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

© 1996-2024, Chris Wensley et al.

The XMod package is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

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

1 Introduction ........................................................................................................... 5

2 2d-groups : crossed modules and cat\(^1\)-groups ........................................... 8
   2.1 Constructions for crossed modules ................................................................. 8
   2.2 Properties of crossed modules ..................................................................... 14
   2.3 Pre-crossed modules ..................................................................................... 16
   2.4 Cat\(^1\)-groups and pre-cat\(^1\)-groups ......................................................... 17
   2.5 Properties of cat\(^1\)-groups and pre-cat\(^1\)-groups ................................... 22
   2.6 Enumerating cat\(^1\)-groups with a given source ...................................... 24
   2.7 Selection of a small cat\(^1\)-group ................................................................. 26
   2.8 More functions for crossed modules and cat\(^1\)-groups ............................ 28
   2.9 The group groupoid associated to a cat\(^1\)-group ....................................... 29

3 2d-mappings ......................................................................................................... 32
   3.1 Morphisms of 2-dimensional groups ........................................................... 32
   3.2 Morphisms of pre-crossed modules ............................................................. 32
   3.3 Morphisms of pre-cat\(^1\)-groups .................................................................. 35
   3.4 Operations on morphisms ............................................................................ 37
   3.5 Quasi-isomorphisms ..................................................................................... 38

4 Isoclimism of groups and crossed modules ..................................................... 40
   4.1 More operations for crossed modules ......................................................... 40
   4.2 Isoclimism for groups ................................................................................ 47
   4.3 Isoclimism for crossed modules .................................................................. 49

5 Whitehead group of a crossed module ............................................................ 51
   5.1 Derivations and Sections ............................................................................ 51
   5.2 Whitehead Groups and Monoids ................................................................ 54

6 Actors of 2d-groups ............................................................................................ 57
   6.1 Actor of a crossed module ........................................................................... 57

7 Induced constructions .......................................................................................... 61
   7.1 Coproducts of crossed modules .................................................................. 61
   7.2 Induced crossed modules .......................................................................... 62
   7.3 Induced cat\(^1\)-groups .................................................................................. 64
Chapter 1

Introduction

The \texttt{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

\begin{verbatim}
gap> LoadPackage( "xmod" );
\end{verbatim}

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.

\texttt{XMod} was originally implemented in 1997 using the \texttt{GAP} 3 language. In April 2002 the first and third parts were converted to \texttt{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 \texttt{GAP} 4.4. Many of the function names have been changed during the conversion, for example \texttt{ConjugationXMod} has become \texttt{XModByNormalSubgroup} (2.1.2). For a list of name changes see the file \texttt{names.pdf} in the \texttt{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 Act(ξ) of a crossed module ξ we require representations for the Whitehead group of regular derivations of ξ and for the group of automorphisms of ξ. The construction also provides an inner morphism from ξ to Act(ξ) whose kernel is the centre of ξ.

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 between 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](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 \( \partial \), 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 \( \alpha : R \to \text{Aut}(S) \) such that, for all \( s, s_1, s_2 \in S \) and \( r \in R \),

\[
\text{XMod 1 : } \partial(s^r) = r^{-1}(\partial s)r = (\partial s)^r, \quad \text{XMod 2 : } s_1^{s_2} = s_2^{-1}s_1s_2 = s_1^{s_2}.
\]

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

\( \text{XMod}(\text{args}) \) (function)

\( \text{XModByBoundaryAndAction(bdy, act)} \) (operation)

The global function \( \text{XMod} \) 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.

\[
\text{gap> c5 := Group( (5,6,7,8,9) );;}
\text{gap> SetName( c5, "c5" );}
\text{gap> id5 := IdentityMapping( c5 );;}
\text{gap> ac5 := AutomorphismGroup( c5 );;}
\text{gap> act := MappingToOne( c5, ac5 );;}
\text{gap> XMod( id5, act ) = XModByBoundaryAndAction( id5, act );}
\text{true}
\]
2.1.2 XModByNormalSubgroup

