XMod

Crossed Modules and Cat1-Groups

2.92

23 January 2024

Chris Wensley

Murat Alp

Alper Odabas

Enver Onder Uslu

Chris Wensley
Email: cdwensley.maths@btinternet.com
Homepage: https://github.com/cdwensley

Murat Alp
Email: muratalp@nigde.edu.tr
Address: Prof. Dr. M. Alp
Ömer Halisdemir University
Art and Science Faculty
Mathematics Department
Nigde
Turkey

Alper Odabas
Email: aodabas@ogu.edu.tr
Address: Dr. A. Odabas
Osmangazi University
Arts and Sciences Faculty
Department of Mathematics and Computer Science
Eskisehir
Turkey

Abstract

The XMod package provides functions for computation with

- finite crossed modules of groups and cat1-groups, and morphisms of these structures;
- finite pre-crossed modules, pre-cat1-groups, and their Peiffer quotients;
- isoclinism classes of groups and crossed modules;
- derivations of crossed modules and sections of cat1-groups;
- crossed squares and their morphisms, including the actor crossed square of a crossed module;
- crossed modules of finite groupoids (experimental version).

XMod was originally implemented in 1996 using the GAP3 language, when the second author was studying for a Ph.D. [Alp97] at Bangor.

In April 2002 the first and third parts were converted to GAP4, the pre-structures were added, and version 2.001 was released. The final two parts, covering derivations, sections and actors, were included in the January 2004 release 2.002 for GAP 4.4.

In October 2015 functions for computing isoclinism classes of crossed modules, written by Alper Odabaş and Enver Uslu, were added. These are contained in Chapter 4, and are described in detail in the paper [IOU16].

Bug reports, suggestions and comments are, of course, welcome. Please submit an issue at `https://github.com/gap-packages/xmod/issues/` or send an email to the first author at `cdwensley@btinternet.com`.

Copyright

Acknowledgements

This documentation was prepared using the GAPDoc [LN17] and AutoDoc [GH17] packages.

The procedure used to produce new releases uses the package GitHubPagesForGAP [Hor17] and the package ReleaseTools.

Contents

Chapter 1

Introduction

The XMod package provides functions for computation with

- finite crossed modules of groups and cat1-groups, and morphisms of these structures;
- finite pre-crossed modules, pre-cat1-groups, and their Peiffer quotients;
- derivations of crossed modules and sections of cat1-groups;
- isoclinism of groups and crossed modules;
- the actor crossed square of a crossed module;
- crossed squares, cat2-groups, and their morphisms (experimental version);
- crossed modules of groupoids (experimental version).

It is loaded with the command

Example

```
gap> LoadPackage( "xmod" );
```

The term crossed module was introduced by J. H. C. Whitehead in [Whi48], [Whi49]. Loday, in [Lod82], reformulated the notion of a crossed module as a cat1-group. Norrie [Nor90], [Nor87] and Gilbert [Gil90] have studied derivations, automorphisms of crossed modules and the actor of a crossed module, while Ellis [Ell84] has investigated higher dimensional analogues. Properties of induced crossed modules have been determined by Brown, Higgins and Wensley in [BH78], [BW95] and [BW96]. For further references see [AW00], where we discuss some of the data structures and algorithms used in this package, and also tabulate isomorphism classes of cat1-groups up to size 30.

XMod was originally implemented in 1997 using the GAP 3 language. In April 2002 the first and third parts were converted to GAP 4, the pre-structures were added, and version 2.001 was released. The final two parts, covering derivations, sections and actors, were included in the January 2004 release 2.002 for GAP 4.4. Many of the function names have been changed during the conversion, for example `ConjugationXMod` has become `XModByNormalSubgroup` (2.1.2). For a list of name changes see the file `names.pdf` in the `doc` directory.

In October 2015 Alper Odabaş and Enver Uslu were added to the list of package authors. Their functions for computing isoclinism classes of groups and crossed modules are contained in Chapter 4, and are described in detail in their paper [IOU16].

The package may be obtained as a compressed tar file XMod-version.number.tar.gz by ftp from one of the following sites:

- the XMod GitHub release site: https://github.com/gap-packages.github.io/xmod/.
- any GAP archive, e.g. https://www.gap-system.org/Packages/packages.html;

The package also has a GitHub repository at: https://github.com/gap-packages/xmod/.

Crossed modules and cat1-groups are special types of *2-dimensional groups* [Bro82], [BHS11], and are implemented as 2DimensionalDomains and 2DimensionalGroups having a Source and a Range.

The package divides into eight parts. The first part is concerned with the standard constructions for pre-crossed modules and crossed modules; together with direct products; normal sub-crossed modules; and quotients. Operations for constructing pre-cat1-groups and cat1-groups, and for converting between cat1-groups and crossed modules, are also included.

The second part is concerned with *morphisms* of (pre-)crossed modules and (pre-)cat1-groups, together with standard operations for morphisms, such as composition, image and kernel.

The third part is the most recent part of the package, introduced in October 2015. Additional operations and properties for crossed modules are included in Section 4.1. Then, in 4.2 and 4.3 there are functions for isoclinism of groups and crossed modules.

The fourth part is concerned with the equivalent notions of *derivation* for a crossed module and *section* for a cat1-group, and the monoids which they form under the Whitehead multiplication.

The fifth part deals with actor crossed modules and actor cat1-groups. For the actor crossed module $\mathrm{Act}(\mathscr{X})$ of a crossed module $\mathscr{X}$ we require representations for the Whitehead group of regular derivations of $\mathscr{X}$ and for the group of automorphisms of $\mathscr{X}$. The construction also provides an inner morphism from $\mathscr{X}$ to $\mathrm{Act}(\mathscr{X})$ whose kernel is the centre of $\mathscr{X}$.

The sixth part, which remains under development, contains functions to compute induced crossed modules.

Since version 2.007 there are experimental functions for *crossed squares* and their morphisms, structures which arise as 3-dimensional groups. Examples of these are inclusions of normal sub-crossed modules, and the inner morphism from a crossed module to its actor.

The eighth part has some experimental functions for crossed modules of groupoids, interacting with the package groupoids. Much more work on this is needed.

Future plans include the implementation of *group-graphs* which will provide examples of pre-crossed modules (their implementation will require interaction with graph-theoretic functions in GAP 4). There are also plans to implement cat2-groups, and conversion betwen these and crossed squares.

The equivalent categories XMod (crossed modules) and Cat1 (cat1-groups) are also equivalent to GpGpd, the subcategory of group objects in the category Gpd of groupoids. Finite groupoids have been implemented in Emma Moore's package groupoids [Moo01] for groupoids and crossed resolutions.

In order that the user has some control of the verbosity of the XMod package's functions, an InfoClass InfoXMod is provided (see Chapter ref:Info Functions in the GAP Reference Manual for a description of the Info mechanism). By default, the InfoLevel of InfoXMod is 0; progressively more information is supplied by raising the InfoLevel to 1, 2 and 3.

Example

```
gap> SetInfoLevel( InfoXMod, 1); #sets the InfoXMod level to 1
```

Once the package is loaded, the manual `doc/manual.pdf` can be found in the documentation folder. The `html` versions, with or without MathJax, should be rebuilt as follows:

Example

```
gap> ReadPackage( "xmod, "makedoc.g" );
```

It is possible to check that the package has been installed correctly by running the test files:

Example

```
gap> ReadPackage( "xmod", "tst/testall.g" );
#I  Testing .../pkg/xmod/tst/gp2obj.tst
...
```

Additional information can be found on the *Computational Higher-dimensional Discrete Algebra* website at: https://github.com/cdwensley.

Chapter 2

2d-groups : crossed modules and cat^1-groups

The term *2d-group* refers to a set of equivalent categories of which the most common are the categories of *crossed modules*; *cat*^{1}*-groups*; and *group-groupoids*, all of which involve a pair of groups.

2.1 Constructions for crossed modules

A crossed module (of groups) $\mathcal{X} = (\partial : S \to R)$ consists of a group homomorphism ∂, called the *boundary* of $\mathcal{X}$, with *source* S and *range* R. The group R acts on itself by conjugation, and on S by an *action* $\alpha : R \to \mathrm{Aut}(S)$ such that, for all $s, s_1, s_2 \in S$ and $r \in R$,

$$\textbf{XMod 1} : \partial(s^r) = r^{-1}(\partial s)r = (\partial s)^r, \qquad \textbf{XMod 2} : s_1^{\partial s_2} = s_2^{-1} s_1 s_2 = {s_1}^{s_2}.$$

When only the first of these axioms is satisfied, the resulting structure is a *pre-crossed module* (see section 2.3). The kernel of ∂ is abelian.

(Much of the literature on crossed modules uses left actions, but we have chosen to use right actions in this package since that is the standard choice for group actions in GAP.)

2.1.1 XMod

▷ XMod(*args*) (function)
▷ XModByBoundaryAndAction(*bdy, act*) (operation)

The global function `XMod` calls one of the standard constructions described in the following subsections. In the example the boundary is the identity mapping on `c5` and the action is trivial.

Example

```
gap> c5 := Group( (5,6,7,8,9) );;
gap> SetName( c5, "c5" );
gap> id5 := IdentityMapping( c5 );;
gap> ac5 := AutomorphismGroup( c5 );;
gap> act := MappingToOne( c5, ac5 );;
gap> XMod( id5, act ) = XModByBoundaryAndAction( id5, act );
true
```

2.1.2 XModByNormalSubgroup

▷ `XModByNormalSubgroup(G, N)` (operation)

A *conjugation crossed module* is the inclusion of a normal subgroup $S \trianglelefteq R$, where R acts on S by conjugation.

2.1.3 XModByTrivialAction

▷ `XModByTrivialAction(bdy)` (operation)

A *trivial action crossed module* $(\partial : S \to R)$ has $s^r = s$ for all $s \in S$, $r \in R$, the source is abelian and the image lies in the centre of the range.

Example

```
gap> q8 := QuaternionGroup( IsPermGroup, 8 );
Group([ (1,5,3,7)(2,8,4,6), (1,2,3,4)(5,6,7,8) ])
gap> SetName( q8, "q8" );
gap> c2 := Centre( q8 );
Group([ (1,3)(2,4)(5,7)(6,8) ])
gap> SetName( c2, "<-1>" );
gap> bdy := InclusionMappingGroups( q8, c2 );;
gap> X8a := XModByTrivialAction( bdy );
[<-1>->q8]
gap> c4 := Subgroup( q8, [q8.1] );;
gap> SetName( c4, "<i>" );
gap> X8b := XModByNormalSubgroup( q8, c4 );
[<i>->q8]
gap> Display(X8b);
Crossed module [<i>->q8] :-
: Source group has generators:
  [ (1,5,3,7)(2,8,4,6) ]
: Range group q8 has generators:
  [ (1,5,3,7)(2,8,4,6), (1,2,3,4)(5,6,7,8) ]
: Boundary homomorphism maps source generators to:
  [ (1,5,3,7)(2,8,4,6) ]
: Action homomorphism maps range generators to automorphisms:
  (1,5,3,7)(2,8,4,6) --> { source gens --> [ (1,5,3,7)(2,8,4,6) ] }
  (1,2,3,4)(5,6,7,8) --> { source gens --> [ (1,7,3,5)(2,6,4,8) ] }
  These 2 automorphisms generate the group of automorphisms.
```

2.1.4 XModByAutomorphismGroup

▷ `XModByAutomorphismGroup(grp)` (attribute)
▷ `XModByInnerAutomorphismGroup(grp)` (attribute)
▷ `XModByGroupOfAutomorphisms(G, A)` (operation)

An *automorphism crossed module* has as range a subgroup R of the automorphism group $\mathrm{Aut}(S)$ of S which contains the inner automorphism group of S. The boundary maps $s \in S$ to the inner automorphism of S by s.

Example

```
gap> X5 := XModByAutomorphismGroup( c5 );
[c5 -> Aut(c5)]
gap> Display( X5 );
Crossed module [c5->Aut(c5)] :-
: Source group c5 has generators:
  [ (5,6,7,8,9) ]
: Range group Aut(c5) has generators:
  [ GroupHomomorphismByImages( c5, c5, [ (5,6,7,8,9) ], [ (5,7,9,6,8) ] ) ]
: Boundary homomorphism maps source generators to:
  [ IdentityMapping( c5 ) ]
: Action homomorphism maps range generators to automorphisms:
  GroupHomomorphismByImages( c5, c5, [ (5,6,7,8,9) ],
[ (5,7,9,6,8) ] ) --> { source gens --> [ (5,7,9,6,8) ] }
  This automorphism generates the group of automorphisms.
```

2.1.5 XModByCentralExtension

▷ `XModByCentralExtension(`*bdy*`)` (operation)

A *central extension crossed module* has as boundary a surjection $\partial : S \to R$, with central kernel, where $r \in R$ acts on S by conjugation with $\partial^{-1}r$.

Example

```
gap> gen12 := [ (1,2,3,4,5,6), (2,6)(3,5) ];;
gap> d12 := Group( gen12 );;
gap> gen6 := [ (7,8,9), (8,9) ];;
gap> s3 := Group( gen6 );;
gap> SetName( d12, "d12" );  SetName( s3, "s3" );
gap> pr12 := GroupHomomorphismByImages( d12, s3, gen12, gen6 );;
gap> Kernel( pr12 ) = Centre( d12 );
true
gap> X12 := XModByCentralExtension( pr12 );;
gap> Display( X12 );
Crossed module [d12->s3] :-
: Source group d12 has generators:
  [ (1,2,3,4,5,6), (2,6)(3,5) ]
: Range group s3 has generators:
  [ (7,8,9), (8,9) ]
: Boundary homomorphism maps source generators to:
  [ (7,8,9), (8,9) ]
: Action homomorphism maps range generators to automorphisms:
  (7,8,9) --> { source gens --> [ (1,2,3,4,5,6), (1,3)(4,6) ] }
  (8,9) --> { source gens --> [ (1,6,5,4,3,2), (2,6)(3,5) ] }
  These 2 automorphisms generate the group of automorphisms.
```

2.1.6 XModByPullback

▷ `XModByPullback(xmod, hom)` (operation)

Let $\mathscr{X}_0 = (\mu : M \to P)$ be a crossed module. If $\nu : N \to P$ is a group homomorphism with the same range as $\mathscr{X}_0$, form the pullback group $L = M \times_P N$, with projection $\lambda : L \to N$ (as defined in the Utils package). Then N acts on L by $(m,n)^{n'} := (m^{\nu n'}, n^{n'})$, so that $\mathscr{X}_1 = (\lambda : L \to N)$ is the *pullback crossed module* determined by $\mathscr{X}_0$ and ν. There is also a morphism of crossed modules $(\kappa, \nu) : \mathscr{X}_1 \to \mathscr{X}_2$.

The example forms a pullback of the crossed module `X12` of the previous subsection.

Example

```
gap> gens4 := [ (11,12), (12,13), (13,14) ];;
gap> s4 := Group( gens4 );;
gap> theta := GroupHomomorphismByImages( s4, s3, gens4, [(7,8),(8,9),(7,8)] );;
gap> X1 := XModByPullback( X12, theta );;
gap> StructureDescription( Source( X1 ) );
"C2 x S4"
gap> SetName( s4, "s4" );  SetName( Source( X1 ), "c2s4" );
gap> infoX1 := PullbackInfo( Source( X1 ) );;
gap> infoX1!.directProduct;
Group([ (1,2,3,4,5,6), (2,6)(3,5), (7,8), (8,9), (9,10) ])
gap> infoX1!.projections[1];
[ (7,8)(9,10), (7,9)(8,10), (2,6)(3,5)(8,9), (1,5,3)(2,6,4)(8,10,9),
  (1,6,5,4,3,2)(8,9,10) ] -> [ (), (), (2,6)(3,5), (1,5,3)(2,6,4),
  (1,6,5,4,3,2) ]
gap> infoX1!.projections[2];
[ (7,8)(9,10), (7,9)(8,10), (2,6)(3,5)(8,9), (1,5,3)(2,6,4)(8,10,9),
  (1,6,5,4,3,2)(8,9,10) ] -> [ (11,12)(13,14), (11,13)(12,14), (12,13),
  (12,14,13), (12,13,14) ]
```

2.1.7 XModByAbelianModule

▷ `XModByAbelianModule(abmod)` (operation)

A *crossed abelian module* has an abelian module as source and the zero map as boundary. See section 14.2 for an example.

2.1.8 DirectProduct (for crossed modules)

▷ `DirectProduct(X1, X2)` (operation)

The direct product $\mathscr{X}_1 \times \mathscr{X}_2$ of two crossed modules has source $S_1 \times S_2$, range $R_1 \times R_2$ and boundary $\partial_1 \times \partial_2$, with R_1, R_2 acting trivially on S_2, S_1 respectively. The embeddings and projections are constructed automatically, and placed in the `DirectProductInfo` attribute, together with the two *objects* $\mathscr{X}_1 \times \mathscr{X}_2$.

The example constructs the product of the two crossed modules formed in subsection `XModByTrivialAction` (2.1.3).

Example

```
gap> X8ab := DirectProduct( X8a, X8b );
[[<-1>->q8]x[<i>->q8]]
gap> infoX8ab := DirectProductInfo( X8ab );
rec(
  embeddings := [ [[<-1>->q8] => [<-1>x<i>->q8xq8]],
      [[<i>->q8] => [<-1>x<i>->q8xq8]] ], objects := [ [<-1>->q8], [<i>->q8] ]
    ,
  projections := [ [[<-1>x<i>->q8xq8] => [<-1>->q8]],
      [[<-1>x<i>->q8xq8] => [<i>->q8]] ] )
gap> DirectProduct( X8a, X8b, X12 );
[[[<-1>->q8]x[<i>->q8]]x[d12->s3]]
```

2.1.9 Source (for crossed modules)

▷ `Source(X0)` (attribute)
▷ `Range(X0)` (attribute)
▷ `Boundary(X0)` (attribute)
▷ `XModAction(X0)` (attribute)

The following attributes are used in the construction of a crossed module `X0`.

- `Source(X0)` and `Range(X0)` are the source S and range R of ∂, the boundary `Boundary(X0)`;
- `XModAction(X0)` is a homomorphism from R to a group of automorphisms of `X0`.

(Up until version 2.63 there was an additional attribute `AutoGroup`, the range of `XModAction(X0)`.)

The example uses the crossed module `X12` constructed in subsection `XModByCentralExtension` (2.1.5).

Example

```
gap> [ Source( X12 ), Range( X12 ) ];
[ d12, s3 ]
gap> Boundary( X12 );
[ (1,2,3,4,5,6), (2,6)(3,5) ] -> [ (7,8,9), (8,9) ]
gap> XModAction( X12 );
[ (7,8,9), (8,9) ] ->
[ [ (1,2,3,4,5,6), (2,6)(3,5) ] -> [ (1,2,3,4,5,6), (1,3)(4,6) ],
  [ (1,2,3,4,5,6), (2,6)(3,5) ] -> [ (1,6,5,4,3,2), (2,6)(3,5) ] ]
```

2.1.10 ImageElmXModAction

▷ `ImageElmXModAction(X0, s, r)` (operation)

This function returns the element s^r given by `XModAction(X0)`.

Example

```
gap> ImageElmXModAction( X12, (1,2,3,4,5,6), (8,9) );
(1,6,5,4,3,2)
```

2.1.11 Size2d (for crossed modules)

▷ `Size2d(X0)` (attribute)

The standard operation `Size` cannot be used for crossed modules because the size of a collection is required to be a number, and we wish to return a list. `Size2d( X0 )` returns the two-element list, `[ Size( Source(X0) ), Size( Range(X0) ) ]`.

In the simple example below, `X5` is the automorphism crossed module constructed in subsection `XModByAutomorphismGroup` (2.1.4).

Example

```
gap> Size2d( X5 );
[ 5, 4 ]
```

2.1.12 Name (for crossed modules)

▷ `Name(X0)` (attribute)
▷ `IdGroup(X0)` (attribute)
▷ `ExternalSetXMod(X0)` (attribute)

More familiar attributes are `Name` and `IdGroup`. The name is formed by concatenating the names of the source and range (if these exist). `IdGroup( X0 )` returns a two-element list `[ IdGroup( Source(X0) ), IdGroup( Range(X0) ) ]`.

The `ExternalSetXMod` for a crossed module is the source group considered as a G-set of the range group using the crossed module action.

The `Display` function is used to print details of 2d-groups.

The `Print` statements at the end of the example list the GAP representations and attributes of `X5`.

Example

```
gap> IdGroup( X5 );
[ [ 5, 1 ], [ 4, 1 ] ]
gap> ext := ExternalSetXMod( X5 );
<xset:[ (), (5,6,7,8,9), (5,7,9,6,8), (5,8,6,9,7), (5,9,8,7,6) ]>
gap> Orbits( ext );
[ [ () ], [ (5,6,7,8,9), (5,7,9,6,8), (5,9,8,7,6), (5,8,6,9,7) ] ]
gap> a := GeneratorsOfGroup( Range( X5 ) )[1]^2;
[ (5,6,7,8,9) ] -> [ (5,9,8,7,6) ]
gap> ImageElmXModAction( X5, (5,7,9,6,8), a );
(5,8,6,9,7)
gap> Print( RepresentationsOfObject(X5), "\n" );
[ "IsComponentObjectRep", "IsAttributeStoringRep", "IsPreXModObj" ]
gap> Print( KnownAttributesOfObject(X5), "\n" );
```

```
[ "Name", "Range", "Source", "IdGroup", "Boundary", "Size2d", "XModAction",
  "ExternalSetXMod", "HigherDimension" ]
```

2.2 Properties of crossed modules

The underlying category structures for the objects constructed in this chapter follow the sequence Is2DimensionalDomain; Is2DimensionalMagma; Is2DimensionalMagmaWithOne; Is2DimensionalMagmaWithInverses, mirroring the situation for (one-dimensional) groups. From these we construct Is2DimensionalSemigroup, Is2DimensionalMonoid and Is2DimensionalGroup.

There are then a variety of properties associated with crossed modules, starting with IsPreXMod and IsXMod.

2.2.1 IsXMod

▷ IsXMod(*X0*) (property)
▷ IsPreXMod(*X0*) (property)
▷ IsPerm2DimensionalGroup(*X0*) (property)
▷ IsPc2DimensionalGroup(*X0*) (property)
▷ IsFp2DimensionalGroup(*X0*) (property)

A structure which has IsPerm2DimensionalGroup is a precrossed module or a pre-cat^1-group (see section 2.4) whose source and range are both permutation groups. The properties IsPc2DimensionalGroup, IsFp2DimensionalGroup are defined similarly. In the example below we see that X5 has IsPreXMod, IsXMod and IsPerm2DimensionalGroup. There are also properties corresponding to the various construction methods listed in section 2.1: IsTrivialAction2DimensionalGroup; IsNormalSubgroup2DimensionalGroup; IsCentralExtension2DimensionalGroup; IsAutomorphismGroup2DimensionalGroup; IsAbelianModule2DimensionalGroup.

Example

```
gap> [ IsTrivial( X5 ),  IsNonTrivial( X5 ),  IsFinite( X5 ) ];
[ false, true, true ]
gap> kpoX5 := KnownPropertiesOfObject(X5);;
gap> ForAll( [ "IsTrivial", "IsNonTrivial", "IsFinite",
>  "CanEasilyCompareElements", "CanEasilySortElements", "IsDuplicateFree",
>  "IsGeneratorsOfSemigroup", "IsPreXModDomain", "IsPreXMod", "IsXMod",
>  "IsAutomorphismGroup2DimensionalGroup" ],
>  s -> s in kpoX5 );
true
```

2.2.2 SubXMod

▷ SubXMod(*X0, src, rng*) (operation)
▷ TrivialSubXMod(*X0*) (attribute)

▷ `NormalSubXMods(X0)` (attribute)

With the standard crossed module constructors listed above as building blocks, sub-crossed modules, normal sub-crossed modules $\mathscr{N} \lhd \mathscr{X}$, and also quotients $\mathscr{X}/\mathscr{N}$ may be constructed. A sub-crossed module $\mathscr{S} = (\delta : N \to M)$ is *normal* in $\mathscr{X} = (\partial : S \to R)$ if

- N, M are normal subgroups of S, R respectively,
- δ is the restriction of ∂,
- $n^r \in N$ for all $n \in N,\ r \in R$,
- $(s^{-1})^m s \in N$ for all $m \in M,\ s \in S$.

These conditions ensure that $M \ltimes N$ is normal in the semidirect product $R \ltimes S$. (Note that $\langle s,m \rangle = (s^{-1})^m s$ is a displacement: see `Displacement` (4.1.3).)

A method for `IsNormal` for precrossed modules is provided. See section 4.1 for factor crossed modules and their natural morphisms.

The five normal subcrossed modules of `X4` found in the following example are `[id,id]`, `[k4,k4]`, `[k4,a4]`, `[a4,a4]` and `X4` itself.

Example

```
gap> s4 := Group( (1,2), (2,3), (3,4) );;
gap> a4 := Subgroup( s4, [ (1,2,3), (2,3,4) ] );;
gap> k4 := Subgroup( a4, [ (1,2)(3,4), (1,3)(2,4) ] );;
gap> SetName(s4,"s4");  SetName(a4,"a4");  SetName(k4,"k4");
gap> X4 := XModByNormalSubgroup( s4, a4 );
[a4->s4]
gap> Y4 := SubXMod( X4, k4, a4 );
[k4->a4]
gap> IsNormal(X4,Y4);
true
gap> NX4 := NormalSubXMods( X4 );;
gap> Length( NX4 );
5
```

2.2.3 KernelCokernelXMod

▷ `KernelCokernelXMod(X0)` (attribute)

Let $\mathscr{X} = (\partial : S \to R)$. If $K \leqslant S$ is the kernel of ∂, and $J \leqslant R$ is the image of ∂, form $C = R/J$. Then $(\nu\partial|_K : K \to C)$ is a crossed module where $\nu : R \to C, r \mapsto Jr$ is the natural map, and the action of C on K is given by $k^{Jr} = k^r$.

Example

```
gap> d8d8 := Group( (1,2,3,4), (1,3), (5,6,7,8), (5,7) );;
gap> X88 := XModByAutomorphismGroup( d8d8 );;
gap> Size2d( X88 );
[ 64, 2048 ]
gap> Y88 := KernelCokernelXMod( X88 );;
```

```
gap> IdGroup(Y88);
[ [ 4, 2 ], [ 128, 928 ] ]
gap> StructureDescription( Y88 );
[ "C2 x C2", "(D8 x D8) : C2" ]
```

2.3 Pre-crossed modules

2.3.1 PreXModByBoundaryAndAction

▷ PreXModByBoundaryAndAction(*bdy, act*) (operation)
▷ PreXModWithTrivialRange(*src, rng*) (operation)
▷ SubPreXMod(*X0, src, rng*) (operation)

If axiom **XMod 2** is *not* satisfied, the corresponding structure is known as a *pre-crossed module*.

A special case of this operation is when the range is a trivial group (not necessarily a subgroup of the source), and so the action is trivial. This case will be used when constructing a special type of double groupoid in Chapter 11.

Example

```
gap> b1 := (11,12,13,14,15,16,17,18);;  b2 := (12,18)(13,17)(14,16);;
gap> d16 := Group( b1, b2 );;
gap> sk4 := Subgroup( d16, [ b1^4, b2 ] );;
gap> SetName( d16, "d16" );  SetName( sk4, "sk4" );
gap> bdy16 := GroupHomomorphismByImages( d16, sk4, [b1,b2], [b1^4,b2] );;
gap> aut1 := GroupHomomorphismByImages( d16, d16, [b1,b2], [b1^5,b2] );;
gap> aut2 := GroupHomomorphismByImages( d16, d16, [b1,b2], [b1,b2^4*b2] );;
gap> aut16 := Group( [ aut1, aut2 ] );;
gap> act16 := GroupHomomorphismByImages( sk4, aut16, [b1^4,b2], [aut1,aut2] );;
gap> P16 := PreXModByBoundaryAndAction( bdy16, act16 );
[d16->sk4]
gap> IsXMod( P16 );
false
gap> Q16 := PreXModWithTrivialRange( d16, d16 );
[d16->Group( [ () ] )]
gap> SQ16 := SubPreXMod( Q16, sk4, Group( [()] ) );;
gap> Display(SQ16);
Crossed module :-
: Source group has generators:
  [ (11,15)(12,16)(13,17)(14,18), (12,18)(13,17)(14,16) ]
: Range group has generators:
  [ () ]
: Boundary homomorphism maps source generators to:
  [ (), () ]
  The automorphism group is trivial
```

2.3.2 PeifferSubgroup

▷ `PeifferSubgroup(X0)` (attribute)
▷ `XModByPeifferQuotient(prexmod)` (attribute)

The *Peiffer subgroup* P of a pre-crossed module $\mathscr{X}$ is the subgroup of $\ker(\partial)$ generated by *Peiffer commutators*

$$\lfloor s_1, s_2 \rfloor \;=\; (s_1^{-1})^{\partial s_2}\, s_2^{-1}\, s_1\, s_2 \;=\; \langle \partial s_2, s_1 \rangle\, [s_1, s_2]\,.$$

Then $\mathscr{P} = (0 : P \to \{1_R\})$ is a normal sub-pre-crossed module of $\mathscr{X}$ and $\mathscr{X}/\mathscr{P} = (\partial : S/P \to R)$ is a crossed module.

In the following example the Peiffer subgroup is cyclic of size 4.

Example

```
gap> P := PeifferSubgroup( P16 );
Group( [ (11,15)(12,16)(13,17)(14,18), (11,17,15,13)(12,18,16,14) ] )
gap> X16 := XModByPeifferQuotient( P16 );
Peiffer([d16->sk4])
gap> Display( X16 );
Crossed module Peiffer([d16->sk4]) :-
: Source group has generators:
  [ f1, f2 ]
: Range group has generators:
  [ (11,15)(12,16)(13,17)(14,18), (12,18)(13,17)(14,16) ]
: Boundary homomorphism maps source generators to:
  [ (12,18)(13,17)(14,16), (11,15)(12,16)(13,17)(14,18) ]
  The automorphism group is trivial
gap> iso16 := IsomorphismPermGroup( Source( X16 ) );;
gap> S16 := Image( iso16 );
Group([ (1,2), (3,4) ])
```

2.4 Cat^1-groups and pre-cat^1-groups

In [Lod82], Loday reformulated the notion of a crossed module as a cat^1-group, namely a group G with a pair of endomorphisms $t,h : G \to G$ having a common image R and satisfying certain axioms. We find it computationally convenient to define a cat^1-group $\mathscr{C} = (e;t,h : G \to R)$ as having source group G, range group R, and three homomorphisms: two surjections $t,h : G \to R$ and an embedding $e : R \to G$ satisfying:

$$\textbf{Cat 1}: \; t \circ e \;=\; h \circ e = \mathrm{id}_R, \qquad \textbf{Cat 2}: \; [\ker t, \ker h] \;=\; \{1_G\}.$$

It follows that $t \circ e \circ h = h$, $h \circ e \circ t = t$, $t \circ e \circ t = t$ and $h \circ e \circ h = h$. (See section 2.5 for the case when t,h are endomorphisms.)

2.4.1 Cat1Group

▷ `Cat1Group(args)` (function)
▷ `PreCat1Group(args)` (function)
▷ `PreCat1GroupByTailHeadEmbedding(t, h, e)` (operation)

▷ `PreCat1GroupWithIdentityEmbedding(t, h)` (operation)

The global functions `Cat1Group` and `PreCat1Group` can be called in various ways.

- as `Cat1Group(t,h,e);` when t,h,e are three homomorphisms, which is equivalent to `PreCat1GroupByTailHeadEmbedding(t,h,e);`
- as `Cat1Group(t,h);` when t,h are two endomorphisms, which is equivalent to `PreCat1GroupWithIdentityEmbedding(t,h);`
- as `Cat1Group(t);` when $t = h$ is an endomorphism, which is equivalent to `PreCat1GroupWithIdentityEmbedding(t,t);`
- as `Cat1Group(t,e);` when $t = h$ and e are homomorphisms, which is equivalent to `PreCat1GroupByTailHeadEmbedding(t,t,e);`
- as `Cat1Group(i,j,k);` when i, j, k are integers, which is equivalent to `Cat1Select(i,j,k);` as described in section 2.7.

Example

```
gap> g18gens := [ (1,2,3), (4,5,6), (2,3)(5,6) ];;
gap> s3agens := [ (7,8,9), (8,9) ];;
gap> g18 := Group( g18gens );;  SetName( g18, "g18" );
gap> s3a := Group( s3agens );;  SetName( s3a, "s3a" );
gap> t1 := GroupHomomorphismByImages(g18,s3a,g18gens,[(7,8,9),(),(8,9)]);
[ (1,2,3), (4,5,6), (2,3)(5,6) ] -> [ (7,8,9), (), (8,9) ]
gap> h1 := GroupHomomorphismByImages(g18,s3a,g18gens,[(7,8,9),(7,8,9),(8,9)]);
[ (1,2,3), (4,5,6), (2,3)(5,6) ] -> [ (7,8,9), (7,8,9), (8,9) ]
gap> e1 := GroupHomomorphismByImages(s3a,g18,s3agens,[(1,2,3),(2,3)(5,6)]);
[ (7,8,9), (8,9) ] -> [ (1,2,3), (2,3)(5,6) ]
gap> C18 := Cat1Group( t1, h1, e1 );
[g18=>s3a]
```

2.4.2 Source (for cat1-groups)

▷ `Source(C)` (attribute)
▷ `Range(C)` (attribute)
▷ `TailMap(C)` (attribute)
▷ `HeadMap(C)` (attribute)
▷ `RangeEmbedding(C)` (attribute)
▷ `KernelEmbedding(C)` (attribute)
▷ `Boundary(C)` (attribute)
▷ `Name(C)` (attribute)
▷ `Size2d(C)` (attribute)

These are the attributes of a cat^1-group $\mathcal{C}$ in this implementation.

