GroupRepresentationsForCAP

Skeletal category of group representations for CAP

2020.09.08

8 September 2020

Sebastian Posur

Sebastian Posur
Email: sebastian.posur@uni-siegen.de
Homepage: https://sebastianpos.github.io
Address: Department Mathematik
Universität Siegen
Walter-Flex-Straße 3
57068 Siegen
Germany

Contents

Chapter 1

Associators

1.1 Introduction

Let G be a finite group and let G-mod be a skeletal version of the monoidal category of finite dimensional complex representations of G. The purpose of these GAP methods is the computation of the associators of G-mod.

1.2 Quickstart

The following commands compute the associator of D_8 and write all data necessary for the reproducibility of the computation to files with the prefix "D8".

Example

```
gap> G := DihedralGroup( 8 );;
gap> ComputeAssociator( G, true, true, false );;
gap> path := Concatenation(
>   PackageInfo( "GroupRepresentationsForCAP" )[1].InstallationPath,
>   "/examples/doc/D8" );;
gap> WriteAssociatorComputationToFiles( path );
```

1.3 Read, Write, and Display

The following intermediate steps of the associator computation can be read from/written to files.

- Irreducible representations of a finite group given by matrices (Data 1).
- Decomposition isomorphisms of tensor products into direct sums of irreducibles (Data 2).

Furthermore, the following data can be written to files.

- A database key for the AssociatorsDatabase/DatabaseKeys.g file.
- The final result, namely the associator (Data 3).

Data 1 and Data 2 involve choices and thus are subject to changes in further versions of this package. However, the process Data 2 -> Data 3 is a mathematical function and thus stable. For reproducibility, it is recommended to store all three data. To facilitate this task, use the function WriteAssociatorComputationToFiles.

1.3.1 WriteDatabaseKeysToFile (for IsString)

▷ `WriteDatabaseKeysToFile(s)` (operation)

Returns: nothing

The argument is a filename s. This operation writes the database keys computed by the last call of InitializeGroupData to the corresponding file.

1.3.2 WriteRepresentationsDataToFile (for IsString)

▷ `WriteRepresentationsDataToFile(s)` (operation)

Returns: nothing

The argument is a filename s. This operation writes the representations computed by the last call of InitializeGroupData to the corresponding file.

1.3.3 WriteSkeletalFunctorDataToFile (for IsString)

▷ `WriteSkeletalFunctorDataToFile(s)` (operation)

Returns: nothing

The argument is a filename s. This operation writes the skeletal functor data computed by the last call of SkeletalFunctorTensorData to the corresponding file.

1.3.4 WriteAssociatorDataToFile (for IsString)

▷ `WriteAssociatorDataToFile(s)` (operation)

Returns: nothing

The argument is a filename s. This operation writes the associator data of the initialized group to the corresponding file. You have to call AssociatorForSufficientlyManyTriples first.

1.3.5 WriteAssociatorComputationToFiles (for IsString)

▷ `WriteAssociatorComputationToFiles(s)` (operation)

Returns: nothing

Only call this function if you did a whole associator computation first (e.g. using ComputeAssociator). The argument is a string s. This function writes 4 files:

- sKey.g: A file for the database key of the associator computation.
- sReps.g: A file containing the irreducible representations used for the associator computation.
- sDec.g: A file for the tensor decompositions used for the associator computation.
- sAss.g or sAssD.g: A file containing the computed associator. The suffix D is used if the associator was not computed for all triples.

1.3.6 ReadDatabaseKeys (for IsString)

▷ `ReadDatabaseKeys(s)` (operation)

Returns: a list

The argument is a filename s of a file written by WriteDatabaseKeysToFile. The output is a list [group, conductor, position of trivial character, field, category].

1.3.7 ReadRepresentationsData (for IsString, IsString)

▷ `ReadRepresentationsData(s_1, s_2)` (operation)

Returns: a list

The arguments are a filename s_1 of a file written by WriteDatabaseKeysToFile, and a filename s_2 of a file written by WriteRepresentationsDataToFile. The output is a list [,number of irreducibles, irreducibles, representations given by images of generators, inverses of these images, vector space objects for the irreducibles].

1.3.8 ReadSkeletalFunctorData (for IsString, IsString)

▷ `ReadSkeletalFunctorData(s_1, s_2)` (operation)

Returns: a list

The arguments are a filename s_1 of a file written by WriteDatabaseKeysToFile, and a filename s_2 of a file written by WriteSkeletalFunctorDataToFile. The output is a list [irreducibles, skeletal functor tensor data, vector space objects for the irreducibles].

1.4 Computing associators

1.4.1 InitializeGroupData (for IsGroup)

▷ `InitializeGroupData(G)` (operation)

Returns: a list

The argument is a group G. This method calls InitializeGroupData(G, false).

1.4.2 InitializeGroupData (for IsGroup, IsBool)

▷ `InitializeGroupData(G, b)` (operation)

Returns: a list

The arguments are a group G and a boolean b. The output is a list [generators of G ,number of irreducibles, irreducibles, representations given by images of generators, inverses of these images, vector space objects for the irreducibles]. Furthermore, this method stores the database key, which can be written using WriteDatabaseKeysToFile. If b is true, then the id of the group in the database key is given by its string, otherwise it is given by its id in the SmallGroupLibrary.

1.4.3 InitializeGroupDataDixon (for IsGroup)

▷ `InitializeGroupDataDixon(G)` (operation)

Returns: a list

The argument is a group G. This method calls InitializeGroupDataDixon(G, false).

1.4.4 InitializeGroupDataDixon (for IsGroup, IsBool)

▷ `InitializeGroupDataDixon(G, b)` (operation)

Returns: a list

The arguments are a group G and a boolean b. This method does the same as InitializeGroupData, but uses IrreducibleRepresentationsDixon for affording irreducible representations.

