Introduction to the CubicBraidGroup class

Sebastian Oehms

18.3.2017

Contents

1 Using the CubicBraidGroup class

1.1 Introduction

This module is devoted to factor groups of the Artin braid groups, such that the images s_i of the braid generators have order three:

$$s_i^3 = 1$$

In general these groups have firstly been investigated by Coxeter, H.S.M in: "Factor groups of the braid groups, Proceedings of the Fourth Candian Mathematical Congress (Vancover 1957), pp. 95-122".

Coxeter showed, that these groups are finite as long as the number of strands is less than 6 and infinite elsewise. More explicitely the factor group on three strand braids is isomorphic to $SL(2,3)$, on four strand braids to $GU(3,2)$ and on five strand braids to $Sp(4,3) \times C_3$. Coxeter realized these groups as subgroups of unitary groups with respect to a certain hermitian form over the complex numbers (in fact over $\mathbb{Q}$ adjoined with a primitive 12-th root of unity).

In "Einige endliche Faktorgruppen der Zopfgruppen" (Math. Z., 163 (1978), 291-302) J. Assion considered two series $S(m)$ and $U(m)$ of finite dimensional factors of these groups. The additional relations on the braid group generators $\{s_1, \cdots, s_{m-1}\}$ are

$$\begin{aligned} s_3 s_1 t_2 s_1 t_2^{-1} t_3 t_2 s_1 t_2^{-1} t_3^{-1} = 1 \quad \text{for} \quad m >= 5 \quad \text{in case of} \quad S(m) \\ t_1 t_3 = 1 \quad \text{for} \quad m >= 5 \quad \text{in case of} \quad U(m) \end{aligned}$$

where $t_i = (s_i s_{i+1})^3$. He showed that each series of finite cubic braid group factors must be an epimorhic image of one of his two series, as long as the groups with less than 5 strands are the full cubic braid groups, whereas the group on 5 strands is not. He realized the groups $S(m)$ as symplectic groups over $GF(3)$ (resp. subgroups therein) and $U(m)$ as general unitary groups over $GF(4)$ (resp. subgroups therein).

This class implements all the groups conidered by Coxeter and Assion as finitely presented groups (via the gap interface) together with the classical realizations given by the authors. It also contains the coercion maps between the two ways of realization. In addition the user can construct other realizations and maps to matrix groups with help of the burau representation. In case gap3 and CHEVIE are installed under sage version 7.2 (or later) the reflection groups via the gap3 interface are availlable, too. The methods for all this functionality are:

- as_classical_group
- as_matrix_group
- as_reflection_group (needs sage version 7.2 up and gap3 + CHEVIE)
- as_permutation_group

Further methods are:

- order
- pre_image_braid_group
- cubic_braid_subgroup
- character_table

AUTHOR
Sebastian Oehms, Sept. 2016

1.2 Getting started

The CubicBraidGroup class is not integrated into the sage library, right now. If you are using sage on your own computer you may install the CubicBraidGoup class following the README.txt file To use the CubicBraidGroup on this plattform you can create a jupyter notebook or a sage worksheet (like this) and start with a cell containing the first line of this:

```
%auto
from cubic_braid import *
```

To execute the command you must click on the "play" symbol (>|) above (instead of carriage return key in a shell sessiion)

On your own computer in a command-line sage-session you must start with this line, as well. For further information on how to get started see the README.txt-File.

The general documentation concerning the CubicBraidGroup class can be shown using the following command.

```
print CubicBraidGroup_class.___doc___
```

Class to handel cubic factors of braid group on n strands

If you don't see this well formated type

sage: print CubicBraidGroup_class.___doc___

This class implements quotient groups of the braid group mapping their generators to elements of order 3
(see the module header for more informations on these groups)

These groups are implemented as a particular case of finitely presented groups similar to the BraidGroup class

A cubic braid group can be created by giving the number of strands, and the name of the generators in a similar
way as it works for the BraidGroup class.

INPUT (to the constructor):

- "names": (see the BraidGroup_class docomentation)

- "AdditionalRelation": (keyword, explantaion see below)

- "verbose": (keyword, explantaion see below)

RAISE (on init):

- ValueError: "the number of strands must be an integer larger than one"

Setting the keyword 'AdditionalRelation' to one on the values 'S' or 'U' the additional relations due to
Assion are added:

'S': $s_3 s_1 t_2 s_1 t_2^{-1} t_3 t_2 s_1 t_2^{-1} t_3^{-1) = 1$ for $m >= 5$
'U': $t_1 t_3 = 1$ for $m >= 5$

where $t_i = (s_i s_{i+1})^3$. If AdditionalRelation='C' (default) only the cubic relation on the generators is
active (Coxeters case of investigation). Note that for n=2,3,4 the groups do not differ between the three possible
values of AdditionalRelation (as finitely presented groups). But anyway, the classes for 'C', 'S' and 'U' are
different, since they have different classical realizations implemented .

Setting the keyword verbose it is possible to print time stamp messages in order to do a performance or
call stack debugging. This keyword uses the timeStampControl class. For more information on this type

```
print setupTimeStamp.___doc___
print timeStampControl.___doc___
print timeStampControl.print_timestamp.___doc___
print print_time_tb.___doc___
```

The creation of instances of this class can also be done more easy by help of the functions CubicBraidGroup,
AssionGroupS and AssionGroupU (similar to the function BraidGroup with respect to the BraidGroup_class)