The maps t,h are often referred to as the *source* and *target*, but we choose to call them the *tail* and *head* of $\mathcal{C}$, because *source* is the GAP term for the domain of a function. The `RangeEmbedding` is the embedding of `R` in `G`, the `KernelEmbedding` is the inclusion of the kernel of `t` in `G`, and the `Boundary` is the restriction of h to the kernel of t. It is frequently the case that $t = h$, but not in the example C18 above.

Example

```
gap> [ Source( C18 ), Range( C18 ) ];
[ g18, s3a ]
gap> TailMap( C18 );
[ (1,2,3), (4,5,6), (2,3)(5,6) ] -> [ (7,8,9), (), (8,9) ]
gap> HeadMap( C18 );
[ (1,2,3), (4,5,6), (2,3)(5,6) ] -> [ (7,8,9), (7,8,9), (8,9) ]
gap> RangeEmbedding( C18 );
[ (7,8,9), (8,9) ] -> [ (1,2,3), (2,3)(5,6) ]
gap> Kernel( C18 );
Group([ (4,5,6) ])
gap> KernelEmbedding( C18 );
[ (4,5,6) ] -> [ (4,5,6) ]
gap> Name( C18 );
"[g18=>s3a]"
gap> Size2d( C18 );
[ 18, 6 ]
gap> StructureDescription( C18 );
[ "(C3 x C3) : C2", "S3" ]
```

The next four subsections contain some more constructors for cat^1-groups.

2.4.3 DiagonalCat1Group

▷ DiagonalCat1Group(*genG*) (operation)

This operation constructs examples of cat^1-groups of the form $G \times G \Rightarrow G$. The tail map is the identity on the first factor and kills of the second, while the head map does the reverse. The range embedding maps G to the diagonal in $G \times G$. The corresponding crossed module is isomorphic to the identity crossed module on G.

Example

```
gap> C4 := DiagonalCat1Group( [ (1,2,3), (2,3,4) ] );;
gap> SetName( Source(C4), "a4a4" );  SetName( Range(C4_, "a4d" );
gap> Display( C4 );
Cat1-group [a4a4=>a4d] :-
: Source group a4a4 has generators:
  [ (1,2,3), (2,3,4), (5,6,7), (6,7,8) ]
: Range group a4d has generators:
  [ ( 9,10,11), (10,11,12) ]
: tail homomorphism maps source generators to:
  [ ( 9,10,11), (10,11,12), (), () ]
: head homomorphism maps source generators to:
  [ (), (), ( 9,10,11), (10,11,12) ]
: range embedding maps range generators to:
  [ (1,2,3)(5,6,7), (2,3,4)(6,7,8) ]
: kernel has generators:
```

```
  [ (5,6,7), (6,7,8) ]
: boundary homomorphism maps generators of kernel to:
  [ ( 9,10,11), (10,11,12) ]
: kernel embedding maps generators of kernel to:
  [ (5,6,7), (6,7,8) ]
```

2.4.4 TransposeCat1Group

▷ `TransposeCat1Group(C0)` (attribute)
▷ `TransposeIsomorphism(C0)` (attribute)

The *transpose* of a cat^1-group C has the same source, range and embedding, but has the tail and head maps interchanged. The `TransposeIsomorphism` gives the isomorphism between the two.

Example

```
gap> R4 := TransposeCat1Group( C4 );
[a4a4=>a4d]
gap> Boundary( R4 );
[ (2,3,4), (1,2,3) ] -> [ (10,11,12), (9,10,11) ]
gap> TailMap( R4 ) = HeadMap( R4 );
false
gap> TailMap( R4 ) = HeadMap( C4 );
true
gap> MappingGeneratorsImages( TransposeIsomorphism(C4) );
[ [ [ (1,2,3), (2,3,4), (5,6,7), (6,7,8) ],
      [ (5,6,7), (6,7,8), (1,2,3), (2,3,4) ] ],
  [ [ (9,10,11), (10,11,12) ], [ (9,10,11), (10,11,12) ] ] ]
```

2.4.5 Cat1GroupByPeifferQuotient

▷ `Cat1GroupByPeifferQuotient(P)` (operation)

If $C = (e;t,h : G \to R)$ is a pre-cat^1-group, its Peiffer subgroup is $P = [\ker t, \ker h]$ and the associated cat^1-group C_2 has source G/P. In the example, $t = h : s4 \to c2$ with $\ker t = \ker h = a4$ and $P = [a4, a4] = k4$, so that $G/P = s4/k4 \cong s3$.

Example

```
gap> s4 := Group( (1,2,3), (3,4) );;  SetName( s4, "s4" );
gap> h := GroupHomomorphismByImages( s4, s4, [(1,2,3),(3,4)], [(),(3,4)] );;
gap> c2 := Image( h );;  SetName( c2, "c2" );
gap> C := PreCat1Group( h, h );
[s4=>c2]
gap> P := PeifferSubgroupPreCat1Group( C );
Group([ (1,3)(2,4), (1,2)(3,4) ])
gap> C2 := Cat1GroupByPeifferQuotient( C );
[Group( [ f1, f2 ] )=>c2]
gap> StructureDescription( C2 );
[ "S3", "C2" ]
```

```
gap> rec2 := PreXModRecordOfPreCat1Group( C );;
gap> XC := rec2.prexmod;;
gap> StructureDescription( XC );
[ "A4", "C2" ]
gap> XC2 := XModByPeifferQuotient( XC );;
gap> StructureDescription( XC2 );
[ "C3", "C2" ]
gap> CXC2 := Cat1GroupOfXMod( XC2 );;
gap> StructureDescription( CXC2 );
[ "S3", "C2" ]
gap> IsomorphismCat1Groups( C2, CXC2 );
[[Group( [ f1, f2 ] ) => c2] => [(..|X..) => c2]]
```

2.4.6 SubCat1Group

▷ SubCat1Group(*C1*, *S1*) (operation)

▷ SubPreCat1Group(*C1*, *S1*) (operation)

S_1 is a sub-cat^1-group of C_1 provided the source and range of S_1 are subgroups of the source and range of C_1 and that the tail, head and embedding of S_1 are the appropriate restrictions of those of C_1.

Example

```
gap> s3 := Subgroup( s4, [(2,3),(3,4)] );;
gap> res := DoublyRestrictedMapping( h, s3, s3 );;
gap> S := PreCat1Group( res, res );
[Group( [ (2,3), (3,4) ] )=>Group( [ (3,4), (3,4) ] )]
```

2.4.7 DirectProduct (for cat1-groups)

▷ DirectProduct(*C1*, *C2*) (operation)

The direct product $\mathscr{C}_1 \times \mathscr{C}_2$ of two cat^1-groups has source $G_1 \times G_2$ and range $R_1 \times R_2$. The tail, head and embedding maps are $t_1 \times t_2$, $h_1 \times h_2$ and $e_1 \times e_2$. The embeddings and projections are constructed automatically, and placed in the `DirectProductInfo` attribute, together with the two *objects* $\mathscr{C}_1$ and $\mathscr{C}_2$.

The example constructs the product of two of the cat^1-groups constructed above.

Example

```
gap> C418 := DirectProduct( C4, C18 );
[(a4a4xg18)=>(a4d x s3a)]
gap> infoC418 := DirectProductInfo( C418 );
rec(
  embeddings := [ [[a4a4=>a4d] => [(a4a4xg18)=>(a4d x s3a)]],
      [[g18=>s3a] => [(a4a4xg18)=>(a4d x s3a)]] ],
  objects := [ [a4a4=>a4d], [g18=>s3a] ],
  projections := [ [[(a4a4xg18)=>(a4d x s3a)] => [a4a4=>a4d]],
      [[(a4a4xg18)=>(a4d x s3a)] => [g18=>s3a]] ] )
```

```
gap> t418 := TailMap( C418 );
[ (1,2,3), (2,3,4), (5,6,7), (6,7,8), (9,10,11), (12,13,14), (10,11)(13,14)
 ] -> [ (1,2,3), (2,3,4), (), (), (5,6,7), (), (6,7) ]
gap> h418 := HeadMap( C418 );
[ (1,2,3), (2,3,4), (5,6,7), (6,7,8), (9,10,11), (12,13,14), (10,11)(13,14)
 ] -> [ (), (), (1,2,3), (2,3,4), (5,6,7), (5,6,7), (6,7) ]
gap> e418 := RangeEmbedding( C418 );
[ (1,2,3), (2,3,4), (5,6,7), (6,7) ] -> [ (1,2,3)(5,6,7), (2,3,4)(6,7,8),
  (9,10,11), (10,11)(13,14) ]
```

2.5 Properties of cat^1-groups and pre-cat^1-groups

Many of the properties listed in section 2.2 apply to pre-cat^1-groups and to cat^1-groups since these are also 2d-groups. There are also more specific properties.

2.5.1 IsCat1Group

▷ `IsCat1Group(C0)` (property)
▷ `IsPreXCat1Group(C0)` (property)
▷ `IsIdentityCat1Group(C0)` (property)

`IsIdentityCat1Group(C0)` is true when the head and tail maps of `C0` are identity mappings.

Example

```
gap> G8 := SmallGroup( 288, 956 );  SetName( G8, "G8" );
<pc group of size 288 with 7 generators>
gap> d12 := DihedralGroup( 12 );  SetName( d12, "d12" );
<pc group of size 12 with 3 generators>
gap> a1 := d12.1;;  a2 := d12.2;;  a3 := d12.3;;  a0 := One( d12 );;
gap> gensG8 := GeneratorsOfGroup( G8 );;
gap> t8 := GroupHomomorphismByImages( G8, d12, gensG8,
>             [ a0, a1*a3, a2*a3, a0, a0, a3, a0 ] );;
gap> h8 := GroupHomomorphismByImages( G8, d12, gensG8,
>             [ a1*a2*a3, a0, a0, a2*a3, a0, a0, a3^2 ] );;
gap> e8 := GroupHomomorphismByImages( d12, G8, [a1,a2,a3],
>          [ G8.1*G8.2*G8.4*G8.6^2, G8.3*G8.4*G8.6^2*G8.7, G8.6*G8.7^2 ] );
[ f1, f2, f3 ] -> [ f1*f2*f4*f6^2, f3*f4*f6^2*f7, f6*f7^2 ]
gap> C8 := PreCat1GroupByTailHeadEmbedding( t8, h8, e8 );
[G8=>d12]
gap> IsCat1Group( C8 );
true
gap> KnownPropertiesOfObject( C8 );
[ "CanEasilyCompareElements", "CanEasilySortElements", "IsDuplicateFree",
  "IsGeneratorsOfSemigroup", "IsPreCat1Domain", "IsPc2DimensionalGroup",
  "IsPreXMod", "IsPreCat1Group", "IsCat1Group", "IsIdentityPreCat1Group",
  "IsPreCat1GroupWithIdentityEmbedding" ]
```

2.5.2 IsPreCat1GroupWithIdentityEmbedding

▷ `IsPreCat1GroupWithIdentityEmbedding(C0)` (property)
▷ `IsomorphicPreCat1GroupWithIdentityEmbedding(C0)` (attribute)
▷ `IsomorphismToPreCat1GroupWithIdentityEmbedding(C0)` (attribute)

`IsPreCat1GroupWithIdentityEmbedding(C0)` is true when the range embedding of `C0` is an inclusion mapping. (This property used to be called `IsPreCat1GroupByEndomorphisms` but, as the example below shows, when the tail and head maps are endomorphisms the range embedding need not be an inclusion.) When this is not the case, replacing t,h,e by $t*e, h*e$ and the inclusion mapping of the image of e gives an isomorphic cat^1-group for which `IsPreCat1GroupWithIdentityEmbedding` is true. This is the `IsomorphicPreCat1GroupWithIdentityEmbedding` of `C0` and `IsomorphismToPreCat1GroupWithIdentityEmbedding` is the isomorphism between them. (See the next chapter for mappings of cat^1-groups.)

Example
```
gap> G5 := Group( (1,2,3,4,5) );;
gap> t := GroupHomomorphismByImages( G5, G5, [(1,2,3,4,5)], [(1,5,4,3,2)] );;
gap> PC5 := PreCat1GroupByTailHeadEmbedding( t, t, t );
[Group( [ (1,2,3,4,5) ] )=>Group( [ (1,2,3,4,5) ] )]
gap> IsPreCat1GroupWithIdentityEmbedding( PC5 );
false
gap> IPC5 := IsomorphicPreCat1GroupWithIdentityEmbedding( PC5 );
[Group( [ (1,2,3,4,5) ] )=>Group( [ (1,2,3,4,5) ] )]
gap> TailMap( IPC5 ); RangeEmbedding( IPC5 );
[ (1,2,3,4,5) ] -> [ (1,2,3,4,5) ]
[ (1,2,3,4,5) ] -> [ (1,2,3,4,5) ]
```

2.5.3 Cat1GroupOfXMod

▷ `Cat1GroupOfXMod(X0)` (attribute)
▷ `XModOfCat1Group(C0)` (attribute)
▷ `PreCat1GroupRecordOfPreXMod(P0)` (attribute)
▷ `PreXModRecordOfPreCat1Group(P0)` (attribute)

The category of crossed modules is equivalent to the category of cat^1-groups, and the functors between these two categories may be described as follows. Starting with the crossed module $\mathscr{X} = (\partial : S \to R)$ the group G is defined as the semidirect product $G = R \ltimes S$ using the action from $\mathscr{X}$, with multiplication rule

$$(r_1,s_1)(r_2,s_2) \;=\; (r_1r_2, {s_1}^{r_2}s_2).$$

The structural morphisms are given by

$$t(r,s) = r, \quad h(r,s) = r(\partial s), \quad er = (r,1).$$

On the other hand, starting with a cat^1-group $\mathscr{C} = (e;t,h : G \to R)$, we define $S = \ker t$, the range R is unchanged, and $\partial = h|_S$. The action of R on S is conjugation in G via the embedding of R in G.

As from version 2.74, the attribute `PreCat1GroupRecordOfPreXMod` of a pre-crossed modute $X = (\partial : S \to R)$ returns a record with fields

- .precat1, the pre-cat1-group $C = (e;t,h : G \to R)$ of X, where $G = R \ltimes S$;
- .iscat1, true if C is a cat1-group;
- .xmodSourceEmbedding, the image S' of S in G;
- .xmodSourceEmbeddingIsomorphism, the isomorphism $S \to S'$;
- .xmodRangeEmbedding, the image R' of R in G;
- .xmodRangeEmbeddingIsomorphism, the isomorphism $R \to R'$;

Example

```
gap> X8 := XModOfCat1Group( C8 );;
gap> Display( X8 );

Crossed module X([G8=>d12]) :-
: Source group has generators:
  [ f1, f4, f5, f7 ]
: Range group d12 has generators:
  [ f1, f2, f3 ]
: Boundary homomorphism maps source generators to:
  [ f1*f2*f3, f2*f3, <identity> of ..., f3^2 ]
: Action homomorphism maps range generators to automorphisms:
  f1 --> { source gens --> [ f1*f5, f4*f5, f5, f7^2 ] }
  f2 --> { source gens --> [ f1*f5*f7^2, f4, f5, f7 ] }
  f3 --> { source gens --> [ f1*f7, f4, f5, f7 ] }
  These 3 automorphisms generate the group of automorphisms.
: associated cat1-group is [G8=>d12]

gap> StructureDescription(X8);
[ "D24", "D12" ]
```

2.6 Enumerating cat^1-groups with a given source

As the size of a group G increases, the number of cat^1-groups with source G increases rapidly. However, one is usually only interested in the isomorphism classes of cat^1-groups with source G. An iterator AllCat1GroupsIterator is provided, which runs through the various cat^1-groups. This iterator finds, for each subgroup R of G, the cat^1-groups with range R. It does this by running through the AllSubgroupsIterator(G) provided by the Utils package, and then using the iterator AllCat1GroupsWithImageIterator(G,R).

2.6.1 AllCat1GroupsWithImage

▷ AllCat1GroupsWithImage(*G*, *R*) (operation)
▷ AllCat1GroupsWithImageIterator(*G*, *R*) (operation)
▷ AllCat1GroupsWithImageNumber(*G*, *R*) (attribute)
▷ AllCat1GroupsWithImageUpToIsomorphism(*G*, *R*) (operation)

The iterator `AllCat1GroupsWithImageIterator(G,R)` iterates through all the cat^1-groups with source `G` and range R. The attribute `AllCat1GroupsWithImageNumber(G)` runs through this iterator to count the number n_R of these cat^1-groups. The operation `AllCat1GroupsWithImage(G)` returns a list containing these n_R cat^1-groups. Since these lists can get very long, this operation should only be used for simple cases. The operation `AllCat1GroupsWithImageUpToIsomorphism(G)` returns representatives of the isomorphism classes of these cat^1-groups.

Example

```
gap> d12 := DihedralGroup( IsPermGroup, 12 );  SetName( d12, "d12" );
Group([ (1,2,3,4,5,6), (2,6)(3,5) ])
gap> c2 := Subgroup( d12, [ (1,6)(2,5)(3,4) ] );;
gap> AllCat1GroupsWithImageNumber( d12, c2 );
1
gap> L12 := AllCat1GroupsWithImage( d12, c2 );
[ [d12=>Group( [ (), (1,6)(2,5)(3,4) ] )] ]
```

2.6.2 AllCat1GroupsMatrix

▷ `AllCat1GroupsMatrix(G)` (attribute)

The operation `AllCat1GroupsMatrix(G)` constructs a symmetric matrix M with rows and columns labelled by the idempotent endomorphisms e_i on G, where $M_{ij} = 2$ if e_i, e_j combine to form a cat^1-group; $M_{ij} = 1$ if they only form a pre-cat^1-group; and $M_{ij} = 0$ otherwise. The matrix is automatically printed out with dots in place of zeroes.

In the example we see that the group QD_{16} has 10 idempotent endomorphisms and 5 cat^1-groups, all of which are symmetric ($t = h$), and a further 9 pre-cat^1-groups, 5 of which are symmetric. (A cat^1-group and its transpose are not counted twice.) This operation is intended to be used to illustrate how cat^1-groups are formed, and should only be used with groups of low order.

The attribute `AllCat1GroupsNumber(G)` returns the number n of these cat^1-groups.

Example

```
gap> qd16 := SmallGroup( 16, 8 );;
gap> AllCat1GroupsMatrix( qd16 );;
number of idempotent endomorphisms found = 10
number of cat1-groups found = 5
number of additional pre-cat1-groups found = 9
1.........
.21.......
.11.......
...21.....
...11.....
.....21...
.....11...
.......21.
.......11.
.........2
```

2.6.3 AllCat1GroupsIterator

▷ `AllCat1GroupsIterator(G)` (operation)
▷ `AllCat1GroupsUpToIsomorphism(G)` (operation)
▷ `AllCat1Groups(G)` (operation)

The iterator `AllCat1GroupsIterator(G)` iterates through all the cat^1-groups with source `G`. The operation `AllCat1Groups(G)` returns a list containing these n cat^1-groups. Since these lists can get very long, this operation should only be used for simple cases. The operation `AllCat1GroupsUpToIsomorphism(G)` returns representatives of the isomorphism classes of these subgroups.

Example

```
gap> iter := AllCat1GroupsIterator( d12 );;
gap> AllCat1GroupsNumber( d12 );
12
gap> iso12 := AllCat1GroupsUpToIsomorphism( d12 );
[ [d12=>Group( [ (), (2,6)(3,5) ] )],
  [d12=>Group( [ (1,4)(2,5)(3,6), (2,6)(3,5) ] )],
  [d12=>Group( [ (1,5,3)(2,6,4), (2,6)(3,5) ] )],
  [d12=>Group( [ (1,2,3,4,5,6), (2,6)(3,5) ] )] ]
```

2.6.4 CatnGroupNumbers (for cat1-groups)

▷ `CatnGroupNumbers(G)` (attribute)
▷ `CatnGroupLists(G)` (attribute)
▷ `InitCatnGroupRecords(G)` (operation)

The attribute `CatnGroupNumbers` for a group G is a mutable record which stores numbers of cat^1-groups, cat^2-groups, etc. as they are calculated. The field `CatnGroupNumbers(G).idem` is the number of idempotent endomorphisms of G. Similarly, `CatnGroupNumbers(G).cat1` is the number of cat^1-groups on G, while `CatnGroupNumbers(G).iso1` is the number of isomorphism classes of these cat^1-groups. Also `CatnGroupNumbers(G).symm` is the number of cat^1-groups whose `TailMap` is the same as the `HeadMap`, while `CatnGroupNumbers(G).siso` is the number of isomorphism classes of these symmetric cat^1-groups. Symmetric cat^1-groups are in one-one correspondence with symmetric cat^2-groups. The attribute `CatnGroupLists` is used for storing results of cat^2-group calculations.

Example

```
gap> CatnGroupNumbers( d12 );
rec( cat1 := 12, idem := 21, iso1 := 4, siso := 4, symm := 12 )
```

2.7 Selection of a small cat^1-group

The `Cat1Group` function may also be used to select a cat^1-group from a data file. All cat^1-structures on groups of size up to 60 (ordered according to the GAP 4 numbering of small groups) are stored in a list in file `cat1data.g`. Global variables `CAT1_LIST_MAX_SIZE := 60`, `CAT1_LIST_CLASS_SIZES` and `CAT1_LIST_NUMBERS` are also stored. The second of these just stores the number of isomorphism classes of groups of size `size`. The third stores the numbers of isomorphism classes of cat^1-groups for each of these groups. The data is read into the list `CAT1_LIST` only when this function is called.

This data was available in early versions of XMod with groups up to order 70 covered. More recently a larger range of groups has become available in the package HAP. The authors are indebted to Van Luyen Le in Galway for pointing out a number of errors in the version of this list distributed up to version 2.24 of this package.

2.7.1 Cat1Select

▷ `Cat1Select(size, gpnum, num)` (operation)

The function `Cat1Select` returns the cat^1-group numbered `num` whose source is the group `G := SmallGroup(size,gpnum)`. When $|G| \leqslant 60$ the data file in this package is used. For larger groups `SmallCat1Group` (see 13.1) is called, accessing the datafile in package HAP.

The example below is the first case in which $t \neq h$ and the associated conjugation crossed module is given by the normal subgroup `c3` of `s3`.

Example

```
gap> L18 := Cat1Select( 18 );
Usage:  Cat1Select( size, gpnum, num );  where gpnum <= 5
fail
gap> ## check the number of cat1-structures on the fourth group of order 18
gap> Cat1Select( 18, 4 );
Usage:  Cat1Select( size, gpnum, num );  where num <= 4
fail
gap> ## select the second of these cat1-structures
gap> B18 := Cat1Select( 18, 4, 2 );
[(C3 x C3) : C2=>Group( [ f1, <identity> of ..., f3 ] )]
gap> ## convert from a pc-cat1-group to a permutation cat1-group
gap> iso18 := IsomorphismPermObject( B18 );;
gap> PB18 := Image( iso18 );;
gap> Display( PB18 );
Cat1-group :-
: Source group has generators:
  [ (4,5,6), (1,2,3), (2,3)(5,6) ]
: Range group has generators:
  [ (1,2,3), (2,3)(5,6) ]
: tail homomorphism maps source generators to:
  [ (), (1,2,3), (2,3)(5,6) ]
: head homomorphism maps source generators to:
  [ (), (1,2,3), (2,3)(5,6) ]
: range embedding maps range generators to:
  [ (1,2,3), (2,3)(5,6) ]
: kernel has generators:
  [ (4,5,6) ]
: boundary homomorphism maps generators of kernel to:
  [ () ]
: kernel embedding maps generators of kernel to:
  [ (4,5,6) ]
```

```
: associated crossed module is [Group( [ (4,5,6) ] ) -> Group(
[ (1,2,3), (2,3)(5,6) ] )]
gap> convert the result to the associated permutation crossed module
gap> Y18 := XModOfCat1Group( PB18 );;
gap> Display( Y18 );
Crossed module :-
: Source group has generators:
  [ (4,5,6) ]
: Range group has generators:
  [ (1,2,3), (2,3)(5,6) ]
: Boundary homomorphism maps source generators to:
  [ () ]
: Action homomorphism maps range generators to automorphisms:
  (1,2,3) --> { source gens --> [ (4,5,6) ] }
  (2,3)(5,6) --> { source gens --> [ (4,6,5) ] }
  These 2 automorphisms generate the group of automorphisms.
: associated cat1-group is [Group( [ (4,5,6), (1,2,3), (2,3)(5,6)
 ] ) => Group( [ (1,2,3), (2,3)(5,6) ] )]
```

2.8 More functions for crossed modules and cat^1-groups

Chapter 4 contains functions for quotient crossed modules; centre of a crossed module; commutator and derived subcrossed modules; etc.

Here we mention two functions for groups which have been extended to the two-dimensional case.

2.8.1 IdGroup (for 2d-groups)

▷ IdGroup(*2DimensionalGroup*) (operation)
▷ StructureDescription(*2DimensionalGroup*) (operation)

These functions return two-element lists formed by applying the function to the source and range of the 2d-group.

Example

```
gap> IdGroup( X8 );
[ [ 24, 6 ], [ 12, 4 ] ]
gap> StructureDescription( C8 );
[ "(S3 x D24) : C2", "D12" ]
```

There are also a number of functions which test for sub-structures.

2.8.2 IsSubXMod

▷ IsSubXMod(*X0, S0*) (operation)
▷ IsSubPreXMod(*X0, S0*) (operation)
▷ IsSubCat1Group(*G0, R0*) (operation)
▷ IsSubPreCat1Group(*G0, R0*) (operation)

▷ IsSub2DimensionalGroup(*G0, R0*) (operation)

These functions test whether the second argument is a sub-2d-group of the first argument. The examples refer back to sub-2d-groups created in sections 2.2 and 2.4.

Example

```
gap> IsSubXMod( X4, Y4 );
true
gap> IsSubPreCat1Group( C, S );
true
```

2.9 The group groupoid associated to a cat^1-group

A *group groupoid* is an algebraic object which is both a groupoid and a group. The category of group groupoids is equivalent to the categories of precrossed modules and precat^1-groups. Starting with a (pre)cat^1-group $\mathscr{C} = (e;t,h : G \to R)$, we form the groupoid $\mathscr{G}$ having the elements of R as objects and the elements of G as arrows. The arrow g has tail tg and head hg. $\mathscr{G}$ has one connected component for each coset of tG in R.

The groupoid (partial) multiplication $*$ on these arrows is defined by:

$$(g_1 : r_1 \to r_2) * (g_2 : r_2 \to r_3) \;=\; (g_1(er_2^{-1})g_2 : r_1 \to r_3).$$

2.9.1 GroupGroupoid

▷ GroupGroupoid(*precat1*) (attribute)

The operation GroupGroupoid implements this construction. In the example we start with a crossed module $(C_3^2 \to S_3)$, form the associated cat^1-group $(S_3 \ltimes C_3^2 \Rightarrow S_3)$, and then form the group groupoid gpd33. Since the image of the boundary of the crossed module is C_3, with index 2 in the range, the groupoid has two connected components, and the root objects are $\{(),(12,13)\}$. The size of the vertex groups is $|\ker t \cap \ker h| = 3$, and the generators at the root objects are $() \to (4,5,6)(7,9,8) \to ()$ and $(12,13) \to (2,3)(4,6)(7,8) \to (12,13)$.

Example

```
gap> s3 := Group( (11,12), (12,13) );;
gap> c3c3 := Group( [ (14,15,16), (17,18,19) ] );;
gap> bdy := GroupHomomorphismByImages( c3c3, s3,
>         [(14,15,16),(17,18,19)], [(11,12,13),(11,12,13)] );;
gap> a := GroupHomomorphismByImages( c3c3, c3c3,
>         [(14,15,16),(17,18,19)], [(14,16,15),(17,19,18)] );;
gap> aut := Group( [a] );;
gap> act := GroupHomomorphismByImages( s3, aut, [(11,12),(12,13)], [a,a] );;
gap> X33 := XModByBoundaryAndAction( bdy, act );;
gap> C33 := Cat1GroupOfXMod( X33 );;
gap> G33 := Source( C33 );;
gap> gpd33 := GroupGroupoid( C33 );
groupoid with 2 pieces:
1:  single piece groupoid with rays: < Group( [ ()>-(4,5,6)(7,9,8)->() ] ),
```

```
[ (), (11,12,13), (11,13,12) ], [ ()>-()->(), ()>-(7,8,9)->(11,12,13),
  ()>-(7,9,8)->(11,13,12) ] >
2:  single piece groupoid with rays: < Group(
[ (12,13)>-(2,3)(4,6)(7,8)->(12,13) ] ), [ (12,13), (11,12), (11,13) ],
[ (12,13)>-(2,3)(5,6)(8,9)->(12,13), (12,13)>-(2,3)(5,6)(7,9)->(11,13),
  (12,13)>-(2,3)(5,6)(7,8)->(11,12) ] >
```

2.9.2 GroupGroupoidElement

▷ GroupGroupoidElement(*precat1, root, g*) (operation)

Since we need to define a second multiplication on the elements of G, we have to convert $g \in G$ into a new type of object, `GroupGroupoidElementType`, a record e with fields:

- `e!.precat1`, the precat^1-group from which $\mathscr{G}$ was formed;
- `e!.root`, the root object of the component containing e;
- `e!.element`, the element $g \in G$;
- `e!.tail`, the tail object of the element e;
- `e!.head`, the head object of the element e;
- `e!.tailid`, the identity element at the tail object;
- `e!.headid`, the identity element at the head object;

In the example we pick a particular pair of elements $g_1, g_2 \in G$, construct group groupoid elements e_1, e_2 from them, and show that $g_1 * g_2$ and $e_1 * e_2$ give very different results. (Warning: at present iterators for object groups and homsets do not work.)

Example

```
gap> piece2 := Pieces( gpd33 )[2];;
gap> obs2 := piece2!.objects;
[ (12,13), (11,12), (11,13) ]
gap> RaysOfGroupoid( piece2 );
[ (12,13)>-(2,3)(5,6)(8,9)->(12,13), (12,13)>-(2,3)(5,6)(7,9)->(11,13),
  (12,13)>-(2,3)(5,6)(7,8)->(11,12) ]
gap> g1 := (1,2)(5,6)(7,9);;
gap> g2 := (2,3)(4,5)(7,8);;
gap> g1 * g2;
(1,3,2)(4,5,6)(7,9,8)
gap> e1 := GroupGroupoidElement( C33, (12,13), g1 );
(11,12)>-(1,2)(5,6)(7,9)->(12,13)
gap> e2 := GroupGroupoidElement( C33, (12,13), g2 );
(12,13)>-(2,3)(4,5)(7,8)->(11,13)
gap> e1*e2;
(11,12)>-(1,2)(4,5)(8,9)->(11,13)
gap> e2^-1;
(11,13)>-(1,3)(4,6)(7,9)->(12,13)
```

```
gap> obgp := ObjectGroup( gpd33, (11,12) );;
gap> GeneratorsOfGroup( obgp )[1];
(11,13)>-( 1, 3)( 4, 6)( 7, 8)->(11,13)
gap> Homset( gpd33, (11,12), (11,13) );
<homset (11,12) -> (11,13) with head group Group(
[ (11,12)>-( 1, 2)( 4, 6)( 7, 8)->(11,12) ] )>
```

Chapter 3

2d-mappings

3.1 Morphisms of 2-dimensional groups

This chapter describes morphisms of (pre-)crossed modules and (pre-)cat1-groups.

3.1.1 Source (for 2d-group mappings)

▷ `Source(map)` (attribute)
▷ `Range(map)` (attribute)
▷ `SourceHom(map)` (attribute)
▷ `RangeHom(map)` (attribute)

Morphisms of *2-dimensional groups* are implemented as *2-dimensional mappings*. These have a pair of 2-dimensional groups as source and range, together with two group homomorphisms mapping between corresponding source and range groups. These functions return `fail` when invalid data is supplied.

3.2 Morphisms of pre-crossed modules

3.2.1 IsXModMorphism

▷ `IsXModMorphism(map)` (property)
▷ `IsPreXModMorphism(map)` (property)

A morphism between two pre-crossed modules $\mathcal{X}_1 = (\partial_1 : S_1 \to R_1)$ and $\mathcal{X}_2 = (\partial_2 : S_2 \to R_2)$ is a pair (σ, ρ), where $\sigma : S_1 \to S_2$ and $\rho : R_1 \to R_2$ commute with the two boundary maps and are morphisms for the two actions:

$$\partial_2 \circ \sigma \ = \ \rho \circ \partial_1, \qquad \sigma(s^r) \ = \ (\sigma s)^{\rho r}.$$

Here σ is the `SourceHom` (3.1.1) and ρ is the `RangeHom` (3.1.1) of the morphism. When $\mathcal{X}_1 = \mathcal{X}_2$ and σ, ρ are automorphisms then (σ, ρ) is an automorphism of $\mathcal{X}_1$. The group of automorphisms is denoted by $\mathrm{Aut}(\mathcal{X}_1)$.

3.2.2 IsInjective (for pre-xmod morphisms)

▷ `IsInjective(map)` (method)
▷ `IsSurjective(map)` (method)
▷ `IsSingleValued(map)` (method)
▷ `IsTotal(map)` (method)
▷ `IsBijective(map)` (method)
▷ `IsEndo2DimensionalMapping(map)` (property)

The usual properties of mappings are easily checked. It is usually sufficient to verify that both the `SourceHom` (3.1.1) and the `RangeHom` (3.1.1) have the required property.