1.4.5 InitializeGroupData (for IsGroup, IsList, IsBool)

▷ `InitializeGroupData(arg1, arg2, arg3)` (operation)

1.4.6 SkeletalFunctorTensorData

▷ `SkeletalFunctorTensorData()` (operation)

Returns: a list

There is no argument. This methods calls SkeletalFunctorTensorData with the output of the last call of InitializeGroupData or InitializeGroupDataDixon.

1.4.7 SkeletalFunctorTensorData (for IsList)

▷ `SkeletalFunctorTensorData(l)` (operation)

Returns: a list

The argument is a list l which is the output of InitializeGroupData, InitializeGroupDataDixon, or ReadRepresentationsData. The output is a triple $[t_1, t_2, t_3]$. t_1 is the list of all characters of G. t_2 is a list such that the (i, j)-th entry, where i, j range from 1 to the number of irreducibles, is a pair of mutual inverse morphisms $[\alpha, \alpha^{-1}]$, and α is a decomposition isomorphism $\bigoplus_{\chi \in \mathrm{Irr}(G)} V_\chi^{n_\chi} \to V_i \otimes V_j$. t_3 is a list of vector space objects for the irreducibles.

1.4.8 AssociatorDataFromSkeletalFunctorTensorData (for IsInt, IsInt, IsInt, IsList)

▷ `AssociatorDataFromSkeletalFunctorTensorData(a, b, c, l)` (operation)

Returns: a list

The arguments are integers a, b, c and a list l which is the output of SkeletalFunctorTensorData. The output is a list containing homalg matrices representing the components of the associator of V_a, V_b, V_c, where the numbers correspond to the enlisting of the irreducible characters given by l.

1.4.9 AssociatorForSufficientlyManyTriples

▷ `AssociatorForSufficientlyManyTriples()` (operation)

Returns: a list

There is no argument. This methods calls AssociatorForSufficientlyManyTriples with the output of the last call of SkeletalFunctorTensorData and false.

1.4.10 AssociatorForSufficientlyManyTriples (for IsList, IsBool)

▷ `AssociatorForSufficientlyManyTriples(l, b)` (operation)

Returns: a list

The arguments are a list l which is the output of SkeletalFunctorTensorData, and a boolean b. The output is a list of lists L such that $L[a][b][c]$ contains the associator computed by AssociatorDataFromSkeletalFunctorTensorData(a,b,c). If b is true, then a, b, c ranges through all possible triples, otherwise, a, b, c are computed for so many triples such that the others can be obtained using braidings.

1.4.11 ComputeAssociator (for IsGroup, IsBool)

▷ `ComputeAssociator(G, b_1)` (operation)

Returns: a list

The arguments are a group G, and a boolean b_1. The output is ComputeAssociator(G, b_1, false, true).

1.4.12 ComputeAssociator (for IsGroup, IsBool, IsBool)

▷ `ComputeAssociator(G, b_1, b_2)` (operation)

Returns: a list

The arguments are a group G, and two booleans b_1, b_2. The output is ComputeAssociator(G, b_1, b_2, true).

1.4.13 ComputeAssociator (for IsGroup, IsBool, IsBool, IsBool)

▷ `ComputeAssociator(G, b_1, b_2, b_3)` (operation)

Returns: a list

The arguments are a group G, and three booleans b_1, b_2, b_3. The output is a list l whose (a,b,c)-th entry contains a string representing the associator of the objects V_a, V_b, V_c in a skeleton of the representation category of G, where V_* are irreducible representations corresponding to the ordering of the irreducible characters Irr(G). If b_1 is true, this method uses IrreducibleRepresentationsDixon, otherwise it uses IrreducibleAffordingRepresentation. If b_2 is true, the associators are computed for all possible triples a,b,c, otherwise only for sufficiently many such that the others can be reproduced using the braiding in the representation category. If b_3 is true, then the id of the group in the database key is given by its string, otherwise it is given by its id in the SmallGroupLibrary. This last boolean is relevant only if you want to write the computed associators to files (e.g. using WriteAssociatorComputationToFiles).

1.5 Technical functions

1.5.1 SetInfoLevelForAssociatorComputations (for IsInt)

▷ `SetInfoLevelForAssociatorComputations(l)` (operation)

Returns: nothing

The argument is an integer l. If $l > 0$, then the functions for computing associators provide information during the computation. This is useful in cases where the computation may take a long time.

1.5.2 DefinedOverCyclotomicField (for IsInt, IsGroupHomomorphism)

▷ `DefinedOverCyclotomicField(n, f)` (operation)

Returns: a boolean

The arguments are an integer n and a group homomorphism f whose images are matrices. The output is true if the entries of the images of f lie in a cyclotomic field generated by a primitive n-th root of unity, false otherwise.

1.5.3 GroupReperesentationByImages (for IsGroup, IsList)

▷ `GroupReperesentationByImages(G, L)` (operation)

Returns: a group homomorphism

The arguments are a group G with generators $g_1,\dots,g_n$ and a list $L=[l_1,\dots,l_n]$. The output is the group homomorphism from G to the group generated by the elements of L, mapping g_i to l_i.

1.5.4 DiagonalizationTransformationOfBraiding (for IsVectorSpaceMorphism)

▷ `DiagonalizationTransformationOfBraiding(e)` (attribute)

Returns: an invertible endomorphism in $\mathrm{Hom}(V,V)$

The argument is an endomorphism $e \in \mathrm{Hom}(V,V)$ of vector spaces whose minimal polynomial divides x^2-1. The output is an invertible endomorphism t such that $t^{-1}\circ e\circ t$ is a diagonal matrix.