EXAMPLES:

```
sage: from cubic_braid import *
sage: U3 = CubicBraidGroup(3, AdditionalRelation = 'U'); U3
Assion group on 3 strands of type U
sage: U3.gens()
(c0, c1)
```

alternate possibilities defining U3:

```
sage: U3 = AssionGroupU(3); U3
Assion group on 3 strands of type U
sage: U3.gens()
(u0, u1)

sage: U3.<u1,u2> = AssionGroupU(3); U3
Assion group on 3 strands of type U
sage: U3.gens()
(u1, u2)
```

alternates naming the generators:

```
sage: U3 = AssionGroupU(3, 'a, b'); U3
Assion group on 3 strands of type U
sage: U3.gens()
(a, b)

sage: C3 = CubicBraidGroup(3, 't'); C3
Cubic Braid group on 3 strands
sage: C3.gens()
(t0, t1)

sage: U3.is_isomorphic( C3 )
True
sage: U3.as_classical_group()
Subgroup of (The projective general unitary group of degree 3 over Finite Field of
size 2) generated by
[(1,7,6) (3,19,14)(4,15,10)(5,11,18)(12,16,20),
(1,12,13)(2,15,19)(4,9,14)(5,18,8)(6,21,16)]
sage: C3.as_classical_group()
Subgroup of General Unitary Group of degree 2 over Universal Cyclotomic Field with
respect to hermitian form
[-E(12)^7 + E(12)^11                  -1]
[                 -1 -E(12)^7 + E(12)^11]
generated by (
[    E(3) E(12)^11]
[       0        1],
[       1        0]
[E(12)^11     E(3)])
```

using verbose mode:

```
sage: C3.<c1,c2> = CubicBraidGroup(3, verbose=True); C3
L: StackInfo    Elapse:     0, Total:   0 Ident: C3 In: ___init___ Line: 403
L: StackInfo    Elapse:   197, Total:   0 Begin In:
___create_classical_realization___ Line: 910
L: StackInfo    Elapse:     8, Total:   0 Begin In: as_matrix_group Line: 1031
L: Body         Elapse:    92, Total:   0 genList prepared In: as_matrix_group
Line: 1038
L: Body         Elapse:    52, Total:   0 MatGroup defined In: as_matrix_group
Line: 1054
L: StackInfo    Elapse:   356, Total:   0 Begin In: gap_hom Line: 143
L: Debug        Elapse:     8, Total:   0 GroupHomomorphis In: gap_hom Line: 188
L: Debug        Elapse:    17, Total:   0 checked if groupMap works In: gap_hom
Line: 199
L: StackInfo    Elapse:     8, Total:   0 End In: gap_hom Line: 226
L: StackInfo    Elapse:    44, Total:   0 Begin In: ___check_homomorphism___ Line:
60
```

L: StackInfo Elapse: 52, Total: 0 End In: ___check_homomorphism___ Line: 625
L: Body Elapse: 3, Total: 0 Hom from self defined In: as_matrix_group Line: 1064
L: StackInfo Elapse: 62, Total: 0 Begin In: gap_hom Line: 143
L: Debug Elapse: 4, Total: 0 GroupHomomorphis In: gap_hom Line: 188
L: Debug Elapse: 20, Total: 0 checked if groupMap works In: gap_hom Line: 199
L: StackInfo Elapse: 4, Total: 0 End In: gap_hom Line: 226
L: Body Elapse: 35, Total: 0 Section to self defined In: as_matrix_group Line: 1077
L: StackInfo Elapse: 3, Total: 0 Ende In: as_matrix_group Line: 1102
L: StackInfo Elapse: 3, Total: 0 End In: ___create_classical_realization___ Line: 975
L: StackInfo Elapse: 12, Total: 0 Ident: C3 In: ___init___ Line: 472
Cubic Braid group on 3 strands
sage:
sage:
sage: C3.<c1,c2> = CubicBraidGroup(3, verbose=30); C3
L: StackInfo Elapse: 62, Total: 0 (truncated 1) Begin In: ___create_classical_realization___ Line: 910
---> ___init___ In: lib/cubic_braid.py Line: 471
---> ___classcall___ In: /opt/sage/sage-6.9-i686-Linux/local/lib/python2.7/site-packages/sage/structure
/unique_representation.py Line: 1021
---> CubicBraidGroup In: lib/cubic_braid.py Line: 1231
L: StackInfo Elapse: 405, Total: 0 Begin In: as_matrix_group Line: 1031
---> ___create_classical_realization___ In: lib/cubic_braid.py Line: 968
---> ___init___ In: lib/cubic_braid.py Line: 471
---> ___classcall___ In: /opt/sage/sage-6.9-i686-Linux/local/lib/python2.7/site-packages/sage/structure
/unique_representation.py Line: 1021
L: Body Elapse: 145, Total: 0 genList prepared In: as_matrix_group Line: 1038
---> ___create_classical_realization___ In: lib/cubic_braid.py Line: 968
---> ___init___ In: lib/cubic_braid.py Line: 471
---> ___classcall___ In: /opt/sage/sage-6.9-i686-Linux/local/lib/python2.7/site-packages/sage/structure
/unique_representation.py Line: 1021
Cubic Braid group on 3 strands
sage:

TESTS:

sage: C4 = CubicBraidGroup(4)
sage: TestSuite(C4).run()
sage: C5 = CubicBraidGroup(5)

```
sage: TestSuite(C5).run()
sage: C6 = CubicBraidGroup(6)
sage: TestSuite(C6).run()
sage: S3 = AssionGroupS(3)
sage: TestSuite(S3).run()
sage: S4 = AssionGroupS(4)
sage: TestSuite(S5).run()
sage: U3 = AssionGroupU(3)
sage: TestSuite(U3).run()
sage: U4 = AssionGroupU(4)
sage: TestSuite(U4).run()
sage: U5 = AssionGroupU(5)
sage: TestSuite(U5).run()
```

METHODS (implemented / overwriten here):

- as_classical_group(): type
  sage: print CubicBraidGroup_class.as_classical_group.___doc___
- as_matrix_group(): type
  sage: print CubicBraidGroup_class.as_matrix_group.___doc___
- as_refection_group(): type
  sage: print CubicBraidGroup_class.as_refection_group.___doc___
- as_permutation_group(): type
  sage: print CubicBraidGroup_class.as_permutation_group.___doc___
- pre_image_braid_group: type
  sage: print CubicBraidGroup_class.pre_image_braid_group.___doc___
- cubic_braid_subgroup(): type
  sage: print CubicBraidGroup_class.cubic_braid_subgroup.___doc___
- centralizing_element(): type
  sage: print CubicBraidGroup_class.centralizing_element.___doc___
- order(): type
  sage: print CubicBraidGroup_class.order.___doc___
- character_table(): type
  sage: print CubicBraidGroup_class.character_table.___doc___

AUTHOR

- Sebastian Oehms, Sept. 2016

REFERENCES:

- Coxeter, H.S.M: “Factor groups of the braid groups, Proceedings of the Fourth Candian Mathematical Congress
(Vancover 1957), pp. 95-122”.

