AssociationSchemes

A GAP package for working with association schemes and homogeneous coherent configurations

1.0.0

10 April 2019

John Bamberg

Akihide Hanaki

Jesse Lansdown

John Bamberg
Email: john.bamberg@uwa.edu.au
Homepage: http://school.maths.uwa.edu.au/~bamberg/
Address: John Bamberg
School of Mathematics and Statistics
The University of Western Australia
35 Stirling Highway
Crawley WA 6009, Perth
Australia

Akihide Hanaki
Email: hanaki@shinshu-u.ac.jp
Homepage: http://math.shinshu-u.ac.jp/~hanaki/
Address: Akihide Hanaki
Department of Mathematics
Faculty of Science, Shinshu University
Matsumoto 390-8621, Japan

Jesse Lansdown
Email: jesse.lansdown@research.uwa.edu.au
Homepage: http://www.jesselansdown.com
Address: Jesse Lansdown
School of Mathematics and Statistics
The University of Western Australia
35 Stirling Highway
Crawley WA 6009, Perth
Australia

Abstract

AssociationSchemes is a GAP package for working with association schemes and homogeneous coherent configurations.

Copyright

Acknowledgements

The third author would like to acknowledge the support of an Australian Government Research Training Program (RTP) Scholarship while writing this software. The first and third authors are also grateful for the 2019 CMSC Retreat for providing an opportunity and environment for some of the founding work on the package.

Contents

Chapter 1

Introduction

1.1 Welcome to AssociationSchemes

AssociationSchemes is a GAP[GAP16] package for working with association schemes and homogeneous coherent configurations.

For definitions and more information on the theory of association schemes and homogeneous coherent configurations, we refer you to [BI84] and [God93].

It is important to note that the term "association scheme" is used differently by different authors. We reserve the term "association scheme" to mean a symmetric coherent configuration, and use "homogeneous coherent configuration" to refer to the more general objects.

1.2 Citing AssociationSchemes

If you use AssociationSchemes in research leading to publication please cite it as you would a paper. Example citations and a BibTeX entry are given below. Please check that the version and DOI match the version of AssociationSchemes used in your research.

Please also inform us by email of the paper, as we are very interested to hear how AssociationSchemes is being used!

Example

```
@article{AssociationSchemes,
Author = {Bamberg, J. and Hanaki, A. and Lansdown, J.},
Doi = {10.5281/zenodo.2634955},
Key = {AssociationSchemes},
Title = {{AssociationSchemes -- AssociationSchemes: A GAP package for working
with association schemes and homogeneous coherent configurations, Version 1.0.0}},
Url = {http://doi.org/10.5281/zenodo.2634955},
Year = 2019        ,
}
```

1.3 Dependencies

AssociationSchemes requires

- GAP 4.8 (or later)

as well as the following GAP packages:

- Digraphs 0.13.0 (or later)
- NautyTracesInterface 0.2 (or later)

You may of course use AssociationSchemes without the above packages, however the corresponding functionality will be unavailable.

Note that NautyTracesInterface is not yet a deposited package. It can be obtained from the link in the appendix.

1.4 Installation

To install AssociationSchemes, simply copy to the "pkg" directory of your GAP installation and unzip.

Alternatively, you may load the package from a location other than the GAP "pkg" directory by using the -L flag when opening GAP. Note that this requires the parent directory of AssociationSchemes to be called "pkg". See the GAP documentation for more details on how to do this. This is useful, for example, when administrative priveledges are required to access the GAP root directory.

Chapter 2

Getting Started

2.1 Tutorial - A first session with AssociationSchemes

In this section we provide a "first session" introduction to the AssociationSchemes package. It is intended to demonstrate the basic functions of the package through a series of small examples. More detailed descriptions of each of the methods are given in the chapter "Functionality". The fundamental method of describing a scheme in the AssociationSchemes package is via its relation matrix. Take for example the following relation matrix:

Example

```
gap> M:=
> [ [  0,  1,  2,  2,  3,  3,  3,  3,  3,  3,  3,  3 ],
>   [  1,  0,  2,  2,  3,  3,  3,  3,  3,  3,  3,  3 ],
>   [  2,  2,  0,  1,  3,  3,  3,  3,  3,  3,  3,  3 ],
>   [  2,  2,  1,  0,  3,  3,  3,  3,  3,  3,  3,  3 ],
>   [  3,  3,  3,  3,  0,  1,  2,  2,  3,  3,  3,  3 ],
>   [  3,  3,  3,  3,  1,  0,  2,  2,  3,  3,  3,  3 ],
>   [  3,  3,  3,  3,  2,  2,  0,  1,  3,  3,  3,  3 ],
>   [  3,  3,  3,  3,  2,  2,  1,  0,  3,  3,  3,  3 ],
>   [  3,  3,  3,  3,  3,  3,  3,  3,  0,  1,  2,  2 ],
>   [  3,  3,  3,  3,  3,  3,  3,  3,  1,  0,  2,  2 ],
>   [  3,  3,  3,  3,  3,  3,  3,  3,  2,  2,  0,  1 ],
>   [  3,  3,  3,  3,  3,  3,  3,  3,  2,  2,  1,  0 ] ];;
```

To construct a scheme from this matrix, we use the CoherentConfiguration command.

Example

```
gap> CC := HomogeneousCoherentConfiguration(M);;
```

CoherentConfiguration performs a number of checks as it constructs the scheme to make sure that it is in fact a homogeneous coherent configuration. However if you are confident that M does in fact define a scheme, then you can skip the checks by using CoherentConfigurationNC. Do not do this unless you are sure! We can display the scheme and see that GAP already knows the class and order of CC, as well that CC is symmetric and commutative.

Example

```
gap> Display(CC);
3-class association scheme of order 12.
  Symmetric: true
  Commutative: true
```

We can directly ask if CC is commutative or symmetric.

Example

```
gap> IsCommutative(CC);
true
gap> IsSymmetricCoherentConfiguration(CC);
true
```

We can retrieve the relation matrix of a scheme

Example

```
gap> relmat := RelationMatrix(CC);;
gap> relmat = M;
true
```

Example

```
gap> P := MatrixOfEigenvalues(CC);;
gap> Display(P);
[ [   1,   1,   2,   8 ],
  [   1,   1,   2,  -4 ],
  [   1,   1,  -2,   0 ],
  [   1,  -1,   0,   0 ] ]
```

If we try displaying again, we will also obtain the matrix of eigenvalues and the dual matrix of eigenvalues.