1.5.5 AffordAllIrreducibleRepresentations (for IsGroup)

▷ `AffordAllIrreducibleRepresentations(G)` (operation)

Returns: a list

The argument is a group G. The output is a list of all irreducible representations of G using the command IrreducibleAffordingRepresentation.

1.5.6 AffordAllIrreducibleRepresentationsDixon (for IsGroup)

▷ `AffordAllIrreducibleRepresentationsDixon(G)` (operation)

Returns: a list

The argument is a group G. The output is a list of all irreducible representations of G using the command IrreducibleRepresentationsDixon.

1.5.7 DefaultFieldForListOfRepresentations (for IsList)

▷ `DefaultFieldForListOfRepresentations(L)` (operation)

Returns: a GAP field

The argument is a list L of representations of a group G. The output is a field over which all representations are defined simultaniously.

1.5.8 RewriteMatrixInCyclotomicGenerator (for IsMatrix, IsInt)

▷ `RewriteMatrixInCyclotomicGenerator(M, n)` (operation)

Returns: a matrix

The arguments are a matrix M and an integer n. The output is a matrix N in $Q[\varepsilon]$. Substituting an n-th root of unity for ε in N yields M.

1.5.9 InternalHomToTensorProductAdjunctionMapTemp (for IsVectorSpaceObject, IsVectorSpaceObject, IsVectorSpaceMorphism)

▷ `InternalHomToTensorProductAdjunctionMapTemp(b, c, g)` (operation)

Returns: a morphism in $\mathrm{Hom}(a\otimes b,c)$.

The arguments are objects b,c and a morphism $g : a \to \underline{\mathrm{Hom}}(b,c)$. The output is a morphism $f : a\otimes b \to c$ corresponding to g under the tensor hom adjunction.

1.5.10 HomalgMatrixAsString (for IsHomalgMatrix)

▷ `HomalgMatrixAsString(M)` (operation)

Returns: a string

The argument is a homalg matrx M. The output is a string consisting of the elements of M, seperated by commas.

1.5.11 DataFromSkeletalFunctorTensorDataAsStringList (for IsList)

▷ `DataFromSkeletalFunctorTensorDataAsStringList(l)` (operation)

Returns: a list of strings and empty entries

The argument is a list l of homalg matrices. In l, empty entries are allowed. The output is a list where each non-empty entry of l is converted to a string using HomalgMatrixAsString.

1.5.12 AsVectorSpaceMorphism (for IsHomalgMatrix)

▷ `AsVectorSpaceMorphism(M)` (attribute)

Returns: a vector space morphism

The argument is a homalg matrix M. The output is a vector space morphism whose underlying matrix is given by M.

1.5.13 CreateEndomorphismFromString (for IsVectorSpaceObject, IsString)

▷ `CreateEndomorphismFromString(V, s)` (operation)

Returns: a vector space morphism

The arguments are a vector space object V and a string s consisting of $\dim(V)^2$ elements of the ground field of V. The output is a vector space endomorphism $V \to V$ defined by s.

Chapter 2

Semisimple Categories

2.1 Introduction

Let k be a field and I be a totally ordered set. We denote the matrix category of k by $k-$vec (see the package `LinearAlgebraForCAP`). The semisimple category $\bigoplus_{i\in I} k-$vec associated to k and I is defined as the full subcategory of the product category $\prod_{i\in I} k-$vec generated by those I-indexed tuples having only finitely many non-zero entries. By χ^i, we denote the object which is 1 at entry i and 0 otherwise. Thus, an arbitrary object in $\bigoplus_{i\in I} k-$vec can be written as $\oplus_{i\in I} a_i \chi^i$ for non-negative numbers a_i for which only finitely many are non-zero.

2.2 Constructors

2.2.1 SemisimpleCategory (for IsFieldForHomalg, IsFunction, IsObject, IsString, IsBool, IsList)

▷ `SemisimpleCategory(k, m, u, s, b, L)` (operation)

Returns: a category

The arguments are:

- a homalg field k,
- a membership function m sending any GAP object to a boolean,
- a GAP object u,
- a string s containing a filename in the folder "/gap/AssociatorsDatabase/" of this package,
- a boolean b,
- a list L containing 4 entries, where the first 3 are filters and the last one is a string.

The output is a CAP category modelling $\bigoplus_{i\in I} k-$vec, where I is the set defined by the membership function m. Note that objects in I are expected to be equipped with operations enlisted in the chapter "Irreducible Objects". Furthermore, this CAP category is a rigid symmetric closed monoidal Abelian category. Its tensor product is defined by the data of the file s, where the boolean b is true if the associator stored in s was computed for all triples, and false otherwise (cf. chapter "Associators"). Its braiding and duality comes from the additional structure required for I. Its tensor unit is modelled by

u. The three filters of the L are filters for the resulting category, its objects, and its morphisms. L_4 is the name of the resulting category.

2.2.2 SemisimpleCategory (for IsFieldForHomalg, IsFunction, IsObject, IsString, IsBool)

▷ `SemisimpleCategory(k, m, u, s, b)` (operation)

Returns: a category

The arguments are:

- a homalg field k,
- a membership function m sending any GAP object to a boolean,
- a GAP object u,
- a string s containing a filename in the folder "/gap/AssociatorsDatabase/" of this package,
- a boolean b.

This function calls SemisimpleCategory on the six arguments [k, m, u, s, b, [IsObject, IsObject, IsObject, automatically generated name]]

2.2.3 SemisimpleCategoryMorphism (for IsSemisimpleCategoryObject, IsList, IsSemisimpleCategoryObject)

▷ `SemisimpleCategoryMorphism(s, L, r)` (operation)

Returns: a morphism

The arguments are an object s in a semisimple category $\bigoplus_{i \in I} k-$vec, a list of pairs $L = [[\phi_1, i_1], \dots [\phi_l, i_l]]$ where ϕ_j are morphisms in the Matrix Category $k-$vec and $i_j \in I$, and another object r in the same semisimple category. The output is a morphism in $\bigoplus_{i \in I} k-$vec from s to r whose i-th component is given by ϕ_i. For this morphism to be well defined, there has to be an ϕ_i for every i in the support of s and r.