▷ XModByNormalSubgroup\( (G, N) \) \hspace{1cm} \text{(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\( (\text{bdy}) \) \hspace{1cm} \text{(operation)}

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

Example

\begin{verbatim}
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.
\end{verbatim}

2.1.4 XModByAutomorphismGroup

▷ XModByAutomorphismGroup\( (\text{grp}) \) \hspace{1cm} \text{(attribute)}
▷ XModByInnerAutomorphismGroup\( (\text{grp}) \) \hspace{1cm} \text{(attribute)}
▷ XModByGroupOfAutomorphisms\( (G, A) \) \hspace{1cm} \text{(operation)}

An automorphism crossed module has as range a subgroup \( R \) of the automorphism group \( \text{Aut}(S) \) 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

\begin{verbatim}
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.
\end{verbatim}

2.1.5 \textbf{XModByCentralExtension}

▷ \textbf{XModByCentralExtension(bdy)} (operation)

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

Example

\begin{verbatim}
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.
\end{verbatim}
2.1.6 XModByPullback

▷ XModByPullback(xmod, hom) (operation)

Let $X_0 = (\mu : M \to P)$ be a crossed module. If $\nu : N \to P$ is a group homomorphism with the same range as $X_0$, 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 $(m,n^n) := (m^{\nu n}, n^\nu)$, so that $X_1 = (\lambda : L \to N)$ is the pullback crossed module determined by $X_0$ and $\nu$. There is also a morphism of crossed modules $(\kappa, \nu) : X_1 \to X_2$.

The example forms a pullback of the crossed module $X_{12}$ of the previous subsection.

```gap
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 $X_1 \times X_2$ 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 $X_1 \times X_2$.

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]]
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)**
- **Range(X0)**
- **Boundary(X0)**
- **XModAction(X0)**

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 ]
Boundary(X12);
[ (1,2,3,4,5,6), (2,6)(3,5) ] -> [ (7,8,9), (8,9) ]
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)**

This function returns the element s' given by XModAction(X0).
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, \( [\text{Size(Source(X0))}, \text{Size(Range(X0))}] \).

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

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

2.1.12 Name (for crossed modules)

▷ Name(X0)
▷ 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 \( [\text{IdGroup(Source(X0))}, \text{IdGroup(Range(X0))}] \).

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.

```
gap> IdGroup( X5 );
[ 5, 1, 4, 1 ]
gap> ext := ExternalSetXMod( X5 );
<xsset:[ (), (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), (5,8,6,9,7), (5,9,8,7,6) ] ]
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" );
```
2.2 Properties of crossed modules

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

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

### 2.2.1 IsXMod

\>
\>
\>
\>
\>
\>

A structure which has \textit{IsPerm2DimensionalGroup} is a precrossed module or a pre-cat\textsuperscript{1}-group (see section 2.4) whose source and range are both permutation groups. The properties \textit{IsPc2DimensionalGroup}, \textit{IsFp2DimensionalGroup} are defined similarly. In the example below we see that \textit{X5} has \textit{IsPreXMod}, \textit{IsXMod} and \textit{IsPerm2DimensionalGroup}.

There are also properties corresponding to the various construction methods listed in section 2.1: \textit{IsTrivialAction2DimensionalGroup}; \textit{IsNormalSubgroup2DimensionalGroup}; \textit{IsCentralExtension2DimensionalGroup}; \textit{IsAutomorphismGroup2DimensionalGroup}; \textit{IsAbelianModule2DimensionalGroup}.

\begin{verbatim}
gap> [ IsTrivial( X5 ), IsNonTrivial( X5 ), IsFinite( X5 ) ];
[ false, true, true ]
gap> kpoX5 := KnownPropertiesOfObject(X5);

\end{verbatim}

### 2.2.2 SubXMod

\>
\>

\begin{verbatim}
  s -> s in kpoX5 );
true
\end{verbatim}
With the standard crossed module constructors listed above as building blocks, sub-crossed modules, normal sub-crossed modules $N \triangleleft \mathcal{X}$, and also quotients $\mathcal{X}/N$ may be constructed. A sub-crossed module $\mathcal{X} = (\partial : N \to M)$ is normal in $\mathcal{X} = (\partial : S \to R)$ if

- $N, M$ are normal subgroups of $S, R$ respectively,
- $\delta$ is the restriction of $\partial$,
- $n' \in N$ 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 $X_4$ found in the following example are $[\text{id}, \text{id}]$, $[\text{k4}, \text{k4}]$, $[\text{k4}, \text{a4}]$, $[\text{a4}, \text{a4}]$ and $X_4$ itself.

Example