- J. Assion: “Einige endliche Faktorgruppen der Zopfgruppen” (Math. Z., 163 (1978), 291-302)

1.3 First steps

To definine the cubic braid group on 3 strand, for example, type:

```
C3 = CubicBraidGroup( 3 ); C3
```
Cubic Braid group on 3 strands

There are several ways to obtain the braid generators as variables: First using the pre defined names:

```
C3.inject_variables()
```
Defining c0, c1

Second, if you like to use your own names it is possiblbe to include a generator declaration inside the group declaration in two ways (caution: sage-kernel needed, this does no work for a pure python kernel):

```
C4.<c1, c2, c3> = CubicBraidGroup(4); print "Element of C4", c1*c2\
    **2*c3
U4 = AssionGroupU(4, 'a, b, c' ); a, b, c = U4.gens(); print "\
    Element of U4", a*b**2*c
```
Element of C4
c1*c2^2*c3
Element of U4
a*b^2*c

But, note:

```
S4 = AssionGroupS(4); x, y, z = S4.gens(); print "Element of S4", x*\
    y**2*z
```
Element of S4
s0*s1^2*s2

1.4 Classical realization

Now, lets define Coxeter's realization of the 3 strand cubic braid group:

```
C3Cl = C3.as_classical_group(); C3Cl
```
Subgroup of General Unitary Group of degree 2 over Universal Cyclotomic Field with respect to hermitian form:
[-E(12)^7 + E(12)^11 -1]
[-1 -E(12)^7 + E(12)^11] generated by:
([E(3) E(12)^11]
[0 1], [1 0]

[E(12)^11 E(3)])

You may apply any method implemented for groups and finitely presented resp. finitely generated matrix groups in sage to C3 resp. C3Cl, for instance to check if they are isomorphic:

```
C3.is_isomorphic( C3Cl )
```
True

To see which methods are available say:

```
print 'Attributes of C3:\n', dir(C3), '\n\n'
print 'Attributes of C3Cl:\n', dir(C3Cl), '\n\n'
```
Attributes of C3:
['CartesianProduct', 'Element', 'Hom', '_AdditionalRelation_', '_TimeStampSub_',
'_TimeStamp_', '__cached_methods', '__call__', '__check_homomorphism__', '__class__',
'__classcall__', '__contains__', '__copy__', '__create_classical_realization__',
'__deepcopy__', '__delattr__', '__dict__', '__dir__', '__div__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__', '__getstate__', '__gt__',
'__hash__', '__init__', '__init_extra__', '__iter__', '__le__', '__len__', '__lt__',
'__make_element_class__', '__metaclass__', '__module__', '__mul__', '__ne__', '__new__
'__nonzero__', '__print_timestamp__', '__pyx_vtable__', '__qualname__', '__rdiv__',
'__reduce__', '__reduce_ex__', '__repr__', '__rmul__', '__setattr__', '__setstate__',
'__sizeof__', '__str__', '__subclasshook__', '__temporarily_change_names', '__weakref__',
'_abstract_element_class', '_ambient', '_an_element_', '_ascii_art_', '_assign_names',
'_axiom_', '_axiom_init_', '_b_', '_base', '_bi_', '_cache_an_element', '_cache_key',
'_centralizing_element_', '_centralizing_matrix_', '_checked_homomorphism_',
'_classical_base_group_', '_classical_base_group_gens_', '_classical_embedding_',
'_classical_group_', '_classical_group_gens_', '_clear_cache_', '_cmp_',
'_coerce_map_from_', '_coerce_map_via', '_coercions_used', '_convert_map_from_',
'_convert_method_name', '_defining_names', '_doccls', '_element_constructor',
'_element_constructor_', '_element_constructor_from_element_class',
'_element_init_pass_parent', '_factory_data', '_first_ngens', '_free_group',
'_free_group_', '_fricas_', '_fricas_init_', '_gap_', '_gap_gens', '_gap_init_',
'_generic_convert_map', '_get_action_', '_giac_', '_giac_init_', '_gp_', '_gp_init_',
'_ident_', '_included_private_doc_', '_init_category_', '_initial_action_list',
'_initial_coerce_list', '_initial_convert_list', '_interface_', '_interface_init_',
'_interface_is_cached_', '_internal_coerce_map_from', '_internal_convert_map_from',
'_introspect_coerce', '_is_category_initialized', '_is_valid_homomorphism_', '_kash_',
'_kash_init_', '_latex_', '_libgap', '_libgap_', '_macaulay2_', '_macaulay2_init_',
'_magma_init_', '_maple_', '_maple_init_', '_mathematica_', '_mathematica_init_',
'_maxima_', '_maxima_init_', '_maxima_lib_', '_maxima_lib_init_', '_names', '_nstrands_',
'_octave_', '_octave_init_', '_pari_', '_pari_init_', '_populate_coercion_lists_',
'_pre_image_braid_group_', '_r_init_', '_reduction', '_refine_category_', '_relations',
'_repr_', '_repr_option', '_richcmp', '_sage_', '_set_element_constructor', '_singular_',
'_singular_init_', '_subgroup_constructor', '_t_', '_test_an_element',
'_test_associativity', '_test_cardinality', '_test_category', '_test_constructions',
'_test_elements', '_test_elements_eq_reflexive', '_test_elements_eq_symmetric',