2.2.4 SemisimpleCategoryMorphismSparse (for IsSemisimpleCategoryObject, IsList, IsSemisimpleCategoryObject)

▷ `SemisimpleCategoryMorphismSparse(s, L, r)` (operation)

Returns: a morphism

The arguments are an object s in a semisimple category $\bigoplus_{i \in I} k-$vec, a list of pairs $L = [[\phi_1, i_1], \dots [\phi_l, i_l]]$ where ϕ_j are morphisms in the Matrix Category $k-$vec and $i_j \in I$, and another object r in the same semisimple category. The output is a morphism in $\bigoplus_{i \in I} k-$vec from s to r whose i_j-th component is given by ϕ_{i_j} for $j = 1, \dots l$, and by the zero morphism otherwise.

2.2.5 ComponentInclusionMorphism (for IsSemisimpleCategoryObject, IsObject)

▷ `ComponentInclusionMorphism(v, j)` (operation)

Returns: a morphism

The arguments are an object $v = \oplus_{i \in I} a_i \chi^i$ in a semisimple category $\bigoplus_{i \in I} k-$vec, and an object $j \in I$. The output is the canonical inclusion $a_j \chi^j \hookrightarrow \oplus_{i \in I} a_i \chi^i$ in $\bigoplus_{i \in I} k-$vec.

2.2.6 ComponentProjectionMorphism (for IsSemisimpleCategoryObject, IsObject)

▷ `ComponentProjectionMorphism(v, j)` (operation)

Returns: a morphism

The arguments are an object $v = \oplus_{i \in I} a_i \chi^i$ in a semisimple category $\bigoplus_{i \in I} k-$vec, and an object $j \in I$. The output is the canonical projection $\oplus_{i \in I} a_i \chi^i \twoheadrightarrow a_j \chi^j$ in $\bigoplus_{i \in I} k-$vec.

2.2.7 SemisimpleCategoryObject (for IsList, IsCapCategory)

▷ `SemisimpleCategoryObject(L, C)` (operation)

Returns: an object

The arguments are a list L and a semisimple category $C = \bigoplus_{i \in I} k-$vec. The list L contains pairs $L = [[a_1, i_1], \ldots, [a_l, i_l]]$ of non-negative integers a_j and objects $i_j \in I$. The output is the object in C given by $\oplus_{j=1}^{l} a_j \chi^{i_j}$.

2.2.8 SemisimpleCategoryObjectConstructorWithFlatList (for IsList, IsCapCategory)

▷ `SemisimpleCategoryObjectConstructorWithFlatList(L, C)` (operation)

Returns: an object

The arguments are a list L and a semisimple category $C = \bigoplus_{i \in I} k-$vec. The list L contains an even number of elements $L = [a_1, i_1, \ldots, a_l, i_l]$ of non-negative integers a_j and objects $i_j \in I$. The output is the object in C given by $\oplus_{j=1}^{l} a_j \chi^{i_j}$.

2.3 Attributes

2.3.1 MembershipFunctionForSemisimpleCategory (for IsSemisimpleCategory)

▷ `MembershipFunctionForSemisimpleCategory(C)` (attribute)

Returns: a function

The argument is a semisimple category $C = \bigoplus_{i \in I} k-$vec. The output is its underlying membership function m for I.

2.3.2 UnderlyingCategoryForSemisimpleCategory (for IsSemisimpleCategory)

▷ `UnderlyingCategoryForSemisimpleCategory(C)` (attribute)

Returns: a category

The argument is a semisimple category $C = \bigoplus_{i \in I} k-$vec. The output is its underlying category $k-$vec.

2.3.3 UnderlyingFieldForHomalgForSemisimpleCategory (for IsSemisimpleCategory)

▷ `UnderlyingFieldForHomalgForSemisimpleCategory(C)` (attribute)

Returns: a homalg field

The argument is a semisimple category $C = \bigoplus_{i \in I} k-$vec. The output is its underlying field k.

2.3.4 GivenObjectFilterForSemisimpleCategory (for IsSemisimpleCategory)

▷ `GivenObjectFilterForSemisimpleCategory(C)` (attribute)

Returns: a filter

The argument is a semisimple category $C = \bigoplus_{i \in I} k-\mathrm{vec}$. The output is its object filter which could be specified in the constructor of C.

2.3.5 GivenMorphismFilterForSemisimpleCategory (for IsSemisimpleCategory)

▷ `GivenMorphismFilterForSemisimpleCategory(C)` (attribute)

Returns: a filter

The argument is a semisimple category $C = \bigoplus_{i \in I} k-\mathrm{vec}$. The output is its morphism filter which could be specified in the constructor of C.

2.3.6 SemisimpleCategoryMorphismList (for IsSemisimpleCategoryMorphism)

▷ `SemisimpleCategoryMorphismList(alpha)` (attribute)

Returns: a list

The argument is a morphism $\alpha = (\alpha_i)_{i \in I}$ in a semisimple category $\bigoplus_{i \in I} k-\mathrm{vec}$. The output is the list of pairs $[[\alpha_{i_1}, i_1], \dots, [\alpha_{i_l}, i_l]]$ where i_j ranges through the support of the source and range of α.

2.3.7 UnderlyingFieldForHomalg (for IsSemisimpleCategoryMorphism)

▷ `UnderlyingFieldForHomalg(alpha)` (attribute)

Returns: a homalg field

The argument is a morphism $\alpha = (\alpha_i)_{i \in I}$ in a semisimple category $\bigoplus_{i \in I} k-\mathrm{vec}$. The output is the homalg field k.