Example

```
gap> Display(CC);
3-class association scheme of order 12.
  Symmetric: true
  Commutative: true
  MatrixOfEigenvalues:
[ [   1,   1,   2,   8 ],
  [   1,   1,   2,  -4 ],
  [   1,   1,  -2,   0 ],
  [   1,  -1,   0,   0 ] ]
  DualMatrixOfEigenvalues:
[ [   1,   2,   3,   6 ],
  [   1,   2,   3,  -6 ],
  [   1,   2,  -3,   0 ],
  [   1,  -1,   0,   0 ] ]
```

If you want to print CC, it will return the relation matrix. This is useful if you want to print to a file for exmaple.

Example

```
gap> Print(CC);
[ [ 0, 1, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3 ], [ 1, 0, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3 ],
  [ 2, 2, 0, 1, 3, 3, 3, 3, 3, 3, 3, 3 ], [ 2, 2, 1, 0, 3, 3, 3, 3, 3, 3, 3, 3 ],
  [ 3, 3, 3, 3, 0, 1, 2, 2, 3, 3, 3, 3 ], [ 3, 3, 3, 3, 1, 0, 2, 2, 3, 3, 3, 3 ],
  [ 3, 3, 3, 3, 2, 2, 0, 1, 3, 3, 3, 3 ], [ 3, 3, 3, 3, 2, 2, 1, 0, 3, 3, 3, 3 ],
  [ 3, 3, 3, 3, 3, 3, 3, 3, 0, 1, 2, 2 ], [ 3, 3, 3, 3, 3, 3, 3, 3, 1, 0, 2, 2 ],
  [ 3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 0, 1 ], [ 3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 1, 0 ] ]
```

You can obtain the adjacency matrices by doing:

Example
```
gap> AdjacencyMatrices(CC);;
```

If you were able to calculate the matrix of eigenvalues, then you can also construct the minimal idempotents E_i

Example
```
gap> MinimalIdempotents(CC);;
```

Note that if CC is Schurian (or has a transitive group associated with it) then MinimalIdempotents will be much faster! You can check if a scheme is schurian by doing

Example
```
gap> IsSchurian(CC);
true
```

In doing this, a graph is constructed and the automorphism group for CC is found. We can also find the automorphism group directly.

Example
```
gap> AutomorphismGroup(CC);
<permutation group with 11 generators>
```

We can define homogeneous coherent figurations from transitive groups. This is typically fast.

Example
```
gap> G := Group( [ ( 6,10)( 7,11)( 8,12)( 9,13)(15,28)(16,29)(17,30)(18,31)
>          (20,37)(21,38)(22,39)(23,40)(24,33)(25,34)(26,35)(27,36),
>      ( 1,15,22,31,18,26,16, 2, 5)( 3,24,21)( 4,20,40,29,11, 6,28,27,25)
>      ( 7,10,14)( 8,33,35,39,38,12,32,13,19)( 9,37,34)(23,36,30),
>      ( 3, 4)( 7,11)( 8, 9)(12,13)(15,28)(17,31)(18,30)(19,32)(20,33)
>      (21,25)(22,36)(23,35)(24,37)(26,40)(27,39)(34,38), () ] );;
gap> HomogeneousCoherentConfigurationByOrbitals(G);;
```

If G is generously transitive, then we can constuct a Schurian scheme

Example
```
gap> IsGenerouslyTransitive(G);
true
gap> SchurianScheme(G);;
```

If we have a group G and subgroup H such that G acts transitively on G/H, then we can also use the following construction.

Example
```
gap> G:=SymmetricGroup(5);;
gap> H:=Stabiliser(G, 1);;
gap> HomogeneousCoherentConfigurationByOrbitals(G, H);;
```

There are a number of special constructors, such as for Johnson schemes

Example
```
gap> JohnsonScheme(10,3);
3-class association scheme of order 120.
```

AssociationSchemes also comes with a library of association schemes on small numbers of vertices, according to [HM].

Example

```
gap> m:=HomogeneousCoherentConfiguration(12, 7);;
```

We can test if two schemes are equal with "=". This will return true if the schemes have the same relation matrix. The previous example from the library is in fact the same as the example constructed from the matrix M at the start.

Example

```
gap> CC = m;
true
```

There is also the option to create a fusion scheme. This takes a partition of the relations, (where [0] must be cell of the partition. If the resulting fusion is not a valid scheme this will return fail;

Example

```
gap> FusionOfHomogeneousCoherentConfigurations(m, [[0], [1,2],[3]]);
2-class association scheme of order 12.
```

Chapter 3

Functionality

3.1 Constructor Methods

3.1.1 HomogeneousCoherentConfiguration (for IsMatrix)

▷ `HomogeneousCoherentConfiguration(M)` (operation)

Returns: homogeneous coherent configuration

Takes the relationship matrix, M, describing a homogensous coherent configuration and returns a HomogeneousCoherentConfiguration object. The matrix $M = \sum_{i=0}^{d} iA_i$, where A_i are the adjacency matrices describing a coherent configuration. Checks that the matrix satisfies the axioms of a homogeneous coherent configuration. (Note that this accepts a matrix of the form $M = \sum_{i=0}^{d} a_i A_i$ where a_i is not equal to i, however, it will first convert to the form $M = \sum_{i=0}^{d} iA_i$).

3.1.2 HomogeneousCoherentConfigurationNC (for IsMatrix)

▷ `HomogeneousCoherentConfigurationNC(M)` (operation)

Returns: homogeneous coherent configuration

Same as HomogeneousCoherentConfiguration but without performing any checks. Use this method only if you know with certainty that M describes a coherent configuration.

3.1.3 AssociationScheme (for IsMatrix)

▷ `AssociationScheme(M)` (operation)

Returns: homogeneous coherent configuration

Takes the relationship matrix, M, describing an associatioin scheme and returns a association scheme (symmetric coherent configuration). This is simply a HomogeneousCoherentConfiguration object, but with the known property of being symmetric. The matrix $M = \sum_{i=0}^{d} iA_i$, where A_i are the adjacency matrices describing an association scheme. Checks that the matrix satisfies the association scheme axioms. (Note that this accepts a matrix of the form $M = \sum_{i=0}^{d} a_i A_i$ where a_i is not equal to i, however, it will first convert to the form $M = \sum_{i=0}^{d} iA_i$).