3.2.3 XModMorphism

▷ `XModMorphism(args)` (function)
▷ `XModMorphismByGroupHomomorphisms(X1, X2, sigma, rho)` (operation)
▷ `PreXModMorphism(args)` (function)
▷ `PreXModMorphismByGroupHomomorphisms(P1, P2, sigma, rho)` (operation)
▷ `InclusionMorphism2DimensionalDomains(X1, S1)` (operation)
▷ `InnerAutomorphismXMod(X1, r)` (operation)
▷ `IdentityMapping(X1)` (attribute)

These are the constructors for morphisms of pre-crossed and crossed modules.

In the following example we construct a simple automorphism of the crossed module `X5` constructed in the previous chapter.

Example

```
gap> sigma5 := GroupHomomorphismByImages( c5, c5, [ (5,6,7,8,9) ]
        [ (5,9,8,7,6) ] );;
gap> rho5 := IdentityMapping( Range( X1 ) );
IdentityMapping( PAut(c5) )
gap> mor5 := XModMorphism( X5, X5, sigma5, rho5 );
[[c5->Aut(c5))] => [c5->Aut(c5))]]
gap> Display( mor5 );
Morphism of crossed modules :-
: Source = [c5->Aut(c5)] with generating sets:
  [ (5,6,7,8,9) ]
  [ GroupHomomorphismByImages( c5, c5, [ (5,6,7,8,9) ], [ (5,7,9,6,8) ] ) ]
: Range = Source
: Source Homomorphism maps source generators to:
  [ (5,9,8,7,6) ]
: Range Homomorphism maps range generators to:
  [ GroupHomomorphismByImages( c5, c5, [ (5,6,7,8,9) ], [ (5,7,9,6,8) ] ) ]
gap> IsAutomorphism2DimensionalDomain( mor5 );
true
gap> Order( mor5 );
2
gap> RepresentationsOfObject( mor5 );
[ "IsComponentObjectRep", "IsAttributeStoringRep", "Is2DimensionalMappingRep" ]
gap> KnownPropertiesOfObject( mor5 );
```

```
[ "CanEasilyCompareElements", "CanEasilySortElements", "IsTotal",
  "IsSingleValued", "IsInjective", "IsSurjective", "RespectsMultiplication",
  "IsPreXModMorphism", "IsXModMorphism", "IsEndomorphism2DimensionalDomain",
  "IsAutomorphism2DimensionalDomain" ]
gap> KnownAttributesOfObject( mor5 );
[ "Name", "Order", "Range", "Source", "SourceHom", "RangeHom" ]
```

3.2.4 IsomorphismPerm2DimensionalGroup (for pre-xmod morphisms)

▷ `IsomorphismPerm2DimensionalGroup(obj)` (attribute)
▷ `IsomorphismPc2DimensionalGroup(obj)` (attribute)
▷ `IsomorphismByIsomorphisms(D, list)` (operation)

When $\mathscr{D}$ is a 2-dimensional domain with source S and range R and $\sigma : S \to S'$, $\rho : R \to R'$ are isomorphisms, then `IsomorphismByIsomorphisms(D,[sigma,rho])` returns an isomorphism $(\sigma,\rho) : \mathscr{D} \to \mathscr{D}'$ where $\mathscr{D}'$ has source S' and range R'. Be sure to test `IsBijective` for the two functions σ,ρ before applying this operation.

Using `IsomorphismByIsomorphisms` with a pair of isomorphisms obtained using `IsomorphismPermGroup` or `IsomorphismPcGroup`, we may construct a crossed module or a cat1-group of permutation groups or pc-groups.

Example

```
gap> q8 := SmallGroup(8,4);;   ## quaternion group
gap> XAq8 := XModByAutomorphismGroup( q8 );
[Group( [ f1, f2, f3 ] )->Group( [ Pcgs([ f1, f2, f3 ]) -> [ f1*f2, f2, f3 ],
  Pcgs([ f1, f2, f3 ]) -> [ f2, f1*f2, f3 ],
  Pcgs([ f1, f2, f3 ]) -> [ f1*f3, f2, f3 ],
  Pcgs([ f1, f2, f3 ]) -> [ f1, f2*f3, f3 ] ] )]
gap> iso := IsomorphismPerm2DimensionalGroup( XAq8 );;
gap> YAq8 := Image( iso );
[Group( [ (1,2,4,6)(3,8,7,5), (1,3,4,7)(2,5,6,8), (1,4)(2,6)(3,7)(5,8)
 ] )->Group( [ (1,3,4,6), (1,2,3)(4,5,6), (1,4)(3,6), (2,5)(3,6) ] )]
gap> s4 := SymmetricGroup(4);;
gap> isos4 := IsomorphismGroups( Range(YAq8), s4 );;
gap> id := IdentityMapping( Source( YAq8 ) );;
gap> IsBijective( id );;  IsBijective( isos4 );;
gap> mor := IsomorphismByIsomorphisms( YAq8, [id,isos4] );;
gap> ZAq8 := Image( mor );
[Group( [ (1,2,4,6)(3,8,7,5), (1,3,4,7)(2,5,6,8), (1,4)(2,6)(3,7)(5,8)
 ] )->SymmetricGroup( [ 1 .. 4 ] )]
```

3.2.5 MorphismOfPullback (for a crossed module by pullback)

▷ `MorphismOfPullback(xmod)` (attribute)

Let $\mathscr{X}_1 = (\lambda : L \to N)$ be the pullback crossed module obtained from a crossed module $\mathscr{X}_0 = (\mu : M \to P)$ and a group homomorphism $\nu : N \to P$. Then the associated crossed module morphism is

$(\kappa,\nu) : \mathscr{X}_1 \to \mathscr{X}_0$ where κ is the projection from L to M.

3.3 Morphisms of pre-cat1-groups

A morphism of pre-cat1-groups from $\mathscr{C}_1 = (e_1;t_1,h_1 : G_1 \to R_1)$ to $\mathscr{C}_2 = (e_2;t_2,h_2 : G_2 \to R_2)$ is a pair (γ,ρ) where $\gamma : G_1 \to G_2$ and $\rho : R_1 \to R_2$ are homomorphisms satisfying

$$h_2 \circ \gamma \;=\; \rho \circ h_1, \qquad t_2 \circ \gamma \;=\; \rho \circ t_1, \qquad e_2 \circ \rho \;=\; \gamma \circ e_1.$$

3.3.1 IsCat1GroupMorphism

▷ `IsCat1GroupMorphism(map)` (property)
▷ `IsPreCat1GroupMorphism(map)` (property)
▷ `Cat1GroupMorphism(args)` (function)
▷ `Cat1GroupMorphismByGroupHomomorphisms(C1, C2, gamma, rho)` (operation)
▷ `PreCat1GroupMorphism(args)` (function)
▷ `PreCat1GroupMorphismByGroupHomomorphisms(P1, P2, gamma, rho)` (operation)
▷ `InclusionMorphism2DimensionalDomains(C1, S1)` (operation)
▷ `InnerAutomorphismCat1(C1, r)` (operation)
▷ `IdentityMapping(C1)` (attribute)

For an example we form a second cat1-group `C2=[g18=>s3a]`, similar to `C1` in 2.4.1, then construct an isomorphism (γ,ρ) between them.

Example

```
gap> t3 := GroupHomomorphismByImages(g18,s3a,g18gens,[(),(7,8,9),(8,9)]);;
gap> e3 := GroupHomomorphismByImages(s3a,g18,s3agens,[(4,5,6),(2,3)(5,6)]);;
gap> C3 := Cat1Group( t3, h1, e3 );;
gap> imgamma := [ (4,5,6), (1,2,3), (2,3)(5,6) ];;
gap> gamma := GroupHomomorphismByImages( g18, g18, g18gens, imgamma );;
gap> rho := IdentityMapping( s3a );;
gap> phi3 := Cat1GroupMorphism( C18, C3, gamma, rho );;
gap> Display( phi3 );;
Morphism of cat1-groups :-
: Source = [g18=>s3a] with generating sets:
  [ (1,2,3), (4,5,6), (2,3)(5,6) ]
  [ (7,8,9), (8,9) ]
:  Range = [g18=>s3a] with generating sets:
  [ (1,2,3), (4,5,6), (2,3)(5,6) ]
  [ (7,8,9), (8,9) ]
: Source Homomorphism maps source generators to:
  [ (4,5,6), (1,2,3), (2,3)(5,6) ]
: Range Homomorphism maps range generators to:
  [ (7,8,9), (8,9) ]
```

3.3.2 Cat1GroupMorphismOfXModMorphism

▷ `Cat1GroupMorphismOfXModMorphism(IsXModMorphism)` (attribute)
▷ `XModMorphismOfCat1GroupMorphism(IsCat1GroupMorphism)` (attribute)

If $(\sigma,\rho) : \mathscr{X}_1 \to \mathscr{X}_2$ and $\mathscr{C}_1, \mathscr{C}_2$ are the cat^1-groups accociated to $\mathscr{X}_1, \mathscr{X}_2$, then the associated morphism of cat^1-groups is (γ,ρ) where $\gamma(r_1,s_1) = (\rho r_1, \sigma s_1)$.

Similarly, given a morphism $(\gamma,\rho) : \mathscr{C}_1 \to \mathscr{C}_2$ of cat1-groups, the associated morphism of crossed modules is $(\sigma,\rho) : \mathscr{X}_1 \to \mathscr{X}_2$ where $\sigma s_1 = \gamma(1,s_1)$. .

Example

```
gap> phi5 := Cat1GroupMorphismOfXModMorphism( mor5 );
[[(Aut(c5) |X c5)=>Aut(c5)] => [(Aut(c5) |X c5)=>Aut(c5)]]
gap> mor3 := XModMorphismOfCat1GroupMorphism( phi3 );;
gap> Display( mor3 );
Morphism of crossed modules :-
: Source = xmod([g18=>s3a]) with generating sets:
  [ (4,5,6) ]
  [ (7,8,9), (8,9) ]
:  Range = xmod([g18=>s3a]) with generating sets:
  [ (1,2,3) ]
  [ (7,8,9), (8,9) ]
: Source Homomorphism maps source generators to:
  [ (1,2,3) ]
: Range Homomorphism maps range generators to:
  [ (7,8,9), (8,9) ]
```

3.3.3 IsomorphismPermObject

▷ `IsomorphismPermObject(`*obj*`)` (function)

▷ `IsomorphismPerm2DimensionalGroup(`*2DimensionalGroup*`)` (attribute)

▷ `IsomorphismFp2DimensionalGroup(`*2DimensionalGroup*`)` (attribute)

▷ `IsomorphismPc2DimensionalGroup(`*2DimensionalGroup*`)` (attribute)

▷ `RegularActionHomomorphism2DimensionalGroup(`*2DimensionalGroup*`)` (attribute)

The global function `IsomorphismPermObject` calls `IsomorphismPerm2DimensionalGroup`, which constructs a morphism whose `SourceHom` (3.1.1) and `RangeHom` (3.1.1) are calculated using `IsomorphismPermGroup` on the source and range.

The global function `RegularActionHomomorphism2DimensionalGroup` is similar, but uses `RegularActionHomomorphism` in place of `IsomorphismPermGroup`.

Example

```
gap> iso8 := IsomorphismPerm2DimensionalGroup( C8 );
[[G8=>d12] => [..]]
```

3.3.4 SmallerDegreePermutationRepresentation2DimensionalGroup (for perm 2d-groups)

▷ `SmallerDegreePermutationRepresentation2DimensionalGroup(`*Perm2DimensionalGroup*`)` (attribute)

The attribute `SmallerDegreePermutationRepresentation2DimensionalGroup` is obtained by calling `SmallerDegreePermutationRepresentation` on the source and range to obtain the an isomorphism for the pre-xmod or pre-cat^1-group.

Example

```
gap> G := Group( (1,2,3,4)(5,6,7,8) );;
gap> H := Subgroup( G, [ (1,3)(2,4)(5,7)(6,8) ] );;
gap> XG := XModByNormalSubgroup( G, H );
[Group( [ (1,3)(2,4)(5,7)(6,8) ] )->Group( [ (1,2,3,4)(5,6,7,8) ] )]
gap> sdpr := SmallerDegreePermutationRepresentation2DimensionalGroup( XG );;
gap> Range( sdpr );
[Group( [ (1,2) ] )->Group( [ (1,2,3,4) ] )]
```

3.4 Operations on morphisms

3.4.1 CompositionMorphism

▷ `CompositionMorphism(map2, map1)` (operation)

Composition of morphisms (written `(<map1> * <map2>)` when maps act on the right) calls the `CompositionMorphism` function for maps (acting on the left), applied to the appropriate type of 2d-mapping.

Example

```
gap> H8 := Subgroup(G8,[G8.3,G8.4,G8.6,G8.7]);  SetName( H8, "H8" );
Group([ f3, f4, f6, f7 ])
gap> c6 := Subgroup( d12, [b,c] );  SetName( c6, "c6" );
Group([ f2, f3 ])
gap> SC8 := Sub2DimensionalGroup( C8, H8, c6 );
[H8=>c6]
gap> IsCat1Group( SC8 );
true
gap> inc8 := InclusionMorphism2DimensionalDomains( C8, SC8 );
[[H8=>c6] => [G8=>d12]]
gap> CompositionMorphism( iso8, inc );
[[H8=>c6] => P[G8=>d12]]
```

3.4.2 Kernel (for 2d-mappings)

▷ `Kernel(map)` (operation)
▷ `Kernel2DimensionalMapping(map)` (attribute)

The kernel of a morphism of crossed modules is a normal subcrossed module whose groups are the kernels of the source and target homomorphisms. The inclusion of the kernel is a standard example of a crossed square, but these have not yet been implemented.

Example

```
gap> c2 := Group( (19,20) );
Group([ (19,20) ])
gap> X0 := XModByNormalSubgroup( c2, c2 );  SetName( X0, "X0" );
[Group( [ (19,20) ] )->Group( [ (19,20) ] )]
gap> SX8 := Source( X8 );;
gap> genSX8 := GeneratorsOfGroup( SX8 );
[ f1, f4, f5, f7 ]
gap> sigma0 := GroupHomomorphismByImages(SX8,c2,genSX8,[(19,20),(),(),()]);
[ f1, f4, f5, f7 ] -> [ (19,20), (), (), () ]
gap> rho0 := GroupHomomorphismByImages(d12,c2,[a1,a2,a3],[(19,20),(),()]);
[ f1, f2, f3 ] -> [ (19,20), (), () ]
gap> mor0 := XModMorphism( X8, X0, sigma0, rho0 );;
gap> K0 := Kernel( mor0 );;
gap> StructureDescription( K0 );
[ "C12", "C6" ]
```

3.5 Quasi-isomorphisms

A morphism of crossed modules $\phi : \mathscr{X} = (\partial : S \to R) \to \mathscr{X}' = (\partial' : S' \to R')$ induces homomorphisms $\pi_1(\phi) : \pi_1(\partial) \to \pi_1(\partial')$ and $\pi_2(\phi) : \pi_2(\partial) \to \pi_2(\partial')$. A morphism ϕ is a *quasi-isomorphism* if both $\pi_1(\phi)$ and $\pi_2(\phi)$ are isomorphisms. Two crossed modules $\mathscr{X}, \mathscr{X}'$ are *quasi-isomorphic* is there exists a sequence of quasi-isomorphisms

$$\mathscr{X} = \mathscr{X}_1 \leftrightarrow \mathscr{X}_2 \leftrightarrow \mathscr{X}_3 \leftrightarrow \cdots \longleftrightarrow \mathscr{X}_\ell = \mathscr{X}'$$

of length $\ell - 1$. Here $\mathscr{X}_i \leftrightarrow \mathscr{X}_j$ means that *either* $\mathscr{X}_i \to \mathscr{X}_j$ *or* $\mathscr{X}_j \to \mathscr{X}_i$. When $\mathscr{X}, \mathscr{X}'$ are quasi-isomorphic we write $\mathscr{X} \simeq \mathscr{X}'$. Clearly $\simeq$ is an equivalence relation. Mac\ Lane and Whitehead in [MLW50] showed that there is a one-to-one correspondence between homotopy 2-types and quasi-isomorphism classes. We say that $\mathscr{X}$ represents a *trivial* quasi-isomorphism class if $\partial = 0$.

Two cat^1-groups are quasi-isomorphic if their corresponding crossed modules are. The procedure for constructing a representative for the quasi-isomorphism class of a cat^1-group $\mathscr{C}$, as described by Ellis and Le in [EL14], is as follows. The *quotient process* consists of finding all normal sub-crossed modules $\mathscr{N}$ of the crossed module $\mathscr{X}$ associated to $\mathscr{C}$; constructing the quotient crossed module morphisms $\nu : \mathscr{X} \to \mathscr{X}/\mathscr{N}$; and converting these ν to morphisms from $\mathscr{C}$.

The *sub-crossed module process* consists of finding all sub-crossed modules $\mathscr{S}$ of $\mathscr{X}$ such that the inclusion $\iota : \mathscr{S} \to \mathscr{X}$ is a quasi-isomorphism; then converting ι to a morphism to $\mathscr{C}$.

The procedure for finding all quasi-isomorphism reductions consists of repeating the quotient process, followed by the sub-crossed module process, until no further reductions are possible.

It may happen that $\mathscr{C}_1 \simeq \mathscr{C}_2$ without either having a quasi-isomorphism reduction. In this case it is necessary to find a suitable $\mathscr{C}_3$ with reductions $\mathscr{C}_3 \to \mathscr{C}_1$ and $\mathscr{C}_3 \to \mathscr{C}_2$. No such automated process is available in XMod.

Functions for these computations were first implemented in the package HAP and are available as `QuotientQuasiIsomorph`, `SubQuasiIsomorph` and `QuasiIsomorph`.

3.5.1 QuotientQuasiIsomorphism

▷ `QuotientQuasiIsomorphism(cat1, bool)` (operation)

This function implements the quotient process. The second parameter is a boolean which, when true, causes the results of some intermediate calculations to be printed. The output shows the identity of the reduced cat1-group, if there is one.

Example

```
gap> C18a := Cat1Select( 18, 4, 4 );;
gap> StructureDescription( C18a );
[ "(C3 x C3) : C2", "S3" ]
gap> QuotientQuasiIsomorphism( C18a, true );
quo: [ f2 ][ f3 ], [ "1", "C2" ]
[ [ 2, 1 ], [ 2, 1 ] ], [ 2, 1, 1 ]
[ [ 2, 1, 1 ] ]
```

3.5.2 SubQuasiIsomorphism

▷ `SubQuasiIsomorphism(cat1, bool)` (operation)

This function implements the sub-crossed module process.

Example

```
gap> SubQuasiIsomorphism( C18a, false );
[ [ 2, 1, 1 ], [ 2, 1, 1 ], [ 2, 1, 1 ] ]
```

3.5.3 QuasiIsomorphism

▷ `QuasiIsomorphism(cat1, list, bool)` (operation)

This function implements the general process.

Example

```
gap> L18a := QuasiIsomorphism( C18a, [18,4,4], false );
[ [ 2, 1, 1 ], [ 18, 4, 4 ] ]
```

The logs above show that `C18a` has just one normal sub-crossed module $\mathscr{N}$ leading to a reduction, and that there are three sub-crossed modules $\mathscr{S}$ all giving the same reduction. The conclusion is that `C18a` is quasi-isomorphic to the identity cat1-group on the cyclic group of order 2.

Chapter 4

Isoclinism of groups and crossed modules

This chapter describes some functions written by Alper Odabaş and Enver Uslu, and reported in their paper [IOU16]. Section 4.1 contains some additional basic functions for crossed modules, constructing quotients, centres, centralizers and normalizers. In Sections 4.2 and 4.3 there are functions dealing specifically with isoclinism for groups and for crossed modules. Since these functions represent a recent addition to the package (as of November 2015), the function names are liable to change in future versions. The notion of isoclinism has been crucial to the enumeration of groups of prime power order, see for example James, Newman and O'Brien, [JNO90].

4.1 More operations for crossed modules

4.1.1 FactorPreXMod

▷ FactorPreXMod(*X1, X2*) (operation)

▷ NaturalMorphismByNormalSubPreXMod(*X1, X2*) (operation)

When $\mathscr{X}_2 = (\partial_2 : S_2 \to R_2)$ is a normal sub-precrossed module of $\mathscr{X}_1 = (\partial_1 : S_1 \to R_1)$, then the quotient precrossed module is $(\partial : S_2/S_1 \to R_2/R_1)$ with the induced boundary and action maps. Quotienting a precrossed module by it's Peiffer subgroup is a special case of this construction. (Permutation representations vary between different versions of GAP.)

Example

```
gap> d24 := DihedralGroup( IsPermGroup, 24 );;
gap> SetName( d24, "d24" );
gap> Y24 := XModByAutomorphismGroup( d24 );;
gap> Size2d( Y24 );
[ 24, 48 ]
gap> X24 := Image( IsomorphismPerm2DimensionalGroup( Y24 ) );
[d24->Group([ (2,4), (1,2,3,4), (6,7), (5,6,7) ])]
gap> nsx := NormalSubXMods( X24 );;
gap> Length( nsx );
40
gap> ids := List( nsx, n -> IdGroup(n) );;
gap> pos1 := Position( ids, [ [4,1], [8,3] ] );;
gap> Xn1 := nsx[pos1];
[Group( [ f2*f4^2, f3*f4 ] )->Group( [ f3, f4, f5 ] )]
```

```
gap> nat1 := NaturalMorphismByNormalSubPreXMod( X24, Xn1 );;
gap> Qn1 := FactorPreXMod( X24, Xn1 );;
gap> [ Size2d( Xn1 ), Size2d( Qn1 ) ];
[ [ 4, 8 ], [ 6, 6 ] ]
```

4.1.2 IntersectionSubXMods

▷ `IntersectionSubXMods(X0, X1, X2)` (operation)

When `X1,X2` are subcrossed modules of `X0`, then the source and range of their intersection are the intersections of the sources and ranges of `X1` and `X2` respectively.

Example

```
gap> pos2 := Position( ids, [ [24,6], [12,4] ] );;
gap> Xn2 := nsx[pos2];;
gap> IdGroup( Xn2 );
[ [ 24, 6 ], [ 12, 4 ] ]
gap> pos3 := Position( ids, [ [12,2], [24,5] ] );;
gap> Xn3 := nsx[pos3];;
gap> IdGroup( Xn3 );
[ [ 12, 2 ], [ 24, 5 ] ]
gap> Xn23 := IntersectionSubXMods( X24, Xn2, Xn3 );;
gap> IdGroup( Xn23 );
[ [ 12, 2 ], [ 6, 2 ] ]
```

4.1.3 Displacement

▷ `Displacement(alpha, r, s)` (operation)
▷ `DisplacementGroup(X0, Q, T)` (operation)
▷ `DisplacementSubgroup(X0)` (attribute)

Commutators may be written $[r,q] = r^{-1}q^{-1}rq = (q^{-1})^r q = r^{-1}r^q$, and satisfy identities

$$[r,q]^p = [r^p,q^p], \qquad [pr,q] = [p,q]^r[r,q], \qquad [r,pq] = [r,q][r,p]^q, \qquad [r,q]^{-1} = [q,r].$$

In a similar way, when a group R acts on a group S, the *displacement* of $s \in S$ by $r \in R$ is defined to be $\langle r,s\rangle := (s^{-1})^r s \in S$. When $\mathscr{X} = (\partial : S \to R)$ is a pre-crossed module, the first crossed module axiom requires $\partial\langle r,s\rangle = [r,\partial s]$. When α is the action of $\mathscr{X}$, the `Displacement` function may be used to calculate $\langle r,s\rangle$. Displacements satisfy the following identities, where $s,t \in S,\ p,q,r \in R$:

$$\langle r,s\rangle^p = \langle r^p,s^p\rangle, \qquad \langle qr,s\rangle = \langle q,s\rangle^r\langle r,s\rangle, \qquad \langle r,st\rangle = \langle r,t\rangle\langle r,s\rangle^t, \qquad \langle r,s\rangle^{-1} = \langle r^{-1},s^r\rangle.$$

The operation `DisplacementGroup` applied to `X0,Q,T` is the subgroup of `S` consisting of all the displacements $\langle r,s\rangle, r \in Q \leqslant R, s \in T \leqslant S$. The `DisplacementSubgroup` of $\mathscr{X}$ is the subgroup $\mathrm{Disp}(\mathscr{X})$ of S given by `DisplacementGroup(X0,R,S)`. The identities imply $\langle r,s\rangle^t = \langle r,st^{r^{-1}}\rangle\langle r^{-1},t\rangle$, so $\mathrm{Disp}(\mathscr{X})$ is normal in S.

Example

```
gap> pos4 := Position( ids, [ [6,2], [24,14] ] );;
gap> Xn4 := nsx[pos4];;
gap> bn4 := Boundary( Xn4 );;
gap> Sn4 := Source(Xn4);;
gap> Rn4 := Range(Xn4);;
gap> genRn4 := GeneratorsOfGroup( Rn4 );;
gap> L := List( genRn4, g -> ( Order(g) = 2 ) and
>                 not ( IsNormal( Rn4, Subgroup( Rn4, [g] ) ) ) );;
gap> pos := Position( L, true );;
gap> s := Sn4.1;  r := genRn4[pos];
(1,3,5,7,9,11)(2,4,6,8,10,12)
(6,7)
gap> act := XModAction( Xn4 );;
gap> d := Displacement( act, r, s );
(1,5,9)(2,6,10)(3,7,11)(4,8,12)
gap> Image( bn4, d ) = Comm( r, Image( bn4, s ) );
true
gap> Qn4 := Subgroup( Rn4, [ (6,7), (1,3), (2,4) ] );;
gap> Tn4 := Subgroup( Sn4, [ (1,3,5,7,9,11)(2,4,6,8,10,12) ] );;
gap> DisplacementGroup( Xn4, Qn4, Tn4 );
Group([ (1,5,9)(2,6,10)(3,7,11)(4,8,12) ])
gap> DisplacementSubgroup( Xn4 );
Group([ (1,5,9)(2,6,10)(3,7,11)(4,8,12) ])
```

4.1.4 CommutatorSubXMod

▷ `CommutatorSubXMod(X, X1, X2)` (operation)
▷ `CrossActionSubgroup(X, X1, X2)` (operation)

When $\mathcal{X}_1 = (N \to Q), \mathcal{X}_2 = (M \to P)$ are two normal subcrossed modules of $\mathcal{X} = (\partial : S \to R)$, the displacements $\langle p,n \rangle$ and $\langle q,m \rangle$ all map by ∂ into $[Q,P]$. These displacements form a normal subgroup of S, called the `CrossActionSubgroup`. The `CommutatorSubXMod` $[\mathcal{X}_1, \mathcal{X}_2]$ has this subgroup as source and $[P,Q]$ as range, and is normal in $\mathcal{X}$.

Example

```
gap> CAn23 := CrossActionSubgroup( X24, Xn2, Xn3 );;
gap> IdGroup( CAn23 );
[ 12, 2 ]
gap> Cn23 := CommutatorSubXMod( X24, Xn2, Xn3 );;
gap> IdGroup( Cn23 );
[ [ 12, 2 ], [ 6, 2 ] ]
gap> Xn23 = Cn23;
true
```

4.1.5 DerivedSubXMod

▷ `DerivedSubXMod(X0)` (attribute)

The `DerivedSubXMod` of $\mathscr{X}$ is the normal subcrossed module $[\mathscr{X},\mathscr{X}] = (\partial' : \mathrm{Disp}(\mathscr{X}) \to [R,R])$ where ∂' is the restriction of ∂ (see page 66 of Norrie's thesis [Nor87]).

Example

```
gap> DXn4 := DerivedSubXMod( Xn4 );;
gap> IdGroup( DXn4 );
[ [ 3, 1 ], [ 3, 1 ] ]
```

4.1.6 FixedPointSubgroupXMod

▷ `FixedPointSubgroupXMod(X0, T, Q)` (operation)
▷ `StabilizerSubgroupXMod(X0, T, Q)` (operation)

The `FixedPointSubgroupXMod(X,T,Q)` for $\mathscr{X} = (\partial : S \to R)$ is the subgroup $\mathrm{Fix}(\mathscr{X},T,Q)$ of elements $t \in T \leqslant S$ individually fixed under the action of $Q \leqslant R$.

The `StabilizerSubgroupXMod(X,T,Q)` for $\mathscr{X}$ is the subgroup $\mathrm{Stab}(\mathscr{X},T,Q)$ of $Q \leqslant R$ whose elements act trivially on the whole of $T \leqslant S$ (see page 19 of Norrie's thesis [Nor87]).

Example

```
gap> fix := FixedPointSubgroupXMod( Xn4, Sn4, Rn4 );
Group([ (1,7)(2,8)(3,9)(4,10)(5,11)(6,12) ])
gap> stab := StabilizerSubgroupXMod( Xn4, Sn4, Rn4 );;
gap> IdGroup( stab );
[ 12, 5 ]
```

4.1.7 CentreXMod

▷ `CentreXMod(X0)` (attribute)
▷ `Centralizer(X, Y)` (operation)
▷ `Normalizer(X, Y)` (operation)

The *centre* $Z(\mathscr{X})$ of $\mathscr{X} = (\partial : S \to R)$ has as source the fixed point subgroup $\mathrm{Fix}(\mathscr{X},S,R)$. The range is the intersection of the centre $Z(R)$ with the stabilizer subgroup.

When $\mathscr{Y} = (T \to Q)$ is a subcrossed module of $\mathscr{X} = (\partial : S \to R)$, the *centralizer* $C_{\mathscr{X}}(\mathscr{Y})$ of $\mathscr{Y}$ has as source the fixed point subgroup $\mathrm{Fix}(\mathscr{X},S,Q)$. The range is the intersection of the centralizer $C_R(Q)$ with $\mathrm{Stab}(\mathscr{X},T,R)$.

The *normalizer* $N_{\mathscr{X}}(\mathscr{Y})$ of $\mathscr{Y}$ has as source the subgroup of S consisting of the displacements $\langle s,q \rangle$ which lie in S.

Example

```
gap> ZXn4 := CentreXMod( Xn4 );
[Group( [ f3*f4 ] )->Group( [ f3, f5 ] )]
```

```
gap> IdGroup( ZXn4 );
[ [ 2, 1 ], [ 4, 2 ] ]
gap> CDXn4 := Centralizer( Xn4, DXn4 );
[Group( [ f3*f4 ] )->Group( [ f2 ] )]
gap> IdGroup( CDXn4 );
[ [ 2, 1 ], [ 3, 1 ] ]
gap> NDXn4 := Normalizer( Xn4, DXn4 );
[Group( <identity> of ... )->Group( [ f5, f2*f3 ] )]
gap> IdGroup( NDXn4 );
[ [ 1, 1 ], [ 12, 5 ] ]
```

4.1.8 CentralQuotient

▷ `CentralQuotient(G)` (attribute)

The `CentralQuotient` of a group G is the crossed module $(G \to G/Z(G))$ with the natural homomorphism as the boundary map. This is a special case of `XModByCentralExtension` (2.1.5).

Similarly, the central quotient of a crossed module $\mathscr{X}$ is the crossed square $(\mathscr{X} \Rightarrow \mathscr{X}/Z(\mathscr{X}))$ (see section 8.2).

Example

```
gap> Q24 := CentralQuotient( d24);  IdGroup( Q24 );
[d24->d24/Z(d24)]
[ [ 24, 6 ], [ 12, 4 ] ]
```

4.1.9 IsAbelian2DimensionalGroup

▷ `IsAbelian2DimensionalGroup(X0)` (property)
▷ `IsAspherical2DimensionalGroup(X0)` (property)
▷ `IsSimplyConnected2DimensionalGroup(X0)` (property)
▷ `IsFaithful2DimensionalGroup(X0)` (property)

A crossed module is *abelian* if it equal to its centre. This is the case when the range group is abelian and the action is trivial.

A crossed module is *aspherical* if the boundary has trivial kernel.

A crossed module is *simply connected* if the boundary has trivial cokernel.

A crossed module is *faithful* if the action is faithful.

Example

```
gap> [ IsAbelian2DimensionalGroup(Xn4), IsAbelian2DimensionalGroup(X24) ];
[ false, false ]
gap> pos7 := Position( ids, [ [3,1], [6,1] ] );;
gap> [ IsAspherical2DimensionalGroup(nsx[pos7]), IsAspherical2DimensionalGroup(X24) ];
[ true, false ]
gap> [ IsSimplyConnected2DimensionalGroup(Xn4), IsSimplyConnected2DimensionalGroup(X24) ];
[ true, true ]
gap> [ IsFaithful2DimensionalGroup(Xn4), IsFaithful2DimensionalGroup(X24) ];
```

```
[ false, true ]
```

4.1.10 LowerCentralSeriesOfXMod

▷ `LowerCentralSeriesOfXMod(X0)` (attribute)
▷ `IsNilpotent2DimensionalGroup(X0)` (property)
▷ `NilpotencyClass2DimensionalGroup(X0)` (attribute)

Let $\mathscr{Y}$ be a subcrossed module of $\mathscr{X}$. A *series of length n* from $\mathscr{X}$ to $\mathscr{Y}$ has the form

$$\mathscr{X} = \mathscr{X}_0 \trianglerighteq \mathscr{X}_1 \trianglerighteq \cdots \trianglerighteq \mathscr{X}_i \trianglerighteq \cdots \trianglerighteq \mathscr{X}_n = \mathscr{Y} \quad (1 \leqslant i \leqslant n).$$

The quotients $\mathscr{F}_i = \mathscr{X}_i / \mathscr{X}_{i-1}$ are the *factors* of the series.

A factor $\mathscr{F}_i$ is *central* if $\mathscr{X}_{i-1} \trianglelefteq \mathscr{X}$ and $\mathscr{F}_i$ is a subcrossed module of the centre of $\mathscr{X}/\mathscr{X}_{i-1}$.

A series is *central* if all its factors are central.

$\mathscr{X}$ is *soluble* if it has a series all of whose factors are abelian.