2.3.8 SemisimpleCategoryObjectList (for IsSemisimpleCategoryObject)

▷ `SemisimpleCategoryObjectList(v)` (attribute)

Returns: a list

The argument is an object $v = \oplus_{j=1}^{l} a_j \chi^{i_j}$ in a semisimple category. The output is the list $[[a_1, i_1], \dots [a_l, i_l]]$.

2.3.9 SemisimpleCategoryObjectListWithActualObjects (for IsSemisimpleCategory-Object)

▷ `SemisimpleCategoryObjectListWithActualObjects(v)` (attribute)

Returns: a list

The argument is an object $v = \oplus_{j=1}^{l} a_j \chi^{i_j}$ in a semisimple category. The output is the list $[[a_1, \chi^{i_1}], \dots [a_l, \chi^{i_l}]]$.

2.3.10 Support (for IsSemisimpleCategoryObject)

▷ `Support(v)` (attribute)

Returns: a list

The argument is an object $v = \oplus_{j=1}^{l} a_j \chi^{i_j}$ in a semisimple category. The output is the list $[i_1, \dots, i_l]$.

2.3.11 UnderlyingFieldForHomalg (for IsSemisimpleCategoryObject)

▷ `UnderlyingFieldForHomalg(v)` (attribute)

Returns: a homalg field

The argument is an object $v = \oplus_{j=1}^{l} a_j \chi^{i_j}$ in a semisimple category $\bigoplus_{i \in I} k-$vec. The output is the homalg field k.

2.3.12 Dimension (for IsSemisimpleCategoryObject)

▷ `Dimension(v)` (attribute)

Returns: an integer

The argument is an object $v = \oplus_{j=1}^{l} a_j \chi^{i_j}$ in a semisimple category $\bigoplus_{i \in I} k-$vec. The output is the integer $\sum_{j=1}^{l} a_j \cdot \dim(i_j)$. This functions works under the assumption that there is a notion of dimension on the objects in I.

2.4 Operations

2.4.1 Component (for IsSemisimpleCategoryMorphism, IsObject)

▷ `Component(alpha, i)` (operation)

Returns: a vector space morphism

The argument is a morphism $\alpha = (\alpha_i)_{i \in I}$ in a semisimple category $\bigoplus_{i \in I} k-$vec and an object $i \in I$. The output is α_i.

2.4.2 LaTeXStringOp (for IsSemisimpleCategoryMorphism)

▷ `LaTeXStringOp(m)` (operation)

Returns: a string

The argument is a morphism `m` in a semisimple category. The output is a LaTeX string (without enclosing dollar signs) that may be used to print out `m` nicely.

2.4.3 NormalizeSemisimpleCategoryObjectList (for IsList)

▷ `NormalizeSemisimpleCategoryObjectList(L)` (operation)

Returns: a list

The argument is a list $L = [[a_1, i_1], \dots, [a_l, i_l]]$ of non-negative integers a_j and objects $i_j \in I$, where I correspond to irreducible objects in a semisimple category $\bigoplus_{i \in I} k-$vec. The output is again a list of pairs consisting of integers an elements in I, but with the following normalization:

- Each a_j is positive,
- i_j is strictly less than i_{j+1}.

2.4.4 Multiplicity (for IsSemisimpleCategoryObject, IsObject)

▷ `Multiplicity(v, i)` (operation)

Returns: an integer

The arguments are an object $v = \oplus_{j=1}^{l} a_j \chi^{i_j}$ in a semisimple category $\bigoplus_{i \in I} k-$vec, and an element $i \in I$. The output is the integer a_i.

2.4.5 Component (for IsSemisimpleCategoryObject, IsObject)

▷ `Component(v, i)` (operation)

Returns: a vector space object

The arguments are an object $v = \oplus_{j=1}^{l} a_j \chi^{i_j}$ in a semisimple category $\bigoplus_{i \in I} k-$vec, and an element $i \in I$. The output is the k-vector space object k^{a_i} in Cap's Matrix Category.

2.4.6 TestPentagonIdentity (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

▷ `TestPentagonIdentity(v_1, v_2, v_3, v_4)` (operation)

Returns: a boolean

This is a debug operation. The arguments are 4 objects v_1, v_2, v_3, v_4 in a category. The output is true if the pentagon identity holds for those 4 objects, false otherwise.

2.4.7 TestPentagonIdentityForAllQuadruplesInList (for IsList)

▷ `TestPentagonIdentityForAllQuadruplesInList(L)` (operation)

Returns: a boolean

This is a debug operation. The argument is a list L consisting of quadruples of objects in a semisimple category $\bigoplus_{i \in I} k-$vec. The output is true if the pentagon identity holds for all those quadruples, false otherwise.

2.4.8 TestBraidingCompatability (for IsSemisimpleCategoryObject, IsSemisimpleCategoryObject, IsSemisimpleCategoryObject)

▷ `TestBraidingCompatability(v_1, v_2, v_3)` (operation)

Returns: a boolean

This is a debug operation. The arguments are 3 objects v_1, v_2, v_3 in a semisimple category $\bigoplus_{i \in I} k-$vec. The output is true if the braiding compatabilities with the associator hold, false otherwise.

2.4.9 TestBraidingCompatabilityForAllTriplesInList (for IsList)

▷ `TestBraidingCompatabilityForAllTriplesInList(L)` (operation)

Returns: a boolean

This is a debug operation. The argument is a list L consisting of triples of objects in a semisimple category $\bigoplus_{i \in I} k-$vec. The output is true if the braiding compatabilities with the associator hold for all those triples false otherwise.