3.1.4 AssociationSchemeNC (for IsMatrix)

▷ `AssociationSchemeNC(M)` (operation)

Returns: homogeneous coherent configuration

Same as AssociationScheme but without performing any checks. Use this method only if you know with certainty that M describes an association scheme (symmetric coherent configuration).

3.1.5 ReorderRelations (for IsHomogeneousCoherentConfiguration, IsList)

▷ `ReorderRelations(CC, L)` (operation)

Returns: coherent configuration

Takes a homogeneous coherent configuration CC and a list L, where L is a reordering of the relations. Returns a homogeneous coherent configuration where the i-th relation of the CC becomes the j-th relation in the new homogeneous coherent configuration, where $j = L_i$. Note that L_i must be equal to $\{0,\ldots,d\}$ as a set, and additionally requires that $L_1 = 0$.

3.1.6 SaveHomogeneousCoherentConfigurationWithCertainAttributes (for IsString, IsHomogeneousCoherentConfiguration, IsList)

▷ `SaveHomogeneousCoherentConfigurationWithCertainAttributes(file, A, L)` (operation)

Returns: true

Saves homogeneous coherent configuration A to file F with the attributes listed in L. Note that L must be a list of strings, where each entry is an attribute known for A. Note that Print or PrintTo will only return the relation matrix of a homogeneous coherent configuration, which contains all necessary information about the homogeneous coherent configuration, but may require a lot of computation to reobtain its attributes. Hence this method is intended to alow saving of computationally difficult or time consuming attributes directly. It also alows the user to choose which attributes to save, since some attributes are very large, but easily recomputed. For example, it is often desirable to save the matrix of eigenvalues, and perhaps the automorphism group and intersection matrices, while it is not generally desirable to also save the adjacency matrices or minimal idempotents.

3.1.7 ReadHomogeneousCoherentConfigurationWithCertainAttributes (for IsString)

▷ `ReadHomogeneousCoherentConfigurationWithCertainAttributes(file, A, L)` (operation)

Returns: homogeneous coherent configuration

Reads in a homogenous coherent configuration from file and sets it to have the attributes stored in the file. This reads files of the type formed by SaveHomogeneousCoherentConfigurationWithCertainAttributes.

3.1.8 BilinearFormsScheme (for IsField, IsPosInt, IsPosInt)

▷ `BilinearFormsScheme(F, n, k)` (operation)

Returns: homogeneous coherent configuration

Returns the bilinear forms scheme for the finite field F with a bilinear form from $F^n \times F^n$ to F^k.

3.1.9 HomogeneousCoherentConfigurationByOrbitals (for IsPermGroup)

▷ `HomogeneousCoherentConfigurationByOrbitals(G)` (operation)

Returns: homogeneous coherent configuration

Constructs a "group-case" coherent configuration, where the relations are defined by the orbitals of G on $\{1,\ldots,n\} \times \{1,\ldots,n\}$. G must be a permutation group which is transitive on $\{1,\ldots,n\}$.

3.1.10 HomogeneousCoherentConfigurationByOrbitals (for IsGroup, IsGroup)

▷ `HomogeneousCoherentConfigurationByOrbitals(G, H)` (operation)

Returns: homogeneous coherent configuration

Constructs a "group-case" coherent configuration, where the relations are defined by the orbitals of G on G/H. G is a group, H is a subgroup of G, G/H is the set of right cosets of G on H, and G must be transitive on G/H.

3.1.11 GrassmannScheme (for IsPosInt, IsPosInt, IsPosInt)

▷ `GrassmannScheme(n, k, q)` (operation)

Returns: homogeneous coherent configuration

Returns the Grassman scheme, $J_q(n,k)$.

3.1.12 GroupCoherentConfiguration (for IsGroup)

▷ `GroupCoherentConfiguration(G)` (operation)

Returns: homogeneous coherent configuration

Returns the coherent configuration on the conjugacy classes of a group G.

3.1.13 HammingScheme (for IsPosInt, IsPosInt)

▷ `HammingScheme(n, q)` (operation)

Returns: homogeneous coherent configuration

Returns the Hamming scheme, $H(n,q)$.

3.1.14 JohnsonScheme (for IsPosInt, IsPosInt)

▷ `JohnsonScheme(n, k)` (operation)

Returns: homogeneous coherent configuration

Returns the Johnson scheme, $J(n,k)$.

3.1.15 HomogeneousCoherentConfiguration (for IsPosInt, IsPosInt)

▷ `HomogeneousCoherentConfiguration(n, k)` (operation)

Returns: homogeneous coherent configuration

Returns the k-th homogeneous coherent configuration of order n. Library is complete for $5 \leq n \leq 38$ excluding $n \in \{31,35,36,37\}$, corresponding to [HM].

3.1.16 NumberOfHomogeneousCoherentConfigurations (for IsPosInt)

▷ `NumberOfHomogeneousCoherentConfigurations(n)` (operation)

Returns: m

Returns the number m of homogeneous coherent configuration of order n contained in the library.

3.1.17 AvailableHomogeneousCoherentConfigurations

▷ `AvailableHomogeneousCoherentConfigurations()` (operation)

Returns: L

Returns a list L of the orders for which the library contains homogeneous coherent configurations.

3.1.18 AllHomogeneousCoherentConfigurations (for IsPosInt)

▷ `AllHomogeneousCoherentConfigurations(n)` (operation)

Returns: L

Returns a list L of all homogeneous coherent configuration of order n.

3.1.19 FusionOfHomogeneousCoherentConfigurations (for IsHomogeneousCoherentConfiguration, IsList)

▷ `FusionOfHomogeneousCoherentConfigurations(CC, L)` (operation)

Returns: homogeneous coherent configuration

Takes a d-class homogeneous coherent configuration CC and returns a fusion scheme corresponding to L, where L is a partion of $\{0,\ldots,d\}$. Returns fail if L is not a valid partition.