```gap
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

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

Example

```gap
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 );
```
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
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)
- XModByPeifferQuotient(prexmod)

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

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

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

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

```gap
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)(14,16) ]
: 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 $t, h : G \rightarrow G$ having a common image $R$ and satisfying certain axioms. We find it computationally convenient to define a cat$^1$-group $\mathcal{G} = (e ; t, h : G \rightarrow R)$ as having source group $G$, range group $R$, and three homomorphisms: two surjections $t, h : G \rightarrow R$ and an embedding $e : R \rightarrow G$ satisfying:

$\text{Cat 1: } t \circ e = h \circ e = \text{id}_R, \quad \text{Cat 2: } [\ker t, \ker h] = \{1_G\}.$

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)
- PreCat1Group(args)
- PreCat1GroupByTailHeadEmbedding(t, h, e)
The global functions \texttt{Cat1Group} and \texttt{PreCat1Group} can be called in various ways.

- as \texttt{Cat1Group}(t,h,e); when \( t,h,e \) are three homomorphisms, which is equivalent to \texttt{PreCat1GroupByTailHeadEmbedding}(t,h,e);
- as \texttt{Cat1Group}(t,h); when \( t,h \) are two endomorphisms, which is equivalent to \texttt{PreCat1GroupWithIdentityEmbedding}(t,h);
- as \texttt{Cat1Group}(t); when \( t = h \) is an endomorphism, which is equivalent to \texttt{PreCat1GroupWithIdentityEmbedding}(t,t);
- as \texttt{Cat1Group}(t,e); when \( t = h \) and \( e \) are homomorphisms, which is equivalent to \texttt{PreCat1GroupByTailHeadEmbedding}(t,t,e);
- as \texttt{Cat1Group}(i,j,k); when \( i,j,k \) are integers, which is equivalent to \texttt{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 \texttt{cat1}-groups)

\begin{itemize}
  \item \texttt{Source(C)} (attribute)
  \item \texttt{Range(C)} (attribute)
  \item \texttt{TailMap(C)} (attribute)
  \item \texttt{HeadMap(C)} (attribute)
  \item \texttt{RangeEmbedding(C)} (attribute)
  \item \texttt{KernelEmbedding(C)} (attribute)
  \item \texttt{Boundary(C)} (attribute)
  \item \texttt{Name(C)} (attribute)
  \item \texttt{Size2d(C)} (attribute)
\end{itemize}

These are the attributes of a \texttt{cat1}-group \( C \) in this implementation.

The maps \( t,h \) are often referred to as the \texttt{source} and \texttt{target}, but we choose to call them the \texttt{tail} and \texttt{head} of \( C \), because \texttt{source} is the GAP term for the domain of a function. The \texttt{RangeEmbedding} is the embedding of \( R \) in \( G \), the \texttt{KernelEmbedding} is the inclusion of the kernel of \( t \) in \( G \), and the \texttt{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

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:
```

2.4.4 TransposeCat1Group

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

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
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

\[ (5,6,7), (6,7,8) \]