2.4.10 TestZigZagIdentitiesForDual (for IsSemisimpleCategoryObject)

▷ `TestZigZagIdentitiesForDual(v)` (operation)

Returns: a boolean

This is a debug operation. The argument is an object v in a semisimple category $\bigoplus_{i \in I} k-$vec. The output is true if the zig zag identity for duals hold, false otherwise.

2.4.11 TestZigZagIdentitiesForDualForAllObjectsInList (for IsList)

▷ `TestZigZagIdentitiesForDualForAllObjectsInList(L)` (operation)

Returns: a boolean

This is a debug operation. The argument is a list L consisting of objects in a semisimple category $\bigoplus_{i \in I} k-$vec. The output is true if the zig zag identity for duals hold for all those objects, false otherwise.

2.4.12 LaTeXStringOp (for IsSemisimpleCategoryObject)

▷ `LaTeXStringOp(c)` (operation)

Returns: a string

The argument is an object `c` in a semisimple category. The output is a LaTeX string (without enclosing dollar signs) that may be used to print out `c` nicely.

2.5 GAP Categories

2.5.1 IsSemisimpleCategoryMorphism (for IsCapCategoryMorphism and IsCellOfSkeletalCategory)

▷ `IsSemisimpleCategoryMorphism(object)` (filter)

Returns: `true` or `false`

The GAP category of morphisms in a semisimple category.

2.5.2 IsSemisimpleCategoryObject (for IsCapCategoryObject and IsCellOfSkeletalCategory)

▷ `IsSemisimpleCategoryObject(object)` (filter)

Returns: `true` or `false`

The GAP category of objects in a semisimple category.

Chapter 3

Irreducible Objects

3.1 Introduction

For a semisimple category C of the form $\bigoplus_{i \in I} k-$vec to become a rigid symmetric closed monoidal skeletal category, the set I has to be equipped with extra strucutre. To become a skeletal category, we need:

- a total ordering on I.

To become a monoidal category, we need:

- a function `IsYieldingIdentities`, deciding whether an object yields the identity whenever it is part of an associator triple or a braiding pair,
- functions `Multiplicity` and `*`, defining the tensor product on objects,
- a function `AssociatorFromData`, defining the tensor product on morphisms.

To become a symmetric monoidal category, we need:

- a function `ExteriorPower`.

To become a rigid symmetric monoidal category, we need:

- a function `Dual`, defining duals on objects.

In the following, two families of such sets I are introduced:

- `GIrreducibleObject`: For a group G, the set I consists of the irreducible characters of G. We call the elements in I the G-irreducible objects.
- `GZGradedIrreducibleObject`: For a group G, the set I consists of the irreducible characters of G together with a degree $n \in \mathbb{Z}$. We call the elements in I the $G-\mathbb{Z}$-irreducible objects.

3.2 Constructors

3.2.1 GIrreducibleObject (for IsCharacter)

▷ `GIrreducibleObject(c)` (attribute)

Returns: a G-irreducible object

The argument is a character c of a group. The output is its associated G-irreducible object.

3.2.2 GZGradedIrreducibleObject (for IsInt, IsCharacter)

▷ `GZGradedIrreducibleObject(n, c)` (operation)

Returns: a $G-\mathbb{Z}$-irreducible object

The argument is an integer n and a character c of a group. The output is their associated $G-\mathbb{Z}$-irreducible object.

3.3 Attributes

3.3.1 UnderlyingCharacter (for IsGIrreducibleObject)

▷ `UnderlyingCharacter(i)` (attribute)

Returns: an irreducible character

The argument is a G-irreducible object i. The output is its underlying character.

3.3.2 UnderlyingGroup (for IsGIrreducibleObject)

▷ `UnderlyingGroup(i)` (attribute)

Returns: a group

The argument is a G-irreducible object i. The output is its underlying group.

3.3.3 UnderlyingCharacterTable (for IsGIrreducibleObject)

▷ `UnderlyingCharacterTable(i)` (attribute)

Returns: a character table

The argument is a G-irreducible object i. The output is the character table of its underlying group.

3.3.4 UnderlyingIrreducibleCharacters (for IsGIrreducibleObject)

▷ `UnderlyingIrreducibleCharacters(i)` (attribute)

Returns: a list

The argument is a G-irreducible object i. The output is a list consisting of the irreducible characters of its underlying group.

3.3.5 UnderlyingCharacterNumber (for IsGIrreducibleObject)

▷ `UnderlyingCharacterNumber(i)` (attribute)

Returns: an integer

The argument is a G-irreducible object i. The output is the integer n such that the n-th entry of the list of the underlying irreducible characters is the underlying irreducible character of i.

3.3.6 Dimension (for IsGIrreducibleObject)

▷ `Dimension(i)` (attribute)

Returns: an integer

The argument is a G-irreducible object i. The output is the dimension of its underlying irreducible character.

3.3.7 Dual (for IsGIrreducibleObject)

▷ `Dual(i)` (attribute)

Returns: a G-irreducible object

The argument is a G-irreducible object i. The output is the G-irreducible object associated to the dual character of c, where c is the associated character of i.

3.3.8 UnderlyingCharacter (for IsGZGradedIrreducibleObject)

▷ `UnderlyingCharacter(i)` (attribute)

Returns: an irreducible character

The argument is a $G-\mathbb{Z}$-irreducible object i. The output is its underlying character.

3.3.9 UnderlyingDegree (for IsGZGradedIrreducibleObject)

▷ `UnderlyingDegree(i)` (attribute)

Returns: an integer

The argument is a $G-\mathbb{Z}$-irreducible object i. The output is its underlying degree.

3.3.10 UnderlyingGroup (for IsGZGradedIrreducibleObject)

▷ `UnderlyingGroup(i)` (attribute)

Returns: a group

The argument is a $G-\mathbb{Z}$-irreducible object i. The output is its underlying group.