'_test_elements_eq_transitive', '_test_elements_neq', '_test_eq', '_test_inverse',
'_test_new', '_test_not_implemented_methods', '_test_one', '_test_pickling', '_test_prod',
'_test_some_elements', '_tester', '_ti_', '_unicode_art_', '_unset_category',
'_unset_coercions_used', '_unset_embedding', '_verbose_', 'abelian_invariants',
'alexander_matrix', 'algebra', 'ambient', 'an_element', 'as_classical_group',
'as_matrix_group', 'as_permutation_group', 'as_reflection_group', 'base', 'base_ring',
'cardinality', 'cartesian_product', 'categories', 'category', 'cayley_graph',
'cayley_table', 'center', 'centralizing_element', 'change_debug_options',
'character_table', 'class_function', 'coerce', 'coerce_embedding', 'coerce_map_from',
'conjugacy_class', 'conjugacy_class_representatives', 'conjugacy_classes', 'construction',
'convert_map_from', 'cubic_braid_subgroup', 'db', 'direct_product', 'dump', 'dumps',
'element_class', 'epimorphisms', 'free_group', 'gap', 'gen', 'generators', 'gens',
'gens_dict', 'gens_dict_recursive', 'get_action', 'group_generators', 'has_base',
'has_coerce_map_from', 'holomorph', 'hom', 'inject_variables', 'intersection',
'irreducible_characters', 'is_abelian', 'is_atomic_repr', 'is_coercion_cached',
'is_commutative', 'is_conversion_cached', 'is_empty', 'is_exact', 'is_finite',
'is_isomorphic', 'is_multiplicative', 'is_parent_of', 'is_subgroup', 'latex_name',
'latex_variable_names', 'list', 'magma_generators', 'monoid_generators',
'multiplication_table', 'ngens', 'normalize_names', 'objgen', 'objgens', 'one',
'one_element', 'order', 'parent', 'pre_image_braid_group', 'prod', 'product',
'product_from_element_class_mul', 'quotient', 'random_element', 'register_action',
'register_coercion', 'register_conversion', 'register_embedding',
'regular_representation', 'relations', 'rename', 'reset_name', 'rewriting_system', 'save',
'semidirect_product', 'semigroup_generators', 'simplification_isomorphism', 'simplified',
'some_elements', 'structure_description', 'subgroup', 'submonoid', 'subsemigroup',
'trivial_representation', 'variable_name', 'variable_names']

Attributes of C3Cl:
['CartesianProduct', 'Element', 'Hom', '_Hom_', '___cached_methods', '___call___',
'___class___', '___cmp___', '___contains___', '___delattr___', '___dict___', '___dir___', '___div___',
'___doc___', '___format___', '___getattribute___', '___getitem___', '___getstate___', '___hash___',
'___init___', '___init_extra___', '___iter___', '___len___', '___make_element_class___',
'___module___', '___mul___', '___new___', '___nonzero___', '___pyx_vtable___', '___qualname___',
'___rdiv___', '___reduce___', '___reduce_ex___', '___repr___', '___rmul___', '___setattr___',
'___setstate___', '___sizeof___', '___str___', '___subclasshook___', '___temporarily_change_names',
'___weakref___', '_abstract_element_class', '_ambient', '_an_element_', '_ascii_art_',
'_assign_names', '_axiom_', '_axiom_init_', '_base', '_cache_an_element', '_cache_key',
'_check_matrix', '_cmp_', '_cmp_generators', '_coerce_map_from_', '_coerce_map_via',
'_coercions_used', '_convert_map_from_', '_convert_method_name', '_defining_names',
'_deg', '_doccls', '_element_constructor', '_element_constructor_',
'_element_constructor_from_element_class', '_element_init_pass_parent', '_factory_data',
'_first_ngens', '_fricas_', '_fricas_init_', '_gap_', '_gap_gens', '_gap_init_',
'_generic_convert_map', '_get_action_', '_giac_', '_giac_init_', '_gp_', '_gp_init_',
'_init_category_', '_initial_action_list', '_initial_coerce_list',
'_initial_convert_list', '_interface_', '_interface_init_', '_interface_is_cached_',
'_internal_coerce_map_from', '_internal_convert_map_from', '_introspect_coerce',
'_is_category_initialized', '_is_valid_homomorphism_', '_kash_', '_kash_init_', '_latex_',