3.1.20 DirectProductOfHomogeneousCoherentConfigurations (for IsHomogeneousCoherentConfiguration, IsHomogeneousCoherentConfiguration)

▷ `DirectProductOfHomogeneousCoherentConfigurations(CC1, CC2)` (operation)

Returns: homogeneous coherent configuration

Takes two homogeneous coherent configurations CC1 and CC2 and returns their direct product.

3.1.21 WreathProductOfHomogeneousCoherentConfigurations (for IsHomogeneousCoherentConfiguration, IsHomogeneousCoherentConfiguration)

▷ `WreathProductOfHomogeneousCoherentConfigurations(CC1, CC2)` (operation)

Returns: homogeneous coherent configuration

Takes two homogeneous coherent configurations CC1 and CC2 and returns their wreath product.

3.1.22 CyclotomicScheme (for IsPosInt, IsPosInt)

▷ `CyclotomicScheme(q, d)` (operation)

Returns: homogeneous coherent configuration

Let q be a prime power, and d a divisor of $q-1$. Returns the cyclotomic scheme $Cyc(q,d)$.

3.1.23 SchurianScheme (for IsPermGroup)

▷ `SchurianScheme(G)` (operation)

Returns: homogeneous coherent configuration

Returns the Schurian scheme defined by G, where G is a generously transitive permutation group. A Schurian scheme is a special case of CoherentConfigurationByOrbitals and is symmetric.

3.2 Matrices Describing Homogeneous Coherent Configurations

3.2.1 RelationMatrix (for IsHomogeneousCoherentConfiguration)

▷ `RelationMatrix(CC)` (operation)

Returns: M

Takes a homogeneous coherent configuration and returns the underlying relation matrix $M = \sum_{i=0}^{d} iA_i$, where A_i are the adjacency matrices of the coherent configuration

3.2.2 AdjacencyMatrices (for IsHomogeneousCoherentConfiguration)

▷ `AdjacencyMatrices(CC)` (attribute)

Returns: L

Returns a list L, where the i-th entry of L is the adjacency matrix A_{i-1}, where $(A_i)_{xy} = 1$ if $(x,y) \in R_i$ and $(A_i)_{xy} = 0$ otherwise.

3.2.3 IntersectionMatrices (for IsHomogeneousCoherentConfiguration)

▷ `IntersectionMatrices(CC)` (attribute)

Returns: L

Returns a list L of the intersection matrices of a homogeneous coherent configuration CC, where the i-th entry of L is B_{i-1} and $(B_i)_{jk} = p_{ji}^k$.

3.2.4 MinimalIdempotents (for IsHomogeneousCoherentConfiguration)

▷ `MinimalIdempotents(CC)` (attribute)

Returns: L

Returns a list L which is a basis of minimal idempotents for the adjacency algebra of a homogeneous coherent configuration CC. The i-th entry of L is E_{i-1}.

3.3 Properties Of Homogeneous Coherent Configurations

3.3.1 IsCommutative (for IsHomogeneousCoherentConfiguration)

▷ `IsCommutative(CC)` (property)

Returns: true or false

Checks if the input is a commutative coherent configuration.

3.3.2 IsSymmetricCoherentConfiguration (for IsHomogeneousCoherentConfiguration)

▷ `IsSymmetricCoherentConfiguration(CC)` (property)

Returns: true or false

Checks if the input is a symmetric coherent configuration.

3.3.3 IsAssociationScheme (for IsHomogeneousCoherentConfiguration)

▷ `IsAssociationScheme(`*CC*`)` (operation)
Returns: true or false
Alias for IsSymmetricCoherentConfiguration

3.3.4 IsStronglyRegularGraph (for IsHomogeneousCoherentConfiguration)

▷ `IsStronglyRegularGraph(`*CC*`)` (property)
Returns: true or false
Check if a coherent configuration is a strongly regular graph (a 2-class homogeneous coherent configuration).

3.3.5 IsPPolynomial (for IsHomogeneousCoherentConfiguration)

▷ `IsPPolynomial(`*CC*`)` (property)
Returns: true or false
Returns if the homogeneous coherent configuration CC is P-polynomial.

3.3.6 IsMetric (for IsHomogeneousCoherentConfiguration)

▷ `IsMetric(`*CC*`)` (operation)
Returns: true or false
Alias for is P-polynomial.

3.3.7 IsQPolynomial (for IsHomogeneousCoherentConfiguration)

▷ `IsQPolynomial(`*CC*`)` (property)
Returns: true or false
Returns if the commutative coherent configuration CC is Q-polynomial.

3.3.8 IsCometric (for IsHomogeneousCoherentConfiguration)

▷ `IsCometric(`*CC*`)` (operation)
Returns: true or false
Alias for is Q-polynomial.

3.3.9 IsThin (for IsHomogeneousCoherentConfiguration)

▷ `IsThin(`*CC*`)` (property)
Returns: true or false
Check if the homogeneous coherent configuration is thin.

3.3.10 IsQuasiThin (for IsHomogeneousCoherentConfiguration)

▷ `IsQuasiThin(`*CC*`)` (property)
Returns: true or false
Check if the homogeneous coherent configuration is quasi thin.

3.3.11 IsPrimitive (for IsHomogeneousCoherentConfiguration)

▷ `IsPrimitive(CC)` (property)

Returns: true or false

Check if the homogeneous coherent configuration is primitve.

3.3.12 IsHomogeneousCoherentConfigurationByOrbitals (for IsHomogeneousCoherentConfiguration)

▷ `IsHomogeneousCoherentConfigurationByOrbitals(CC)` (property)

Returns: true or false

Checks if the coherent configuration CC can be constructed from relations defined by the orbitals of a group G acting transitively on a set X.

3.3.13 IsGenerouslyTransitive (for IsPermGroup)

▷ `IsGenerouslyTransitive(G)` (property)

Returns: true or false

Checks if the permutation group G is generously transitive.

3.3.14 IsGenerouslyTransitive (for IsPermGroup, IsList)

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

Returns: true or false

Checks that the permutation group G acts generously transitive on the set L.

3.3.15 IsSchurian (for IsHomogeneousCoherentConfiguration)

▷ `IsSchurian(CC)` (property)

Returns: true or false

Checks if the input is a Schurian scheme, that is, if the automorphism group is generously transitive.

3.4 Attributes Of Homogeneous Coherent Configurations

3.4.1 NumberOfClasses (for IsHomogeneousCoherentConfiguration)

▷ `NumberOfClasses(CC)` (attribute)

Returns: d

Returns d for a d-class association scheme.