If \(C = (e; t, h : G \to R)\) is a pre-cat\(^1\)-group, its Peiffer subgroup is \(P = \left\{ \ker t, \ker h \right\}\) and the associated cat\(^1\)-group \(C2\) 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
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 );
XC := rec2.prexmod;;
StructureDescription( XC );
[ "A4", "C2" ]
XC := XModByPeifferQuotient( XC );
StructureDescription( XC2 );
[ "C3", "C2" ]
CXC2 := Cat1GroupOfXMod( XC2 );
StructureDescription( CXC2 );
[ "S3", "C2" ]
IsomorphismCat1Groups( C2, CXC2 );
[[Group( [ f1, f2 ] ) => c2] => [(..|X..) => c2]]
```

### 2.4.6 SubCat1Group

- SubCat1Group(\(C_1, S_1\))
- SubPreCat1Group(\(C_1, S_1\))

\(S_1\) 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 \(C_1\).

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

### 2.4.7 DirectProduct (for cat1-groups)

- DirectProduct(\(C_1, C_2\))

The direct product \(\mathcal{C}_1 \times \mathcal{C}_2\) 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 \(\mathcal{C}_1\) and \(\mathcal{C}_2\).

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

```gap
C418 := DirectProduct( C4, C18 );
[(a4a4xg18)=>(a4d x s3a)]
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] ] )
```
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
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 );
```
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
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 );
gap> IsPreCat1GroupWithIdentityEmbedding( PC5 );
false
gap> IPC5 := IsomorphicPreCat1GroupWithIdentityEmbedding( PC5 );
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 $\mathcal{X} = (\partial : S \to R)$ the group $G$ is defined as the semidirect product $G = R \rtimes S$ using the action from $\mathcal{X}$, with multiplication rule

$$(r_1, s_1)(r_2, s_2) = (r_1r_2, s_1^\partial s_2).$$

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 $\mathcal{G} = (e, t, h : G \to R)$, 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 module $X = (\partial : S \to R)$ 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
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 \texttt{AllCat1GroupsWithImageIterator}(G, R) iterates through all the \texttt{cat}^1\texttt{-groups with source }G\texttt{ and range }R. The attribute \texttt{AllCat1GroupsWithImageNumber}(G) runs through this iterator to count the number \(n_R\) of these \texttt{cat}^1\texttt{-groups. The operation }\texttt{AllCat1GroupsWithImage}(G)\texttt{ returns a list containing these }n_R\texttt{ \texttt{cat}^1\texttt{-groups. Since these lists can get very long, this operation should only be used for simple cases. The operation }\texttt{AllCat1GroupsWithImageUpToIsomorphism}(G)\texttt{ returns representatives of the isomorphism classes of these \texttt{cat}^1\texttt{-groups.}

Example

\begin{verbatim}
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) ] )] ]
\end{verbatim}

2.6.2 \texttt{AllCat1GroupsMatrix}

\begin{verbatim}
▷ \texttt{AllCat1GroupsMatrix}(G)
\end{verbatim}

The operation \texttt{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 \texttt{cat}^1\texttt{-group; }M_{ij} = 1\) if they only form a \texttt{pre-cat}^1\texttt{-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 \texttt{cat}^1\texttt{-groups, all of which are symmetric }\(t = h\), and a further 9 \texttt{pre-cat}^1\texttt{-groups, 5 of which are symmetric. (A \texttt{cat}^1\texttt{-group and its transpose are not counted twice.) This operation is intended to be used to illustrate how \texttt{cat}^1\texttt{-groups are formed, and should only be used with groups of low order.}}

The attribute \texttt{AllCat1GroupsNumber}(G) returns the number \(n\) of these \texttt{cat}^1\texttt{-groups.

Example

\begin{verbatim}
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
\end{verbatim}
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.

```gap
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 cat\(^1\)-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.