'_libgap', '_libgap_', '_macaulay2_', '_macaulay2_init_', '_magma_init_', '_maple_', '_maple_init_', '_mathematica_', '_mathematica_init_', '_maxima_', '_maxima_init_', '_maxima_lib_', '_maxima_lib_init_', '_names', '_octave_', '_octave_init_', '_pari_', '_pari_init_', '_populate_coercion_lists_', '_r_init_', '_reduction', '_refine_category_', '_repr_', '_repr_option', '_richcmp', '_sage_', '_set_element_constructor', '_singular_', '_singular_init_', '_subgroup_constructor', '_test_an_element', '_test_associativity', '_test_cardinality', '_test_category', '_test_elements', '_test_elements_eq_reflexive', '_test_elements_eq_symmetric', '_test_elements_eq_transitive', '_test_elements_neq', '_test_eq', '_test_inverse', '_test_new', '_test_not_implemented_methods', '_test_one', '_test_pickling', '_test_prod', '_test_some_elements', '_tester', '_unicode_art_', '_unset_category', '_unset_coercions_used', '_unset_embedding', 'algebra', 'ambient', 'an_element', 'as_matrix_group', 'as_permutation_group', 'base', 'base_ring', 'cardinality', 'cartesian_product', 'categories', 'category', 'cayley_graph', 'cayley_table', 'center', 'class_function', 'coerce', 'coerce_embedding', 'coerce_map_from', 'conjugacy_class', 'conjugacy_class_representatives', 'conjugacy_classes', 'construction', 'convert_map_from', 'db', 'degree', 'dump', 'dumps', 'element_class', 'gap', 'gen', 'generators', 'gens', 'gens_dict', 'gens_dict_recursive', 'get_action', 'group_generators', 'has_base', 'has_coerce_map_from', 'holomorph', 'hom', 'inject_variables', 'intersection', 'invariant_generators', 'irreducible_characters', 'is_abelian', 'is_atomic_repr', 'is_coercion_cached', 'is_commutative', 'is_conversion_cached', 'is_empty', 'is_exact', 'is_finite', 'is_isomorphic', 'is_multiplicative', 'is_parent_of', 'is_subgroup', 'latex_name', 'latex_variable_names', 'list', 'magma_generators', 'matrix_space', 'module_composition_factors', 'monoid_generators', 'multiplication_table', 'ngens', 'normalize_names', 'objgen', 'objgens', 'one', 'one_element', 'order', 'parent', 'prod', 'product', 'product_from_element_class_mul', 'quotient', 'random_element', 'register_action', 'register_coercion', 'register_conversion', 'register_embedding', 'regular_representation', 'rename', 'reset_name', 'save', 'semidirect_product', 'semigroup_generators', 'some_elements', 'structure_description', 'subgroup', 'submonoid', 'subsemigroup', 'trivial_representation', 'variable_name', 'variable_names']

The above used method "as_classical_group" is a special method of the CubicBraidGroup class. It returns the realization of the cubic braid group as classical group. These are (in principal) symplectic groups over $GF(3)$ resp. unitary groups over $GF(4)$ in the case of Assion groups of type "S" and "U" respectively (realization of Assion) and subgroups of the unitary groups over $\mathbb{Q}$ adjoined with a primitive 12-th root of unity with respect to a certain hermitian form in the case of the ordinary cubic braid groups (Coxeters realization):

```
S3 = AssionGroupS(3); print S3; print ""
S3Cl = S3.as_classical_group(); print S3Cl; print ""
U3 = AssionGroupU(3); print U3; print ""
U3Cl = U3.as_classical_group(); print U3Cl
```

Assion group on 3 strands of type S

Symplectic Group of degree 2 over Finite Field of size 3
Assion group on 3 strands of type U

Subgroup of (The projective general unitary group of degree 3 over Finite Field of size 2) generated by [(1,7,6)(3,19,14)(4,15,10)(5,11,18)(12,16,20),
(1,12,13)(2,15,19)(4,9,14)(5,18,8)(6,21,16)]

1.5 Exceptions in Assions series

U3Cl is an exception to the statement above! It is not identical to the unitary group of degree 2. This exception occurs if the number of strands is zero mululo three in the case of Assion groups of type U resp. if the number of strands is even concerning the Assion groups of type S. But, since U3Cl is a subgroup of U4Cl (the unitary group of degree 3 over $GF(4)$) you may obtain it like this:

```
U3Clemb = U3.as_classical_group( embedded = True); print U3Clemb; \
    print ""
um1, um2 =U3Clemb.gens()

U4 = AssionGroupU(4)
U4Cl = U4.as_classical_group(); print U4Cl; print ""

print 'first generator in U4Cl?', um1 in U4Cl
print 'second generator in U4Cl?', um2 in U4Cl
```

```
Matrix group over Finite Field in a of size 2^2 with 2 generators (
[0 0 a] [a + 1     a     a]
[0 1 0] [    a a + 1     a]
[a 0 a], [    a     a a + 1]
)
General Unitary Group of degree 3 over Finite Field in a of size 2^2 with respect to
hermitian form:
[0 0 1]
[0 1 0]
[1 0 0]
first generator in U4Cl? True
second generator in U4Cl? True
```

Don't confuse about the difference with respect to the base field concerning GU(3,2) (size 2^2) and PGU(3,2) (size 2) which is caused by non conform conventions inside sage.

1.5.1 Description of the exceptions as centralizer

Assion described the exceptional cases of the groups $S(m)$ and $U(m)$ as centralizer groups of certain elements in the projective counterpart of the corresponding classical group. These elements can be obtained by the following method:

```
u3cent = U3.centralizing_element(); print "centalizing element (\
    u3cent):\n%s\n" %( u3cent )
print "u3cent in U3Cl? %s" %( u3cent in U3Cl )
print "u3cent in U3Clemb? %s\n" %(u3cent in U3Clemb )
```

```
u3centP = U3.centralizing_element(projective = True ); print "\
    projective centalizing element (u3centP):\n%s\n" %( u3centP )
print "u3centP in U3Cl? %s" %(u3centP in U3Cl)
print "u3centP in U3Clemb? %s\n" %(u3centP in U3Clemb)

PU3 = U3Cl.ambient_group(); print "Ambient Group of U3Cl (PU3):\n%s\\
    n" %(PU3)
PU3cent = PU3.centralizer(u3centP); print "Centralizer of u3centP in\
     PU3 (PU3cent):\n%s\n" %(PU3cent)

print "Check PU3cent == U3Cl: %s" %( U3Cl == PU3cent)
```

```
centalizing element (u3cent):
[a + 1 a + 1     1]
[a + 1     0     a]
[    1     a     a]
u3cent in U3Cl? True
u3cent in U3Clemb? True
projective centalizing element (u3centP):
(1,16)(2,9)(3,10)(4,19)(6,12)(7,20)(13,21)(14,15)
u3centP in U3Cl? True
u3centP in U3Clemb? False
Ambient Group of U3Cl (PU3):
The projective general unitary group of degree 3 over Finite Field of size 2
Centralizer of u3centP in PU3 (PU3cent):
Subgroup of (The projective general unitary group of degree 3 over Finite Field of size 2)
generated by [(2,3,4)(6,13,20)(7,12,21)(8,11,18)(9,10,19),
(1,6,7)(3,14,19)(4,10,15)(5,18,11)(12,20,16),
(1,16)(2,9)(3,10)(4,19)(6,12)(7,20)(13,21)(14,15)]
Check PU3cent == U3Cl: True
```