3.4.2 Order (for IsHomogeneousCoherentConfiguration)

▷ `Order(CC)` (attribute)

Returns: n

Returns the order n (number of vertices) of the coherent configuration.

3.4.3 Valencies (for IsHomogeneousCoherentConfiguration)

▷ `Valencies(CC)` (attribute)

Returns: L

Returns a list L of valencies of a coherent configuration CC. The i-th entry of L is k_{i-1}.

3.4.4 NumberOfCharacters (for IsHomogeneousCoherentConfiguration)

▷ `NumberOfCharacters(CC)` (attribute)

Returns: n

Returns the number n of characters of CC.

3.4.5 MatrixOfEigenvalues (for IsHomogeneousCoherentConfiguration)

▷ `MatrixOfEigenvalues(CC)` (attribute)

Returns: P

Returns a the matrix of eigenvalues (or character table), P, for a homogeneous coherent configuration CC.

3.4.6 CharacterTable (for IsHomogeneousCoherentConfiguration)

▷ `CharacterTable(CC)` (operation)

Returns: P

Alias for MatrixOfEigenvalues.

3.4.7 DualMatrixOfEigenvalues (for IsHomogeneousCoherentConfiguration)

▷ `DualMatrixOfEigenvalues(CC)` (attribute)

Returns: Q

Returns a the dual matrix of eigenvalues, Q, for a homogeneous coherent configuration CC.

3.4.8 AutomorphismGroup (for IsHomogeneousCoherentConfiguration)

▷ `AutomorphismGroup(CC)` (attribute)

Returns: G

Returns the automorphism group G of the coherent configuration CC. G is a permutation group acting on the index set of the vertices.

3.4.9 AllPPolynomialOrderings (for IsHomogeneousCoherentConfiguration)

▷ `AllPPolynomialOrderings(CC)` (attribute)

Returns: L

Calculate the list L of all P-polynomial orderings of a homogeneous coherent configuration.

3.4.10 KreinParameters (for IsHomogeneousCoherentConfiguration)

▷ `KreinParameters(CC)` (attribute)

Returns: L

Return a list L of all Krein parameters of a commutative homogeneous coherent configuration, where $L[i][j,k] = q^k_{i,j}$.

3.4.11 AllQPolynomialOrderings (for IsHomogeneousCoherentConfiguration)

▷ `AllQPolynomialOrderings(CC)` (attribute)

Returns: L

Calculate a list L of all Q-polynomial orderings of a homogeneous coherent configuration.

3.4.12 ConstructorGroup (for IsHomogeneousCoherentConfiguration)

▷ `ConstructorGroup(CC)` (attribute)

Returns: group or false

Checks if the coherent configuration was constructed by a group and returns it if it was, or returns false otherwise.

3.5 Methods

3.5.1 Relation (for IsHomogeneousCoherentConfiguration, IsPosInt, IsPosInt)

▷ `Relation(CC, x, y)` (operation)

Returns: i

Takes a CC and two points, x and y, and returns i such that $(x,y) \in R_i$.

3.5.2 Neighbours (for IsHomogeneousCoherentConfiguration, IsPosInt, IsInt)

▷ `Neighbours(CC, p, k)` (operation)

Returns: L

Returns a list L of all the points y of CC such that $(p,y) \in R_k$.

3.5.3 Neighbours (for IsHomogeneousCoherentConfiguration, IsInt, IsList)

▷ `Neighbours(CC, p, L)` (operation)

Returns: L

Returns a list L of all the points y of CC such that $(p,y) \in R_k$ for some $k \in L$.

3.5.4 IntersectionNumber (for IsHomogeneousCoherentConfiguration, IsInt, IsInt, IsInt)

▷ `IntersectionNumber(CC, i, j, k)` (operation)

Returns: p^k_{ij}

Returns the intersection number p^k_{ij} for a coherent configuration CC.

3.5.5 KreinParameter (for IsHomogeneousCoherentConfiguration, IsInt, IsInt, IsInt)

▷ `KreinParameter(CC, i, j, k)` (operation)

Returns: $q^k_{i,j}$

Compute the krein parameter $q^k_{i,j}$ of a commutative homogeneous coherent configuration.

3.5.6 MatrixOfEigenvaluesOfHammingScheme (for IsPosInt, IsPosInt)

▷ `MatrixOfEigenvaluesOfHammingScheme(n, q)` (operation)

Returns: P

Returns matrix of eigenvalue P for the Hamming scheme, $H(n,q)$.

3.5.7 MatrixOfEigenvaluesOfJohnsonScheme (for IsPosInt, IsPosInt)

▷ `MatrixOfEigenvaluesOfJohnsonScheme(n, k)` (operation)

Returns: P

Returns the matrix of eigenvalues P of the Johnson scheme $J(n,k)$.

3.6 Algebras

3.6.1 IntersectionAlgebra (for IsHomogeneousCoherentConfiguration)

▷ `IntersectionAlgebra(CC)` (operation)

Returns: A

Returns the intersection algebra A of a homogeneous coherent configuration.

3.6.2 BoseMesnerAlgebra (for IsHomogeneousCoherentConfiguration)

▷ `BoseMesnerAlgebra(CC)` (operation)

Returns: A

Returns the Bose-Mesner algebra A of a homogeneous coherent configuration.

3.6.3 AdjacencyAlgebra (for IsHomogeneousCoherentConfiguration)

▷ `AdjacencyAlgebra(CC)` (operation)

Returns: A

Returns the adjacency algebra A of a homogeneous coherent configuration. This is an alias for BoseMesnerAlgebra.

3.6.4 TerwilligerAlgebra (for IsHomogeneousCoherentConfiguration, IsInt)

▷ `TerwilligerAlgebra(CC, p)` (operation)

Returns: T

Returns the Terwilliger algebra T of a homogeneous coherent configuration with respect to the point p.

3.6.5 TerwilligerAlgebra (for IsHomogeneousCoherentConfiguration)

▷ `TerwilligerAlgebra(CC)` (operation)

Returns: T

Returns the Terwilliger algebra T of a homogeneous coherent configuration with respect to the first point.