$\mathscr{X}$ is *nilpotent* is it has a series all of whose factors are central factors of $\mathscr{X}$.

The *lower central series* of $\mathscr{X}$ is the sequence (see [Nor87], p.77):

$$\mathscr{X} = \Gamma_1(\mathscr{X}) \trianglerighteq \Gamma_2(\mathscr{X}) \trianglerighteq \cdots \qquad \text{where} \qquad \Gamma_j(\mathscr{X}) = [\Gamma_{j-1}(\mathscr{X}), \mathscr{X}].$$

If $\mathscr{X}$ is nilpotent, then its lower central series is its most rapidly descending central series.

The least integer c such that $\Gamma_{c+1}(\mathscr{X})$ is the trivial crossed module is the *nilpotency class* of $\mathscr{X}$.

Example

```
gap> lcs := LowerCentralSeries( X24 );;
gap> List( lcs, g -> IdGroup(g) );
[ [ [ 24, 6 ], [ 48, 38 ] ], [ [ 12, 2 ], [ 6, 2 ] ], [ [ 6, 2 ], [ 3, 1 ] ],
  [ [ 3, 1 ], [ 3, 1 ] ] ]
gap> IsNilpotent2DimensionalGroup( X24 );
false
gap> NilpotencyClassOf2DimensionalGroup( X24 );
0
```

4.1.11 IsomorphismXMods

▷ `IsomorphismXMods(X1, X2)` (operation)

The function `IsomorphismXMods` computes an isomorphism $\mu : \mathscr{X}_1 \to \mathscr{X}_2$, provided one exists, or else returns `fail`.

Example

```
gap> gend24 := GeneratorsOfGroup( d24 );;
gap> a := gend24[1];; b:= gend24[2];;
gap> J := Subgroup( d24, [a^2,b] );
Group([ (1,3,5,7,9,11)(2,4,6,8,10,12), (2,12)(3,11)(4,10)(5,9)(6,8) ])
gap> K := Subgroup( d24, [a^2,a*b] );
```

```
Group([ (1,3,5,7,9,11)(2,4,6,8,10,12), (1,12)(2,11)(3,10)(4,9)(5,8)(6,7) ])
gap> XJ := XModByNormalSubgroup( d24, J );;
gap> XK := XModByNormalSubgroup( d24, K );;
gap> iso := IsomorphismXMods( XJ, XK );;
gap> SourceHom( iso );
[ (1,3,5,7,9,11)(2,4,6,8,10,12), (2,12)(3,11)(4,10)(5,9)(6,8) ] ->
[ (1,3,5,7,9,11)(2,4,6,8,10,12), (1,12)(2,11)(3,10)(4,9)(5,8)(6,7) ]
gap> RangeHom( iso );
[ (1,2,3,4,5,6,7,8,9,10,11,12), (2,12)(3,11)(4,10)(5,9)(6,8) ] ->
[ (1,2,3,4,5,6,7,8,9,10,11,12), (1,12)(2,11)(3,10)(4,9)(5,8)(6,7) ]
```

4.1.12 AllXMods

▷ `AllXMods(args)` (function)

▷ `AllXModsWithGroups(src, rng)` (operation)

▷ `AllXModsUpToIsomorphism(src, rng)` (operation)

▷ `IsomorphismClassRepresentatives2dGroups(L)` (operation)

The global function `AllXMods` may be called in three ways. Firstly, as `AllXMods(S,R)` to compute all crossed modules with chosen source and range groups: this calls `AllXModsWithGroups(S,R)`. Secondly, `AllXMods([n,m])` computes all crossed modules with a given size `[n,m]`. Thirdly `AllXMods(ord)` to compute all crossed modules whose associated cat1-groups have a given size `ord`.

The function `AllXModsUpToIsomorphism(S,R)` returns a list of representatives of the isomorphism classes of crossed modules with source S and range R.

If `L` is a list returned by, for example, `AllXModsWithGroups(S,R)`, then the isomorphism class representatives for this list is returned by `IsomorphismClassRepresentatives2dGroups(L)`. This result is the same as that given by `AllXModsUpToIsomorphism(S,R)`.

In the example we see that there are 4 crossed modules, in 3 isomorphism classes, $(C_6 \to S_3)$; forming a subset of the 17 crossed modules with size `[6,6]`; and that these form a subset of the 205 crossed modules whose cat1-group has size 36. There are 40 precrossed modules with size `[6,6]`.

Example

```
gap> c6 := SmallGroup( 6, 2 );;
gap> s3 := SmallGroup( 6, 1 );;
gap> Ac6s3 := AllXMods( c6, s3 );;
gap> Length( Ac6s3 );
4
gap> Ic6s3 := AllXModsUpToIsomorphism( c6, s3 );;
gap> List( Ic6s3, obj -> IsTrivialAction2DimensionalGroup( obj ) );
[ true, false, false ]
gap> Kc6s3 := List( Ic6s3, obj -> KernelCokernelXMod( obj ) );;
gap> List( Kc6s3, obj -> IdGroup( obj ) );
[ [ [ 6, 2 ], [ 6, 1 ] ], [ [ 6, 2 ], [ 6, 1 ] ], [ [ 2, 1 ], [ 2, 1 ] ] ]
[ ]
gap> A66 := AllXMods( [6,6] );;
gap> Length( A66 );
17
```

```
gap> IA66 := IsomorphismClassRepresentatives2dGroups( A66 );;
gap> Length( IA66 );
9
gap> x36 := AllXMods( 36 );;
gap> Length( x36 );
205
gap> size36 := List( x36, x -> Size2d( x ) );;
gap> Collected( size36 );
[ [ [ 1, 36 ], 14 ], [ [ 2, 18 ], 7 ], [ [ 3, 12 ], 21 ], [ [ 4, 9 ], 14 ],
  [ [ 6, 6 ], 17 ], [ [ 9, 4 ], 102 ], [ [ 12, 3 ], 8 ], [ [ 18, 2 ], 18 ],
  [ [ 36, 1 ], 4 ] ]
```

4.2 Isoclinism for groups

4.2.1 Isoclinism (for groups)

▷ Isoclinism(*G*, *H*) (operation)
▷ AreIsoclinicDomains(*G*, *H*) (operation)

Let G,H be groups with central quotients $Q(G)$ and $Q(H)$ and derived subgroups $[G,G]$ and $[H,H]$ respectively. Let $c_G : G/Z(G) \times G/Z(G) \to [G,G]$ and $c_H : H/Z(H) \times H/Z(H) \to [H,H]$ be the two commutator maps. An *isoclinism* $G \sim H$ is a pair of isomorphisms (η,ξ) where $\eta : G/Z(G) \to H/Z(H)$ and $\xi : [G,G] \to [H,H]$ such that $c_G * \xi = (\eta \times \eta) * c_H$. Isoclinism is an equivalence relation, and all abelian groups are isoclinic to the trivial group.

Example

```
gap> G := SmallGroup( 64, 6 );;  StructureDescription( G );
"(C8 x C4) : C2"
gap> QG := CentralQuotient( G );;  IdGroup( QG );
[ [ 64, 6 ], [ 8, 3 ] ]
gap> H := SmallGroup( 32, 41 );;  StructureDescription( H );
"C2 x Q16"
gap> QH := CentralQuotient( H );;  IdGroup( QH );
[ [ 32, 41 ], [ 8, 3 ] ]
gap> Isoclinism( G, H );
[ [ f1, f2, f3 ] -> [ f1, f2*f3, f3 ], [ f3, f5 ] -> [ f4*f5, f5 ] ]
gap> K := SmallGroup( 32, 43 );;  StructureDescription( K );
"(C2 x D8) : C2"
gap> QK := CentralQuotient( K );;  IdGroup( QK );
[ [ 32, 43 ], [ 16, 11 ] ]
gap> AreIsoclinicDomains( G, K );
false
```

4.2.2 IsStemDomain (for groups)

▷ IsStemDomain(*G*) (property)
▷ IsoclinicStemDomain(*G*) (attribute)

▷ `AllStemGroupIds(n)` (operation)
▷ `AllStemGroupFamilies(n)` (operation)

A group G is a *stem group* if $Z(G) \leq [G,G]$. Every group is isoclinic to a stem group, but distinct stem groups may be isoclinic. For example, groups D_8, Q_8 are two isoclinic stem groups.

The function `IsoclinicStemDomain` returns a stem group isoclinic to G.

The function `AllStemGroupIds` returns the `IdGroup` list of the stem groups of a specified size, while `AllStemGroupFamilies` splits this list into isoclinism classes.

Example

```
gap> DerivedSubgroup(G);
Group([ f3, f5 ])
gap> IsStemDomain( G );
false
gap> IsoclinicStemDomain( G );
<pc group of size 16 with 4 generators>
gap> AllStemGroupIds( 32 );
[ [ 32, 6 ], [ 32, 7 ], [ 32, 8 ], [ 32, 18 ], [ 32, 19 ], [ 32, 20 ],
  [ 32, 27 ], [ 32, 28 ], [ 32, 29 ], [ 32, 30 ], [ 32, 31 ], [ 32, 32 ],
  [ 32, 33 ], [ 32, 34 ], [ 32, 35 ], [ 32, 43 ], [ 32, 44 ], [ 32, 49 ],
  [ 32, 50 ] ]
gap> AllStemGroupFamilies( 32 );
[ [ [ 32, 6 ], [ 32, 7 ], [ 32, 8 ] ], [ [ 32, 18 ], [ 32, 19 ], [ 32, 20 ] ],
  [ [ 32, 27 ], [ 32, 28 ], [ 32, 29 ], [ 32, 30 ], [ 32, 31 ], [ 32, 32 ],
      [ 32, 33 ], [ 32, 34 ], [ 32, 35 ] ], [ [ 32, 43 ], [ 32, 44 ] ],
  [ [ 32, 49 ], [ 32, 50 ] ] ]
```

4.2.3 IsoclinicRank (for groups)

▷ `IsoclinicRank(G)` (attribute)
▷ `IsoclinicMiddleLength(G)` (attribute)

Let G be a finite p-group. Then $\log_p |[G,G]/(Z(G) \cap [G,G])|$ is called the *middle length* of G. Also $\log_p |Z(G) \cap [G,G]| + \log_p |G/Z(G)|$ is called the *rank* of G. These invariants appear in the tables of isoclinism families of groups of order 128 in [JNO90].

Example

```
gap> IsoclinicMiddleLength( G );
1
gap> IsoclinicRank( G );
4
```

4.3 Isoclinism for crossed modules

4.3.1 Isoclinism (for crossed modules)

▷ Isoclinism(*X0, Y0*) (operation)
▷ AreIsoclinicDomains(*X0, Y0*) (operation)

Let $\mathscr{X},\mathscr{Y}$ be crossed modules with central quotients $Q(\mathscr{X})$ and $Q(\mathscr{Y})$, and derived subcrossed modules $[\mathscr{X},\mathscr{X}]$ and $[\mathscr{Y},\mathscr{Y}]$ respectively. Let $c_{\mathscr{X}} : Q(\mathscr{X}) \times Q(\mathscr{X}) \to [\mathscr{X},\mathscr{X}]$ and $c_{\mathscr{Y}} : Q(\mathscr{Y}) \times Q(\mathscr{Y}) \to [\mathscr{Y},\mathscr{Y}]$ be the two commutator maps. An *isoclinism* $\mathscr{X} \sim \mathscr{Y}$ is a pair of bijective morphisms (η,ξ) where $\eta : Q(\mathscr{X}) \to Q(\mathscr{Y})$ and $\xi : [\mathscr{X},\mathscr{X}] \to [\mathscr{Y},\mathscr{Y}]$ such that $c_{\mathscr{X}} * \xi = (\eta \times \eta) * c_{\mathscr{Y}}$. Isoclinism is an equivalence relation, and all abelian crossed modules are isoclinic to the trivial crossed module.

Example

```
gap> C8 := Cat1Group( 16, 8, 2 );;
gap> X8 := XMod(C8);  IdGroup( X8 );
[Group( [ f1*f2*f3, f3, f4 ] )->Group( [ f2, f2 ] )]
[ [ 8, 1 ], [ 2, 1 ] ]
gap> C9 := Cat1Group( 32, 9, 2 );
[(C8 x C2) : C2 => Group( [ f2, f2 ] )]
gap> X9 := XMod( C9 );  IdGroup( X9 );
[Group( [ f1*f2*f3, f3, f4, f5 ] )->Group( [ f2, f2 ] )]
[ [ 16, 5 ], [ 2, 1 ] ]
gap> AreIsoclinicDomains( X8, X9 );
true
gap> ism89 := Isoclinism( X8, X9 );;
gap> Display( ism89 );
[ [[Group( [ f1, f2, <identity> of ... ] ) -> Group( [ f2, f2 ] )] => [Group(
    [ f1, f2, <identity> of ..., <identity> of ... ] ) -> Group(
    [ f2, f2 ] )]],
  [[Group( [ f3 ] ) -> Group( <identity> of ... )] => [Group(
    [ f3 ] ) -> Group( <identity> of ... )]] ]
```

4.3.2 IsStemDomain (for crossed modules of groups)

▷ IsStemDomain(*X0*) (property)
▷ IsoclinicStemDomain(*X0*) (attribute)

A crossed module $\mathscr{X}$ is a *stem crossed module* if $Z(\mathscr{X}) \leq [\mathscr{X},\mathscr{X}]$. Every crossed module is isoclinic to a stem crossed module, but distinct stem crossed modules may be isoclinic.

A method for IsoclinicStemDomain has yet to be implemented.

Example

```
gap> IsStemDomain(X8);
true
gap> IsStemDomain(X9);
false
```

4.3.3 IsoclinicRank (for crossed modules of groups)

▷ `IsoclinicRank(X0)` (attribute)
▷ `IsoclinicMiddleLength(X0)` (attribute)

The formulae in subsection 4.2.3 are applied to the crossed module.

Example

```
gap> IsoclinicMiddleLength(X8);
[ 1, 0 ]
gap> IsoclinicRank(X8);
[ 3, 1 ]
```

Chapter 5

Whitehead group of a crossed module

5.1 Derivations and Sections

The Whitehead monoid $\mathrm{Der}(\mathscr{X})$ of $\mathscr{X}$ was defined in [Whi48] to be the monoid of all *derivations* from R to S, that is the set of all maps $\chi : R \to S$, with *Whitehead multiplication* $\star$ (on the *right*) satisfying:

$$\textbf{Der 1} : \chi(qr) \;=\; (\chi q)^r\,(\chi r), \qquad \textbf{Der 2} : (\chi_1 \star \chi_2)(r) \;=\; (\chi_2 r)(\chi_1 r)(\chi_2 \partial \chi_1 r).$$

The zero map is the identity for this composition. Invertible elements in the monoid are called *regular*. The Whitehead group of $\mathscr{X}$ is the group of regular derivations in $\mathrm{Der}(\mathscr{X})$. In the next chapter the *actor* of $\mathscr{X}$ is defined as a crossed module whose source and range are permutation representations of the Whitehead group and the automorphism group of $\mathscr{X}$.

The construction for cat1-groups equivalent to the derivation of a crossed module is the *section*. The monoid of sections of $\mathscr{C} = (e;t,h : G \to R)$ is the set of group homomorphisms $\xi : R \to G$, with Whitehead multiplication $\star$ (on the *right*) satisfying:

$$\textbf{Sect 1} : t \circ \xi \;=\; \mathrm{id}_R, \quad \textbf{Sect 2} : (\xi_1 \star \xi_2)(r) \;=\; (\xi_1 r)(eh\xi_1 r)^{-1}(\xi_2 h \xi_1 r) \;=\; (\xi_2 h \xi_1 r)(eh\xi_1 r)^{-1}(\xi_1 r).$$

The embedding e is the identity for this composition, and $h(\xi_1 \star \xi_2) = (h\xi_1)(h\xi_2)$. A section is *regular* when $h\xi$ is an automorphism, and the group of regular sections is isomorphic to the Whitehead group.

If ε denotes the inclusion of $S = \ker\, t$ in G then $\partial = h\varepsilon : S \to R$ and

$$\xi r \;=\; (er)(e\chi r), \quad \text{which equals} \quad (r,\chi r) \;\in\; R \ltimes S,$$

determines a section ξ of $\mathscr{C}$ in terms of the corresponding derivation χ of $\mathscr{X}$, and conversely.

5.1.1 DerivationByImages

▷ `DerivationByImages(X0, ims)` (operation)
▷ `IsDerivation(map)` (property)
▷ `IsUp2DimensionalMapping(chi)` (property)
▷ `UpGeneratorImages(chi)` (attribute)
▷ `UpImagePositions(chi)` (attribute)
▷ `DerivationImage(chi, r)` (operation)

A derivation χ is stored like a group homomorphisms by specifying the images of the generating set `StrongGeneratorsStabChain( StabChain(R) )` of the range R. This set of images is stored as the attribute `UpGeneratorImages` of χ. The function `IsDerivation` is automatically called to check that this procedure is well-defined.

Images of the remaining elements may be obtained using axiom **Der 1**. `UpImagePositions(chi)` is the list of the images under χ of `Elements(R)` and `DerivationImage(chi,r)` returns χr.

In the following example a cat1-group `C3` and the associated crossed module `X3` are constructed, where `X3` is isomorphic to the inclusion of the normal cyclic group `c3` in the symmetric group `s3`. The derivation χ_1 maps `c3` to the identity and the other 3 elements to $(1,2,3)(4,6,5)$.

Example

```
gap> g18 := Group( (1,2,3), (4,5,6), (2,3)(5,6) );;
gap> SetName( g18, "g18" );
gap> gen18 := GeneratorsOfGroup( g18 );;
gap> g1 := gen18[1];;  g2 := gen18[2];;  g3 := gen18[3];;
gap> s3 := Subgroup( g18, gen18{[2..3]} );;
gap> SetName( s3, "s3" );;
gap> t := GroupHomomorphismByImages( g18, s3, gen18, [g2,g2,g3] );;
gap> h := GroupHomomorphismByImages( g18, s3, gen18, [(),g2,g3] );;
gap> e := GroupHomomorphismByImages( s3, g18, [g2,g3], [g2,g3] );;
gap> C3 := Cat1Group( t, h, e );
[g18=>s3]
gap> SetName( Kernel(t), "c3" );;
gap> X3 := XModOfCat1Group( C3 );
[c3->s3]
gap> R3 := Range( X3 );;
gap> StrongGeneratorsStabChain( StabChain( R3 ) );
[ (4,5,6), (2,3)(5,6) ]
gap> chi1 := DerivationByImages( X3, [ (), (1,2,3)(4,6,5) ] );
DerivationByImages( s3, c3, [ (4,5,6), (2,3)(5,6) ],
[ (), (1,2,3)(4,6,5) ] )
gap> [ IsUp2DimensionalMapping( chi1 ), IsDerivation( chi1 ) ];
[ true, true ]
gap> UpGeneratorImages( chi1 );
[ (), (1,2,3)(4,6,5) ]
gap> UpImagePositions( chi1 );
[ 1, 1, 1, 2, 2, 2 ]
gap> DerivationImage( chi1, (2,3)(4,5) );
(1,2,3)(4,6,5)
```

5.1.2 PrincipalDerivation

▷ `PrincipalDerivation(X0, s)` (operation)

The *principal derivation* determined by $s \in S$ is the derivation $\eta_s : R \to S,\ r \mapsto (s^{-1})^r s$.

Example

```
gap> eta := PrincipalDerivation( X3, (1,2,3)(4,6,5) );
```

```
DerivationByImages( s3, c3, [ (4,5,6), (2,3)(5,6) ], [ (), (1,3,2)(4,5,6) ] )
```

5.1.3 SectionByHomomorphism

▷ `SectionByHomomorphism(C, hom)` (operation)
▷ `IsSection(hom)` (property)
▷ `SectionByDerivation(chi)` (operation)
▷ `DerivationBySection(xi)` (operation)

Sections *are* group homomorphisms, so do not need a special representation. Operations `SectionByDerivation` and `DerivationBySection` convert derivations to sections, and vice-versa, calling `Cat1GroupOfXMod` (2.5.3) and `XModOfCat1Group` (2.5.3) automatically.

Two strategies for calculating derivations and sections are implemented, see [AW00]. The default method for `AllDerivations` (5.2.1) is to search for all possible sets of images using a backtracking procedure, and when all the derivations are found it is not known which are regular. In early versions of this package, the default method for `AllSections( <C> )` was to compute all endomorphisms on the range group `R` of `C` as possibilities for the composite $h\xi$. A backtrack method then found possible images for such a section. In the current version the derivations of the associated crossed module are calculated, and these are all converted to sections using `SectionByDerivation`.

Example
```
gap> hom2 := GroupHomomorphismByImages( s3, g18, [ (4,5,6), (2,3)(5,6) ],
> [ (1,3,2)(4,6,5), (1,2)(4,6) ] );;
gap> xi2 := SectionByHomomorphism( C3, hom2 );
SectionByHomomorphism( s3, g18, [ (4,5,6), (2,3)(5,6) ],
[ (1,3,2)(4,6,5), (1,2)(4,6) ] )
gap> [ IsUp2DimensionalMapping( xi2 ), IsSection( xi2 ) ];
[ true, true ]
gap> chi2 := DerivationBySection( xi2 );
DerivationByImages( s3, c3, [ (4,5,6), (2,3)(5,6) ],
[ (1,3,2)(4,5,6), (1,2,3)(4,6,5) ] )
gap> xi1 := SectionByDerivation( chi1 );
SectionByHomomorphism( s3, g18, [ (4,5,6), (2,3)(5,6) ],
[ (1,2,3), (1,2)(4,6) ] )
```

5.1.4 IdentityDerivation

▷ `IdentityDerivation(X0)` (attribute)
▷ `IdentitySection(C0)` (attribute)

The identity derivation maps the range group to the identity subgroup of the source, while the identity section is just the range embedding considered as a section.

Example
```
gap> IdentityDerivation( X3 );
DerivationByImages( s3, c3, [ (4,5,6), (2,3)(5,6) ], [ (), () ] )
```

```
gap> IdentitySection(C3);
SectionByHomomorphism( s3, g18, [ (4,5,6), (2,3)(5,6) ],
[ (4,5,6), (2,3)(5,6) ] )
```

5.1.5 WhiteheadProduct

▷ `WhiteheadProduct(chi1, chi2)` (operation)
▷ `WhiteheadOrder(chi)` (operation)

The `WhiteheadProduct` may be applied to two derivations to form $\chi_1 \star \chi_2$, or to two sections to form $\xi_1 \star \xi_2$. The `WhiteheadOrder` of a regular derivation χ is the smallest power of χ, using this product, equal to the `IdentityDerivation` (5.1.4).

Example

```
gap> chi12 := WhiteheadProduct( chi1, chi2 );
DerivationByImages( s3, c3, [ (4,5,6), (2,3)(5,6) ], [ (1,2,3)(4,6,5), () ] )
gap> xi12 := WhiteheadProduct( xi1, xi2 );
SectionByHomomorphism( s3, g18, [ (4,5,6), (2,3)(5,6) ],
[ (1,2,3), (2,3)(5,6) ] )
gap> xi12 = SectionByDerivation( chi12 );
true
gap> [ WhiteheadOrder( chi2 ), WhiteheadOrder( xi2 ) ];
[ 2, 2 ]
```

5.2 Whitehead Groups and Monoids

As mentioned at the beginning of this chapter, the Whitehead monoid $\mathrm{Der}(\mathscr{X})$ of $\mathscr{X}$ is the monoid of all derivations from R to S. Monoids of derivations have representation `IsMonoidOfUp2DimensionalMappingsObj`. Multiplication tables for Whitehead monoids enable the construction of transformation representations.

5.2.1 AllDerivations

▷ `AllDerivations(X0)` (attribute)
▷ `ImagesTable(obj)` (attribute)
▷ `DerivationClass(mon)` (attribute)
▷ `WhiteheadMonoidTable(X0)` (attribute)
▷ `WhiteheadTransformationMonoid(X0)` (attribute)

Using our example `X3` we find that there are just nine derivations.

Example

```
gap> all3 := AllDerivations( X3 );
monoid of derivations with images list:
[ (), () ]
[ (), (1,3,2)(4,5,6) ]
```

```
[ (), (1,2,3)(4,6,5) ]
[ (1,3,2)(4,5,6), () ]
[ (1,3,2)(4,5,6), (1,3,2)(4,5,6) ]
[ (1,3,2)(4,5,6), (1,2,3)(4,6,5) ]
[ (1,2,3)(4,6,5), () ]
[ (1,2,3)(4,6,5), (1,3,2)(4,5,6) ]
[ (1,2,3)(4,6,5), (1,2,3)(4,6,5) ]
gap> DerivationClass( all3 );
"all"
gap> Perform( ImagesTable( all3 ), Display );
[ 1, 1, 1, 1, 1, 1 ]
[ 1, 1, 1, 3, 3, 3 ]
[ 1, 1, 1, 2, 2, 2 ]
[ 1, 3, 2, 1, 3, 2 ]
[ 1, 3, 2, 3, 2, 1 ]
[ 1, 3, 2, 2, 1, 3 ]
[ 1, 2, 3, 1, 2, 3 ]
[ 1, 2, 3, 3, 1, 2 ]
[ 1, 2, 3, 2, 3, 1 ]
gap> wmt3 := WhiteheadMonoidTable( X3 );;
gap> Perform( wmt3, Display );
[ 1, 2, 3, 4, 5, 6, 7, 8, 9 ]
[ 2, 3, 1, 5, 6, 4, 8, 9, 7 ]
[ 3, 1, 2, 6, 4, 5, 9, 7, 8 ]
[ 4, 6, 5, 1, 3, 2, 7, 9, 8 ]
[ 5, 4, 6, 2, 1, 3, 8, 7, 9 ]
[ 6, 5, 4, 3, 2, 1, 9, 8, 7 ]
[ 7, 7, 7, 7, 7, 7, 7, 7, 7 ]
[ 8, 8, 8, 8, 8, 8, 8, 8, 8 ]
[ 9, 9, 9, 9, 9, 9, 9, 9, 9 ]
gap> wtm3 := WhiteheadTransformationMonoid( X3 );
<transformation monoid of degree 9 with 3 generators>
gap> GeneratorsOfMonoid( wtm3 );
[ Transformation( [ 2, 3, 1, 5, 6, 4, 8, 9, 7 ] ),
  Transformation( [ 4, 6, 5, 1, 3, 2, 7, 9, 8 ] ),
  Transformation( [ 7, 7, 7, 7, 7, 7, 7, 7, 7 ] ) ]
```

5.2.2 RegularDerivations

▷ `RegularDerivations(X0)` (attribute)
▷ `ImagesList(obj)` (attribute)
▷ `WhiteheadGroupTable(X0)` (attribute)
▷ `WhiteheadPermGroup(X0)` (attribute)

`RegularDerivations` are those derivations which are invertible in the monoid. Multiplication tables for the Whitehead group enable the construction of permutation representations.

Of the nine derivations of `X3` just six are regular. The associated group is isomorphic to the symmetric group s3.

Example

```
gap> reg3 := RegularDerivations( X3 );
monoid of derivations with images list:
[ (), () ]
[ (), (1,3,2)(4,5,6) ]
[ (), (1,2,3)(4,6,5) ]
[ (1,3,2)(4,5,6), () ]
[ (1,3,2)(4,5,6), (1,3,2)(4,5,6) ]
[ (1,3,2)(4,5,6), (1,2,3)(4,6,5) ]
gap> wgt3 := WhiteheadGroupTable( X3 );;
gap> Perform( wgt3, Display );
[ [ 1, 2, 3, 4, 5, 6 ],
  [ 2, 3, 1, 5, 6, 4 ],
  [ 3, 1, 2, 6, 4, 5 ],
  [ 4, 6, 5, 1, 3, 2 ],
  [ 5, 4, 6, 2, 1, 3 ],
  [ 6, 5, 4, 3, 2, 1 ] ]
gap> wpg3 := WhiteheadPermGroup( X3 );
Group([ (1,2,3)(4,5,6), (1,4)(2,6)(3,5) ])
```

5.2.3 PrincipalDerivations

▷ `PrincipalDerivations(X0)` (attribute)

The principal derivations form a subgroup of the Whitehead group.

Example

```
gap> PDX3 := PrincipalDerivations( X3 );
monoid of derivations with images list:
[ (), () ]
[ (), (1,3,2)(4,5,6) ]
[ (), (1,2,3)(4,6,5) ]
```

5.2.4 AllSections

▷ `AllSections(C0)` (attribute)
▷ `RegularSections(C0)` (attribute)

These operations have been declared but are not yet implemented. The interested user should, instead, work with the corresponding derivations.

Chapter 6

Actors of 2d-groups

6.1 Actor of a crossed module

The *actor* of $\mathscr{X}$ is a crossed module $\mathrm{Act}(\mathscr{X}) = (\Delta : \mathscr{W}(\mathscr{X}) \to \mathrm{Aut}(\mathscr{X}))$ which was shown by Lue and Norrie, in [Nor87] and [Nor90] to give the automorphism object of a crossed module $\mathscr{X}$. In this implementation, the source of the actor is a permutation representation W of the Whitehead group of regular derivations, and the range of the actor is a permutation representation A of the automorphism group $\mathrm{Aut}(\mathscr{X})$ of $\mathscr{X}$.

6.1.1 AutomorphismPermGroup

▷ `AutomorphismPermGroup(`*xmod*`)` (attribute)
▷ `GeneratingAutomorphisms(`*xmod*`)` (attribute)
▷ `PermAutomorphismAsXModMorphism(`*xmod, perm*`)` (operation)

The automorphisms (σ,ρ) of $\mathscr{X}$ form a group $\mathrm{Aut}(\mathscr{X})$ of crossed module isomorphisms. The function `AutomorphismPermGroup` finds a set of `GeneratingAutomorphisms` for $\mathrm{Aut}(\mathscr{X})$, and then constructs a permutation representation of this group, which is used as the range of the actor crossed module of $\mathscr{X}$. The individual automorphisms can be constructed from the permutation group using the function `PermAutomorphismAsXModMorphism`. The example below uses the crossed module `X3=[c3->s3]` constructed in section 5.1.

Example

```
gap> APX3 := AutomorphismPermGroup( X3 );
Group([ (5,7,6), (1,2)(3,4)(6,7) ])
gap> Size( APX3 );
6
gap> genX3 := GeneratingAutomorphisms( X3 );
[ [[c3->s3] => [c3->s3]], [[c3->s3] => [c3->s3]] ]
gap> e6 := Elements( APX3 )[6];
(1,2)(3,4)(5,7)
gap> m6 := PermAutomorphismAsXModMorphism( X3, e6 );;
gap> Display( m6 );
Morphism of crossed modules :-
: Source = [c3->s3] with generating sets:
  [ (1,2,3)(4,6,5) ]
```

```
  [ (4,5,6), (2,3)(5,6) ]
: Range = Source
: Source Homomorphism maps source generators to:
  [ (1,3,2)(4,5,6) ]
: Range Homomorphism maps range generators to:
  [ (4,6,5), (2,3)(4,5) ]
```

6.1.2 WhiteheadXMod

▷ `WhiteheadXMod(xmod)` (attribute)
▷ `LueXMod(xmod)` (attribute)
▷ `NorrieXMod(xmod)` (attribute)
▷ `ActorXMod(xmod)` (attribute)

An automorphism (σ,ρ) of `X` acts on the Whitehead monoid by $\chi^{(\sigma,\rho)} = \sigma\circ\chi\circ\rho^{-1}$, and this determines the action for the actor. In fact the four groups S,W,R,A, the homomorphisms between them, and the various actions, give five crossed modules forming a *crossed square* (see `ActorCrossedSquare` (8.2.5)).

- $\mathscr{W}(\mathscr{X}) = (\eta : S \to W)$, the Whitehead crossed module of $\mathscr{X}$, at the top,
- $\mathscr{X} = (\partial : S \to R)$, the initial crossed module, on the left,
- $\mathrm{Act}(\mathscr{X}) = (\Delta : W \to A)$, the actor crossed module of $\mathscr{X}$, on the right,
- $\mathscr{N}(X) = (\alpha : R \to A)$, the Norrie crossed module of $\mathscr{X}$, on the bottom, and
- $\mathscr{L}(\mathscr{X}) = (\Delta\circ\eta = \alpha\circ\partial : S \to A)$, the Lue crossed module of $\mathscr{X}$, along the top-left to bottom-right diagonal.