1.6 Conversion maps

Observe, that the cubic braid groups do not differ from the corresponding Assion groups in terms of isomorphism as long as the number of strand is less than five:

```
print 'C3 isomorphic to S3?', C3.is_isomorphic(S3)
print 'C3 isomorphic to U3?', C3.is_isomorphic(U3)
print 'C3 isomorphic to S3Cl?', C3.is_isomorphic(S3Cl)
print 'C3 isomorphic to U3Cl?', C3.is_isomorphic(U3Cl)
print 'C3 isomorphic to U3Clemb?', C3.is_isomorphic(U3Clemb)
```

```
C3 isomorphic to S3?
True
C3 isomorphic to U3?
True
C3 isomorphic to S3Cl? True
C3 isomorphic to U3Cl? True
```

```
C3 isomorphic to U3Clemb? True
True
C3 isomorphic to S3Cl? True
C3 isomorphic to U3Cl? True
C3 isomorphic to U3Clemb? True
```

These isomorphisms are canonical in the case of S3 and U3 since all these groups are defined as finitely presented groups with identical number of generators and according relations:

```
print 'Relations of C3:\n%s\n'%( list(C3.relations()) )
print 'Relations of S3:\n%s\n'%( list(S3.relations()) )
```

```
Relations of C3:
[c0*c1*c0*c1^-1*c0^-1*c1^-1, c0^3, c1^3]
Relations of S3:
[s0*s1*s0*s1^-1*s0^-1*s1^-1, s0^3, s1^3]
```

In the other cases the maps are realized as conversion maps, that is:

```
C3.inject_variables(); element = c0*c1**2; image_element = C3Cl(\
    element)
print 'Image of %s in C3Cl:\n%s\n' %(element, image_element)
U3.inject_variables(); element = u0*u1**2; image_element = U3Cl(\
    element)
print 'Image of %s in U3Cl:\n%s\n' %(element, image_element)
image_element = U3Clemb(element)
print 'Image of %s in U3Clemb:\n%s\n' %(element, image_element)
```

```
Defining c0, c1
Image of c0*c1^2 in C3Cl:
[ -E(3)^2  E(12)^7]
[-E(12)^7   E(3)^2]
Defining u0, u1
Image of u0*u1^2 in U3Cl:
(1,7,16,20)(2,19,9,4)(3,15,10,14)(5,11)(6,13,12,21)(8,18)
Image of u0*u1^2 in U3Clemb:
[    1     1 a + 1]
[a + 1      a a + 1]
[    a     0     a]
```

You may also convert in the opposite direction:

```
elements = list(S3Cl.some_elements())
image_elements = [ S3(element) for element in elements ]

print 'Images of\n\n%s\n\nand\n\n%s\n\nin S3:\n\n%s\n' %(elements\
    [0], elements[1], image_elements)
```

```
Images of

[1 1]
```

[0 1]

and

[0 1]
[2 0]

in S3:

[s1, s0*s1*s0]

1.7 Preimages in the Artin braid group

For each element of a cubic braid group you can define a preimage in the braid group as instance of the element class of the braid group class:

```
braid_elements = [element.braid() for element in image_elements ]
print 'Braid preimages of\n%s\nand\n%s:\n\n%s\n' %(elements[0], \
    elements[1], braid_elements)
type(braid_elements[0])
jones_polynomial = [element.jones_polynomial() for element in \
    braid_elements ]
print 'Coresponding Jones polynomials:\n%s\n' %(jones_polynomial)
burau_matrix = braid_elements[1].burau_matrix()
print 'Burau matrix to second braid preimage:\n%s' %(burau_matrix)
```

```
Braid preimages of
[1 1]
[0 1]
and
[0 1]
[2 0]:

[s1, s0*s1*s0]
<class 'lib.local_braid.local_BraidGroup_class_with_category.element_class'>
Coresponding Jones polynomials:
[-sqrt(t) - 1/sqrt(t), -t^(5/2) - sqrt(t)]
Burau matrix to second braid preimage:
[  1 - t t - t^2     t^2]
[  1 - t       t       0]
[      1       0       0]
```

1.8 Burau matrices for the cubic braid groups

You can calculate the burau matrix correspondig to elements of the cubic braid groups, as well. By default you will get it as matrix over \QQ adjoint with a third root of unity (zeta3):

```
elem = image_elements[1]
print "Burau:\n%s" %(elem.burau_matrix())
```
```
Burau:
[  -zeta3        1     zeta3]
[  -zeta3 zeta3 + 1         0]
[       1         0         0]
```

Using the additional options of this method, you can construct representations in finite matrix groups, as well:

```
bur_mat = elem.burau_matrix(characteristic = 5)
print "Burau in characteristic 5:\n%s\nbase_ring: %s" %(bur_mat, \
    bur_mat.base_ring())
```
```
Burau in characteristic 5:
[2*rI + 2       1 3*rI + 3]
[2*rI + 2 3*rI + 4        0]
[       1       0        0]
base_ring: Finite Field in rI of size 5^2
```

To see all possible options type (but again, note that carriage return is ignored in the sage worksheet. To see it better use jupyter notebook or command line):

```
print CubicBraidElement.burau_matrix.__doc__
```
```
    Return the Burau matrix of the cubic braid coset.

    If you don't see this well formated type

    sage: print CubicBraidElement.burau_matrix.__doc_

    This method uses the same method belonging to the Braid class, but reduces the
parameter to a primitive six
    root of unity, respectivly an element vanishing on the polynomial $x^2-x+1$

    INPUT: (all parameters are optional by keyword )

    - "rootBur": six root of unity in some field (default six root of unity over
$\QQ$)
    - "Domain":  base_ring for the burau matrix (default is Cyclotomic Field of order
3 and degree 2, resp. the
                domain of rootBur if given)
    - "reduced": boolean (default: False); whether to return the reduced or unreduced
Burau representation
                (see Braid class )
    - "characteristic": integer giving the characteristic of the domain (default is 0
or the characteristic of
                the doamain if given)
    - "version":  values:
```

