Delta_5^{-1}\,(\sigma_3\,\sigma_2\,\sigma_4\,\sigma_3\,\sigma_2\,\sigma_1)\,(\sigma_3\,\sigma_4\,\sigma_2\,\sigma_1),\ \Delta_5^{-1}\,(\sigma_2\,\sigma_3\,\sigma_1\,\sigma_4\,\sigma_3\,\sigma_2\,\sigma_1)\,(\sigma_1\,\sigma_2\,\sigma_3)\}$

The difference between the first and the second input was that at first we operated with standard list expressions representing left canonical forms. At the second input we first constructed an `ArtinBraid`-expression and then applied algorithms to that expression.

The Burau representation

The following lines show how the Burau matrix and its preimage can be calculated.

```
n = 4;
W = {-3, 2, 1, -3, 2, 2, 1, 1, 1, -2, -3, 2, 3};
bm = Burau[W, n, t];
MatrixForm[bm]
W2 = InvertBurauH[bm, t, 20];
WordToLCF[W, n]
WordToLCF[W2, n]
```

$$\begin{pmatrix} 1 - t + t^3 - t^4 + t^5 & t - 2t^2 + t^3 & 2t^2 - 3t^3 + 2t^4 - t^5 & t^3 - t^4 \\ 1 - 2t + 2t^2 - 2t^3 + t^4 & 1 & -1 + 2t - 2t^2 + 2t^3 - t^4 & 0 \\ -2 + \frac{1}{t} + 2t - t^2 & -1 - \frac{1}{t^2} + \frac{2}{t} & 4 + \frac{1}{t^2} - \frac{3}{t} - 3t + t^2 & t \\ -5 - \frac{1}{t^2} + \frac{4}{t} + 4t - 2t^2 & -1 + \frac{1}{t^3} - \frac{3}{t^2} + \frac{3}{t} & 8 - \frac{1}{t^3} + \frac{4}{t^2} - \frac{7}{t} - 5t + 2t^2 & -1 + t \end{pmatrix}$$

```
{-3, {4, 3, 1, 2}, {3, 2, 4, 1}, {2, 4, 3, 1},
 {3, 1, 4, 2}, {2, 4, 3, 1}, {1, 3, 2, 4}, {3, 1, 2, 4}}

{-3, {4, 3, 1, 2}, {3, 2, 4, 1}, {2, 4, 3, 1},
 {3, 1, 4, 2}, {2, 4, 3, 1}, {1, 3, 2, 4}, {3, 1, 2, 4}}
```

The Lawrence-Krammer representation

The next lines show the input and the output for calculating and inverting a Krammer matrix. The preimage is returned directly in its left canonical form.

```
n = 4;
W = {1, -2, -3};
WordToLCF[W, n]
km = Krammer[W, n, t, q];
MatrixForm[km]
InvertKrammer[km, t, q]
{-1, {2, 1, 4, 3}, {4, 1, 2, 3}}
```

$$\begin{pmatrix} -q t + q^2 t & -q t + q^2 t & \frac{t}{q} - q t + q^2 t & 0 & 0 & 0 \\ 1 - q & 0 & -1 - \frac{1}{q^2} + \frac{2}{q} & 0 & \frac{1}{q^3 t} & \frac{1}{q^4 t} - \frac{1}{q^3 t} \\ 0 & 1 - q & 2 - \frac{1}{q} - q & 0 & 0 & \frac{1}{q^3 t} \\ 1 - q + q^2 & 0 & -2 - \frac{1}{q^2} + \frac{2}{q} + q & 0 & \frac{1}{q^3 t} - \frac{1}{q^2 t} & \frac{1}{q^4 t} - \frac{2}{q^3 t} + \frac{1}{q^2 t} \\ 0 & 1 - q + q^2 & 2 - \frac{1}{q} - 2q + q^2 & 0 & 0 & \frac{1}{q^3 t} - \frac{1}{q^2 t} \\ 0 & 0 & 0 & 1 & 1 - \frac{1}{q} & 0 \end{pmatrix}$$

```
{-1, {2, 1, 4, 3}, {4, 1, 2, 3}}
```

`BraidDiagramPlot`

By typing the following lines, we get figure 6:

```
W = {1, 2, 1, -3, -3, 1};
g = BraidDiagramPlot[W, 4, {Hue[0.2], Hue[0.4], Hue[0.6], Hue[0.8]}];
Show[g, PlotRange → All, Prolog → {AbsoluteThickness[2]}];
```

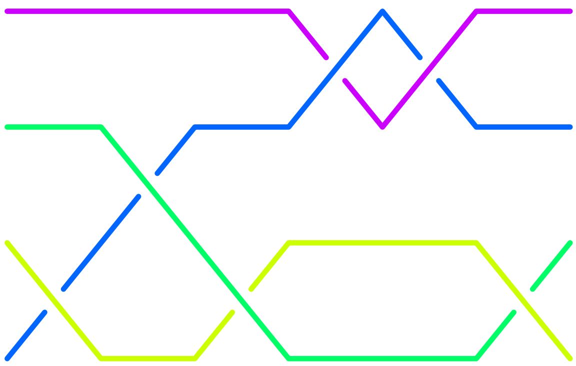

Figure 6: `BraidDiagramPlot`-picture for list {1,2,1,-3,-3,1}.