3.7 Subsets And Codes

3.7.1 InnerDistribution (for IsList, IsHomogeneousCoherentConfiguration)

▷ `InnerDistribution(v, CC)` (operation)

Returns: a

Returns the inner distribution a of a vector v with respect to the adjacency matrices of the coherent configuration CC. Note that v must be a vector over R^n where n is the order of CC. CC must be commutative.

3.7.2 MacWilliamsTransform (for IsList, IsHomogeneousCoherentConfiguration)

▷ `MacWilliamsTransform(v, CC)` (operation)

Returns: aQ

Returns the MacWilliams transform aQ of a vector v with respect to a coherent confiiguration CC. Takes either a vector v in R^n and converts it to its inner distribution vector first, or takes the inner distribution directly.

3.7.3 DualBoseMesnerBasis (for IsHomogeneousCoherentConfiguration, IsPosInt)

▷ `DualBoseMesnerBasis(CC, p)` (operation)

Returns: L

Returns a list L with the dual Bose-Mesner basis of a homogeneous coherent configuration with respect to the point p, such that $L_i = \tilde{E}_{i-1}$.

3.7.4 DualBoseMesnerBasis (for IsHomogeneousCoherentConfiguration)

▷ `DualBoseMesnerBasis(CC)` (operation)

Returns: L

Returns a list L with the dual Bose-Mesner basis of a homogeneous coherent configuration with respect to the first point, such that $L_i = \tilde{E}_{i-1}$.

3.7.5 OuterDistribution (for IsList, IsHomogeneousCoherentConfiguration)

▷ `OuterDistribution(v, CC)` (operation)

Returns: B

Returns the outer distribution B of a vector v with respect to the adjacency matrices of the coherent configuration CC. Note that v must be a vector over R^n where n is the order of CC. CC must be commutative.

3.7.6 CharacteristicVector (for IsList, IsList)

▷ `CharacteristicVector(X, Omega)` (operation)

Returns: χ_X

Takes a subset X of Omega and returns the characteristic vector. The characteristic vector is a 0,1-vector indexed by the entries of Omega, with a 1 at position x if x is in X, and 0 otherwise.

3.7.7 CharacteristicVector (for IsList, IsPosInt)

▷ `CharacteristicVector(X, n)` (operation)

Returns: χ_X

Takes a subset X of of [1 .. n] and returns the characteristic vector chi_X.

Chapter 4

Examples

4.1 Example 1 – Constructing groups

In this example, we show how we can use coherent configurations to construct an entriely different almost simple permutation group from another one. We first show how $PSU(4,3)$ can be made out of its subgroup $PSL(3,4)$.

Example

```
gap> psl34 := PSL(3,4);;
gap> sylow3 := SylowSubgroup(psl34, 3);;
gap> normaliser := Normaliser(psl34, sylow3);;
gap> G := Image( FactorCosetAction(psl34, normaliser) );;
```

At this stage, we have constructed the unique permutation representation of degree 280, for $PSL(3,4)$.

Example

```
gap> A := HomogeneousCoherentConfigurationByOrbitals(G);
7-class homogeneous coherent configuration of order 280
gap> mat := RelationMatrix(A);;
gap> P := MatrixOfEigenvalues(A);;
gap> Print(P);
[ [ 1, 18, 18, 18, 72, 72, 72, 9 ], [ 1, 4, 4, 4, 16, -12, -12, -5 ],
  [ 1, -2, -2, 10, -8, 0, 0, 1 ], [ 1, -2, 10, -2, -8, 0, 0, 1 ],
  [ 1, 10, -2, -2, -8, 0, 0, 1 ],
  [ 1, -2, -2, -2, 0, -8*E(7)^3-8*E(7)^5-8*E(7)^6, -8*E(7)-8*E(7)^2-8*E(7)^4, -3 ],
  [ 1, -2, -2, -2, 0, -8*E(7)-8*E(7)^2-8*E(7)^4, -8*E(7)^3-8*E(7)^5-8*E(7)^6, -3 ],
  [ 1, -2, -2, -2, 7, -3, -3, 4 ] ]
```

We now take a particular fusion of this coherent configuration to obtain a 2-class association scheme.

Example

```
gap> valency18 := Filtered([1..7], j -> Number(mat[1], i -> i = j) = 18);
[1,2,3]
gap> fusions := List(Combinations(valency18,2), t ->
>                 FusionOfHomogeneousCoherentConfigurations(A, [[0], t,
>                         Difference([1..7],t)]) );;
```

Any of these three fusions will do:

Example

```
gap> autgroup := AutomorphismGroup( fusions[1] );;
gap> DisplayCompositionSeries( autgroup );
```

```
G (11 gens, size 26127360)
 | Z(2)
S (4 gens, size 13063680)
 | Z(2)
S (3 gens, size 6531840)
 | Z(2)
S (2 gens, size 3265920)
 | 2A(3,3) = U(4,3) ~ 2D(3,3) = O-(6,3)
1 (0 gens, size 1)
gap> socle := Socle(autgroup);;
gap> StructureDescription(socle);
"PSU(4,3)"
```

4.2 Example 2 – Dual polar spaces and their graphs

For this example, we also use the package FinInG [BBDB+18]. We will construct a metric association scheme coming from a dual polar space.

Example

```
gap> LoadPackage("FinInG", false);;
gap> quadric := EllipticQuadric(7, 2);
Q-(7, 2)
gap> points := AsList( Planes(quadric) );;
gap> mat := NullMat(Length(points), Length(points));;
gap> for i in [1..Length(points)] do
>         for j in [i+1..Length(points)] do
>                 intersection := Meet( points{[i,j]} );
gap>                    mat[i][j] := 2 - ProjectiveDimension( intersection );
gap>                    mat[j][i] := mat[i][j];
gap>         od;
gap> od;
```

So far we have constructed the relation matrix arising from the dual polar space.