'unitary': gives the unitary form according to Squier (see Braid._unitary_burau_matrix_())
'default': the method behaves like the original one of the Braid -class
any value else: gives the reduced form given on wikipedia

OUTPUT:

The Burau matrix of the cubic braid coset with entries in the domain given by the options

If you need the values of the reconstructed keywords “rootBur”, “Domain” or “characteristic” use the internal
version _burau_matrix_ of this method

RAISE:

- ValueError: 'characteristic must be in integer'

- ValueError: 'characteristic must be a prime'

- ValueError: 'characteristic of Domain does not match given characteristic'

- ValueError: 'rootBur must belong to a domain containing 1'

- ValueError: 'rootBur must vanish on x^2-x+1 default case

- ValueError: 'rootBur must vanish on x^4-x^2+1 in case of call with version = 'unitary'

EXAMPLES::

```
sage: C3.<c1, c2> = CubicBraidGroup(3)
sage: ele1 = c1*c2*c1
sage: BurauTest = ele1.burau_matrix(); print BurauTest
[   -zeta3         1     zeta3]
[   -zeta3 zeta3 + 1         0]
[        1         0         0]
sage: BurauTest.base_ring()
Cyclotomic Field of order 3 and degree 2
sage:
sage: BurauTest = ele1.burau_matrix( characteristic = 0 ); print BurauTest
[   -zeta3         1     zeta3]
[   -zeta3 zeta3 + 1         0]
```

```
[       1        0        0]
sage: BurauTest.base_ring()
Cyclotomic Field of order 3 and degree 2
sage:
sage: BurauTest = ele1.burau_matrix( Domain = QQ ); print BurauTest
Warning: Domain extended to splitting field of tT^2 - tT + 1
[-rI + 1       1  rI - 1]
[-rI + 1      rI       0]
[      1       0       0]
sage: BurauTest.base_ring()
Number Field in rI with defining polynomial tT^2 - tT + 1
sage:
sage: BurauTest = ele1.burau_matrix( Domain = QQ[I, sqrt(3)] ); print
BurauTest
[ 1/2*sqrt3*I + 1/2                  1 -1/2*sqrt3*I - 1/2]
[ 1/2*sqrt3*I + 1/2 -1/2*sqrt3*I + 1/2                  0]
[                 1                  0                  0]
sage: BurauTest.base_ring()
Number Field in I with defining polynomial x^2 + 1 over its base field
sage:
sage: BurauTest = ele1.burau_matrix( characteristic = 7 ); print BurauTest
[3 1 4]
[3 5 0]
[1 0 0]
sage: BurauTest.base_ring()
Finite Field of size 7
sage:
sage: BurauTest = ele1.burau_matrix( characteristic = 2 ); print BurauTest
[rI + 1      1 rI + 1]
[rI + 1     rI       0]
[     1      0       0]
sage: BurauTest.base_ring()
Finite Field in rI of size 2^2
sage:
sage:
sage: F4.<r64> = GF(4)
sage: BurauTest = ele1.burau_matrix( rootBur=r64 ); print BurauTest
[r64 + 1       1 r64 + 1]
[r64 + 1     r64       0]
[      1       0       0]
sage: BurauTest.base_ring()
Finite Field in r64 of size 2^2
sage:
sage: BurauTest = ele1.burau_matrix( Domain = GF(5) ); print BurauTest
Warning: Domain extended to splitting field of tT^2 + 4*tT + 1
[2*rI + 2         1 3*rI + 3]
[2*rI + 2 3*rI + 4         0]
```

```
        [       1        0        0]
        sage: BurauTest.base_ring()
        Finite Field in rI of size 5^2
        sage:
        sage: BurauTest = ele1.burau_matrix( version = 'unitary' ); print BurauTest
        [        0 -zeta12^3]
        [-zeta12^3         0]
        sage: BurauTest.base_ring()
        Cyclotomic Field of order 12 and degree 4
        sage:
        sage: BurauTest = ele1.burau_matrix( Domain = QQ[I, sqrt(3)], version =
'unitary' ); print BurauTest
        [ 0 -I]
        [-I  0]
        sage: BurauTest.base_ring()
        Number Field in I with defining polynomial x^2 + 1 over its base field
```

AUTHOR:

- Sebastian Oehms, Sept. 2016

REFERENCES:

- :wikipedia: “Burau_representation”

for more inormation type
sage: print Braid.burau_matrix.___doc___
sage: print local_Braid.burau_matrix.___doc___
sage: print local_Braid.___burau_matrix_wikipedia___.___doc___
sage: print local_Braid.___burau_matrix_unitary___.___doc___

1.9 Other matrix group realizations via the Burau representation

If you want to create the image of the Burau representation as as matrix group together with the corresponding group homorphism you just need to type, for instance:

```
C3c7 = C3.as_matrix_group( Domain = GF(7) ); C3c7
Matrix group over Finite Field of size 7 with 2 generators (
[3 5 0]  [1 0 0]
[1 0 0]  [0 3 5]
[0 0 1], [0 1 0]
)
```

Conversion maps are availlable as in the case of the classical groups:

```
c1, c2 = C3.gens(); elemC3 = c1*c2
elemC3c7 = C3c7( elemC3 ); print "elemC3c7:\n", elemC3c7
elemC3back = C3( elemC3c7 ); print "elemC3back:\n", elemC3back
print "Check: ", elemC3back == elemC3
```

```
elemC3c7:
[3 1 4]
[1 0 0]
[0 1 0]
elemC3back:
c0*c1
Check:  True
```

Of course, the map backwards is a group homomorphism only if the representation is faithfull. In general this conversion map is a section. Further, note that the Burau representation does factor through the relations of Assion, just for certain characteristic.