`BraidDiagramPlot3D`

By typing the following lines, we get a similar figure as with `BraidDiagramPlot`:

```
W = {1, 2, 1, -3, -3, 1};
g = BraidDiagramPlot3D [W, 4,
   {Hue[0.2], Hue[0.4], Hue[0.6], Hue[0.8]}, 0.1, Edges → Null];
Show[g, PlotRange → All, ViewPoint → {0.000, 0.000, 4.880}, Boxed → False];
```

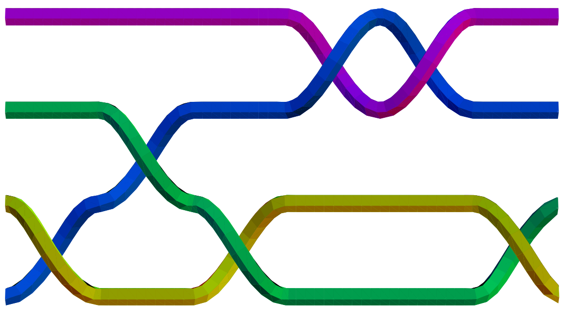

Figure 7: `BraidDiagramPlot3D`-picture for list {1,2,1,-3,-3,1}.

A final word

About the package This package is far from being finished. There are many topics (better inversion algorithm for Burau representation, braid classification, root extraction, Gassner representation, Birman-Wenzl and Iwahori-Hecke algebras, knot invariants, etc.) in braid theory that have not (yet) been implemented in this package.

Acknowledgements I would like to thank professor J. Kari and assistant professor A. Renvall for useful discussions and advices.

References

[1] I. Anshel, M. Anshel, B. Fisher, and D. Golfeld. New key agreement protocols in braid group cryptography. In D. Naccache, editor, *Topics in Cryptology*, volume 2020 of *LNCS*, pages 13–27. Springer-Verlag, 2001. 7, 12

[2] E. Artin. Theory of braids. *Annals of Mathematics*, 48(1):101–126, 1947. 1, 3

[3] S. Bachmuth. Braid groups are linear groups. *Advances in Mathematics*, 121:50–61, 1996. 3

[4] S. J. Bigelow. The Burau representation of the braid group B_n is not faithful for $n = 5$,. *Geometry and Topology*, 3:397–404, 1999. 7

[5] S. J. Bigelow. Braid groups are linear. *Journal of American Mathematical Society*, 14(2):471–486, 2001. 3, 8

[6] J. S. Birman. *Braids, Links and Mapping Class Groups*, volume 82 of *Annals of Mathematics Studies*. Princeton University Press, 1975. Errata: `http://www.math.columbia.edu/~jb/`. 0, 6, 7, 12

[7] J. S. Birman, K. H. Ko, and S. J. Lee. A new approach to the word and conjugacy problems in the braid groups. *Advances in Mathematics*, 139:322–353, 1998. 3, 6

[8] J. S. Birman, K. H. Ko, and S. J. Lee. The infimum, supremum and geodesic length of a braid conjugacy class. *Advances in Mathematics*, 164:41–56, 2001. `http://www.arxiv.org/math.GT/0003125`. 6

[9] F. Bohnenblust. The algebraical braid group. *Annals of Mathematics*, 48(1):126–136, 1947. 1

[10] J. C. Cha. CBraid: a C++ library for computation in braid groups. `http://knot.kaist.ac.kr/~jccha/cbraid/`, 2003. 0

[11] J. C. Cha, K. H. Ko, S. J. Lee, J. W. Han, and J. H. Cheon. An efficient implementation of braid groups. In C. Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 144–156. Springer-Verlag, 2001. 8, 11

[12] J. H. Cheon and B. Jun. A polynomial time algorithm for the braid Diffie-Hellman conjugacy problem. In D. Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 212–225. Springer-Verlag, 2003. 8, 13

[13] W. Chow. On the algebraical braid group. *Annals of Mathematics*, 49(2):654–658, 1948. 1

[14] P. Dehorney. Braid groups and left distributive operations. *Transactions of the American mathematical society*, 345:115–151, 1994. 2, 3