3.3.11 UnderlyingCharacterTable (for IsGZGradedIrreducibleObject)

▷ `UnderlyingCharacterTable(i)` (attribute)

Returns: a character table

The argument is a $G-\mathbb{Z}$-irreducible object i. The output is the character table of its underlying group.

3.3.12 UnderlyingIrreducibleCharacters (for IsGZGradedIrreducibleObject)

▷ `UnderlyingIrreducibleCharacters(i)` (attribute)

Returns: a list

The argument is a $G-\mathbb{Z}$-irreducible object i. The output is a list consisting of the irreducible characters of its underlying group.

3.3.13 UnderlyingCharacterNumber (for IsGZGradedIrreducibleObject)

▷ `UnderlyingCharacterNumber(i)` (attribute)

Returns: an integer

The argument is a $G-\mathbb{Z}$-irreducible object i. The output is the integer n such that the n-th entry of the list of the underlying irreducible characters is the underlying irreducible character of i.

3.3.14 Dimension (for IsGZGradedIrreducibleObject)

▷ `Dimension(i)` (attribute)

Returns: an integer

The argument is a $G-\mathbb{Z}$-irreducible object i. The output is the dimension of its underlying irreducible character.

3.3.15 Dual (for IsGZGradedIrreducibleObject)

▷ `Dual(i)` (attribute)

Returns: a $G-\mathbb{Z}$-irreducible object

The argument is a $G-\mathbb{Z}$-irreducible object i. The output is the $G-\mathbb{Z}$-irreducible object associated to the degree $-n$ and the dual character of c, where n is the underlying degree and c is the underlying character of i.

3.4 Properties

3.4.1 IsYieldingIdentities (for IsGIrreducibleObject)

▷ `IsYieldingIdentities(i)` (property)

Returns: a boolean

The argument is a G-irreducible object i. The output is true if the underlying character of i is the trivial one, false otherwise.

3.4.2 IsYieldingIdentities (for IsGZGradedIrreducibleObject)

▷ `IsYieldingIdentities(i)` (property)

Returns: a boolean

The argument is a $G-\mathbb{Z}$-irreducible object i. The output is true if the underlying character of i is the trivial one, false otherwise.

3.5 Operations

3.5.1 Multiplicity (for IsGIrreducibleObject, IsGIrreducibleObject, IsGIrreducibleObject)

▷ `Multiplicity(i, j, k)` (operation)

Returns: an integer

The arguments are 3 G-irreducible objects i,j,k. Let their underlying characters be denoted by a,b,c, respectively. Then the output is the number $\langle a, b\cdot c\rangle$, i.e., the multiplicity of a in the product of characters $b\cdot c$.

3.5.2 * (for IsGIrreducibleObject, IsGIrreducibleObject)

▷ `\*(i, j)` (operation)

Returns: a list

The arguments are 2 G-irreducible objects i,j with underlying irreducible characters a,b, respectively. The output is a list L = $[[n_1,k_1],\dots,[n_l,k_l]]$ consisting of positive integers n_c and G-irreducible objects k_c representing the character decomposition into irreducibles of the product $a \cdot b$.

3.5.3 AssociatorFromData (for IsGIrreducibleObject, IsGIrreducibleObject, IsGIrreducibleObject, IsList, IsFieldForHomalg, IsList)

▷ `AssociatorFromData(i, j, k, A, F, L)` (operation)

Returns: a list

The arguments are

- three G-irreducible objects i,j,k,
- a list A containing the associator on all irreducibles as strings, e.g., the list constructed by the methods provided in this package,
- a homalg field F,
- a list L = $[[n_1,h_1],\dots,[n_l,h_l]]$ consisting of positive integers n_c and G-irreducible objects h_c representing the character decomposition into irreducibles of the product of i,j,k.

The output is the list $[[\alpha_{h_1},h_1],\dots,[\alpha_{h_l},h_l]]$, where α_{h_c} is the F-vector space homomorphism representing the h_c-th component of the associator of i,j,k.

3.5.4 ExteriorPower (for IsGIrreducibleObject, IsGIrreducibleObject)

▷ `ExteriorPower(i, j)` (operation)

Returns: a list

The arguments are two G-irreducible objects i,j. The output is the empty list if i is not equal to j. Otherwise, the output is a list $L = [[n_1,k_1],\dots,[n_1,k_l]]$ consisting of positive integers n_j and G-irreducible objects k_j, corresponding to the decomposition of the second exterior power character $\wedge^2 c$ into irreducibles. Here, c is the associated character of i.

3.5.5 Multiplicity (for IsGZGradedIrreducibleObject, IsGZGradedIrreducibleObject, IsGZGradedIrreducibleObject)

▷ `Multiplicity(i, j, k)` (operation)

Returns: an integer

The arguments are 3 $G-\mathbb{Z}$-irreducible objects i,j,k. Let their underlying characters be denoted by a,b,c, respectively, and their underlying degrees by n_i,n_j,n_k, respectively. The output is 0 if n_i is not equal to n_j+n_k. Otherwise, the output is the number $\langle a, b\cdot c\rangle$, i.e., the multiplicity of a in the product of characters $b \cdot c$.

Let their underlying characters be denoted by a,b, respectively, and their underlying degrees by n_i,n_j, respectively. if $n_i = n_j$ and the underlying character number of j

3.5.6 * (for IsGZGradedIrreducibleObject, IsGZGradedIrreducibleObject)

▷ `\*(i, j)` (operation)

Returns: a list

The arguments are 2 $G-\mathbb{Z}$-irreducible objects i,j with underlying irreducible characters a,b, respectively. The output is a list L = $[[n_1,k_1],\dots,[n_l,k_l]]$ consisting of positive integers n_c and G-irreducible objects k_c representing the character decomposition into irreducibles of the product $a\cdot b$. The underlying degrees of k_c are given by the sum of the underlying degrees of i and j.