Example

```
gap> WGX3 := WhiteheadPermGroup( X3 );
Group([ (1,2,3)(4,5,6), (1,4)(2,6)(3,5) ])
gap> WX3 := WhiteheadXMod( X3 );;
gap> Display( WX3 );
Crossed module Whitehead[c3->s3] :-
: Source group has generators:
  [ (1,2,3)(4,6,5) ]
: Range group has generators:
  [ (1,2,3)(4,5,6), (1,4)(2,6)(3,5) ]
: Boundary homomorphism maps source generators to:
  [ (1,2,3)(4,5,6) ]
: Action homomorphism maps range generators to automorphisms:
  (1,2,3)(4,5,6) --> { source gens --> [ (1,2,3)(4,6,5) ] }
  (1,4)(2,6)(3,5) --> { source gens --> [ (1,3,2)(4,5,6) ] }
  These 2 automorphisms generate the group of automorphisms.
gap> LX3 := LueXMod( X3 );;
gap> Display( LX3 );
Crossed module Lue[c3->s3] :-
: Source group has generators:
```

```
  [ (1,2,3)(4,6,5) ]
: Range group has generators:
  [ (5,7,6), (1,2)(3,4)(6,7) ]
: Boundary homomorphism maps source generators to:
  [ (5,7,6) ]
: Action homomorphism maps range generators to automorphisms:
  (5,7,6) --> { source gens --> [ (1,2,3)(4,6,5) ] }
  (1,2)(3,4)(6,7) --> { source gens --> [ (1,3,2)(4,5,6) ] }
  These 2 automorphisms generate the group of automorphisms.
gap> NX3 := NorrieXMod( X3 );;
gap> Display( NX3 );
Crossed module Norrie[c3->s3] :-
: Source group has generators:
  [ (4,5,6), (2,3)(5,6) ]
: Range group has generators:
  [ (5,7,6), (1,2)(3,4)(6,7) ]
: Boundary homomorphism maps source generators to:
  [ (5,6,7), (1,2)(3,4)(6,7) ]
: Action homomorphism maps range generators to automorphisms:
  (5,7,6) --> { source gens --> [ (4,5,6), (2,3)(4,5) ] }
  (1,2)(3,4)(6,7) --> { source gens --> [ (4,6,5), (2,3)(5,6) ] }
  These 2 automorphisms generate the group of automorphisms.
gap> AX3 := ActorXMod( X3 );;
gap> Display( AX3);
Crossed module Actor[c3->s3] :-
: Source group has generators:
  [ (1,2,3)(4,5,6), (1,4)(2,6)(3,5) ]
: Range group has generators:
  [ (5,7,6), (1,2)(3,4)(6,7) ]
: Boundary homomorphism maps source generators to:
  [ (5,7,6), (1,2)(3,4)(6,7) ]
: Action homomorphism maps range generators to automorphisms:
  (5,7,6) --> { source gens --> [ (1,2,3)(4,5,6), (1,6)(2,5)(3,4) ] }
  (1,2)(3,4)(6,7) --> { source gens --> [ (1,3,2)(4,6,5), (1,4)(2,6)(3,5) ] }
  These 2 automorphisms generate the group of automorphisms.
```

The main methods for these operations are written for permutation crossed modules. For other crossed modules an isomorphism to a permutation crodssed module is found first, and then the main method is applied to the image. In the example the crossed module XAq8 is the automorphism crossed module of the quaternion group.

Example

```
gap> StructureDescription( WhiteheadXMod( XAq8 ) );
[ "Q8", "C2 x C2 x C2" ]
gap> StructureDescription( LueXMod( XAq8 ) );
[ "Q8", "S4" ]
gap> StructureDescription( NorrieXMod( XAq8 ) );
[ "S4", "S4" ]
gap> StructureDescription( ActorXMod( XAq8 ) );
[ "C2 x C2 x C2", "S4" ]
```

6.1.3 XModCentre

▷ `XModCentre(xmod)` (attribute)
▷ `InnerActorXMod(xmod)` (attribute)
▷ `InnerMorphism(xmod)` (attribute)

Pairs of boundaries or identity mappings provide six morphisms of crossed modules. In particular, the boundaries of $\mathscr{W}(\mathscr{X})$ and $\mathscr{N}(\mathscr{X})$ form the *inner morphism* of $\mathscr{X}$, mapping source elements to principal derivations and range elements to inner automorphisms. The image of $\mathscr{X}$ under this morphism is the *inner actor* of $\mathscr{X}$, while the kernel is the *centre* of $\mathscr{X}$. In the example which follows, the inner morphism of `X3=(c3->s3)`, from Chapter 5, is an inclusion of crossed modules.

Note that we appear to have defined *two* sorts of *centre* for a crossed module: `XModCentre` here, and `CentreXMod` (4.1.7) in the chapter on isoclinism. We suspect that these two definitions give the same answer, but this remains to be resolved.

Example

```
gap> IMX3 := InnerMorphism( X3 );;
gap> Display( IMX3 );
Morphism of crossed modules :-
: Source = [c3->s3] with generating sets:
  [ (1,2,3)(4,6,5) ]
  [ (4,5,6), (2,3)(5,6) ]
:  Range = Actor[c3->s3] with generating sets:
  [ (1,2,3)(4,5,6), (1,4)(2,6)(3,5) ]
  [ (5,7,6), (1,2)(3,4)(6,7) ]
: Source Homomorphism maps source generators to:
  [ (1,2,3)(4,5,6) ]
: Range Homomorphism maps range generators to:
  [ (5,6,7), (1,2)(3,4)(6,7) ]
gap> IsInjective( IMX3 );
true
gap> ZX3 := XModCentre( X3 );
[Group( () )->Group( () )]
gap> IAX3 := InnerActorXMod( X3 );;
gap> Display( IAX3 );
Crossed module InnerActor[c3->s3] :-
: Source group has generators:
  [ (1,2,3)(4,5,6) ]
: Range group has generators:
  [ (5,6,7), (1,2)(3,4)(6,7) ]
: Boundary homomorphism maps source generators to:
  [ (5,7,6) ]
: Action homomorphism maps range generators to automorphisms:
  (5,6,7) --> { source gens --> [ (1,2,3)(4,5,6) ] }
  (1,2)(3,4)(6,7) --> { source gens --> [ (1,3,2)(4,6,5) ] }
  These 2 automorphisms generate the group of automorphisms.
```

Chapter 7

Induced constructions

Before describing general functions for computing induced structures, we consider coproducts of crossed modules which provide induced crossed modules in certain cases.

7.1 Coproducts of crossed modules

Need to add here a reference (or two) for coproducts.

7.1.1 CoproductXMod

▷ CoproductXMod(*X1, X2*) (operation)
▷ CoproductInfo(*X0*) (attribute)

This function calculates the coproduct crossed module of two or more crossed modules which have a common range R. The standard method applies to $\mathscr{X}_1 = (\partial_1 : S_1 \to R)$ and $\mathscr{X}_2 = (\partial_2 : S_2 \to R)$. See below for the case of three or more crossed modules.

The source S_2 of $\mathscr{X}_2$ acts on S_1 via ∂_2 and the action of $\mathscr{X}_1$, so we can form a precrossed module $(\partial' : S_1 \ltimes S_2 \to R)$ where $\partial'(s_1,s_2) = (\partial_1 s_1)(\partial_2 s_2)$. The action of this precrossed module is the diagonal action $(s_1,s_2)^r = (s_1^r, s_2^r)$. Factoring out by the Peiffer subgroup, we obtain the coproduct crossed module $\mathscr{X}_1 \circ \mathscr{X}_2$.

In the example the structure descriptions of the precrossed module, the Peiffer subgroup, and the resulting coproduct are printed out when InfoLevel(InfoXMod) is at least 1. The coproduct comes supplied with attribute CoproductInfo, which includes the embedding morphisms of the two factors.

Example

```
gap> q8 := Group( (1,2,3,4)(5,8,7,6), (1,5,3,7)(2,6,4,8) );;
gap> XAq8 := XModByAutomorphismGroup( q8 );;
gap> s4b := Range( XAq8 );;
gap> SetName( q8, "q8" );  SetName( s4b, "s4b" );
gap> a := q8.1;;  b := q8.2;;
gap> alpha := GroupHomomorphismByImages( q8, q8, [a,b], [a^-1,b] );;
gap> beta := GroupHomomorphismByImages( q8, q8, [a,b], [a,b^-1] );;
gap> k4b := Subgroup( s4b, [ alpha, beta ] );;  SetName( k4b, "k4b" );
gap> Z8 := XModByNormalSubgroup( s4b, k4b );;
gap> SetName( XAq8, "XAq8" );  SetName( Z8, "Z8" );
gap> SetInfoLevel( InfoXMod, 1 );
```

```
gap> XZ8 := CoproductXMod( XAq8, Z8 );
#I  prexmod is [ [ 32, 47 ], [ 24, 12 ] ]
#I  peiffer subgroup is C2, [ 2, 1 ]
#I  the coproduct is [ "C2 x C2 x C2 x C2", "S4" ], [ [ 16, 14 ], [ 24, 12 ] ]
[Group( [ f1, f2, f3, f4 ] )->s4b]
gap> SetName( XZ8, "XZ8" );
gap> info := CoproductInfo( XZ8 );
rec( embeddings := [ [XAq8 => XZ8], [Z8 => XZ8] ], xmods := [ XAq8, Z8 ] )
gap> SetInfoLevel( InfoXMod, 0 );
```

Given a list of more than two crossed modules with a common range R, then an iterated coproduct is formed:

$$\bigcirc\ [\mathscr{X}_1, \mathscr{X}_2, \ldots, \mathscr{X}_n] \ = \ \mathscr{X}_1 \circ (\mathscr{X}_2 \circ (\ldots(\mathscr{X}_{n-1} \circ \mathscr{X}_n)\ldots)).$$

The embeddings field of the CoproductInfo of the resulting crossed module $\mathscr{Y}$ contains the n morphisms $\varepsilon_i : \mathscr{X}_i \to \mathscr{Y} (1 \leqslant i \leqslant n)$.

Example

```
gap> Y := CoproductXMod( [ XAq8, XAq8, Z8, Z8 ] );
[Group( [ f1, f2, f3, f4, f5, f6, f7, f8 ] )->s4b]
gap> StructureDescription( Y );
[ "C2 x C2 x C2 x C2 x C2 x C2 x C2 x C2", "S4" ]
gap> CoproductInfo( Y );
rec(
  embeddings :=
    [ [XAq8 => [Group( [ f1, f2, f3, f4, f5, f6, f7, f8 ] ) -> s4b]],
      [XAq8 => [Group( [ f1, f2, f3, f4, f5, f6, f7, f8 ] ) -> s4b]],
      [Z8 => [Group( [ f1, f2, f3, f4, f5, f6, f7, f8 ] ) -> s4b]],
      [Z8 => [Group( [ f1, f2, f3, f4, f5, f6, f7, f8 ] ) -> s4b]] ],
  xmods := [ XAq8, XAq8, Z8, Z8 ] )
```

7.2 Induced crossed modules

7.2.1 InducedXMod

▷ InducedXMod(*args*) (function)
▷ IsInducedXMod(*xmod*) (property)
▷ InducedXModBySurjection(*xmod, hom*) (operation)
▷ InducedXModByCopower(*xmod, hom, list*) (operation)
▷ MorphismOfInducedXMod(*xmod*) (attribute)

A morphism of crossed modules $(\sigma, \rho) : \mathscr{X}_1 \to \mathscr{X}_2$ factors uniquely through an induced crossed module $\rho_* \mathscr{X}_1 = (\delta : \rho_* S_1 \to R_2)$. Similarly, a morphism of cat1-groups factors through an induced cat1-group. Calculation of induced crossed modules of $\mathscr{X}$ also provides an algebraic means of determining the homotopy 2-type of homotopy pushouts of the classifying space of $\mathscr{X}$. For more background from algebraic topology see references in [BH78], [BW95], [BW96]. Induced crossed modules and induced cat1-groups also provide the building blocks for constructing pushouts in the categories *XMod* and *Cat1*.

Data for the cases of algebraic interest is provided by a crossed module $\mathscr{X} = (\partial : S \to R)$ and a homomorphism $\iota : R \to Q$. The output from the calculation is a crossed module $\iota_* \mathscr{X} = (\delta : \iota_* S \to Q)$ together with a morphism of crossed modules $\mathscr{X} \to \iota_* \mathscr{X}$. When ι is a surjection with kernel K then $\iota_* S = S/[K,S]$ where $[K,S]$ is the subgroup of S generated by elements of the form $s^{-1}s^k, s \in S, k \in K$ (see [BH78], Prop.9). (For many years, up until June 2018, this manual has stated the result to be $[K,S]$, though the correct quotient had been calculated.) When ι is an inclusion the induced crossed module may be calculated using a copower construction [BW95] or, in the case when R is normal in Q, as a coproduct of crossed modules ([BW96], but not yet implemented). When ι is neither a surjection nor an inclusion, ι is factored as the composite of the surjection onto the image and the inclusion of the image in Q, and then the composite induced crossed module is constructed. These constructions use Tietze transformation routines in the library file `tietze.gi`.

As a first, surjective example, we take for $\mathscr{X}$ the normal inclusion crossed module of `a4` in `s4`, and for ι the surjection from `s4` to `s3` with kernel `k4`. The induced crossed module is isomorphic to `X3 = [c3->s3]`.

Example

```
gap> s4gens := GeneratorsOfGroup( s4 );
[ (1,2), (2,3), (3,4) ]
gap> a4gens := GeneratorsOfGroup( a4 );
[ (1,2,3), (2,3,4) ]
gap> s3b := Group( (5,6),(6,7) );;  SetName( s3b, "s3b" );
gap> epi := GroupHomomorphismByImages( s4, s3b, s4gens, [(5,6),(6,7),(5,6)] );;
gap> X4 := XModByNormalSubgroup( s4, a4 );;
gap> indX4 := InducedXModBySurjection( X4, epi );
[a4/ker->s3b]
gap> Display( indX4 );

Crossed module [a4/ker->s3b] :-
: Source group a4/ker has generators:
  [ (1,3,2), (1,2,3) ]
: Range group s3b has generators:
  [ (5,6), (6,7) ]
: Boundary homomorphism maps source generators to:
  [ (5,6,7), (5,7,6) ]
: Action homomorphism maps range generators to automorphisms:
  (5,6) --> { source gens --> [ (1,2,3), (1,3,2) ] }
  (6,7) --> { source gens --> [ (1,2,3), (1,3,2) ] }
  These 2 automorphisms generate the group of automorphisms.

gap> morX4 := MorphismOfInducedXMod( indX4 );
[[a4->s4] => [a4/ker->s3b]]
```

For a second, injective example we take for $\mathscr{X}$ the automorphism crossed module `XAq8` of `CoproductXMod` (7.1.1), and for ι an inclusion of `s4b` in `s5`. The resulting source group is `SL(2,5)`.

Example

```
gap> iso4 := IsomorphismGroups( s4b, s4 );;
```

```
gap> s5 := Group( (1,2,3,4,5), (4,5) );;
gap> SetName( s5, "s5" );
gap> inc45 := InclusionMappingGroups( s5, s4 );;
gap> iota45 := iso4 * inc45;;
gap> indXAq8 := InducedXMod( XAq8, iota45 );
i*(XAq8)
gap> Size2d( indXAq8 );
[ 120, 120 ]
gap> StructureDescription( indXAq8 );
[ "SL(2,5)", "S5" ]
```

For a third example we use the version `InducedXMod(Q,R,S)` of this global function, with $Q \geqslant R \trianglerighteq S$. We take the identity mapping on `s3c` as boundary, and the inclusion of `s3c` in `s4` as ι. The induced group is a general linear group `GL(2,3)`.

Example
```
gap> s3c := Subgroup( s4, [ (2,3), (3,4) ] );;
gap> SetName( s3c, "s3c" );
gap> indXs3c := InducedXMod( s4, s3c, s3c );
i*([s3c->s3c])
gap> StructureDescription( indXs3c );
[ "GL(2,3)", "S4" ]
```

7.2.2 AllInducedXMods

▷ `AllInducedXMods(Q)` (operation)

This function calculates all the induced crossed modules `InducedXMod(Q,R,S)`, where `R` runs over all conjugacy classes of subgroups of `Q` and `S` runs over all non-trivial normal subgroups of `R`.

Example
```
gap> all := AllInducedXMods( q8 );;
gap> ids := List( all, x -> IdGroup(x) );;
gap> Sort( ids );
gap> ids;
[ [ [ 1, 1 ], [ 8, 4 ] ], [ [ 1, 1 ], [ 8, 4 ] ], [ [ 1, 1 ], [ 8, 4 ] ],
  [ [ 1, 1 ], [ 8, 4 ] ], [ [ 4, 2 ], [ 8, 4 ] ], [ [ 4, 2 ], [ 8, 4 ] ],
  [ [ 4, 2 ], [ 8, 4 ] ], [ [ 16, 2 ], [ 8, 4 ] ], [ [ 16, 2 ], [ 8, 4 ] ],
  [ [ 16, 2 ], [ 8, 4 ] ], [ [ 16, 14 ], [ 8, 4 ] ] ]
```

7.3 Induced cat^1-groups

7.3.1 InducedCat1Group

▷ `InducedCat1Group(args)` (function)

▷ `InducedCat1GroupByFreeProduct(grp, hom)` (property)

This area awaits development.

Chapter 8

Crossed squares and Cat^2-groups

The term *3d-group* refers to a set of equivalent categories of which the most common are the categories of *crossed squares* and *cat^2-groups*. A *3d-mapping* is a function between two 3d-groups which preserves all the structure.

The material in this chapter should be considered experimental. A major overhaul took place in time for XMod version 2.73, with the names of a number of operations being changed.

8.1 Definition of a crossed square and a crossed *n*-cube of groups

Crossed squares were introduced by Guin-Waléry and Loday (see, for example, [BL87]) as fundamental crossed squares of commutative squares of spaces, but are also of purely algebraic interest. We denote by $[n]$ the set $\{1,2,\ldots,n\}$. We use the $n=2$ version of the definition of crossed n-cube as given by Ellis and Steiner [ES87].

A *crossed square* $\mathscr{S}$ consists of the following:

- groups S_J for each of the four subsets $J \subseteq [2]$ (we often find it convenient to write $L = S_{[2]}$, $M = S_{\{1\}}$, $N = S_{\{2\}}$ and $P = S_{\emptyset}$);
- a commutative diagram of group homomorphisms:
  $$\ddot{\partial}_1 : S_{[2]} \to S_{\{2\}}, \quad \ddot{\partial}_2 : S_{[2]} \to S_{\{1\}}, \quad \dot{\partial}_2 : S_{\{2\}} \to S_{\emptyset}, \quad \dot{\partial}_1 : S_{\{1\}} \to S_{\emptyset}$$
  (again we often write $\kappa = \ddot{\partial}_1$, $\lambda = \ddot{\partial}_2$, $\mu = \dot{\partial}_2$ and $\nu = \dot{\partial}_1$);
- actions of $S_{\emptyset}$ on $S_{\{1\}}, S_{\{2\}}$ and $S_{[2]}$ which determine actions of $S_{\{1\}}$ on $S_{\{2\}}$ and $S_{[2]}$ via $\dot{\partial}_1$ and actions of $S_{\{2\}}$ on $S_{\{1\}}$ and $S_{[2]}$ via $\dot{\partial}_2$;
- a function $\boxtimes : S_{\{1\}} \times S_{\{2\}} \to S_{[2]}$.

Here is a picture of the situation:

$$\mathscr{S} \quad = \quad \begin{tikzcd} S_{[2]} \arrow[r, "\ddot{\partial}_1"] \arrow[d, "\ddot{\partial}_2"'] & S_{\{2\}} \arrow[d, "\dot{\partial}_2"] \\ S_{\{1\}} \arrow[r, "\dot{\partial}_1"'] & S_{\emptyset} \end{tikzcd} \quad = \quad \begin{tikzcd} L \arrow[r, "\kappa"] \arrow[d, "\lambda"'] & M \arrow[d, "\mu"] \\ N \arrow[r, "\nu"'] & P \end{tikzcd}$$

The following axioms must be satisfied for all $l \in L,\ m, m_1, m_2 \in M,\ n, n_1, n_2 \in N,\ p \in P$.

- The homomorphisms κ, λ preserve the action of P.
- Each of the upper, left-hand, right-hand and lower sides of the square,

  $$\ddot{\mathscr{S}}_1 = (\kappa : L \to M), \quad \ddot{\mathscr{S}}_2 = (\lambda : L \to N), \quad \dot{\mathscr{S}}_2 = (\mu : M \to P), \quad \dot{\mathscr{S}}_1 = (\nu : N \to P),$$

  and the diagonal

  $$\mathscr{S}_{12} = (\partial_{12} := \mu \circ \kappa = \nu \circ \lambda : L \to P)$$

  are crossed modules (with actions via P).

  These will be called the *up, left, right, down* and *diagonal* crossed modules of $\mathscr{S}$.
- $\boxtimes$ is a *crossed pairing*:
  - $(n_1 n_2 \boxtimes m) \ = \ (n_1 \boxtimes m)^{n_2}\, (n_2 \boxtimes m)$,
  - $(n \boxtimes m_1 m_2) \ = \ (n \boxtimes m_2)\, (n \boxtimes m_1)^{m_2}$,
  - $(n \boxtimes m)^p \ = \ (n^p \boxtimes m^p)$.
- $\kappa(n \boxtimes m) \ = \ (m^{-1})^n\, m$ and $\lambda(n \boxtimes m) \ = \ n^{-1}\, n^m$.
- $(n \boxtimes \kappa l) \ = \ (l^{-1})^n\, l$ and $(\lambda l \boxtimes m) \ = \ l^{-1}\, l^m$.

Note that the actions of M on N and N on M via P are compatible since

$$n_1{}^{(m^n)} \ = \ n_1{}^{\mu(m^n)} \ = \ n_1{}^{n^{-1}(\mu m) n} \ = \ ((n_1{}^{n^{-1}})^m)^n.$$

(A *precrossed square* is a similar structure which satisfies some subset of these axioms. This notion needs to be clarified.)

Crossed squares are the $k = 2$ case of a crossed k-cube of groups, defined as follows. (This is an attempt to translate Definition 2.1 in Ronnie Brown's *Computing homotopy types using crossed n-cubes of groups* into right actions – but this definition is not yet completely understood!)

A *crossed k-cube of groups* consists of the following:

- groups S_A for every subset $A \subseteq [k]$;
- a commutative diagram of group homomorphisms $\partial_i : S_A \to S_{A \setminus \{i\}},\ i \in [k]$; with composites $\partial_B : S_A \to S_{A \setminus B},\ B \subseteq [k]$;
- actions of $S_\emptyset$ on each S_A; and hence actions of S_B on S_A via ∂_B for each $B \subseteq [k]$;
- functions $\boxtimes_{A,B} : S_A \times S_B \to S_{A \cup B}, (A, B \subseteq [k])$.

There is then a long list of axioms which must be satisfied.

8.2 Constructions for crossed squares

Analogously to the data structure used for crossed modules, crossed squares are implemented as `3d-groups`. There are also experimental implementations of cat^2-groups, with conversion between the two types of structure. Some standard constructions of crossed squares are listed below. At present, a limited number of constructions is implemented. Morphisms of crossed squares have also been implemented, though there is still a great deal to be done.

8.2.1 CrossedSquareByXMods

▷ CrossedSquareByXMods(*up, left, right, down, diag, pairing*) (operation)
▷ PreCrossedSquareByPreXMods(*up, left, right, down, diag, pairing*) (operation)

If *up,left,right,down,diag* are five (pre-)crossed modules whose sources and ranges agree, as above, then we just have to add a crossed pairing to complete the data for a (pre-)crossed square.

The Display function is used to print details of 3d-groups.

We take as our example a simple, but significant case. We start with five crossed modules formed from subgroups of D_8 with generators $[(1,2,3,4),(3,4)$. The result is a pre-crossed square which is *not* a crossed square.

Example

```
gap> b := (2,4);; c := (1,2)(3,4);; p := (1,2,3,4);;
gap> d8 := Group( b, c );;
gap> SetName( d8, "d8" );;
gap> L := Subgroup( d8, [p^2] );;
gap> M := Subgroup( d8, [b] );;
gap> N := Subgroup( d8, [c] );;
gap> P := TrivialSubgroup( d8 );;
gap> kappa := GroupHomomorphismByImages( L, M, [p^2], [b] );;
gap> lambda := GroupHomomorphismByImages( L, N, [p^2], [c] );;
gap> delta := GroupHomomorphismByImages( L, P, [p^2], [()] );;
gap> mu := GroupHomomorphismByImages( M, P, [b], [()] );;
gap> nu := GroupHomomorphismByImages( N, P, [c], [()] );;
gap> up := XModByTrivialAction( kappa );;
gap> left := XModByTrivialAction( lambda );;
gap> diag := XModByTrivialAction( delta );;
gap> right := XModByTrivialAction( mu );;
gap> down := XModByTrivialAction( nu );;
gap> xp := CrossedPairingByCommutators( N, M, L );;
gap> Print( "xp([c,b]) = ", ImageElmCrossedPairing( xp, [c,b] ), "\n" );
xp([c,b]) = (1,3)(2,4)
gap> PXS := PreCrossedSquareByPreXMods( up, left, right, down, diag, xp );;
gap> Display( PXS );
(pre-)crossed square with (pre-)crossed modules:
      up = [Group( [ (1,3)(2,4) ] ) -> Group( [ (2,4) ] )]
    left = [Group( [ (1,3)(2,4) ] ) -> Group( [ (1,2)(3,4) ] )]
   right = [Group( [ (2,4) ] ) -> Group( () )]
    down = [Group( [ (1,2)(3,4) ] ) -> Group( () )]
gap>  IsCrossedSquare( PXS );
false
```

8.2.2 Size3d (for 3d-objects)

▷ Size3d(*XS*) (attribute)

Just as Size2d was used in place of Size for crossed modules, so Size3d is used for crossed squares: Size3d(XS) returns a four-element list containing the sizes of the four groups at the corners of the square.

Example

```
gap> Size3d( PXS );
[ 2, 2, 2, 1 ]
```

8.2.3 CrossedSquareByNormalSubgroups

▷ `CrossedSquareByNormalSubgroups(L, M, N, P)` (operation)
▷ `CrossedPairingByCommutators(N, M, L)` (operation)

If L,M,N are normal subgroups of a group P, and $[M,N] \leqslant L \leqslant M \cap N$, then the four inclusions $L \to M$, $L \to N$, $M \to P$, $N \to P$, together with the actions of P on M,N and L given by conjugation, form a crossed square with crossed pairing

$$\boxtimes \;:\; N \times M \to L, \quad (n,m) \mapsto [n,m] = n^{-1}m^{-1}nm = (m^{-1})^n m = n^{-1}n^m .$$

This construction is implemented as `CrossedSquareByNormalSubgroups(L,M,N,P)` (note that the parent group comes last).

Example

```
gap> d20 := DihedralGroup( IsPermGroup, 20 );;
gap> gend20 := GeneratorsOfGroup( d20 );
[ (1,2,3,4,5,6,7,8,9,10), (2,10)(3,9)(4,8)(5,7) ]
gap> p1 := gend20[1];;  p2 := gend20[2];;  p12 := p1*p2;
(1,10)(2,9)(3,8)(4,7)(5,6)
gap> d10a := Subgroup( d20, [ p1^2, p2 ] );;
gap> d10b := Subgroup( d20, [ p1^2, p12 ] );;
gap> c5d := Subgroup( d20, [ p1^2 ] );;
gap> SetName( d20, "d20" );  SetName( d10a, "d10a" );
gap> SetName( d10b, "d10b" );  SetName( c5d, "c5d" );
gap> XSconj := CrossedSquareByNormalSubgroups( c5d, d10a, d10b, d20 );
[  c5d -> d10a ]
[   |       |  ]
[ d10b -> d20  ]
gap> xpc := CrossedPairing( XSconj );;
gap> ImageElmCrossedPairing( xpc, [ p2, p12 ] );
(1,9,7,5,3)(2,10,8,6,4)
```

8.2.4 CrossedSquareByNormalSubXMod

▷ `CrossedSquareByNormalSubXMod(X0, X1)` (operation)
▷ `CrossedPairingBySingleXModAction(X0, X1)` (operation)

If $\mathscr{X}_1 = (\partial_1 : S_1 \to R_1)$ is a normal sub-crossed module of $\mathscr{X}_0 = (\partial_0 : S_0 \to R_0)$ then the inclusion morphism gives a crossed square with crossed pairing

$$\boxtimes \;:\; R_1 \times S_0 \to S_1, \quad (r_1,s_0) \mapsto (s_0^{-1})^{r_1} s_0.$$

The example constructs the same crossed square as in the previous subsection.

Example

```
gap> X20 := XModByNormalSubgroup( d20, d10a );;
gap> X10 := XModByNormalSubgroup( d10b, c5d );;
gap> ok := IsNormalSub2DimensionalDomain( X20, X10 );
true
gap> XS20 := CrossedSquareByNormalSubXMod( X20, X10 );
[  c5d -> d10a ]
[   |       |   ]
[ d10b -> d20  ]
gap> xp20 := CrossedPairing( XS20 );;
gap> ImageElmCrossedPairing( xp20, [ p1^2, p2 ] );
(1,7,3,9,5)(2,8,4,10,6)
```

8.2.5 ActorCrossedSquare

▷ ActorCrossedSquare(*X0*) (attribute)
▷ CrossedPairingByDerivations(*X0*) (operation)

The actor $\mathscr{A}(\mathscr{X}_0)$ of a crossed module $\mathscr{X}_0$ has been described in Chapter 5 (see `ActorXMod` (6.1.2)). The crossed pairing is given by

$$\boxtimes \; : \; R \times W \to S, \quad (r,\chi) \mapsto \chi r \, .$$

This is implemented as `ActorCrossedSquare(X0);`.

Example

```
gap> XSact := ActorCrossedSquare( X20 );
crossed square with:
      up = Whitehead[d10a->d20]
    left = [d10a->d20]
   right = Actor[d10a->d20]
    down = Norrie[d10a->d20]
gap> W := Range( Up2DimensionalGroup( XSact ) );
c5:c4
gap> w1 := GeneratorsOfGroup( W )[1];
(1,2)(3,4)(5,18)(6,17)(7,20)(8,19)(9,14)(10,13)(11,16)(12,15)
gap> xpa := CrossedPairing( XSact );;
gap> ImageElmCrossedPairing( xpa, [ p1, w1 ] );
(1,9,7,5,3)(2,10,8,6,4)
```

8.2.6 CrossedSquareByAutomorphismGroup

▷ CrossedSquareByAutomorphismGroup(*G*) (operation)
▷ CrossedPairingByConjugators(*G*) (operation)

For G a group let $\mathrm{Inn}(G)$ be its inner automorphism group and $\mathrm{Aut}(G)$ its full automorphism group. Then there is a crossed square with groups $[G, \mathrm{Inn}(G), \mathrm{Inn}(G), \mathrm{Aut}(G)]$ where the upper and left boundaries are the maps $g \mapsto \iota_g$, where ι_g is conjugation of G by g, and the right and down boundaries are inclusions. The crossed pairing is gived by $\iota_g \boxtimes \iota_h = [g,h]$.

Example

```
gap> AXS20 := CrossedSquareByAutomorphismGroup( d20 );
[      d20 -> Inn(d20) ]
[     |           |     ]
[ Inn(d20) -> Aut(d20) ]

gap> StructureDescription( AXS20 );
[ "D20", "D10", "D10", "C2 x (C5 : C4)" ]
gap> I20 := Range( Up2DimensionalGroup( AXS20 ) );;
gap> genI20 := GeneratorsOfGroup( I20 );
[ ^(1,2,3,4,5,6,7,8,9,10), ^(2,10)(3,9)(4,8)(5,7) ]
gap> xpi := CrossedPairing( AXS20 );;
gap> ImageElmCrossedPairing( xpi, [ genI20[1], genI20[2] ] );
(1,9,7,5,3)(2,10,8,6,4)
```

8.2.7 CrossedSquareByPullback

▷ `CrossedSquareByPullback(X1, X2)` (operation)

If crossed modules $\mathscr{X}_1 = (\nu : N \to P)$ and $\mathscr{X}_2 = (\mu : M \to P)$ have a common range P, let L be the pullback of $\{\nu, \mu\}$. Then N acts on L by $(n,m)^{n'} = (n^{n'}, m^{\nu n'})$, and M acts on L by $(n,m)^{m'} = (n^{\mu m'}, m^{m'})$. So $(\pi_1 : L \to N)$ and $(\pi_2 : L \to M)$ are crossed modules, where π_1, π_2 are the two projections. The crossed pairing is given by:

$$\boxtimes \ : \ N \times M \to L, \quad (n,m) \mapsto (n^{-1}n^{\mu m}, (m^{-1})^{\nu n} m).$$

The second example below uses the central extension crossed module `X12=(D12->S3)` which was constructed in subsection (`XModByCentralExtension` (2.1.5)), with pullback group `D12xC2`.

Example

```
gap> dn := Down2DimensionalGroup( XSconj );;
gap> rt := Right2DimensionalGroup( XSconj );;
gap> XSP := CrossedSquareByPullback( dn, rt );
[ (d10b x_d20 d10a) -> d10a ]
[          |              |    ]
[               d10b -> d20    ]
gap> StructureDescription( XSP );
[ "C5", "D10", "D10", "D20" ]
gap> XS12 := CrossedSquareByPullback( X12, X12 );;
gap> StructureDescription( XS12 );
[ "C2 x C2 x S3", "D12", "D12", "S3" ]
gap> xp12 := CrossedPairing( XS12 );;
gap> ImageElmCrossedPairing( xp12, [ (1,2,3,4,5,6), (2,6)(3,5) ] );
(1,5,3)(2,6,4)(7,11,9)(8,12,10)
```

8.2.8 CrossedSquareByXModSplitting

▷ `CrossedSquareByXModSplitting(X0)` (attribute)
▷ `CrossedPairingByPreImages(X1, X2)` (operation)

For $\mathscr{X} = (\partial : S \to R)$ let Q be the image of ∂. Then $\partial = \partial' \circ \iota$ where $\partial' : S \to Q$ and ι is the inclusion of Q in R. The diagonal of the square is then the initial $\mathscr{X}$, and the crossed pairing is given by commutators of preimages.