```gap
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

The function Cat1Select returns the cat$^1$-group numbered num whose source is the group $G :=$
SmallGroup(size,gpnum). When $|G| \leq 60$ 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 $c_3$ of $s_3$.

```
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 );
[\langle C3 x C3 \rangle : 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 \([\text{Group}(\{(4,5,6)\}) \rightarrow \text{Group}(\{(1,2,3),(2,3)(5,6)\})]\)

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:
\[
(1,2,3) \rightarrow \{\text{source gens} \rightarrow (4,5,6)\}
\]
\[
(2,3)(5,6) \rightarrow \{\text{source gens} \rightarrow (4,6,5)\}
\]
These 2 automorphisms generate the group of automorphisms.
: associated cat1-group is \([\text{Group}(\{(4,5,6),(1,2,3),(2,3)(5,6)\}) \Rightarrow \text{Group}(\{(1,2,3),(2,3)(5,6)\})]\)

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)

▷ \text{IdGroup(2DimensionalGroup)} \hspace{1cm} \text{(operation)}

▷ \text{StructureDescription(2DimensionalGroup)} \hspace{1cm} \text{(operation)}

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

\text{Example}

\text{gap> IdGroup(X8);}
\[
[[24,6],[12,4]]
\]

\text{gap> StructureDescription(C8);} 
\[
"(S3 \times D24) : C2", "D12"
\]

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

2.8.2 IsSubXMod

▷ \text{IsSubXMod(X0, S0)} \hspace{1cm} \text{(operation)}

▷ \text{IsSubPreXMod(X0, S0)} \hspace{1cm} \text{(operation)}

▷ \text{IsSubCat1Group(G0, R0)} \hspace{1cm} \text{(operation)}

▷ \text{IsSubPreCat1Group(G0, R0)} \hspace{1cm} \text{(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.

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

### 2.9 The group groupoid associated to a cat\(^1\)-group

A groupoid \( G \) 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 \( \mathcal{C} = (e, r : h : G \to R) \), we form the groupoid \( \mathcal{G} \) having the elements of \( R \) as objects and the elements of \( G \) as arrows. The arrow \( g \) has tail \( tG \) and head \( hG \). \( \mathcal{G} \) has one connected component for each coset of \( tG \) in \( R \).

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

\[
(g_1 : r_1 \to r_2) * (g_2 : r_2 \to r_3) = (g_1(e_{r_2}^{-1})g_2 : r_1 \to r_3).
\]

#### 2.9.1 GroupGroupoid

```
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)->() ] ),
2.9.2 GroupGroupoidElement

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 $G$ 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 $g_1 \ast g_2$ and $e_1 \ast e_2$ give very different results. (Warning: at present iterators for object groups and homsets do not work.)
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)

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

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 \((\sigma, \rho)\), 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:

\[
\partial_2 \circ \sigma = \rho \circ \partial_1, \quad \sigma(s') = (\sigma s)\rho r.
\]

Here \( \sigma \) is the SourceHom (3.1.1) and \( \rho \) is the RangeHom (3.1.1) of the morphism. When \( \mathcal{X}_1 = \mathcal{X}_2 \) and \( \sigma, \rho \) are automorphisms then \((\sigma, \rho)\) is an automorphism of \( \mathcal{X}_1 \). The group of automorphisms is denoted by \( \text{Aut}(\mathcal{X}_1) \).
3.2.2 IsInjective (for pre-xmod morphisms)

- IsInjective(map)
- IsSurjective(map)
- IsSingleValued(map)
- IsTotal(map)
- IsBijective(map)
- IsEndo2DimensionalMapping(map)

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)
- XModMorphismByGroupHomomorphisms(X1, X2, sigma, rho)
- PreXModMorphism(args)
- PreXModMorphismByGroupHomomorphisms(P1, P2, sigma, rho)
- InclusionMorphism2DimensionalDomains(X1, S1)
- InnerAutomorphismXMod(X1, r)
- IdentityMapping(X1)