3.5.7 AssociatorFromData (for IsGZGradedIrreducibleObject, IsGZGradedIrreducibleObject, IsGZGradedIrreducibleObject, IsList, IsFieldForHomalg, IsList)

▷ `AssociatorFromData(i, j, k, A, F, L)` (operation)

Returns: a list

The arguments are

- three $G-\mathbb{Z}$-irreducible objects i,j,k,
- a list A containing the associator on all irreducibles (of G-irreducible objects) as strings, e.g., the list constructed by the methods provided in this package,
- a homalg field F,
- a list L = $[[n_1,h_1],\dots,[n_l,h_l]]$ consisting of positive integers n_c and $G-\mathbb{Z}$-irreducible objects h_c representing the character decomposition into irreducibles of the product of i,j,k.

The output is the list $[[\alpha_{h_1},h_1],\dots,[\alpha_{h_l},h_l]]$, where α_{h_c} is the F-vector space homomorphism representing the h_c-th component of the associator of i,j,k.

3.5.8 ExteriorPower (for IsGZGradedIrreducibleObject, IsGZGradedIrreducibleObject)

▷ `ExteriorPower(i, j)` (operation)

Returns: a list

The arguments are two $G-\mathbb{Z}$-irreducible objects i,j. The output is the empty list if the underlying characters of i and j are unequal. Otherwise, the output is a list $L=[[n_1,k_1],\dots,[n_1,k_l]]$ consisting of positive integers n_j and $G-\mathbb{Z}$-irreducible objects k_a, corresponding to the decomposition of the second exterior power character $\wedge^2 c$ into irreducibles. Here, c is the associated character of i and j. The underlying degree of k_a is the sum of the underlying degrees of i and j.

Chapter 4

Representation Category of Groups

4.1 Introduction

For a finite group G, the following methods provide computational tools for working with G-mod, a skeletal version of the monoidal category of finite dimensional complex representations of G, and with $G-\mathbb{Z}$-mod, a skeletal version of the monoidal category of finite dimensional complex representations of G equipped with a degree in $\mathbb{Z}$.

4.2 Quickstart

The following commands construct the category D_8-mod, the unique object v corresponding to the irreducible character of degree 2, and perform some computations.

Example

```
gap> RepG := RepresentationCategory( 8, 3 );
The representation category of Group( [ f1, f2, f3 ] )
gap> G := UnderlyingGroupForRepresentationCategory( RepG );
<pc group of size 8 with 3 generators>
gap> StructureDescription( G );
"D8"
gap> c := First( Irr( G ), i -> Degree( i ) = 2 );
Character( CharacterTable( D8 ), [ 2, 0, 0, -2, 0 ] )
gap> v := RepresentationCategoryObject( c, RepG );
1*(x_5)
gap> Dimension( v );
2
gap> Display( AssociatorLeftToRight( v, v, v ) );
Component: (x_5)

1/2,-1/2,1/2, 1/2,
1/2,-1/2,-1/2,-1/2,
1/2,1/2, 1/2, -1/2,
1/2,1/2, -1/2,1/2

A morphism in Category of matrices over Q
------------------------
gap> Display( Braiding( v, v ) );
Component: (x_1)
```

```
1

A morphism in Category of matrices over Q
-------------------------
Component: (x_2)

1

A morphism in Category of matrices over Q
-------------------------
Component: (x_3)

1

A morphism in Category of matrices over Q
-------------------------
Component: (x_4)

-1

A morphism in Category of matrices over Q
-------------------------
gap> alpha := IdentityMorphism( TensorProductOnObjects( v, v ) ) + Braiding( v, v );
<A morphism in The representation category of Group( [ f1, f2, f3 ] )>
gap> CokernelObject( alpha );
1*(x_4)
gap> TensorUnit( RepG );
1*(x_1)
```

4.3 Constructors

4.3.1 RepresentationCategory (for IsGroup)

▷ `RepresentationCategory(G)` (attribute)

Returns: a Cap category

The argument is a group G. The output is the Cap category G-mod. This method uses `String( G )` as an identifier of G.

4.3.2 RepresentationCategory (for IsInt, IsInt)

▷ `RepresentationCategory(o, n)` (operation)

Returns: a Cap category

The arguments are 2 integers o,n. The output is the Cap category G-mod, where G is the group of order o corresponding to the SmallGroupLibrary identification number n.

4.3.3 RepresentationCategoryObject (for IsList, IsCapCategory)

▷ `RepresentationCategoryObject(L, C)` (operation)

Returns: an object in G-mod

There are 2 arguments. The first argument is a list $L = [[n_1, c_1], \dots, [n_l, c_l]]$ consisting of non-negative integers n_i and characters c_i of the same group. Alternatively, the first argument can simply be an irreducible character c, which will be then interpreted as giving the input $[[1, c]]$. The second argument is the Cap category $C = G$-mod. The output is the unique object in G-mod having L as its character decomposition.

4.3.4 RepresentationCategoryObject (for IsCharacter, IsCapCategory, IsString)

▷ `RepresentationCategoryObject(c, C, str)` (operation)

Returns: an object in G-mod

There are 3 arguments. The first argument is an irreducible character `c`. The second argument is the CAP category $C = G$-mod. The third argument is a string used as follows: `SetString( GIrreducibleObject( c ), str )`. The output is the unique object in G-mod having $[[1, c]]$ as its character decomposition.

Chapter 5

Tools

5.1 Helper functions

5.1.1 LaTeXStringOfSemisimpleCategoryObjectList

▷ `LaTeXStringOfSemisimpleCategoryObjectList(L)` (function)

Returns: a string

The argument is a list *L* defining an object `c` in a semisimple category. The output is a LaTeX string (without enclosing dollar signs) that may be used to print out `c` nicely.

Index