A particular case is when S is an R-module A and ∂ is the zero map.

$$\begin{tikzcd} S \arrow[r, "\partial'"] \arrow[d, "\partial'"'] & Q \arrow[d, "\iota"] \\ Q \arrow[r, "\iota"'] & R \end{tikzcd} \qquad \begin{tikzcd} A \arrow[r, "0"] \arrow[d, "0"'] & 1 \arrow[d, "\iota"] \\ 1 \arrow[r, "\iota"'] & R \end{tikzcd}$$

Example

```
gap> k4 := Group( (1,2), (3,4) );;
gap> AX4 := XModByAutomorphismGroup( k4 );;
gap> X4 := Image( IsomorphismPermObject( AX4 ) );;
gap> XSS4 := CrossedSquareByXModSplitting( X4 );;
gap> StructureDescription( XSS4 );
[ "C2 x C2", "1", "1", "S3" ]
gap> XSS20 := CrossedSquareByXModSplitting( X20 );;
gap> up20 := Up2DimensionalGroup( XSS20 );;
gap> Range( up20 ) = d10a;
true
gap> SetName( Range( up20 ), "d10a" );
gap> Name( XSS20 );
"[d10a->d10a,d10a->d20]"
gap> xp12 := CrossedPairing( XS12 );;
gap> ImageElmCrossedPairing( xp12, [ (1,2,3,4,5,6), (2,6)(3,5) ] );
(1,5,3)(2,6,4)(7,11,9)(8,12,10)
gap> XSS20;
[d10a->d10a,d10a->d20]
gap> xps := CrossedPairing( XSS20 );;
gap> ImageElmCrossedPairing( xps, [ p1^2, p2 ] );
(1,7,3,9,5)(2,8,4,10,6)
```

8.2.9 CrossedSquare

▷ `CrossedSquare(args)` (function)

The function `CrossedSquare` may be used to call some of the constructions described in the previous subsections.

- `CrossedSquare(X0)` calls `CrossedSquareByXModSplitting`.

- `CrossedSquare(C0)` calls `CrossedSquareOfCat2Group`.
- `CrossedSquare(X0,X1)` calls `CrossedSquareByPullback` when there is a common range.
- `CrossedSquare(X0,X1)` calls `CrossedSquareByNormalXMod` when `X1` is normal in `X0` .
- `CrossedSquare(L,M,N,P)` calls `CrossedSquareByNormalSubgroups`.

Example

```
gap> diag := Diagonal2DimensionalGroup( AXS20 );
[d20->Aut(d20)]
gap> XSdiag := CrossedSquare( diag );;
gap> StructureDescription( XSdiag );
[ "D20", "D10", "D10", "C2 x (C5 : C4)" ]
```

8.2.10 Transpose3DimensionalGroup (for crossed squares)

▷ `Transpose3DimensionalGroup(S0)` (attribute)

The *transpose* of a crossed square $\mathscr{S}$ is the crossed square $\tilde{\mathscr{S}}$ obtained by interchanging M with N, κ with λ, and ν with μ. The crossed pairing is given by

$$\tilde{\boxtimes} \;:\; M \times N \to L, \quad (m,n) \;\mapsto\; m \tilde{\boxtimes} n := (n \boxtimes m)^{-1} \;.$$

Example

```
gap> XStrans := Transpose3DimensionalGroup( XSconj );
[  c5d -> d10b ]
[   |       |  ]
[ d10a -> d20  ]
```

8.2.11 CentralQuotient (for crossed modules)

▷ `CentralQuotient(X0)` (attribute)

The central quotient of a crossed module $\mathscr{X} = (\partial : S \to R)$ is the crossed square where:

- the left crossed module is $\mathscr{X}$;
- the right crossed module is the quotient $\mathscr{X}/Z(\mathscr{X})$ (see `CentreXMod` (4.1.7));
- the up and down homomorphisms are the natural homomorphisms onto the quotient groups;
- the crossed pairing $\boxtimes : (R \times F) \to S$, where $F = \mathrm{Fix}(\mathscr{X}, S, R)$, is the displacement element $\boxtimes(r, Fs) = \langle r, s\rangle = (s^{-1})^r s$ (see `Displacement` (4.1.3) and section 4.3).

This is the special case of an intended function `CrossedSquareByCentralExtension` which has not yet been implemented. In the example `Xn7` $\trianglelefteq$ `X24`, constructed in section 4.1.

Example

```
gap> pos7 := Position( ids, [ [12,2], [24,5] ] );;
gap> Xn7 := nsx[pos7];;
gap> IdGroup( Xn7 );
[ [ 12, 2 ], [ 24, 5 ] ]
gap> IdGroup( CentreXMod( Xn7 ) );
[ [ 4, 1 ], [ 4, 1 ] ]
gap> CQXn7 := CentralQuotient( Xn7 );;
gap> StructureDescription( CQXn7 );
[ "C12", "C3", "C4 x S3", "S3" ]
```

8.2.12 IsCrossedSquare

▷ `IsCrossedSquare(obj)` (property)
▷ `IsPreCrossedSquare(obj)` (property)
▷ `Is3dObject(obj)` (property)
▷ `IsPerm3dObject(obj)` (property)
▷ `IsPc3dObject(obj)` (property)
▷ `IsFp3dObject(obj)` (property)

These are the basic properties for 3d-groups, and crossed squares in particular.

8.2.13 Up2DimensionalGroup

▷ `Up2DimensionalGroup(XS)` (attribute)
▷ `Left2DimensionalGroup(XS)` (attribute)
▷ `Down2DimensionalGroup(XS)` (attribute)
▷ `Right2DimensionalGroup(XS)` (attribute)
▷ `CrossDiagonalActions(XS)` (attribute)
▷ `Diagonal2DimensionalGroup(XS)` (attribute)
▷ `Name(SO)` (method)

These are the basic attributes of a crossed square $\mathcal{S}$. The six objects used in the construction of $\mathcal{S}$ are the four crossed modules (2d-groups) on the sides of the square (up; left; right and down); the diagonal action of P on L; and the crossed pairing $\{M,N\} \to L$ (see the next subsection). The diagonal crossed module $(L \to P)$ is an additional attribute.

Example

```
gap> Up2DimensionalGroup( XSconj );
[c5d->d10a]
gap> Right2DimensionalGroup( XSact );
Actor[d10a->d20]
gap> Name( XSconj );
"[c5d->d10a,d10b->d20]"
gap> cross1 := CrossDiagonalActions( XSconj )[1];;
gap> gensa := GeneratorsOfGroup( d10a );;
gap> gensb := GeneratorsOfGroup( d10a );;
```

```
gap> act1 := ImageElm( cross1, gensb[1] );;
gap> gensa[2]; ImageElm( act1, gensa[2] );
(2,10)(3,9)(4,8)(5,7)
(1,5)(2,4)(6,10)(7,9)
```

8.2.14 IsSymmetric3DimensionalGroup

▷ IsSymmetric3DimensionalGroup(*obj*) (property)
▷ IsAbelian3DimensionalGroup(*obj*) (property)
▷ IsTrivialAction3DimensionalGroup(*obj*) (property)
▷ IsNormalSub3DimensionalGroup(*obj*) (property)
▷ IsCentralExtension3DimensionalGroup(*obj*) (property)
▷ IsAutomorphismGroup3DimensionalGroup(*obj*) (property)

These are further properties for 3d-groups, and crossed squares in particular. A 3d-group is *symmetric* if its Up2DimensionalGroup is equal to its Left2DimensionalGroup.

8.2.15 CrossedPairing

▷ CrossedPairing(*XS*) (attribute)
▷ CrossedPairingMap(*xpair*) (attribute)
▷ ImageElmCrossedPairing(*XS, pair*) (operation)
▷ Mapping2ArgumentsByFunction(*MxN, L, map*) (operation)

Crossed pairings have been implemented using an operation Mapping2ArgumentsByFunction. This encodes a map $\{M,N\} \to L$ as a map $M \times N \to L$.

The operation ImageElmCrossedPairing returns the image when a crossed pairing $\{M,N\} \to L$ is applied to the pair $[m,n]$ with $m \in M,\ n \in N$.

The first example shows the crossed pairing in the crossed square XSconj.

Example

```
gap> xp := CrossedPairing( XSconj );
crossed pairing: Group( [ ( 1, 3, 5, 7, 9)( 2, 4, 6, 8,10),
  ( 1,10)( 2, 9)( 3, 8)( 4, 7)( 5, 6), (11,13,15,17,19)(12,14,16,18,20),
  (12,20)(13,19)(14,18)(15,17) ] ) -> c5d
gap> ImageElmCrossedPairing( xp,
>      [ (1,6)(2,5)(3,4)(7,10)(8,9), (1,5)(2,4)(6,9)(7,8) ] );
(1,7,8,5,3)(2,9,10,6,4)
```

The second example shows how to construct a crossed pairing.

Example

```
gap> F := FreeGroup(1);;
gap> x := GeneratorsOfGroup(F)[1];;
gap> z := GroupHomomorphismByImages( F, F, [x], [x^0] );;
gap> id := GroupHomomorphismByImages( F, F, [x], [x] );;
gap> map := Mapping2ArgumentsByFunction( [F,F], F, function(c)
```

```
>           return x^(ExponentSumWord(c[1],x)*ExponentSumWord(c[2],x)); end );;
gap> h := CrossedPairingObj( [F,F], F, map );;
gap> ImageElmCrossedPairing( h, [x^3,x^4] );
f1^12
gap> A := AutomorphismGroup( F );;
gap> a := GeneratorsOfGroup(A)[1];;
gap> act := GroupHomomorphismByImages( F, A, [x], [a^2] );;
gap> X0 := XModByBoundaryAndAction( z, act );;
gap> X1 := XModByBoundaryAndAction( id, act );;
gap> XSF := PreCrossedSquareByPreXMods( X0, X0, X1, X1, X0, h );;
gap> IsCrossedSquare( XSF );
true
```

8.3 Morphisms of crossed squares

This section describes an initial implementation of morphisms of (pre-)crossed squares.

8.3.1 CrossedSquareMorphism

▷ `CrossedSquareMorphism(args)` (function)
▷ `CrossedSquareMorphismByXModMorphisms(src, rng, mors)` (operation)
▷ `CrossedSquareMorphismByGroupHomomorphisms(src, rng, homs)` (operation)
▷ `PreCrossedSquareMorphismByPreXModMorphisms(src, rng, mors)` (operation)
▷ `PreCrossedSquareMorphismByGroupHomomorphisms(src, rng, homs)` (operation)

8.3.2 Source

▷ `Source(map)` (attribute)
▷ `Range(map)` (attribute)
▷ `Up2DimensionalMorphism(map)` (attribute)
▷ `Left2DimensionalMorphism(map)` (attribute)
▷ `Down2DimensionalMorphism(map)` (attribute)
▷ `Right2DimensionalMorphism(map)` (attribute)

Morphisms of `3dObjects` are implemented as `3dMappings`. These have a pair of 3d-groups as source and range, together with four 2d-morphisms mapping between the four pairs of crossed modules on the four sides of the squares. These functions return `fail` when invalid data is supplied.

8.3.3 IsCrossedSquareMorphism

▷ `IsCrossedSquareMorphism(map)` (property)
▷ `IsPreCrossedSquareMorphism(map)` (property)
▷ `IsBijective(mor)` (method)
▷ `IsEndomorphism3dObject(mor)` (property)
▷ `IsAutomorphism3dObject(mor)` (property)

A morphism mor between two pre-crossed squares $\mathcal{S}_1$ and $\mathcal{S}_2$ consists of four crossed module morphisms `Up2DimensionalMorphism(mor)`, mapping the `Up2DimensionalGroup` of $\mathcal{S}_1$ to that of $\mathcal{S}_2$, `Left2DimensionalMorphism(mor)`, `Right2DimensionalMorphism(mor)` and `Down2DimensionalMorphism(mor)`. These four morphisms are required to commute with the four boundary maps and to preserve the rest of the structure. The current version of `IsCrossedSquareMorphism` does not perform all the required checks.

Example

```
gap> ad20 := GroupHomomorphismByImages( d20, d20, [p1,p2], [p1,p2^p1] );;
gap> ad10a := GroupHomomorphismByImages( d10a, d10a, [p1^2,p2], [p1^2,p2^p1] );;
gap> ad10b := GroupHomomorphismByImages( d10b, d10b, [p1^2,p12], [p1^2,p12^p1] );;
gap> idc5d := IdentityMapping( c5d );;
gap> up := Up2DimensionalGroup( XSconj );;
gap> lt := Left2DimensionalGroup( XSconj );;
gap> rt := Right2DimensionalGroup( XSconj );;
gap> dn := Down2DimensionalGroup( XSconj );;
gap> mup := XModMorphismByGroupHomomorphisms( up, up, idc5d, ad10a );
[[c5d->d10a] => [c5d->d10a]]
gap> mlt := XModMorphismByGroupHomomorphisms( lt, lt, idc5d, ad10b );
[[c5d->d10b] => [c5d->d10b]]
gap> mrt := XModMorphismByGroupHomomorphisms( rt, rt, ad10a, ad20 );
[[d10a->d20] => [d10a->d20]]
gap> mdn := XModMorphismByGroupHomomorphisms( dn, dn, ad10b, ad20 );
[[d10b->d20] => [d10b->d20]]
gap> autoconj := CrossedSquareMorphism( XSconj, XSconj, [mup,mlt,mrt,mdn] );;
gap> ord := Order( autoconj );;
gap> Display( autoconj );
Morphism of crossed squares :-
: Source = [c5d->d10a,d10b->d20]
: Range = [c5d->d10a,d10b->d20]
:     order = 5
:    up-left: [ [ ( 1, 3, 5, 7, 9)( 2, 4, 6, 8,10) ],
  [ ( 1, 3, 5, 7, 9)( 2, 4, 6, 8,10) ] ]
:   up-right:
[ [ ( 1, 3, 5, 7, 9)( 2, 4, 6, 8,10), ( 2,10)( 3, 9)( 4, 8)( 5, 7) ],
  [ ( 1, 3, 5, 7, 9)( 2, 4, 6, 8,10), ( 1, 3)( 4,10)( 5, 9)( 6, 8) ] ]
:  down-left:
[ [ ( 1, 3, 5, 7, 9)( 2, 4, 6, 8,10), ( 1,10)( 2, 9)( 3, 8)( 4, 7)( 5, 6) ],
  [ ( 1, 3, 5, 7, 9)( 2, 4, 6, 8,10), ( 1, 2)( 3,10)( 4, 9)( 5, 8)( 6, 7) ] ]
: down-right:
[ [ ( 1, 2, 3, 4, 5, 6, 7, 8, 9,10), ( 2,10)( 3, 9)( 4, 8)( 5, 7) ],
  [ ( 1, 2, 3, 4, 5, 6, 7, 8, 9,10), ( 1, 3)( 4,10)( 5, 9)( 6, 8) ] ]
gap> IsAutomorphismHigherDimensionalDomain( autoconj );
true
gap> KnownPropertiesOfObject( autoconj );
[ "CanEasilyCompareElements", "CanEasilySortElements", "IsTotal",
  "IsSingleValued", "IsInjective", "IsSurjective",
  "IsPreCrossedSquareMorphism", "IsCrossedSquareMorphism",
  "IsEndomorphismHigherDimensionalDomain",
  "IsAutomorphismHigherDimensionalDomain" ]
```

8.3.4 InclusionMorphismHigherDimensionalDomains

▷ `InclusionMorphismHigherDimensionalDomains(obj, sub)` (operation)

8.4 Definitions and constructions for cat^2-groups and their morphisms

We give here three equivalent definitions of cat^2-groups. When we come to define cat^n-groups we shall give a similar set of definitions.

Firstly, we take the definition of a cat^2-group from Section 5 of Brown and Loday [BL87], suitably modified. A cat^2-group $\mathscr{C} = (C_{[2]}, C_{\{2\}}, C_{\{1\}}, C_{\emptyset})$ comprises four groups (one for each of the subsets of $[2]$) and 15 homomorphisms, as shown in the following diagram:

$$\mathscr{C} \quad = \quad \begin{tikzcd}[row sep=huge, column sep=huge]
C_{[2]} \arrow[r, shift left=2, "\ddot{t}_1{,}\ddot{h}_1"] \arrow[r, shift left=1] \arrow[d, shift right=2, "\ddot{t}_2{,}\ddot{h}_2"'] \arrow[d, shift right=1] \arrow[dr, shift right=1, "t_{[2]}{,}h_{[2]}"'] \arrow[dr, shift right=2] & C_{\{2\}} \arrow[l, "\ddot{e}_1"] \arrow[d, shift left=2, "\dot{t}_2{,}\dot{h}_2"] \arrow[d, shift left=1] \\
C_{\{1\}} \arrow[u, "\ddot{e}_2"'] \arrow[r, shift right=2, "\dot{t}_1{,}\dot{h}_1"'] \arrow[r, shift right=1] & C_{\emptyset} \arrow[l, "\dot{e}_1"'] \arrow[u, "\dot{e}_2"] \arrow[ul, shift right=1, "e_{[2]}"']
\end{tikzcd}$$

The following axioms are satisfied by these homomorphisms:

- the four sides of the square (up, left, right, down) are cat^1-groups, denoted $\ddot{\mathscr{C}}_1, \ddot{\mathscr{C}}_2, \dot{\mathscr{C}}_1, \dot{\mathscr{C}}_2$;
- $\dot{t}_1 \circ \ddot{h}_2 = \dot{h}_2 \circ \ddot{t}_1$, $\dot{t}_2 \circ \ddot{h}_1 = \dot{h}_1 \circ \ddot{t}_2$, $\dot{e}_1 \circ \dot{t}_2 = \ddot{t}_2 \circ \ddot{e}_1$, $\dot{e}_2 \circ \dot{t}_1 = \ddot{t}_1 \circ \ddot{e}_2$, $\dot{e}_1 \circ \dot{h}_2 = \ddot{h}_2 \circ \ddot{e}_1$, $\dot{e}_2 \circ \dot{h}_1 = \ddot{h}_1 \circ \ddot{e}_2$;
- $\dot{t}_1 \circ \ddot{t}_2 = \dot{t}_2 \circ \ddot{t}_1 = t_{[2]}$, $\dot{h}_1 \circ \ddot{h}_2 = \dot{h}_2 \circ \ddot{h}_1 = h_{[2]}$, $\dot{e}_1 \circ \ddot{e}_2 = \dot{e}_2 \circ \ddot{e}_1 = e_{[2]}$, making the diagonal a pre-cat^1-group $(e_{[2]}; t_{[2]}, h_{[2]} : C_{[2]} \to C_{\emptyset})$.

It follows from these identities that $(\ddot{t}_1, \dot{t}_1)$, $(\ddot{h}_1, \dot{h}_1)$ and $(\ddot{e}_1, \dot{e}_1)$ are morphisms of cat^1-groups, and similarly in the vertical direction.

Secondly, we give the simplest of the three definitions, adapted from Ellis-Steiner [ES87]. A cat^2-group $\mathscr{C}$ consists of groups G, R_1, R_2 and six homomorphisms $t_1, h_1 : G \to R_2$, $e_1 : R_2 \to G$, $t_2, h_2 : G \to R_1$, $e_2 : R_1 \to G$, satisfying the following axioms for all $1 \leqslant i \leqslant 2$,

- $(t_i \circ e_i)r = r$, $(h_i \circ e_i)r = r$, $\forall r \in R_{[2]\setminus\{i\}}$, $\quad [\ker t_i, \ker h_i] = 1$,
- $(e_1 \circ t_1) \circ (e_2 \circ t_2) = (e_2 \circ t_2) \circ (e_1 \circ t_1)$, $\quad (e_1 \circ h_1) \circ (e_2 \circ h_2) = (e_2 \circ h_2) \circ (e_1 \circ h_1)$,
- $(e_1 \circ t_1) \circ (e_2 \circ h_2) = (e_2 \circ h_2) \circ (e_1 \circ t_1)$, $\quad (e_2 \circ t_2) \circ (e_1 \circ h_1) = (e_1 \circ h_1) \circ (e_2 \circ t_2)$.

Our third definition defines a cat^2-group as a "cat^1-group of cat^1-groups". A cat^2-group $\mathscr{C}$ consists of two cat^1-groups $\mathscr{C}_1 = (e_1; t_1, h_1 : G_1 \to R_1)$ and $\mathscr{C}_2 = (e_2; t_2, h_2 : G_2 \to R_2)$ and cat^1-morphisms $t = (\ddot{t}, \dot{t})$, $h = (\ddot{h}, \dot{h}) : \mathscr{C}_1 \to \mathscr{C}_2$, $e = (\ddot{e}, \dot{e}) : \mathscr{C}_2 \to \mathscr{C}_1$, subject to the following conditions:

$$(t \circ e) \text{ and } (h \circ e) \text{ are the identity mapping on } \mathscr{C}_2, \qquad [\ker t, \ker h] = \{1_{\mathscr{C}_1}\},$$

where $\ker t = (\ker \ddot{t}, \ker \dot{t})$, and similarly for $\ker h$.

A recent paper *Computing 3-Dimensional Groups :L Crossed Squares and Cat2-Groups*, by Arvasi, Odabas and Wensley [AOWar], contains tables listing the numbers of isomorphism classes of cat2-groups on groups of order at most 30 – a total of 1007 cat2-groups.

8.4.1 Cat2Group

▷ `Cat2Group(args)` (function)
▷ `PreCat2Group(args)` (function)
▷ `IsCat2Group(C)` (property)
▷ `PreCat2GroupByPreCat1Groups(L)` (operation)

The global functions `Cat2Group` and `PreCat2Group` are normally called with two arguments - the generating up and left cat^1-groups - or with a single argument which is a crossed square. The operation `PreCat2GroupByPreCat1Groups` has five arguments - the up, left, right, down and diagonal cat^1-groups.

The two cat^2-groups `C2a, C2b` constructed in the following example are isomorphic. They differ in the down-left group `P`.

Example

```
gap> a := (1,2,3,4,5,6);;  b := (2,6)(3,5);;
gap> G := Group( a, b );;  SetName( G, "d12" );
gap> t1 := GroupHomomorphismByImages( G, G, [a,b], [a^3,b] );;
gap> up := PreCat1GroupByEndomorphisms( t1, t1 );;
gap> t2 := GroupHomomorphismByImages( G, G, [a,b], [a^4,b] );;
gap> left := PreCat1GroupByEndomorphisms( t2, t2 );;
gap> C2a := Cat2Group( up, left );
(pre-)cat2-group with generating (pre-)cat1-groups:
1 : [d12 => Group( [ (1,4)(2,5)(3,6), (2,6)(3,5) ] )]
2 : [d12 => Group( [ (1,5,3)(2,6,4), (2,6)(3,5) ] )]
gap> IsCat2Group( C2a );
true
gap> genR := [ (1,4)(2,5)(3,6), (2,6)(3,5) ];;
gap> R := Subgroup( G, genR );;
gap> genQ := [ (1,3,5)(2,4,6), (2,6)(3,5) ];;
gap> Q := Subgroup( G, genQ );;
gap> Pa := Group( b );;  SetName( Pa, "c2a" );
gap> Pb := Group( (7,8) );;  SetName( Pb, "c2b" );
gap> t3 := GroupHomomorphismByImages( R, P, genR, [(),(7,8)] );;
gap> e3 := GroupHomomorphismByImages( P, R, [(7,8)], [(2,6)(3,5)] );;
gap> right := PreCat1GroupByTailHeadEmbedding( t3, t3, e3 );;
gap> t4 := GroupHomomorphismByImages( Q, P, genQ, [(),(7,8)] );;
gap> e4 := GroupHomomorphismByImages( P, Q, [(7,8)], [(2,6)(3,5)] );;
gap> down := PreCat1GroupByTailHeadEmbedding( t4, t4, e4 );;
gap> t0 := t1 * t3;;
gap> e0 := GroupHomomorphismByImages( P, G, [(7,8)], [(2,6)(3,5)] );;
gap> diag := PreCat1GroupByTailHeadEmbedding( t0, t0, e0 );;
gap> C2b := PreCat2GroupByPreCat1Groups( up, left, right, down, diag );
(pre-)cat2-group with generating (pre-)cat1-groups:
1 : [d12 => Group( [ (1,4)(2,5)(3,6), (2,6)(3,5) ] )]
2 : [d12 => Group( [ (1,5,3)(2,6,4), (2,6)(3,5) ] )]
```

```
gap> IsPreCatnGroupWithIdentityEmbeddings( C2b );
false
```

8.4.2 DirectProduct

▷ DirectProduct(*C2a, C2b*) (operation)

The direct product $\mathscr{C}_1 \times \mathscr{C}_2$ has as its four up, left, right and down cat^1-groups the direct products of those in $\mathscr{C}_1$ and $\mathscr{C}_2$. The embeddings and projections are constructed automatically, and placed in the DirectProductInfo attribute, together with the two *objects* $\mathscr{C}_1$ and $\mathscr{C}_2$.

Example

```
gap> C2ab := DirectProductOp( [ C2a, C2b ], C2a );
(pre-)cat2-group with generating (pre-)cat1-groups:
1 : [Group( [ (1,2,3,4,5,6), (2,6)(3,5), ( 7, 8, 9,10,11,12), ( 8,12)( 9,11)
 ] ) => Group( [ (1,4)(2,5)(3,6), (2,6)(3,5), ( 7,10)( 8,11)( 9,12),
  ( 8,12)( 9,11) ] )]
2 : [Group( [ (1,2,3,4,5,6), (2,6)(3,5), ( 7, 8, 9,10,11,12), ( 8,12)( 9,11)
 ] ) => Group( [ (1,5,3)(2,6,4), (2,6)(3,5), ( 7, 9,11)( 8,10,12),
  ( 8,12)( 9,11) ] )]
gap> StructureDescription( C2ab );
[ "C2 x C2 x S3 x S3", "C2 x C2 x C2 x C2", "S3 x S3", "C2 x C2" ]
gap> SetName( C2ab, "C2ab" );
gap> Embedding( C2ab, 1 );
<mapping: (pre-)cat2-group with generating (pre-)cat1-groups:
1 : [d12 => Group( [ (1,4)(2,5)(3,6), (2,6)(3,5) ] )]
2 : [d12 => Group( [ (1,5,3)(2,6,4), (2,6)(3,5) ] )] -> C2ab >
gap> Projection( C2ab, 2 );
<mapping: C2ab -> (pre-)cat2-group with generating (pre-)cat1-groups:
1 : [d12 => Group( [ (1,4)(2,5)(3,6), (2,6)(3,5) ] )]
2 : [d12 => Group( [ (1,5,3)(2,6,4), (2,6)(3,5) ] )] >
```

8.4.3 DisplayLeadMaps

▷ DisplayLeadMaps(*C0*) (operation)

This operation provides an alternative to Display giving a shorter output. Generators of the up-left group are output, together with their images under the up and left tail and head maps.

Example

```
gap> DisplayLeadMaps( C2b );
(pre-)cat2-group with up-left group: [ (1,2,3,4,5,6), (2,6)(3,5) ]
   up tail=head images: [ (1,4)(2,5)(3,6), (2,6)(3,5) ]
 left tail=head images: [ (1,5,3)(2,6,4), (2,6)(3,5) ]
```

8.4.4 Transpose3DimensionalGroup (for cat2-groups)

▷ `Transpose3DimensionalGroup(S0)` (attribute)

The *transpose* of a cat^2-group $\mathscr{C}$ with groups $[G,R,Q,P]$ is the cat^2-group $\tilde{\mathscr{C}}$ with groups $[G,Q,R,P]$.

Example

```
gap> TC2a := Transpose3DimensionalGroup( C2a );
(pre-)cat2-group with generating (pre-)cat1-groups:
1 : [d12 => Group( [ (1,5,3)(2,6,4), (2,6)(3,5) ] )]
2 : [d12 => Group( [ (1,4)(2,5)(3,6), (2,6)(3,5) ] )]
```

8.4.5 Cat2GroupMorphism

▷ `Cat2GroupMorphism(args)` (function)
▷ `Cat2GroupMorphismByCat1GroupMorphisms(src, rng, upmor, ltmor)` (operation)
▷ `Cat2GroupMorphismByGroupHomomorphisms(src, rng, homs)` (operation)
▷ `PreCat2GroupMorphism(args)` (function)
▷ `PreCat2GroupMorphismByPreCat1GroupMorphisms(src, rng, upmor, ltmor)` (operation)
▷ `PreCat2GroupMorphismByGroupHomomorphisms(src, rng, homs)` (operation)

A (pre-)cat^2-group morphism $\mu : \mathscr{C} = (G,R,Q,P) \to \mathscr{C}' = (G',R',Q',P')$ is a list of four group homomorphisms $\gamma : G \to G'$, $\rho : R \to R'$, $\xi : Q \to Q'$ and $\pi : P \to P'$ which commute with all the tail, head and embedding maps so that $(\gamma,\rho),(\gamma,\xi),(\rho,\pi)$ and (ξ,π) are all (pre-)cat^1-group morphisms.

For the operations `(Pre)Cat2GroupMorphismByPreCat1GroupMorphisms` the third and fourth parameters `upmor`, `ltmor` are two cat^1-group morphisms with source the up and left cat^1-groups in $\mathscr{C}$.

For the operations `(Pre)Cat2GroupMorphismByGroupHomomorphisms` the third parameter `mors` is the list $[\gamma,\rho,\xi,\pi]$.

The example constructs an automorphism of `c2a` is two ways, using the two methods described above, an d verifies that the result is the same in each case.

Example

```
gap> gamma := GroupHomomorphismByImages( G, G, [a,b], [a^-1,b] );;
gap> rho := IdentityMapping( R );;
gap> xi := GroupHomomorphismByImages( Q, Q, [a^2,b], [a^-2,b] );;
gap> pi := IdentityMapping( Pa );;
gap> homs := [ gamma, rho, xi, pi ];;
gap> mor1 := Cat2GroupMorphismByGroupHomomorphisms( C2a, C2a, homs );
<mapping: (pre-)cat2-group with generating (pre-)cat1-groups:
1 : [d12 => Group( [ (1,4)(2,5)(3,6), (2,6)(3,5) ] )]
2 : [d12 => Group( [ (1,5,3)(2,6,4), (2,6)(3,5) ] )] -> (pre-)cat
2-group with generating (pre-)cat1-groups:
1 : [d12 => Group( [ (1,4)(2,5)(3,6), (2,6)(3,5) ] )]
2 : [d12 => Group( [ (1,5,3)(2,6,4), (2,6)(3,5) ] )] >
gap> upmor := Cat1GroupMorphism( up, up, gamma, rho );;
gap> ltmor := Cat1GroupMorphism( left, left, gamma, xi );;
```

```
gap> mor2 := Cat2GroupMorphismByCat1GroupMorphisms( C2a, C2a, upmor, ltmor );;
gap> mor1 = mor2;
true
```

8.4.6 Cat2GroupOfCrossedSquare

▷ `Cat2GroupOfCrossedSquare(xsq)` (attribute)
▷ `CrossedSquareOfCat2Group(CC)` (attribute)

These functions provide for conversion between crossed squares and cat^2-groups. (They are the 3-dimensional equivalents of `Cat1GroupOfXMod` (2.5.3) and `XModOfCat1Group` (2.5.3).) The actor crossed square `XSact` was constructed in section `ActorCrossedSquare` (8.2.5).

Example

```
gap> xsC2a := CrossedSquareOfCat2Group( C2a );
crossed square with crossed modules:
      up = [Group( () ) -> Group( [ (1,4)(2,5)(3,6) ] )]
    left = [Group( () ) -> Group( [ (1,3,5)(2,4,6) ] )]
   right = [Group( [ (1,4)(2,5)(3,6) ] ) -> Group( [ (2,6)(3,5) ] )]
    down = [Group( [ (1,3,5)(2,4,6) ] ) -> Group( [ (2,6)(3,5) ] )]

gap> IdGroup( xsC2a );
[ [ 1, 1 ], [ 2, 1 ], [ 3, 1 ], [ 2, 1 ] ]

gap> SetName( Source( Right2DimensionalGroup( XSact ) ), "c5:c4" );
gap> SetName( Range( Right2DimensionalGroup( XSact ) ), "c5:c4" );
gap> Name( XSact );
"[d10a->c5:c4,d20->c5:c4]"

gap> C2act := Cat2GroupOfCrossedSquare( XSact );
(pre-)cat2-group with generating (pre-)cat1-groups:
1 : [((c5:c4 |X c5:c4) |X (d20 |X d10a))=>(c5:c4 |X c5:c4)]
2 : [((c5:c4 |X c5:c4) |X (d20 |X d10a))=>(c5:c4 |X d20)]
gap> Size3d( C2act );
[ 80000, 400, 400, 20 ]
```

8.4.7 Subdiagonal2DimensionalGroup

▷ `Subdiagonal2DimensionalGroup(obj)` (attribute)

The diagonal of a crossed square is always a crossed module, but the diagonal of a cat^2-group need only be a pre-cat^1-group. There is, however, a sub-cat^1-group of this diagonal which, in the case of a cat^2-group constructed from a crossed square, is $(P \ltimes L => P)$. (The name of this operation is very provisional.)

Example

```
gap> G24 := SmallGroup( 24, 10 );;
```

```
gap> w := G24.1;; x := G24.2;; y := G24.3;; z := G24.4;; o := One(G24);;
gap> R := Subgroup( G24, [x,y] );;
gap> txy := GroupHomomorphismByImages( G24, R, [w,x,y,z], [o,x,y,o] );;
gap> exy := GroupHomomorphismByImages( R, G24, [x,y], [x,y] );;
gap> C1xy := PreCat1GroupByTailHeadEmbedding( txy, txy, exy );;
gap> Q := Subgroup( G24, [w,y] );;
gap> twy := GroupHomomorphismByImages( G24, Q, [w,x,y,z], [w,o,y,o] );;
gap> ewy := GroupHomomorphismByImages( Q, G24, [w,y], [w,y] );;
gap> C1wy := PreCat1GroupByTailHeadEmbedding( twy, twy, ewy );;
gap> C2wxy := PreCat2Group( C1xy, C1xy );;
gap> dg := Diagonal2DimensionalGroup( C2wxy );;
gap> IsCat1Group( dg );
false
gap> C1sub := Subdiagonal2DimensionalGroup( C2wxy );;
gap> IsCat1Group( C1sub );
true
gap> IsSub2DimensionalGroup( dg, C1sub );
true
```

8.5 Enumerating cat^2-groups with a given source

This section mirrors that for cat^1-groups (2.6). As the size of a group G increases, the number of cat^2-groups with source G increases rapidly. However, one is usually only interested in the isomorphism classes of cat^2-groups with source G. An iterator `AllCat2GroupsIterator` is provided, which runs through the various cat^2-groups. This iterator finds, for each unordered pair of subgroups R,Q of G, the cat^2-groups whose `Up2DimensionalGroup` has range R, and whose `Left2DimensionalGroup` has range Q. It does this by running through `UnoderedPairsIterator(AllSubgroupsIterator(G))` provided by the Utils package, and then using the iterator `AllCat2GroupsWithImagesIterator(G,R,Q)`.