1.10 Realization as complex reflection groups

A third kind of realization of the cubic braid groups leads to complex reflection groups. This realization is availlable starting wirh sage version 7.2 if in addition gap3 with the CHEVIE package is installed. In this case the complex reflection group is obtained by:

```
R3 = C3.as_reflection_group(); R3
```

```
Irreducible complex reflection group of rank 2 and type ST4
```

All methods implemented for the IrreduclibeComplexReflectionGroup class can be used for R3, for example:

```
ctmat       = R3.cartan_matrix();           print "Cartan matrix \
   corresponding to %s:\n%s\n" %(C3, ctmat)
simproots   = R3.simple_roots();            print "simple roots \
   corresponding to %s:\n%s\n" %(C3, simproots)
reflhplanes = R3.reflection_hyperplanes(); print "reflection \
   hyperplanes corresponding to %s:\n%s\n" %(C3, reflhplanes)
coxele      = R3.coxeter_element();         print "Coxeter element \
   corresponding to %s:\n%s" %(C3, coxele)
```

```
Cartan matrix corresponding to Cubic Braid group on 3 strands:
[-2*E(3) - E(3)^2         E(3)^2]
[       -E(3)^2 -2*E(3) - E(3)^2]
simple roots corresponding to Cubic Braid group on 3 strands:
Finite family {1: (0, -2*E(3) - E(3)^2), 2: (2*E(3)^2, E(3)^2)}
reflection hyperplanes corresponding to Cubic Braid group on 3 strands:
Finite family {1: Vector space of degree 2 and dimension 1 over Universal Cyclotomic Field
Basis matrix:
```

[1 0], 2: Vector space of degree 2 and dimension 1 over Universal Cyclotomic Field
Basis matrix:
[1 -1], 3: Vector space of degree 2 and dimension 1 over Universal Cyclotomic Field
Basis matrix:
[1 -E(3)], 4: Vector space of degree 2 and dimension 1 over Universal Cyclotomic Field
Basis matrix:
[1 -E(3)^2]}

Coxeter element corresponding to Cubic Braid group on 3 strands:
(1,7,6,12,23,20)(2,8,17,24,9,5)(3,16,10,19,15,21)(4,14,11,22,18,13)

Elements in R3 may be interpreted as element of the cubic braid group or the classical realization via conversion:

```
coxeleC3  = C3(coxele);  print "Coxeter element as element of %s:\\
    n%s\n" %(C3, coxeleC3)
coxeleC3Cl = C3Cl(coxele); print "Coxeter element as element of the \
    classical group corresponding to the %s:\n%s\n" %(C3, coxeleC3Cl)
```

Coxeter element as element of Cubic Braid group on 3 strands:
c0*c1

Coxeter element as element of the classical group corresponding to the Cubic Braid group on 3 strands:
[0 E(4)]
[E(12)^11 E(3)]

Conversion backwards is posible, as well:

```
c1, c2 = C3.gens()
element = c1*c2/c1/c2; elementcl = C3Cl(element)
image = R3(element);  print "Image of %s in %s:\n%s\n" %(element, \
    R3, image)
image = R3(elementcl); print "Image of \n%s\nin %s:\n%s\n" %(\
    elementcl, R3, image)
```

Image of c0*c1*c0^-1*c1^-1 in Irreducible complex reflection group of rank 2 and type ST4:
(1,15,12,16)(2,18,24,14)(3,17,19,5)(4,20,22,6)(7,9,23,8)(10,11,21,13)
Image of
[E(3) E(12)^11]
[-E(12)^11 -E(3)]
in Irreducible complex reflection group of rank 2 and type ST4:
(1,15,12,16)(2,18,24,14)(3,17,19,5)(4,20,22,6)(7,9,23,8)(10,11,21,13)

1.11 Realization as permutation groups

A third kind of realization of the cubic braid groups is that as permutation groups. This realization is in sage available for all finetely presented groups via the gap interface. In addition you will get coercion maps here, as well and a permutation group constructed with respect to the classical realization. This is the default in the call:

```
C3.as_permutation_group()
```
Subgroup of (Symmetric group of order 8! as a permutation group) generated by
[(2,3,5)(4,6,8), (1,2,4)(5,7,6)]

To obtain the permutation group calculated for the finitely presented group you must type:

```
C3.as_permutation_group( native = True )
```
Subgroup of (Symmetric group of order 8! as a permutation group) generated by
[(2,4,5)(3,6,7), (1,2,3)(5,8,6)]

1.12 Other useful methods

In principal you can use all gap functions being available for these types of groups and being implemented via the gap interface of sage. Use the online help ("print dir(C3)" for example) as shown above to see them all. Here are some examples:

```
S3.character_table()
```
```
[       1        1        1        1        1        1        1]
[       1 -zeta3 - 1   zeta3        1   zeta3 -zeta3 - 1        1]
[       1   zeta3 -zeta3 - 1        1 -zeta3 - 1   zeta3        1]
[       2       -1       -1       -2        1        1        0]
[       2   -zeta3 zeta3 + 1       -2 -zeta3 - 1   zeta3        0]
[       2 zeta3 + 1   -zeta3       -2   zeta3 -zeta3 - 1        0]
[       3        0        0        3        0        0       -1]
```

```
U7 = AssionGroupU( 7 ); print "Order of U7:", U7.order()
```
Order of U7:
82771476480

```
conCls1s2 = S4.conjugacy_class( x*y**2*z ); conCls1s2
```
Conjugacy class of s0*s1^2*s2 in Assion group on 4 strands of type S