Example

```
gap> a := HomogeneousCoherentConfiguration( mat );
3-class association scheme of order 765
gap> P := MatrixOfEigenvalues(a);;
gap> Q := DualMatrixOfEigenvalues(a);;
gap> Display(P);
[ [    1,   28,  224,  512 ],
  [    1,   11,   20,  -32 ],
  [    1,   -7,   14,   -8 ],
  [    1,    1,  -10,    8 ] ]
gap> Display(Q);
[ [       1,      84,     204,     476 ],
  [       1,      33,     -51,      17 ],
  [       1,    15/2,    51/4,   -85/4 ],
  [       1,   -21/4,  -51/16,  119/16 ] ]
gap> IsPPolynomial(a);
true
gap> IsQPolynomial(a);
true
```

A simpler way (perhaps) uses the automorphism group of the ambient polar space:

Example

```
gap> cgroup := CollineationGroup(quadric);
PGO(-1,8,2)
gap> G := Action(cgroup, points);
<permutation group with 3 generators>
gap> a := SchurianScheme(G);
3-class homogeneous coherent configuration of order 765
gap> IsPPolynomial(a);
true
```

The automorphism group of the association scheme should be the same:

Example

```
gap> autgroup := AutomorphismGroup(a);;
gap> autgroup = G;
true
```

Now (for the purist!) we see if there are interesting subsets. Take a nondegenerate hyperplane section defining a parabolic quadric.

Example

```
gap> hyperplane := First(Hyperplanes(PG(7,2)), h ->
>                 TypeOfSubspace(quadric, h) = "parabolic");
<a proj. 6-space in ProjectiveSpace(7, 2)>
gap> insidehyp := Filtered(points, t -> t * hyperplane);;
gap> vector := CharacteristicVector(insidehyp,points);;
gap> dist := InnerDistribution(vector, a);
[ 1, 56, 64, 14 ]
gap> macw := MacWilliamsTransform(dist, a);
[ 135, 630, 0, 0 ]
```

Therefore, a hyperplane section gives rise to a design that is not a code, in this association scheme. Now we produce the dual polar graph.

Example

```
gap> P := MatrixOfEigenvalues(a);;
gap> Display(P);
[ [   1,  224,  512,   28 ],
  [   1,   20,  -32,   11 ],
  [   1,   14,   -8,   -7 ],
  [   1,  -10,    8,    1 ] ]
gap> position := Position(P[1], 28);
4
gap> M := AdjacencyMatrices(a)[ position ];;
gap> graph := Graph(G, [1..Order(a)], OnPoints, {x,y} -> M[x][y] = 1);;
gap> IsDistanceRegular(graph);
true
gap> GlobalParameters(graph);
[ [ 0, 0, 28 ], [ 1, 3, 24 ], [ 3, 9, 16 ], [ 7, 21, 0 ] ]
```

4.3 Example 3 – Codes

For this example, we use the package Guava[BBC+18] for its facility with block codes. We will see that the inner distribution vector of a subset coincides with the weight enumerator of a code when the

association scheme is a Hamming scheme.

Example

```
gap> hammingscheme := HammingScheme(7,2);
7-class homogeneous coherent configuration of order 128
gap> LoadPackage("Guava", false);;
gap> hammingcode := HammingCode(3, GF(2));
a linear [7,4,3]1 Hamming (3,2) code over GF(2)
```

We now use an operation from Guava:

Example

```
gap> InnerDistribution(hammingcode);
[ 1, 0, 0, 7, 7, 0, 0, 1 ]
```

From the association scheme perspective ...

Example

```
gap> codewords := List( hammingcode, VectorCodeword );;
gap> vector := CharacteristicVector( codewords,  AsList(GF(2)^7) );;
gap> Collected(vector);
[ [ 0, 112 ], [ 1, 16 ] ]
gap> InnerDistribution(vector, hammingscheme);
[ 1, 0, 0, 7, 7, 0, 0, 1 ]
```

The MacWilliams transform coincides with the distribution vector of the dual code:

Example

```
gap> 1/16 * MacWilliamsTransform(vector, hammingscheme);
[ 1, 0, 0, 0, 7, 0, 0, 0 ]
gap> dualcode := DualCode( hammingcode );
a linear [7,3,4]2..3 dual code
gap> InnerDistribution( dualcode );
[ 1, 0, 0, 0, 7, 0, 0, 0 ]
```

4.4 Example 4 – Using the library

In this package, we also have a library of all small homogeneous coherent configurations, of order at most 38 (except 31, ,35, 36, 37), corresponding to [HM].

Example

```
gap> for i in [5..20] do
>         Print(i,"    ",NumberOfHomogeneousCoherentConfigurations(i),"\n");
gap> od;
5    2
6    6
7    3
8    16
9    10
10    11
11    3
12    54
13    5
14    14
15    24
```

```
16    208
17    4
18    90
19    6
20    90
gap> order7 := List([1..3], i -> HomogeneousCoherentConfiguration(7, i));
 1-class homogeneous coherent configuration of order 7,
 2-class homogeneous coherent configuration of order 7,
 3-class homogeneous coherent configuration of order 7 ]
```

The first of these is trivial, so we look at the other two. The first arises from the Paley graph of order 7.

Example

```
gap> a1 := order7[2];
2-class homogeneous coherent configuration of order 7
gap> IsStronglyRegularGraph( a1 );
true
gap> autgroup := AutomorphismGroup(a1);
Group([ (2,3,4)(5,7,6), (1,2,3,5,4,6,7) ])
gap> StructureDescription(autgroup);
"C7 : C3"
```

The last one is a 3-class association scheme:

Example

```
gap> a2 := order7[3];
3-class homogeneous coherent configuration of order 7
gap> IsAssociationScheme(a2);
true
gap> IsPPolynomial( a2 );
true
gap> IsPrimitive(a2);
rue
gap> Valencies(a2);
[ 1, 2, 2, 2 ]
gap> autgroup := AutomorphismGroup(a2);
Group([ (2,3)(4,5)(6,7), (1,2)(3,4)(5,6) ])
gap> StructureDescription(autgroup);
"D14"
gap> P := MatrixOfEigenvalues(a2);;
gap> Display(P);
[ [             1,             2,             2,             2 ],
  [             1,  E(7)^3+E(7)^4,    E(7)+E(7)^6,  E(7)^2+E(7)^5 ],
  [             1,  E(7)^2+E(7)^5,  E(7)^3+E(7)^4,    E(7)+E(7)^6 ],
  [             1,    E(7)+E(7)^6,  E(7)^2+E(7)^5,  E(7)^3+E(7)^4 ] ]
gap> AllPPolynomialOrderings(a2);
[ [ 0, 1, 2, 3 ], [ 0, 2, 3, 1 ], [ 0, 3, 1, 2 ] ]
gap> IsQPolynomial(a2);
true
gap> AllQPolynomialOrderings(a2);
[ [ 0, 1, 3, 2 ], [ 0, 2, 1, 3 ], [ 0, 3, 2, 1 ] ]
```

4.5 Example 5 – Constructing HS (advanced example)

We redo an example that appears in Section 3.6 of Peter Cameron's "Permutation Groups" book [Cam99] and construct the Higman-Sims group.