8.5.1 AllCat2GroupsWithImagesIterator

▷ `AllCat2GroupsWithImagesIterator(G, R, Q)` (operation)
▷ `AllCat2GroupsWithImagesNumber(G, R, Q)` (attribute)
▷ `AllCat2GroupsWithImages(G, R, Q)` (operation)
▷ `AllCat2GroupsWithImagesUpToIsomorphism(G, R, Q)` (operation)

The iterator `AllCat2GroupsWithImagesIterator(G)` iterates through all the cat^2-groups with source `G` and generating cat^1-groups `(G=>R)` and `(G=>Q)`. The attribute `AllCat2GroupsWithImagesNumber(G)` runs through this iterator to determine the number n of these cat^2-groups. The operation `AllCat2GroupsWithImages(G)` returns a list containing these n cat^2-groups. Since these lists can get very long, this operation should only be used for simple cases. The operation `AllCat2GroupsWithImagesUpToIsomorphism(G)` returns representatives of the isomorphism classes of these cat^2-groups.

Example

```
gap> G8 := Group( (1,2), (3,4), (5,6) );;
```

```
gap> A := Subgroup( G8, [ (1,2) ] );;
gap> B := Subgroup( G8, [ (3,4) ] );;
gap> AllCat2GroupsWithImagesNumber( G8, A, A );
4
gap> all := AllCat2GroupsWithImages( G8, A, A );;
gap> for C2 in all do DisplayLeadMaps( C2 ); od;
(pre-)cat2-group with up-left group: [ (1,2), (3,4), (5,6) ]
   up tail=head images: [ (1,2), (1,2), () ]
 left tail=head images: [ (1,2), (1,2), () ]
(pre-)cat2-group with up-left group: [ (1,2), (3,4), (5,6) ]
   up tail=head images: [ (1,2), (), () ]
 left tail=head images: [ (1,2), (), () ]
(pre-)cat2-group with up-left group: [ (1,2), (3,4), (5,6) ]
   up tail=head images: [ (1,2), (), (1,2) ]
 left tail=head images: [ (1,2), (), (1,2) ]
(pre-)cat2-group with up-left group: [ (1,2), (3,4), (5,6) ]
   up tail=head images: [ (1,2), (1,2), (1,2) ]
 left tail=head images: [ (1,2), (1,2), (1,2) ]
gap> AllCat2GroupsWithImagesNumber( G8, A, B );
16
gap> iso := AllCat2GroupsWithImagesUpToIsomorphism( G8, A, B );;
gap> for C2 in iso do DisplayLeadMaps( C2 ); od;
(pre-)cat2-group with up-left group: [ (1,2), (3,4), (5,6) ]
   up tail=head images: [ (1,2), (), () ]
 left tail=head images: [ (), (3,4), () ]
(pre-)cat2-group with up-left group: [ (1,2), (3,4), (5,6) ]
   up tail=head images: [ (1,2), (), () ]
 left tail/head images: [ (), (3,4), () ], [ (), (3,4), (3,4) ]
(pre-)cat2-group with up-left group: [ (1,2), (3,4), (5,6) ]
   up tail/head images: [ (1,2), (), () ], [ (1,2), (), (1,2) ]
 left tail/head images: [ (), (3,4), () ], [ (), (3,4), (3,4) ]
```

8.5.2 AllCat2GroupsWithFixedUp

▷ `AllCat2GroupsWithFixedUp(`*C*`)` (operation)

▷ `AllCat2GroupsWithFixedUpAndLeftRange(`*C, R*`)` (operation)

The operation `AllCat2GroupsWithFixedUp(C)` constructs all the cat^2-groups with a fixed `Up2DimensionalGroup` *C*. In the second operation the user may also specify the range of the `Left2DimensionalGroup`.

Example

```
gap> up := Up2DimensionalGroup( iso[1] );
[Group( [ (1,2), (3,4), (5,6) ] )=>Group( [ (1,2), (), () ] )]
gap> AllCat2GroupsWithFixedUp( up );;
gap> Length(last);
28
gap> L := AllCat2GroupsWithFixedUpAndLeftRange( up, B );;
gap> for C in L do DisplayLeadMaps( C ); od;
(pre-)cat2-group with up-left group: [ (1,2), (3,4), (5,6) ]
```

```
   up tail=head images: [ (1,2), (), () ]
 left tail=head images: [ (), (3,4), () ]
(pre-)cat2-group with up-left group: [ (1,2), (3,4), (5,6) ]
   up tail=head images: [ (1,2), (), () ]
 left tail/head images: [ (), (3,4), () ], [ (), (3,4), (3,4) ]
(pre-)cat2-group with up-left group: [ (1,2), (3,4), (5,6) ]
   up tail=head images: [ (1,2), (), () ]
 left tail/head images: [ (), (3,4), (3,4) ], [ (), (3,4), () ]
(pre-)cat2-group with up-left group: [ (1,2), (3,4), (5,6) ]
   up tail=head images: [ (1,2), (), () ]
 left tail=head images: [ (), (3,4), (3,4) ]
```

8.5.3 AllCat2GroupsMatrix

▷ `AllCat2GroupsMatrix(G)` (attribute)

The operation `AllCat2GroupsMatrix(G)` constructs a symmetric matrix M with rows and columns labelled by the cat^1-groups C_i on G, where M_{ij} is 1 if C_i, C_j combine to form a cat^2-group, and 0 otherwise. The matrix is automatically printed out with dots in place of zeroes.

In the example we see that the dihedral group D_{12} has 12 cat^1-groups and 41 cat^2-groups, 12 of which are symmetric. This operation is intended to be used to illustrate how cat^2-groups are formed, and should only be used with groups of low order.

The attribute `AllCat2GroupsNumber(G)` returns the number n of these cat^2-groups.

Example

```
gap> AllCat2GroupsMatrix(d12);;
number of cat2-groups found = 41
1.....1..1.1
.1.....1.1.1
..1.....11.1
...1....1.11
....1.1...11
.....1.1..11
1...1.1..111
.1...1.1.111
..11....1111
111...1111.1
...111111.11
111111111111
gap> AllCat2GroupsNumber(d12);
41
```

8.5.4 AllCat2GroupsIterator

▷ `AllCat2GroupsIterator(G)` (operation)
▷ `AllCat2Groups(G)` (operation)
▷ `AllCat2GroupsUpToIsomorphism(G)` (operation)

▷ `AllCat2GroupFamilies(G)` (operation)
▷ `CatnGroupNumbers(G)` (attribute)
▷ `CatnGroupLists(G)` (attribute)

The iterator `AllCat2GroupsIterator(G)` iterates through all the cat^2-groups with source `G`. The operation `AllCat2Groups(G)` returns a list containing these n cat^2-groups. Since these lists can get very long, this operation should only be used for simple cases. The operation `AllCat2GroupsUpToIsomorphism(G)` returns representatives of the isomorphism classes of these subgroups. The operation `AllCat2GroupFamilies(G)` returns a list of lists. The k-th list contains the positions of the cat^2-groups in `AllCat2Groups(G)` which are isomorphic to the k-th representative. So, for `d12`, the 41 cat^2-groups form 10 classes, and the sizes of these classes are `[6,6,6,6,3,6,3,2,2,1]`. Four of these classes contain symmetric cat^2-groups.

The field `CatnGroupNumbers(G).cat2` is the number of cat^2-groups on G, while `CatnGroupNumbers(G).iso2` is the number of isomorphism classes of these cat^2-groups. Also `CatnGroupNumbers(G).symm` is the number of cat^2-groups whose `Up2DimensionalGroup` is the same as the `Left2DimensionalGroup`, while `CatnGroupNumbers(G).siso` is the number of isomorphism classes of these symmetric cat^2-groups.

Provided that `CatnGroupLists(G).omit` is not set to `true`, *sorted* lists of generating pairs, and of the classes they belong to, are added to the record `CatnGroupLists`. For example `[5,7]` in these lists for `d12` indicates that there is a cat^2-group generated by the fifth and seventh cat^1-groups and that this is in the second class whose representative is `[1,7]`. Classes `[1,5,8,10]` contain symmetric cat^2-groups.

Example

```
gap> AllCat2GroupsNumber( d12 );
41
gap> reps2 := AllCat2GroupsUpToIsomorphism( d12 );;
gap> Length( reps2 );
10
gap> List( reps2, C -> StructureDescription( C ) );
[ [ "D12", "C2", "C2", "C2" ], [ "D12", "C2", "C2 x C2", "C2" ],
  [ "D12", "C2", "S3", "C2" ], [ "D12", "C2", "D12", "C2" ],
  [ "D12", "C2 x C2", "C2 x C2", "C2 x C2" ], [ "D12", "C2 x C2", "S3", "C2" ]
    , [ "D12", "C2 x C2", "D12", "C2 x C2" ], [ "D12", "S3", "S3", "S3" ],
  [ "D12", "S3", "D12", "S3" ], [ "D12", "D12", "D12", "D12" ] ]
gap> fams := AllCat2GroupFamilies( d12 );
[ [ 1, 2, 3, 4, 5, 6 ], [ 7, 8, 10, 11, 13, 14 ], [ 16, 17, 18, 23, 24, 25 ],
  [ 30, 31, 32, 33, 34, 35 ], [ 9, 12, 15 ], [ 19, 20, 21, 26, 27, 28 ],
  [ 36, 37, 38 ], [ 22, 29 ], [ 39, 40 ], [ 41 ] ]
gap> CatnGroupNumbers( d12 );
rec( cat1 := 12, cat2 := 41, idem := 21, iso1 := 4, iso2 := 10,
  isopredg := 0, predg := 0, siso := 4, symm := 12 )
gap> CatnGroupLists( d12 );
rec( allcat2pos := [ 1, 7, 9, 16, 19, 22, 30, 36, 39, 41 ],
  cat2classes :=
    [ [ [ 1, 1 ], [ 2, 2 ], [ 3, 3 ], [ 4, 4 ], [ 5, 5 ], [ 6, 6 ] ],
      [ [ 1, 7 ], [ 5, 7 ], [ 2, 8 ], [ 6, 8 ], [ 3, 9 ], [ 4, 9 ] ],
      [ [ 1, 10 ], [ 2, 10 ], [ 3, 10 ], [ 4, 11 ], [ 5, 11 ], [ 6, 11 ] ],
      [ [ 1, 12 ], [ 2, 12 ], [ 3, 12 ], [ 4, 12 ], [ 5, 12 ], [ 6, 12 ] ],
      [ [ 7, 7 ], [ 8, 8 ], [ 9, 9 ] ],
```

```
    [ [ 7, 10 ], [ 8, 10 ], [ 9, 10 ], [ 7, 11 ], [ 8, 11 ], [ 9, 11 ] ],
    [ [ 7, 12 ], [ 8, 12 ], [ 9, 12 ] ], [ [ 10, 10 ], [ 11, 11 ] ],
    [ [ 10, 12 ], [ 11, 12 ] ], [ [ 12, 12 ] ] ],
cat2pairs := [ [ 1, 1 ], [ 1, 7 ], [ 1, 10 ], [ 1, 12 ], [ 2, 2 ],
    [ 2, 8 ], [ 2, 10 ], [ 2, 12 ], [ 3, 3 ], [ 3, 9 ], [ 3, 10 ],
    [ 3, 12 ], [ 4, 4 ], [ 4, 9 ], [ 4, 11 ], [ 4, 12 ], [ 5, 5 ],
    [ 5, 7 ], [ 5, 11 ], [ 5, 12 ], [ 6, 6 ], [ 6, 8 ], [ 6, 11 ],
    [ 6, 12 ], [ 7, 7 ], [ 7, 10 ], [ 7, 11 ], [ 7, 12 ], [ 8, 8 ],
    [ 8, 10 ], [ 8, 11 ], [ 8, 12 ], [ 9, 9 ], [ 9, 10 ], [ 9, 11 ],
    [ 9, 12 ], [ 10, 10 ], [ 10, 12 ], [ 11, 11 ], [ 11, 12 ], [ 12, 12 ] ],
omit := false, pisopos := [  ], sisopos := [ 1, 5, 8, 10 ] )
```

Chapter 9

Crossed cubes and Cat^3-groups

The term *4d-group* refers to a set of equivalent categories of which the most common are the categories of *crossed cubes* and *cat^3-groups*. A *4d-mapping* is a function between two 4d-groups which preserves all the structure.

The material in this chapter should be considered very experimental. As yet there are no functions for crossed cubes.

9.1 Functions for (pre-)cat^3-groups

We shall use the following standard orientation of a cat^3-group $\mathcal{E}$ on a group G. $\mathcal{E}$ contains 8 groups; 12 cat^1-groups and 6 cat^2-groups forming the vertices; edges and faces of a cube, as shown in the following diagram.

$$\begin{tikzcd}
H \arrow[rr, shift left=2] \arrow[rr, shift left] \arrow[rr, leftarrow, shift right] \arrow[dr, shift left=2] \arrow[dr, shift left] \arrow[dr, leftarrow, shift right] \arrow[ddd, shift left=2] \arrow[ddd, shift left] \arrow[ddd, leftarrow, shift right] & & N \arrow[dr, shift left=2] \arrow[dr, shift left] \arrow[dr, leftarrow, shift right] \arrow[ddd, shift left=2] \arrow[ddd, shift left] \arrow[ddd, leftarrow, shift right] & \\
& G \arrow[rr, shift left=2] \arrow[rr, shift left] \arrow[rr, leftarrow, shift right] \arrow[ddd, shift left=2] \arrow[ddd, shift left] \arrow[ddd, leftarrow, shift right] & & R \arrow[ddd, shift left=2] \arrow[ddd, shift left] \arrow[ddd, leftarrow, shift right] \\
& & & \\
M \arrow[rr, shift left=2] \arrow[rr, shift left] \arrow[rr, leftarrow, shift right] \arrow[dr, shift left=2] \arrow[dr, shift left] \arrow[dr, leftarrow, shift right] & & L \arrow[dr, shift left=2] \arrow[dr, shift left] \arrow[dr, leftarrow, shift right] & \\
& Q \arrow[rr, shift left=2] \arrow[rr, shift left] \arrow[rr, leftarrow, shift right] & & P
\end{tikzcd}$$

By definition, $\mathcal{E}$ is generated by three commuting cat^1-groups $(G \Rightarrow R), (G \Rightarrow Q)$ and $(G \Rightarrow H)$, but it is more convenient to think of $\mathcal{E}$ as generated by two cat^2-groups

- *front*$(\mathcal{E})$, generated by $(G \Rightarrow R)$ and $(G \Rightarrow Q)$;
- *left*$(\mathcal{E})$, generated by $(G \Rightarrow Q)$ and $(G \Rightarrow H)$.

Because the tail, head and embedding maps all commute, it follows that *up*$(\mathcal{E})$, generated by $(G \Rightarrow H)$ and $(G \Rightarrow R)$, is a third cat^2-group. The three remaining faces (cat^2-groups) *right*$(\mathcal{E})$, *down*$(\mathcal{E})$ and *back*($\mathscr{E}$) are then easily constructed. We shall always use the order [*front,up,left,right,down,back*] for the six faces.

9.1.1 Cat3Group

▷ `Cat3Group(args)` (function)
▷ `PreCat3Group(args)` (function)
▷ `IsCat3Group(C)` (property)
▷ `PreCat3GroupByPreCat2Groups(L)` (operation)

The global functions `Cat3Group` and `PreCat3Group` are normally take as arguments a pair of cat^2-groups or a trio of cat^1-groups. In subsection `AllCat2GroupsIterator` (8.5.4) the list of pairs `CatnGroupLists(d12).pairs` contains the three entries `[6,8]`, `[8,11]` and `[6,11]`. It follows that the sixth, eighth and eleventh cat^1-groups for `d12` generate a cat^3-group.

Example

```
gap> all1 := AllCat1Groups( d12 );;
gap> C68 := Cat2Group( all1[6], all1[8] );;
gap> C811 := Cat2Group( all1[8], all1[11] );;
gap> C3Ga := Cat3Group( C68, C811 );
cat3-group with generating (pre-)cat1-groups:
1 : [d12 => Group( [ (), (1,6)(2,5)(3,4) ] )]
2 : [d12 => Group( [ (1,4)(2,5)(3,6), (1,3)(4,6) ] )]
3 : [d12 => Group( [ (1,5,3)(2,6,4), (1,4)(2,3)(5,6) ] )]
gap> C3Gb := Cat3Group( all1[6], all1[8], all1[11] );
cat3-group with generating (pre-)cat1-groups:
1 : [d12 => Group( [ (), (1,6)(2,5)(3,4) ] )]
2 : [d12 => Group( [ (1,4)(2,5)(3,6), (1,3)(4,6) ] )]
3 : [d12 => Group( [ (1,5,3)(2,6,4), (1,4)(2,3)(5,6) ] )]
gap> C3Ga = C3Gb;
true
```

9.1.2 Front3DimensionalGroup

▷ `Front3DimensionalGroup(C3)` (attribute)
▷ `Up3DimensionalGroup(C3)` (attribute)
▷ `Left3DimensionalGroup(C3)` (attribute)
▷ `Right3DimensionalGroup(C3)` (attribute)
▷ `Down3DimensionalGroup(C3)` (attribute)
▷ `Back3DimensionalGroup(C3)` (attribute)

The six faces of a cat^3-group are stored as these attributes.

Example

```
gap> C116 := Cat2Group( all1[11], all1[6] );;
gap> Up3DimensionalGroup( C3Ga ) = C116;
true
```

9.2 Enumerating cat^3-groups with a given source

Once the list `CatnGroupLists(G).pairs` has been obtained we may seek all triples $[i,j],[j,k]$ and $[k,i]$ or $[i,k]$ of pairs in this list and then, for each such triple, construct a cat^3-group generated by the i-th, j-th and k-th cat^1-group on G.

9.2.1 AllCat3GroupTriples

▷ `AllCat3GroupTriples(G)` (operation)
▷ `AllCat3GroupsNumber(G)` (attribute)
▷ `AllCat3Groups(G)` (operation)

The list of triples returned by the operation `AllCat3GroupTriples` is saved as `CatnGroupLists(G).cat3triples`. The length of this list is the number of cat^3-groups on G, and is saved as `CatnGroupNumbers(G).cat3`.

As yet there is no operation `AllCat3GroupsUpToIsomorphism(G)`.

Example

```
gap> triples := AllCat3GroupTriples( d12 );;
gap> CatnGroupNumbers( d12 ).cat3;
94
gap> triples[46];
[ 5, 7, 11 ]
gap> all1 := AllCat1Groups( d12 );;
gap> Cat3Group( all1[5], all1[7], all1[11] );
cat3-group with generating (pre-)cat1-groups:
1 : [d12 => Group( [ (), (1,4)(2,3)(5,6) ] )]
2 : [d12 => Group( [ (1,4)(2,5)(3,6), (2,6)(3,5) ] )]
3 : [d12 => Group( [ (1,5,3)(2,6,4), (1,4)(2,3)(5,6) ] )]
```

9.3 Definition and constructions for cat^n-groups and their morphisms

In this chapter and the previous one we are interested in cat^2-groups and cat^3-groups, and it is convenient in this section to give the more general definition. There are three equivalent descriptions of a cat^n-group.

A *cat^n-group* consists of the following.

- 2^n groups G_A, one for each subset A of $[n]$, the *vertices* of an n-cube.
- Group homomorphisms forming $n2^{n-1}$ commuting cat^1-groups,
$$\mathscr{C}_{A,i} \;=\; (e_{A,i};\, t_{A,i},\, h_{A,i} \;:\; G_A \to G_{A\setminus\{i\}}), \quad \text{for all} \quad A \subseteq [n],\ i \in A,$$
the *edges* of the cube.
- These cat^1-groups combine (in sets of 4) to form $n(n-1)2^{n-3}$ cat^2-groups $\mathscr{C}_{A,\{i,j\}}$ for all $\{i,j\} \subseteq A \subseteq [n],\ i \neq j$, the *faces* of the cube.

Note that, since the $t_{A,i}, h_{A,i}$ and $e_{A,i}$ commute, composite homomorphisms $t_{A,B}, h_{A,B} : G_A \to G_{A\setminus B}$ and $e_{A,B} : G_{A\setminus B} \to G_A$ are well defined for all $B \subseteq A \subseteq [n]$.

Secondly, we give the simplest of the three descriptions, again adapted from Ellis-Steiner [ES87].

A cat^n-group $\mathscr{C}$ consists of 2^n groups G_A, one for each subset A of $[n]$, and $3n$ homomorphisms

$$t_{[n],i}, h_{[n],i} : G_{[n]} \to G_{[n]\setminus\{i\}},\ e_{[n],i} : G_{[n]\setminus\{i\}} \to G_{[n]},$$

satisfying the following axioms for all $1 \leqslant i \leqslant n$,}

- the $\mathscr{C}_{[n],i} = (e_{[n],i};\ t_{[n],i},\ h_{[n],i} : G_{[n]} \to G_{[n]\setminus\{i\}})$ are *commuting* cat^1-groups, so that:
- $(e_1 \circ t_1) \circ (e_2 \circ t_2) = (e_2 \circ t_2) \circ (e_1 \circ t_1), \quad (e_1 \circ h_1) \circ (e_2 \circ h_2) = (e_2 \circ h_2) \circ (e_1 \circ h_1),$
- $(e_1 \circ t_1) \circ (e_2 \circ h_2) = (e_2 \circ h_2) \circ (e_1 \circ t_1), \quad (e_2 \circ t_2) \circ (e_1 \circ h_1) = (e_1 \circ h_1) \circ (e_2 \circ t_2).$

Our third description defines a cat^n-group as a "cat^1-group of cat$^{(n-1)}$-groups".

A *cat^n-group* $\mathscr{C}$ consists of two cat$^{(n-1)}$-groups:

- $\mathscr{A}$ with groups G_A, $A \subseteq [n-1]$, and homomorphisms $\ddot{t}_{A,i}, \ddot{h}_{A,i}, \ddot{e}_{A,i}$,
- $\mathscr{B}$ with groups H_B, $B \subseteq [n-1]$, and homomorphisms $\dot{t}_{B,i}, \dot{h}_{B,i}, \dot{e}_{B,i}$, and
- cat$^{(n-1)}$-morphisms $t, h : \mathscr{A} \to \mathscr{B}$ and $e : \mathscr{B} \to \mathscr{A}$ subject to the following conditions:

$$(t \circ e) \text{ and } (h \circ e) \text{ are the identity mapping on } \mathscr{B}, \qquad [\ker t, \ker h] = \{1_{\mathscr{A}}\}.$$

9.3.1 PreCatnGroup

▷ `PreCatnGroup(L)` (operation)
▷ `CatnGroup(L)` (operation)

The operation `(Pre)CatnGroup` expects as input a list of cat^1-groups.

Example

```
gap> PC4 := PreCatnGroup( [ all1[5], all1[7], all1[11], all1[12] ] );
(pre-)cat4-group with generating (pre-)cat1-groups:
1 : [d12 => Group( [ (), (1,4)(2,3)(5,6) ] )]
2 : [d12 => Group( [ (1,4)(2,5)(3,6), (2,6)(3,5) ] )]
3 : [d12 => Group( [ (1,5,3)(2,6,4), (1,4)(2,3)(5,6) ] )]
4 : [d12 => Group( [ (1,2,3,4,5,6), (2,6)(3,5) ] )]
gap> IsCatnGroup( PC4 );
true
gap> HigherDimension( PC4 );
5
```

Chapter 10

Crossed modules of groupoids

The material documented in this chapter is experimental, and is likely to be changed in due course.

10.1 Constructions for crossed modules of groupoids

A typical example of a crossed module $\mathcal{X}$ over a groupoid has for its range a connected groupoid. This is a direct product of a group with a complete graph, and we call the vertices of the graph the *objects* of the crossed module. The source of $\mathcal{X}$ is a groupoid, with the same objects, which is either discrete or connected. The boundary morphism is constant on objects. For details and other references see [AW10].

10.1.1 SinglePiecePreXModWithObjects

▷ `SinglePiecePreXModWithObjects(pxmod, obs, isdisc)` (operation)

At present the experimental operation `SinglePiecePreXModWithObjects` accepts a precrossed module `pxmod`, a set of objects `obs`, and a boolean `isdisc` which is `true` when the source groupoid is homogeneous and discrete and `false` when the source groupoid is connected. Other operations will be added as time permits.

In the example the crossed module `DX4` has discrete source, while the crossed module `CX4` has connected source. These are groupoid equivalents of `XModByNormalSubgroup` (2.1.2).

Example

```
gap> s4 := Group( (1,2,3,4), (3,4) );;
gap> SetName( s4, "s4" );
gap> a4 := Subgroup( s4, [ (1,2,3), (2,3,4) ] );;
gap> SetName( a4, "a4" );
gap> X4 := XModByNormalSubgroup( s4, a4 );;
gap> DX4 := SinglePiecePreXModWithObjects( X4, [-9,-8,-7], true );
single piece crossed module with objects
  source groupoid:
    homogeneous, discrete groupoid: < a4, [ -9, -8, -7 ] >
  and range groupoid:
    single piece groupoid: < s4, [ -9, -8, -7 ] >
gap> Da4 := Source( DX4 );;
gap> Ds4 := Range( DX4 );;
```

```
gap> CX4 := SinglePiecePreXModWithObjects( X4, [-9,-8,-7], false );
single piece crossed module with objects
  source groupoid:
    single piece groupoid: < a4, [ -9, -8, -7 ] >
  and range groupoid:
    single piece groupoid: < s4, [ -9, -8, -7 ] >
gap> Ca4 := Source( CX4 );;
gap> Cs4 := Range( CX4 );;
```

10.1.2 IsXModWithObjects

▷ `IsXModWithObjects(pxmod)` (property)
▷ `IsPreXModWithObjects(pxmod)` (property)
▷ `IsDirectProductWithCompleteDigraphDomain(pxmod)` (property)

The precrossed module `DX4` belongs to the category `Is2DimensionalGroupWithObjects` and is, of course, a crossed module.

Example

```
gap> Set( KnownPropertiesOfObject( DX4 ) );
[ "CanEasilyCompareElements", "CanEasilySortElements", "IsAssociative",
  "IsDirectProductWithCompleteDigraphDomain", "IsDuplicateFree",
  "IsGeneratorsOfSemigroup", "IsPreXModWithObjects", "IsSinglePieceDomain",
```

10.1.3 IsPermPreXModWithObjects

▷ `IsPermPreXModWithObjects(pxmod)` (property)
▷ `IsPcPreXModWithObjects(pxmod)` (property)
▷ `IsFpPreXModWithObjects(pxmod)` (property)

To test these properties we test the precrossed modules from which they were constructed.

Example

```
gap> IsPermPreXModWithObjects( CX4 );
true
gap> IsPcPreXModWithObjects( CX4 );
false
gap> IsFpPreXModWithObjects( CX4 );
false
```

10.1.4 Root2dGroup

▷ `Root2dGroup(pxmod)` (attribute)
▷ `XModAction(pxmod)` (attribute)

The attributes of a precrossed module with objects include the standard `Source`; `Range`; `Boundary` (2.1.9); and `XModAction` (2.1.9) as with precrossed modules of groups. There is also `ObjectList`, as in the groupoids package. Additionally there is `Root2dGroup` which is the underlying precrossed module used in the construction.

Note that `XModAction` is now a groupoid homomorphism from the source groupoid to a one-object groupoid (with object `0`) where the group is the automorphism group of the range groupoid.

Example

```
gap> Set( KnownAttributesOfObject( CX4 ) );
[ "Boundary", "ObjectList", "Range", "Root2dGroup", "Source", "XModAction" ]
gap> Root2dGroup( CX4 );
[a4->s4]
gap> act := XModAction( CX4 );;
gap> Size( Range( act ) );
20736
gap> r := Arrow( Cs4, (1,2,3,4), -4, -5 );;
gap> ImageElm( act, r );
[groupoid homomorphism :
[ [ [(1,2,3) : -6 -> -6], [(2,3,4) : -6 -> -6], [() : -6 -> -5],
      [() : -6 -> -4] ],
  [ [(2,3,4) : -6 -> -6], [(1,3,4) : -6 -> -6], [() : -6 -> -4],
      [() : -6 -> -5] ] ] : 0 -> 0]
gap> s := Arrow( Ca4, (1,2,4), -5, -5 );;
gap> ##  calculate s^r
gap> ims := ImageElmXModAction( CX4, s, r );
[(1,2,3) : -4 -> -4]
```

There is much more to be done with these constructions.

Chapter 11

Double Groupoids

A *double groupoid* is a *double category* in which all the category structures are groupoids. There is also a pre-crossed module associated to the double groupoid. In a double groupoid, as well as objects and arrows we need a set of *squares*. A square is bounded by four arrows, two horizantal and two vertical, and there is a *horizantal* groupoid structure and a *vertical* groupoid structure on these squares. An element of the source of the pre-crossed module is located at the centre of the square, and its image under the boundary map is equal to the boundary of the square.

The double groupoids constructed here are special in that all four arrows come from the same groupoid. We call these *edge-symmetric* double groupoids.

It is assumed in this chapter that the reader is familiar with constructions for groupoids given in the Groupoids package, such as `SinglePieceBasicDoubleGroupoid`. Such groupoids are *basic*, in that there is no pre-crossed module involvement.

This chapter is experimental, and will be extended as soon as possible.