[15] P. Dehorney. A fast method for comparing braids. *Advances in Mathematics*, 125:200–235, 1997. 3

[16] P. Dehorney. The group of fractions of a torsion free lcm monoid is torsion free. *Journal of Algebra*, 281:303–305, 2004. (formerly a preprint with title 'Addendum to "Gaussian groups are torsion free"'). 2

[17] J. L. Dyer. The algebraic braid groups are torsion-free: an algebraic proof. *Mathematische Zeitschrift*, 172:157–160, 1980. 2

[18] E. A. Elrifai and H. R. Morton. Algorithms for positive braids. *Quart. J. Math. Oxford*, 45:479–497, 1994. 3, 4, 5, 6, 8, 9

[19] D. B. A. Epstein, J. W. Cannon, D. F. Holt, S. V. F. Levy, M. S. Paterson, and W. P. Thurston. *Word Processing in Groups*. Jones and Bartlett publishers, 1992. 2, 3, 4, 5, 8, 9, 10, 11

[20] R. Fox and L. Neuwirth. The braid groups. *Mathematica Scandinavica*, 10:119–126, 1962. 1, 2

[21] N. Franco and J. González-Meneses. Conjugacy problem for braid groups and garside groups. *Journal of Algebra*, 266:112–132, 2003. `http://www.arxiv.org/math.GT/0112310/`. 6, 8, 12

[22] D. Garber, S. Kaplan, and M. Teicher. A new algorithm for solving the word problem in braid groups. *Advances in Mathematics*, 167:142–159, 2002. 3

[23] F. A. Garside. The braid groups and other groups. *Quart. J. Math. Oxford*, 20:235–254, 1969. 3, 6

[24] V. Gebhardt. A new approach to the conjugacy problem in garside groups. 2003. `http://www.arxiv.org/math.GT/0306199/`. 6, 7, 8, 12

[25] T. Harju and J. Karhumäki. Morphisms. In G. Rozenberg and A. Salomaa, editors, *Word, Language, Grammar*, volume 1 of *Handbook of Formal Languages*, chapter 7, pages 439–510. Springer-Verlag, 1997. 3

[26] S.-T. Hu. *Homotopy Theory*, volume 8 of *Pure and applied mathematics*. Academic Press, 1959. 0

[27] J. Hughes. A linear algebraic attack on the AAFG1 braid group cryptosystem. In *ACISP 2002*, volume 2384 of *LNCS*, pages 176–189. Springer-Verlag, 2002. 7, 12

[28] K. H. Ko, S. J. Lee, J. H. Cheon, J. W. Han, J.-S. Kang, and C. Park. New public-key cryptosystem using braid groups. In *Proceedings of Crypto 2000*, volume 1880 of *LNCS*, pages 166–183. Springer-Verlag, 2000. 8

[29] D. Krammer. Braid groups are linear. *Annals of Mathematics*, 155:131–156, 2002. 3, 8, 9, 10, 13

[30] S. J. Lee and E. Lee. Potential weaknesses of the commutator key agreement protocol. In *Eurocrypt 2002*, volume 2332 of *LNCS*, pages 14–28. Springer-Verlag, 2002. 7

[31] V. Lukkarila. Palmikkoryhmistä kryptografiassa / On braid groups in cryptography. Master's thesis, University of Turku, 2004. (in Finnish only). 0, 7, 13

[32] W. Magnus, A. Karrass, and D. Solitar. *Combinatorial Group Theory*, volume 13 of *Pure and applied mathematics*. John Wiley & Sons, Inc., 1966. 0, 1

[33] H. R. Morton. The multivariable Alexander polynomial for a closed braid. In Funchal, editor, *Low-dimensional topology*, volume 233 of *Contemp. Math.*, pages 167–172. Amer. Math. Soc., 1999. `http://www.arxiv.org/math.GT/9803138/`. 7, 9, 12

[34] M. S. Paterson and A. A. Razborov. The set of minimal braids is co-NP-complete. *Journal of Algorithms*, 12:393–408, 1991. 3

[35] B. Wiest. An algorithm for the word problem in braid groups. 2002. `http://www.arxiv.org/math.GT/0211169/`. 3

Index

TURKU

CENTRE *for*

COMPUTER

SCIENCE

Lemminkäisenkatu 14 A, 20520 Turku, Finland | www.tucs.fi

University of Turku

- Department of Information Technology
- Department of Mathematical Sciences

Åbo Akademi University

- Department of Computer Science
- Institute for Advanced Management Systems Research

Turku School of Economics and Business Administration

- Institute of Information Systems Sciences

ISBN 952-12-1555-0
ISSN 1239-1891