First we construct the Hoffman-Singleton graph from the alternating group of degree 7.

Example

```
gap> A7 := AlternatingGroup(7);;
gap> Pi := [ [ 1, 2, 4 ], [ 1, 3, 7 ], [ 1, 5, 6 ],
>  [ 2, 3, 5 ], [ 2, 6, 7 ], [ 3, 4, 6 ], [ 4, 5, 7 ] ];;
gap> OnSetsRecursive := function(x,g)
>         if not IsSet(x) then
>                 return x^g;
gap>         else
>                 return Set(x,y->OnSetsRecursive(y,g));
gap>         fi;
gap> end;;
gap> triples := Combinations([1..7], 3);;
gap> allFanos := Orbit(A7, Pi, OnSetsSets);;
gap> fifty := Concatenation(triples, allFanos);;
gap> A7action := Action(A7, fifty, OnSetsRecursive);
<permutation group with 2 generators>
gap> orbitals := Orbits(A7action, Combinations([1..50],2), OnSets);;
gap> List(orbitals, Size);
[ 210, 315, 70, 420, 105, 105 ]
```

We will now make a homogeneous coherent configuration from scratch, from these orbitals.

Example

```
gap> mat := NullMat(50,50);;
gap> for i in [1..50] do
>         for j in [i+1..50] do
>                 pos := First([1..Length(orbitals)], k -> [i,j] in orbitals[k]);
gap>                    mat[i][j] := pos;
gap>                    mat[j][i] := pos;
gap>         od;
gap> od;
```

This is not a CC yet. We will fuse the relations of valency 3 and 4:

Example

```
gap> l := Collected(mat[1]);
[ [ 0, 1 ], [ 1, 12 ], [ 2, 18 ], [ 3, 4 ], [ 4, 12 ], [ 5, 3 ] ]
gap> to_fuse := Filtered([1..Length(l)], t -> l[t][2] in [3,4])-1;
[ 3, 5 ]
gap> to_fuse2 := Difference([1..6],to_fuse);
[ 1, 2, 4, 6 ]
gap> poly := InterpolatedPolynomial(Rationals, Concatenation([0], to_fuse,
>         to_fuse2), [0,1,1,2,2,2,2] );;
gap> newmat := List(mat, row -> List(row, x -> Value(poly,x)));;
gap> Collected(newmat[1]);
[ [ 0, 1 ], [ 1, 7 ], [ 2, 42 ] ]
```

This now leads us directly to the Hoffman-Singleton graph:

Example

```
gap> cc := HomogeneousCoherentConfiguration( newmat );
2-class association scheme of order 50
gap> autHoffSing := AutomorphismGroup( cc );
<permutation group with 7 generators>
gap> StructureDescription( autHoffSing );
"PSU(3,5) : C2"
```

We will now construct the Mesner-Higman-Sims graph

Example

```
gap> vals := Valencies(cc);
[ 1, 7, 42 ]
gap> adjmat := AdjacencyMatrices(cc)[ Position(vals, 42) ];;
gap> graph := Graph(autHoffSing, [1..50], OnPoints, {x,y} -> adjmat[x][y]=1);;
gap> one_coclique := CompleteSubgraphsOfGivenSize(graph, 15)[1];;
gap> all_cocliques := Orbit(autHoffSing,
>         Set(VertexNames(graph){one_coclique}), OnSets);;
gap> Size(all_cocliques);
100
gap> G := Action(autHoffSing, all_cocliques, OnSets);;
gap> a := SchurianScheme(G);
4-class homogeneous coherent configuration of order 100
```

Now fuse the relations with valencies 7 and 15 (and the complement)

Example

```
gap> vals := Valencies(a);
[ 1, 35, 42, 15, 7 ]
gap> to_fuse := Filtered([1..Length(vals)], t -> vals[t] in [7,15])-1;;
gap> to_fuse2 := Difference([1..4], to_fuse);;
gap> fusion := FusionOfHomogeneousCoherentConfigurations(a, [[0], to_fuse,
>         to_fuse2]);
2-class association scheme of order 100
gap> autgroup2 := AutomorphismGroup(fusion);
<permutation group with 10 generators>
gap> StructureDescription(autgroup2);
"HS : C2"
```

Chapter 5

Appendix

5.1 AssociationSchemes Links

- Homepage: http://www.jesselansdown.com/AssociationSchemes
- Issue tracker: https://github.com/jesselansdown/AssociationSchemes/issues
- DOI: 10.5281/zenodo.2634955

5.2 GAP Links

- Homepage: http://gap-system.org
- NautyTracesInterface: https://github.com/sebasguts/NautyTracesInterface

References

[BBC+18] R. Baart, T. Boothby, J. Cramwinckel, J. Fields, D. Joyner, R. Miller, E. Minkes, E. Roijackers, L. Ruscio, and C. Tjhai. GUAVA, a gap package for computing with error-correcting codes, Version 3.14. https://gap-packages.github.io/guava, Mar 2018. Refereed GAP package. 24

[BBDB+18] J. Bamberg, A. Betten, J. De Beule, P. Cara, M. Lavrauw, and M. Neunhoeffer. FinInG, finite incidence geometry, Version 1.4.1. http://www.fining.org, Mar 2018. Refereed GAP package. 23

[BI84] Eiichi Bannai and Tatsuro Ito. *Algebraic combinatorics. I.* The Benjamin/Cummings Publishing Co., Inc., Menlo Park, CA, 1984. Association schemes. 4

[Cam99] Peter J. Cameron. *Permutation Groups*. London Mathematical Society Student Texts. Cambridge University Press, 1999. 27

[GAP16] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.8.6*, 2016. 4

[God93] C. D. Godsil. *Algebraic combinatorics*. Chapman and Hall Mathematics Series. Chapman & Hall, New York, 1993. 4

[HM] Akihide Hanaki and Izumi Miyamoto. 8, 12, 25

Index