11.1 Double groupoid squares

Let G be a groupoid with object set Ω. Let $\square$ be the set of squares with objects from Ω at each corner; plus two vertical arrows and two horizantal arrows from $\mathrm{Arr}(G)$. Further, let $\mathscr{P} = (\partial : S \to R)$ be a pre-crossed module, and let $m_1 \in S$ be placed at the centre of the square. The following picture illustrates the situation:

$$\begin{tikzcd}
u_1 \arrow[r, "a_1"] \arrow[d, "d_1"'] \arrow[dr, phantom, "m_1"] & u_2 \arrow[d, "e_1"] \\
v_1 \arrow[r, "b_1"'] & v_2
\end{tikzcd}$$

We think of the square being *based* at the bottom, right-hand corner, v_2. The *boundary* of the square is the loop $(v_2, b_1^{-1}d_1^{-1}a_1e_1, v_2) = (v_2, p_1, v_2)$. The *boundary condition* which m_1 has to satisfy is that

$\partial m_1 = p_1$. When defining a *horizantal composition*, as illustrated by

$$\begin{tikzcd} u_1 \arrow[r, "a_1"] \arrow[d, "d_1"'] \arrow[dr, phantom, "m_1"] & u_2 \arrow[r, "a_2"] \arrow[d, "e_1"] \arrow[dr, phantom, "m_2"] & u_3 \arrow[d, "f_1"] \\ v_1 \arrow[r, "b_1"'] & v_2 \arrow[r, "b_2"'] & v_3 \end{tikzcd} \quad = \quad \begin{tikzcd} u_1 \arrow[r, "a_1a_2"] \arrow[d, "d_1"'] \arrow[dr, phantom, "m_1^{b_2}m_2"] & u_3 \arrow[d, "f_1"] \\ v_1 \arrow[r, "b_1b_2"'] & v_3 \end{tikzcd}$$

we have to move m_1, based at v_2, to the new base v_3, and we do this by using the action of the pre-crossed module of b_2 on m_1. Notice that the boundary condition is satisfied, since the first pre-crossed module axiom applies:

$$\partial(m_1^{b_2} m_2) \;=\; \partial(m_1^{b_2})(\partial m_2) \;=\; b_2^{-1}(b_1^{-1}d_1^{-1}a_1e_1)b_2(b_2^{-1}e_1^{-1}a_2f_1) \;=\; (b_1b_2)^{-1}d_1^{-1}(a_1a_2)f_1.$$

Similarly, vertical composition is illustrated by

$$\begin{tikzcd} u_1 \arrow[r, "a_1"] \arrow[d, "d_1"'] \arrow[dr, phantom, "m_1"] & u_2 \arrow[d, "e_1"] \\ v_1 \arrow[r, "b_1"'] \arrow[d, "d_2"'] \arrow[dr, phantom, "m_3"] & v_2 \arrow[d, "e_2"] \\ w_1 \arrow[r, "c_1"'] & w_2 \end{tikzcd} \quad = \quad \begin{tikzcd} u_1 \arrow[r, "a_1"] \arrow[d, "d_1d_2"'] \arrow[dr, phantom, "m_3m_1^{e_2}"] & u_2 \arrow[d, "e_1e_2"] \\ w_1 \arrow[r, "c_1"'] & w_2 \end{tikzcd}$$

Again the boundary condition is satisfied:

$$\partial(m_3 m_1^{e_2}) \;=\; (\partial m_3)\partial(m_1^{e_2}) \;=\; (c_1^{-1}d_2^{-1}b_1e_2)e_2^{-1}(b_1^{-1}d_1^{-1}a_1e_1)e_2 \;=\; c_1^{-1}(d_1d_2)^{-1}a_1(e_1e_2).$$

These two compositions commute, so we may construct products such as:

$$\begin{tikzcd} u_1 \arrow[r, "a_1"] \arrow[d, "d_1"'] \arrow[dr, phantom, "m_1"] & u_2 \arrow[r, "a_2"] \arrow[d, "e_1"] \arrow[dr, phantom, "m_2"] & u_3 \arrow[d, "f_1"] \\ v_1 \arrow[r, "b_1"] \arrow[d, "d_2"'] \arrow[dr, phantom, "m_3"] & v_2 \arrow[r, "b_2"] \arrow[d, "e_2"] \arrow[dr, phantom, "m_4"] & v_3 \arrow[d, "f_2"] \\ w_1 \arrow[r, "c_1"'] & w_2 \arrow[r, "c_2"'] & w_3 \end{tikzcd} \quad = \quad \begin{tikzcd} u_1 \arrow[r, "a_1a_2"] \arrow[d, "d_1d_2"'] \arrow[dr, phantom, "m_3^{c_2}m_4\left(m_1^{b_2}m_2\right)^{f_2}"] & u_3 \arrow[d, "f_1f_2"] \\ w_1 \arrow[r, "c_1c_2"'] & w_3 \end{tikzcd}$$

where

$$m_3^{c_2} m_4 (m_1^{b_2} m_2)^{f_2} \;=\; (m_3 m_1^{e_2})^{c_2} m_4 m_2^{f_2} \;=\; (c_1c_2)^{-1}(d_1d_2)^{-1}(a_1a_2)(f_1f_2).$$

For an example we take for our groupoid the product of the group $S_3 = \langle (7,8),(7,9) \rangle$ with the complete graph on $[-6\ldots-1]$ and, for our pre-crossed module, the `X12`, isomorphic to $(D_{12} \to S_3)$, constructed using `XModByCentralExtension` (2.1.5). The source of `X12` has generating set $\{g = (11,12,13,14,15,16),\ h = (12,16)(13,15)\}$. We check that the two ways of computing the product of four squares below agree.

$$\begin{tikzcd}
-6 \arrow[r, "(7{,}8)"] \arrow[d, "(7{,}9)"'] \arrow[dr, phantom, "gh"] & -5 \arrow[r, "(8{,}9)"] \arrow[d, "(8{,}9)"] \arrow[dr, phantom, "g^2"] & -4 \arrow[d, "(7{,}8)"] \\
-1 \arrow[r, "(7{,}8{,}9)"] \arrow[d, "(8{,}9)"'] \arrow[dr, phantom, "g"] & -3 \arrow[r, "(7{,}9)"] \arrow[d, "(7{,}9{,}8)"] \arrow[dr, phantom, "h"] & -4 \arrow[d, "(8{,}9)"] \\
-2 \arrow[r, "(7{,}9)"'] & -2 \arrow[r, "(7{,}8)"'] & -3
\end{tikzcd}
\quad = \quad
\begin{tikzcd}
-6 \arrow[r, "(7{,}9{,}8)"] \arrow[d, "(7{,}8{,}9)"'] \arrow[dr, phantom, "(11{,}15{,}13)(12{,}16{,}14)"] & -4 \arrow[d, "(7{,}9{,}8)"] \\
-2 \arrow[r, "(7{,}9{,}8)"'] & -3
\end{tikzcd}$$

Example

```
gap> g := (11,12,13,14,15,16);;  h := (12,16)(13,15);;
gap> gend12 := [ g, h ];;
gap> d12 := Group( gend12 );;
gap> SetName( d12, "d12" );
gap> gens3 := [ (7,8,9), (8,9) ];;
gap> s3 := Group( gens3 );;
gap> SetName( s3, "s3" );
gap> pr12 := GroupHomomorphismByImages( d12, s3, gend12, gens3 );;
gap> X12 := XModByCentralExtension( pr12 );;
gap> SetName( X12, "X12" );
gap> Display( X12 );

Crossed module X12 :-
: Source group d12 has generators:
  [ (11,12,13,14,15,16), (12,16)(13,15) ]
: Range group s3 has generators:
  [ (7,8,9), (8,9) ]
: Boundary homomorphism maps source generators to:
  [ (7,8,9), (8,9) ]
: Action homomorphism maps range generators to automorphisms:
  (7,8,9) --> { source gens --> [ (11,12,13,14,15,16), (11,13)(14,16) ] }
  (8,9) --> { source gens --> [ (11,16,15,14,13,12), (12,16)(13,15) ] }
  These 2 automorphisms generate the group of automorphisms.

gap> Gs3 := Groupoid( s3, [-6..-1] );;
gap> SetName( Gs3, "Gs3" );
```

```
gap> D1 := SinglePieceDoubleGroupoid( Gs3, X12 );;
gap> D1!.groupoid;
Gs3
gap> D1!.prexmod;
X12
gap> a1 := Arrow(Gs3,(7,8),-6,-5);;    a2 := Arrow(Gs3,(8,9),-5,-4);;
gap> b1 := Arrow(Gs3,(7,8,9),-1,-3);;  b2 := Arrow(Gs3,(7,9),-3,-4 );;
gap> c1 := Arrow(Gs3,(7,9),-2,-2);;    c2 := Arrow(Gs3,(7,8),-2,-3);;
gap> d1 := Arrow(Gs3,(7,9),-6,-1);;    d2 := Arrow(Gs3,(8,9),-1,-2);;
gap> e1 := Arrow(Gs3,(8,9),-5,-3);;    e2 := Arrow(Gs3,(7,9,8),-3,-2);;
gap> f1 := Arrow(Gs3,(7,8),-4,-4);;    f2 := Arrow(Gs3,(8,9),-4,-3);;
gap> ## now define four squares
gap> sq1 := SquareOfArrows( D1, g*h, a1, d1, e1, b1 );
[-6] ---- (7,8) ---> [-5]
  |                          |
(7,9)    (11,16)(12,15)(13,14)    (8,9)
  V                          V
[-1] ---- (7,8,9) ---> [-3]
gap> sq2 := SquareOfArrows( D1, g^2, a2, e1, f1, b2 );;
gap> sq3 := SquareOfArrows( D1, g, b1, d2, e2, c1 );;
gap> sq4 := SquareOfArrows( D1, h, b2, e2, f2, c2 );;
gap> ## then form two horizontal and two vertical products:
gap> sq12 := LeftRightProduct( D1, sq1, sq2 );;
gap> sq34 := LeftRightProduct( D1, sq3, sq4 );;
gap> sq13 := UpDownProduct( D1, sq1, sq3 );;
gap> sq24 := UpDownProduct( D1, sq2, sq4 );;
gap> ## combine in two ways to get a single square:
gap> sq1324 := LeftRightProduct( D1, sq13, sq24 );
[-6] ---- (7,9,8) ---> [-4]
  |                          |
(7,8,9)    (11,15,13)(12,16,14)    (7,9,8)
  V                          V
[-2] ---- (7,9,8) ---> [-3]
gap> sq1234 := UpDownProduct( D1, sq12, sq34 );;
gap> sq1324 = sq1234;
true
```

11.2 Basic double groupoids

As mentioned earlier, double groupoids were introduced in the Groupoids package, but these were *basic double groupoids*, without any pre-crossed module. The element of a square was simply its boundary. Here we introduce an operation which converts such a basic double groupoid into the more general case considered in this package.

11.2.1 EnhancedBasicDoubleGroupoid

▷ `EnhancedBasicDoubleGroupoid(bdg)` (operation)

We need to add a pre-crossed module to the definition of such a double groupoid. We choose $(G \to G)$ where G is the root group of the underlying groupoid. (This is only valid for groupoids which are the direct product with a complete graph.) The example is taken from section 7.1 of the Groupoids package, converting basic `B0` to `D0`, and we check that the same square is produced in each case.

Example

```
gap> g := (1,2,3,4);;  h := (1,3);;
gap> gend8 := [ g, h ];;
gap> d8 := Group( gend8 );;
gap> SetName( d8, "d8" );
gap> Gd8 := Groupoid( d8, [-9..-7] );;
gap> SetName( Gd8, "Gd8" );
gap> B0 := SinglePieceBasicDoubleGroupoid( Gd8 );;
gap> B0!.groupoid;
Gd8
gap> B0!.objects;
[ -9 .. -7 ]
gap> a0 := Arrow(Gd8,(),-9,-7);;           b0 := Arrow(Gd8,(2,4),-9,-8);;
gap> d0 := Arrow(Gd8,g,-9,-9);;            e0 := Arrow(Gd8,(1,3),-7,-8);;
gap> bdy0 := b0![1]^-1 * d0![1]^-1 * a0![1] * e0![1];;
gap> bsq0 := SquareOfArrows( B0, bdy0, a0, d0, e0, b0 );
[-9] ---- () ---> [-7]
  |                   |
(1,2,3,4)    (1,4,3,2)    (1,3)
  V                   V
[-9] ---- (2,4) ---> [-8]

gap> D0 := EnhancedBasicDoubleGroupoid( B0 );;
gap> D0!.prexmod;
[d8->d8]
gap> bsq0 = SquareOfArrows( D0, bdy0, a0, d0, e0, b0 );
true
```

11.3 Commutative double groupoids

A double groupoid square

$$\begin{tikzcd} u_1 \arrow[r, "a_1"] \arrow[d, "d_1"'] \arrow[dr, phantom, "1"] & u_2 \arrow[d, "e_1"] \\ v_1 \arrow[r, "b_1"'] & v_2 \end{tikzcd}$$

is *commutative* if $a_1 e_1 = d_1 b_1$, which means that its boundary is the identity. So a double groupoid which consists only of commutative squares must have a pre-crossed module with zero boundary. Commutative squares compose horizantally and vertically provided only that they have the correct common arrow.

11.3.1 DoubleGroupoidWithZeroBoundary

▷ `DoubleGroupoidWithZeroBoundary(gpd, src)` (operation)

The data for a double groupoid of commutative squares therefore consists of a groupoid and a source group. We may use the operation `PreXModWithTrivialRange` (2.3.1) to provide a pre-crossed module. We take for our example the groupoid `Gd8` and the pre-crossed module `Q16` of section 2.3. We introduce a new right arrow to construct a square which commutes.

Example

```
gap> D16 := DoubleGroupoidWithZeroBoundary( Gs3, d16 );;
gap> D16!.prexmod;
[d16->Group( [ () ] )]
gap> e16 := Arrow( Gs3, (7,9,8), -5, -3 );;
gap> sq16 := SquareOfArrows( D16, (), a1, d1, e16, b1 );
[-6] ---- (7,8) ---> [-5]
  |                    |
(7,9)     ()     (7,9,8)
  V                    V
[-1] ---- (7,8,9) ---> [-3]
gap> D16 := DoubleGroupoidWithZeroBoundary( Gs3, d16 );;
gap> D16!.prexmod;
[d16->Group( [ () ] )]
gap> e16 := Arrow( Gs3, (7,9,8), -5, -3 );;
gap> sq16 := SquareOfArrows( D16, (), a1, d1, e16, b1 );
[-6] ---- (7,8) ---> [-5]
  |                    |
(7,9)     ()     (7,9,8)
  V                    V
[-1] ---- (7,8,9) ---> [-3]
```

Chapter 12

Applications

This chapter was added in April 2018 for version 2.66 of XMod. Initially it describes crossed modules for free loop spaces. Further applications may arise in due course.

12.1 Free Loop Spaces

These functions have been used to produce examples for Ronald Brown's paper *Crossed modules, and the homotopy* 2*-type of a free loop space* [Bro18]. The relevant theorem in that paper is as follows.

THEOREM 2.1 *Let $\mathscr{M} = (\partial : M \to P)$ be a crossed module of groups and let $X = B\mathscr{M}$ be the classifying space of $\mathscr{M}$. Then the components of LX, the free loop space on X, are determined by equivalence classes of elements $a \in P$ where a, a' are equivalent if and only if there are elements $m \in M$, $p \in P$ such that $a' = p + a - \partial m - p$.*

Further the homotopy 2*-type of a component of LX given by $a \in P$ is determined by the crossed module of groups $L\mathscr{M}[a] = (\partial_a : M \to P(a))$ where:*

- *$P(a)$ is the subgroup of the cat^1-group $G = P \ltimes M$ such that $\partial m = [p,a] = -p-a+p+a$;*
- *$\partial_a(m) = (\partial m, m^{-1}m^a)$ for $m \in M$;*
- *the action of $P(a)$ on M is given by $n^{(p,m)} = n^p$ for $n \in M$, $(p,m) \in P(a)$.*

In particular $\pi_1(LX,a)$ is isomorphic to $\mathrm{cokernel}(\partial_a)$*, and $\pi_2(LX,a) \cong \pi_2(X,*)^{\bar{a}}$, the elements of $\pi_2(X,*)$ fixed under the action of $\bar{a}$, the class of a in $\pi_1(X,*)$.*

There is an exact sequence $\pi \xrightarrow{\phi} \pi \to \pi_1(LX,a) \to C_{\bar{a}}(\pi_1(X,*)) \to 1$, in which $\pi = \pi_2(X,*)$, and ϕ is the morphism $m \mapsto m^{-1}m^a$.

12.1.1 LoopClasses

▷ `LoopClasses(M)` (operation)
▷ `LoopsXMod(M, a)` (operation)
▷ `AllLoopsXMod(M)` (operation)

The operation `LoopClasses` computes the equivalence classes $[a]$ described above. These are all unions of conjugacy classes.

The operation `LoopsXMod(M,a)` calculates the crossed module $L\mathscr{M}[a]$ described in the theorem.

The operation `AllLoopsXMod(M)` returns a list of crossed modules, one for each equivalence class of elements $[a] \subseteq P$.

In the example below the automorphism crossed module `X8` has $M \cong C_2^3$ and $P = PSL(3,2)$ is the automorphism group of M. There are 6 equivalence classes which, in this case, are identical with the conjugacy classes. For each LX calculated, the `IdGroup` (2.8.1) is printed out.

Example

```
gap> SetName( k8, "k8" );
gap> Y8 := XModByAutomorphismGroup( k8 );;
gap> X8 := Image( IsomorphismPerm2DimensionalGroup( Y8 ) );;
gap> SetName( X8, "X8" );
gap> Print( "X8: ", Size( X8 ), " : ", StructureDescription( X8 ), "\n" );
X8: [ 8, 168 ] : [ "C2 x C2 x C2", "PSL(3,2)" ]
gap> classes := LoopClasses( X8 );;
gap> List( classes, c -> Length(c) );
[ 1, 21, 56, 42, 24, 24 ]
gap> LX := LoopsXMod( X8, (1,2)(5,6) );;
gap> Size2d( LX );
[ 8, 64 ]
gap> IdGroup( LX );
[ [ 8, 5 ], [ 64, 138 ] ]
gap> SetInfoLevel( InfoXMod, 1 );
gap> LX8 := AllLoopsXMod( X8 );;
#I  LoopsXMod with a = (),  IdGroup = [ [ 8, 5 ], [ 1344, 11686 ] ]
#I  LoopsXMod with a = (4,5)(6,7),  IdGroup = [ [ 8, 5 ], [ 64, 138 ] ]
#I  LoopsXMod with a = (2,3)(4,6,5,7),  IdGroup = [ [ 8, 5 ], [ 32, 6 ] ]
#I  LoopsXMod with a = (2,4,6)(3,5,7),  IdGroup = [ [ 8, 5 ], [ 24, 13 ] ]
#I  LoopsXMod with a = (1,2,4,3,6,7,5),  IdGroup = [ [ 8, 5 ], [ 56, 11 ] ]
#I  LoopsXMod with a = (1,2,4,5,7,3,6),  IdGroup = [ [ 8, 5 ], [ 56, 11 ] ]
gap> iso := IsomorphismGroups( Range( LX ), Range( LX8[2] ) );;
gap> iso = fail;
false
```

Chapter 13

Interaction with HAP

This chapter describes functions which allow functions in the package HAP to be called from XMod.

13.1 Calling HAP functions

In HAP a cat[1]-group is called a `CatOneGroup` and the traditional terms *source* and *target* are used for the `TailMap` and `HeadMap`. A `CatOneGroup` is a record `C` with fields `C!.sourceMap` and `C!.targetMap`.

13.1.1 SmallCat1Group

▷ `SmallCat1Group(n, i, j)` (operation)

This operation calls the HAP function `SmallCatOneGroup(n,i,j)` which returns a `CatOneGroup` from the HAP database. This is then converted into an XMod cat[1]-group. Note that the numbering is not the same as that used by the XMod operation `Cat1Select`. In the example `C12` is the converted form of `H12`.

Example

```
gap> H12 := SmallCatOneGroup( 12, 4, 3 );
Cat-1-group with underlying group Group( [ f1, f2, f3 ] ) .
gap> C12 := SmallCat1Group( 12, 4, 3 );
[Group( [ f1, f2, f3 ] )=>Group( [ f1, f2, <identity> of ... ] )]
```

13.1.2 CatOneGroupToXMod

▷ `CatOneGroupToXMod(C)` (operation)
▷ `Cat1GroupToHAP(C)` (operation)

These two functions convert between the two alternative implementations.

Example

```
gap> C12 := CatOneGroupToXMod( H12 );
[Group( [ f1, f2, f3 ] )=>Group( [ f1, f2, <identity> of ... ] )]
```

```
gap> C18 := Cat1Select( 18, 4, 3 );
[(C3 x C3) : C2=>Group( [ f1, <identity> of ..., f3 ] )]
gap> H18 := Cat1GroupToHAP( C18 );
Cat-1-group with underlying group (C3 x C3) : C2 .
```

13.1.3 IdCat1Group

▷ `IdCat1Group(C)` (operation)

This function calls the HAP function `IdCatOneGroup` on a cat^1-group C. This returns $[n,i,j]$ if the cat^1-group is the j-th structure on the `SmallGroup(n,i)`.

Example

```
gap> IdCatOneGroup( H18 );
[ 18, 4, 4 ]
gap> IdCat1Group( C18 );
[ 18, 4, 4 ]
```

Chapter 14

Utility functions

By a utility function we mean a GAP function which is

- needed by other functions in this package,
- not (as far as we know) provided by the standard GAP library,
- more suitable for inclusion in the main library than in this package.

Sections on *Printing Lists* and *Distinct and Common Representatives* were moved to the Utils package with version 2.56.

14.1 Mappings

The following two functions have been moved to the gpd package, but are still documented here.

14.1.1 InclusionMappingGroups

▷ `InclusionMappingGroups(G, H)` (operation)
▷ `MappingToOne(G, H)` (operation)

This set of utilities concerns mappings. The map `incd8` is the inclusion of `d8` in `d16` used in Section 3.4. `MappingToOne(G,H)` maps the whole of G to the identity element in H.

Example

```
gap> Print( incd8, "\n" );
[ (11,13,15,17)(12,14,16,18), (11,18)(12,17)(13,16)(14,15) ] ->
[ (11,13,15,17)(12,14,16,18), (11,18)(12,17)(13,16)(14,15) ]
gap> imd8 := Image( incd8 );;
gap> MappingToOne( c4, imd8 );
[ (11,13,15,17)(12,14,16,18) ] -> [ () ]
```

14.1.2 InnerAutomorphismsByNormalSubgroup

▷ InnerAutomorphismsByNormalSubgroup(*G*, *N*) (operation)

Inner automorphisms of a group G by the elements of a normal subgroup N are calculated, often with G = N.

Example

```
gap> autd8 := AutomorphismGroup( d8 );;
gap> innd8 := InnerAutomorphismsByNormalSubgroup( d8, d8 );;
gap> GeneratorsOfGroup( innd8 );
[ ^(1,2,3,4), ^(1,3) ]
```

14.1.3 IsGroupOfAutomorphisms

▷ IsGroupOfAutomorphisms(*A*) (property)

Tests whether the elements of a group are automorphisms.

Example

```
gap> IsGroupOfAutomorphisms( innd8 );
true
```

14.2 Abelian Modules

14.2.1 AbelianModuleObject

▷ AbelianModuleObject(*grp*, *act*) (operation)
▷ IsAbelianModule(*obj*) (property)
▷ AbelianModuleGroup(*obj*) (attribute)
▷ AbelianModuleAction(*obj*) (attribute)

An abelian module is an abelian group together with a group action. These are used by the crossed module constructor XModByAbelianModule (2.1.7).

The resulting Xabmod is isomorphic to the output from XModByAutomorphismGroup(k4);.

Example

```
gap> x := (6,7)(8,9);;  y := (6,8)(7,9);;  z := (6,9)(7,8);;
gap> k4a := Group( x, y );;  SetName( k4a, "k4a" );
gap> gens3a := [ (1,2), (2,3) ];;
gap> s3a := Group( gens3a );;  SetName( s3a, "s3a" );
gap> alpha := GroupHomomorphismByImages( k4a, k4a, [x,y], [y,x] );;
gap> beta := GroupHomomorphismByImages( k4a, k4a, [x,y], [x,z] );;
gap> auta := Group( alpha, beta );;
gap> acta := GroupHomomorphismByImages( s3a, auta, gens3a, [alpha,beta] );;
gap> abmod := AbelianModuleObject( k4a, acta );;
gap> Xabmod := XModByAbelianModule( abmod );
```

```
[k4a->s3a]
gap> Display( Xabmod );

Crossed module [k4a->s3a] :-
: Source group k4a has generators:
  [ (6,7)(8,9), (6,8)(7,9) ]
: Range group s3a has generators:
  [ (1,2), (2,3) ]
: Boundary homomorphism maps source generators to:
  [ (), () ]
: Action homomorphism maps range generators to automorphisms:
  (1,2) --> { source gens --> [ (6,8)(7,9), (6,7)(8,9) ] }
  (2,3) --> { source gens --> [ (6,7)(8,9), (6,9)(7,8) ] }
  These 2 automorphisms generate the group of automorphisms.
```

Chapter 15

Development history

This chapter, which contains details of the major changes to the package as it develops, was first created in April 2002. Details of the changes from XMod 1 to XMod 2.001 are far from complete. Starting with version 2.009 the file `CHANGES` lists the minor changes as well as the more fundamental ones.

The inspiration for this package was the need, in the mid-1990's, to calculate induced crossed modules (see [BW95], [BW96], [BW03]). GAP was chosen over other computational group theory systems because the code was freely available, and it was possible to modify the Tietze transformation code so as to record the images of the original generators of a presentation as words in the simplified presentation. (These modifications are now a standard part of the Tietze transformation package in GAP.)

15.1 Changes from version to version

15.1.1 Version 1 for GAP 3

The first version of XMod became an accepted package for GAP 3.4.3 in December 1996.

15.1.2 Version 2

Conversion of XMod 1 from GAP 3.4.3 to the new GAP syntax began soon after GAP 4 was released, and had a lengthy gestation. The new GAP syntax encouraged a re-naming of many of the function names. An early decision was to introduce generic categories `2dDomain` for (pre-)crossed modules and (pre-)cat1-groups, and `2dMapping` for the various types of morphism. In 2.009 `3dDomain` was used for crossed squares and cat2-groups, and `3dMapping` for their morphisms. A generic name for derivations and sections is also required, and `Up2dMapping` is currently used.

15.1.3 Version 2.001 for GAP 4

This was the first version of XMod for GAP 4, completed in April 2002 in time for the release of GAP 4.3. Functions for actors and induced crossed modules were not included, nor many of the functions for derivations and sections, for example `InnerDerivation`.

15.1.4 Induced crossed modules

During May 2002 converted the code for induced crossed modules. (Induced cat1-groups may be converted one day.)

15.1.5 Versions 2.002 – 2.006

Version 2.004 of April 14th 2004 added the `Cat1Select` (2.7.1) functionality of version 1 to the `Cat1Group` (2.4.1) function.

A significant addition in Version 2.005 was the conversion of the actor crossed module functions from the `3.4.4` version. This included `AutomorphismPermGroup` (6.1.1) for a crossed module; `WhiteheadXMod` (6.1.2); `NorrieXMod` (6.1.2); `LueXMod` (6.1.2); `ActorXMod` (6.1.2); `CentreXMod` (4.1.7) of a crossed module; `InnerMorphism` (6.1.3); and `InnerActorXMod` (6.1.3).

15.1.6 Versions 2.007 – 2.010

These versions contain changes made between September 2004 and October 2007.

- Added basic functions for crossed squares, considered as `3dObjects` with crossed pairings, and their morphisms. Groups with two normal subgroups, and the actor of a crossed module, provide standard examples of crossed squares. (Cat2-groups are not yet implemented.)
- Converted the documentation to the format of the GAPDoc package.
- Improved `AutomorphismPermGroup` (6.1.1) for crossed modules, and introduced a special method for conjugation crossed modules.
- Substantial revisons made to `XModByCentralExtension` (2.1.5); `NorrieXMod` (6.1.2); `LueXMod` (6.1.2); `ActorXMod` (6.1.2); and `InducedXModByCopower` (7.2.1).
- Version 2.010, of October 2007, was timed to coincide with the release of GAP 4.4.10, and included a change of filenames; and correct file protection codes.

15.2 Versions for GAP [4.5 .. 4.12]

Version 2.19, released on 9th June 2012, included the following changes:

- The file `makedocrel.g` was copied, with appropriate changes, from GAPDoc, and now provides the correct way to update the documentation.
- The first functions for crossed modules of groupoids were introduced.
- A GNU General Public License declaration was added.

15.2.1 AllCat1s

Version 2.21 contained major changes to the `Cat1Select` (2.7.1) operation: the list `CAT1_LIST` of cat1-structures in the data file `cat1data.g` was changed from permutation groups to pc-groups, with the generators of subgroups; images of the tail map; and images of the head map being given as `ExtRepOfObj` of words in the generators.

The `AllCat1s` function was reintroduced from the GAP3 version, and renamed as the operation `AllCat1DataGroupsBasic`.

In version 2.25 the data in `cat1data.g` was extended from groups of size up to 48 to groups of size up to 70. In particular, the 267 groups of size 64 give rise to a total of 1275 cat1-groups. The authors are indebted to Van Luyen Le in Galway for pointing out a number of errors in the version of this list distributed with version 2.24 of this package.

15.2.2 Versions 2.43 - 2.56

Version 2.43, released on 11th November 2015, included the following changes:

- The material on isoclinism in Chapter 4 was added.
- The package webpage has moved to `https://github.com/cdwensley`.
- A GitHub repository was started at: `https://github.com/gap-packages/xmod`.
- The section on *Distinct and Common Representatives* was moved to the Utils package.

15.2.3 Version 2.61

Major changes in names took place, with `2dDomain`, `2dGroup`, `2dMapping`, etc. becoming `2DimensionalDomain`, `2DimensionalGroup`, `2DimensionalMapping`, etc., and similarly for 3-dimensional versions. Also `HigherDimensionalDomain` and related categories, domains, properties, attributes and operations were introduced. At the same time, functions for cat2-groups were introduced by Alper Odabas.

15.2.4 Versions 2.63 - 2.74

- Added an implementation of crossed modules of groupoids.
- Lots more work on crossed squares and cat2-groups.
- Added an implementation of group groupoids.

15.2.5 Versions 2.75 - 2.85

- Added conversion functions between XMod and Hap and a new chapter in the manual about these functions.
- Added functions for quasi-isomorphisms.

15.2.6 Versions 2.86 - 2.91

- Added attributes `Size2d` for 2d-objects and `Size3d` for 3d-objects since lists are inappropriate values for the standard function `Size`.
- Added `PreXModWithTrivialRange` and started work on double groupoids.

15.3 What needs doing next?

- Speed up the calculation of Whitehead groups.
- Add more functions for `3dObjects` and implement `cat2-groups`.
- Improve interaction with the package groupoids implementing the group groupoid version of a crossed module, and adding more functions for crossed modules of groupoids.
- Add interaction with IdRel (and possibly XRes and natp) .
- Need `InverseGeneralMapping` for morphisms and more features for `FpXMods`, `PcXMods`, etc.
- Implement actions of a crossed module.
- Implement `FreeXMods` and an operation `Isomorphism2dDomains`.
- Allow the construction of a group of morphisms of crossed modules.
- Complete the conversion from Version 1 of the calculation of sections using `EndoClasses`.
- More crossed square constructions:
  - If M,N are ordinary P-modules and A is an arbitrary abelian group on which P acts trivially, then there is a crossed square with sides
  $$0 : A \to N, \quad 0 : A \to M, \quad 0 : M \to P, \quad 0 : N \to P.$$
  - For a group L, the automorphism crossed module Act $L = (\iota : L \to \mathrm{Aut}\ L)$ splits to form the square with $(\iota_1 : L \to \mathrm{Inn}\ L)$ on two sides, and $(\iota_2 : \mathrm{Inn}\ L \to \mathrm{Aut}\ L)$ on the other two sides, where ι_1 maps $l \in L$ to the inner automorphism $\beta_l : L \to L$, $l' \mapsto l^{-1}l'l$, and ι_2 is the inclusion of Inn L in Aut L. The actions are standard, and the crossed pairing is
  $$\boxtimes \ : \ \mathrm{Inn}\ L \times \mathrm{Inn}\ L \to L, \quad (\beta_l, \beta_{l'}) \ \mapsto \ [l, l'] \ .$$
- Improve the interaction with the HAP package.
- Implement cat1-groups with objects.
- Lots more work on double groupoids.

References

[Alp97] M. Alp. *GAP, crossed modules, cat1-groups: applications of computational group theory*. PhD thesis, University of Wales, Bangor, 1997. 2

[AOWar] Z. Arvasi, A. Odabas, and C. D. Wensley. Computing 3-dimensional groups : Crossed squares and cat2-groups. *J. Symbolic Computation*, (to appear). 79

[AW00] M. Alp and C. D. Wensley. Enumeration of cat1-groups of low order. *Int. J. Algebra and Computation*, 10:407–424, 2000. 5, 53

[AW10] M. Alp and C. D. Wensley. Automorphisms and homotopies of groupoids and crossed modules. *Applied Categorical Structures*, 18:473–495, 2010. 92

[BH78] R. Brown and P. J. Higgins. On the connection between the second relative homotopy group and some related spaces. *Proc. London Math. Soc.*, 36:193–212, 1978. 5, 62, 63

[BHS11] R. Brown, P. J. Higgins, and R. Sivera. *Nonabelian algrebraic topology*, volume 15 of *Tracts in Mathematics*. European Mathematical Society, 2011. 6

[BL87] R. Brown and J.-L. Loday. Van kampen theorems for diagram of spaces. *Topology*, 26:311–335, 1987. 66, 78

[Bro82] R. Brown. Higher-dimensional group theory. In R. Brown and T. L. Thickstun, editors, *Low-dimensional topology*, volume 48 of *London Math. Soc. Lecture Note Series*, pages 215–238. Cambridge University Press, 1982. 6

[Bro18] R. Brown. Crossed modules, and the homotopy 2-type of a free loop space. *arXiv:1003.5617 [math.AT]*, pages 1–11, 2018. 101

[BW95] R. Brown and C. D. Wensley. On finite induced crossed modules, and the homotopy 2-type of mapping cones. *Theory and Applications of Categories*, 1:54–71, 1995. 5, 62, 63, 108

[BW96] R. Brown and C. D. Wensley. Computing crossed modules induced by an inclusion of a normal subgroup, with applications to homotopy 2-types. *Theory and Applications of Categories*, 2:3–16, 1996. 5, 62, 63, 108

[BW03] R. Brown and C. D. Wensley. Computation and homotopical applications of induced crossed modules. *J. Symbolic Computation*, 35:59–72, 2003. 108

[EL14] G. J. Ellis and V. L. Le. Homotopy 2-types of low order. *Experimental Math.*, 23:383–389, 2014. 38

[Ell84] G. Ellis. *Crossed modules and their higher dimensional analogues*. PhD thesis, University of Wales, Bangor, 1984. 5

[ES87] G. Ellis and R. Steiner. Higher dimensional crossed modules and the homotopy groups of (n+1)-ads. *J. Pure and Appl. Algebra*, 46:117–136, 1987. 66, 78, 91

[GH17] S. Gutsche and M. Horn. *AutoDoc - Generate documentation from GAP source code (Version 2017.09.15)*, 2017. GAP package, `https://github.com/gap-packages/AutoDoc`. 2

[Gil90] N. D. Gilbert. Derivations, automorphisms and crossed modules. *Comm. in Algebra*, 18:2703–2734, 1990. 5

[Hor17] M. Horn. *GitHubPagesForGAP - Template for easily using GitHub Pages within GAP packages (Version 0.2)*, 2017. GAP package, `https://gap-system.github.io/GitHubPagesForGAP/`. 2

[IOU16] E. Ilgaz, A. Odabas, and E. O. Uslu. Isoclinism of crossed modules. *J. Symb. Comput.*, pages 1–17, 2016. https://doi.org/10.1016/j.jsc.2015.08.006. 2, 5, 40

[JNO90] R. James, M. F. Newman, and E. A. O'Brien. The groups of order 128. *J. Algebra*, 129:136–158, 1990. 40, 48

[LN17] F. Lübeck and M. Neunhöffer. *GAPDoc (version 1.6)*. RWTH Aachen, 2017. GAP package, `https://www.math.rwth-aachen.de/~Frank.Luebeck/GAPDoc/index.html`. 2

[Lod82] J.-L. Loday. Spaces with finitely many non-trivial homotopy groups. *J. App. Algebra*, 24:179–202, 1982. 5, 17

[MLW50] S. Mac Lane and J. H. C. Whitehead. On the 3-type of a complex. *Proc. Nat. Acad. Sci. U.S.A.*, 37:41–48, 1950. 38

[Moo01] E. J. Moore. *Graphs of Groups: Word Computations and Free Crossed Resolutions*. PhD thesis, University of Wales, Bangor, 2001. 6

[Nor87] K. J. Norrie. *Crossed modules and analogues of group theorems*. PhD thesis, King's College, University of London, 1987. 5, 43, 45, 57

[Nor90] K. J. Norrie. Actions and automorphisms of crossed modules. *Bull. Soc. Math. France*, 118:129–146, 1990. 5, 57

[Whi48] J. H. C. Whitehead. On operators in relative homotopy groups. *Ann. of Math.*, 49:610–640, 1948. 5, 51

[Whi49] J. H. C. Whitehead. Combinatorial homotopy II. *Bull. Amer. Math. Soc.*, 55:453–496, 1949. 5

Index