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 $X_5$ constructed in the previous chapter.

```gap
gap> sigma5 := GroupHomomorphismByImages( c5, c5, [ (5,6,7,8,9) ]
                                       [ (5,8,7,6) ] );
gap> rho5 := IdentityMapping( Range( X1 ) );
gap> mor5 := XModMorphism( X5, X5, sigma5, rho5 );
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
: Range 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 );
```
3.2.4 IsomorphismPerm2DimensionalGroup (for pre-xmod morphisms)

▷ IsomorphismPerm2DimensionalGroup(obj)  
(attribute)

▷ IsomorphismPc2DimensionalGroup(obj)  
(attribute)

▷ IsomorphismByIsomorphisms(D, list)  
(operation)

When $\mathcal{D}$ 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 $(\sigma, \rho) : \mathcal{D} \to \mathcal{D}'$ where $\mathcal{D}'$ has source $S'$ and range $R'$. Be sure to test IsBijective for the two functions $\sigma, \rho$ 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
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*f3, f2, f3 ])]
gap> iso := IsomorphismPerm2DimensionalGroup( XAq8 );;
[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) ] )]
```

3.2.5 MorphismOfPullback (for a crossed module by pullback)

▷ MorphismOfPullback(xmod)  
(attribute)

Let $\mathcal{X}_1 = (\lambda : L \to N)$ be the pullback crossed module obtained from a crossed module $\mathcal{X}_0 = (\mu : M \to P)$ and a group homomorphism $\nu : N \to P$. Then the associated crossed module morphism is
(κ, ν) : 𝒳₁ → 𝒳₀ where κ is the projection from L to M.

### 3.3 Morphisms of pre-cat1-groups

A morphism of pre-cat1-groups from 𝒢₁ = (e₁; t₁, h₁ : G₁ → R₁) to 𝒢₂ = (e₂; t₂, h₂ : G₂ → R₂) is a pair (γ, ρ) where γ : G₁ → G₂ and ρ : R₁ → R₂ are homomorphisms satisfying

\[ h₂ \circ γ = ρ \circ h₁, \quad t₂ \circ γ = ρ \circ t₁, \quad e₂ \circ ρ = γ \circ e₁. \]

#### 3.3.1 IsCat1GroupMorphism

- IsCat1GroupMorphism(map)  
- IsPreCat1GroupMorphism(map)
- Cat1GroupMorphism(args)
- Cat1GroupMorphismByGroupHomomorphisms(C1, C2, gamma, rho)
- PreCat1GroupMorphism(args)
- PreCat1GroupMorphismByGroupHomomorphisms(P1, P2, gamma, rho)
- InclusionMorphism2DimensionalDomains(C1, S1)
- InnerAutomorphismCat1(C1, r)
- IdentityMapping(C1)

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.

```gap
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)
- XModMorphismOfCat1GroupMorphism(IsCat1GroupMorphism)
If \((\sigma, \rho) : \mathcal{X}_1 \to \mathcal{X}_2\) and \(G_1, G_2\) are the cat^1-groups associated to \(\mathcal{X}_1, \mathcal{X}_2\), then the associated morphism of cat^1-groups is \((\gamma, \rho)\) where \(\gamma(r_1, s_1) = (\rho r_1, \sigma s_1)\).

Similarly, given a morphism \((\gamma, \rho) : G_1 \to G_2\) of cat^1-groups, the associated morphism of crossed modules is \((\sigma, \rho) : \mathcal{X}_1 \to \mathcal{X}_2\) where \(\sigma s_1 = \gamma(1, s_1)\).

```gap
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)`

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`.

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

### 3.3.4 SmallerDegreePermutationRepresentation2DimensionalGroup (for perm 2d-groups)

- `SmallerDegreePermutationRepresentation2DimensionalGroup(Perm2DimensionalGroup)`

```
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
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)`

Composition of morphisms (written \((<\text{map1}> \ast <\text{map2}>)\) 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
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> SC8 := Sub2DimensionalGroup( C8, H8, c6 );
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)`

▷ `Kernel2DimensionalMapping(map)`

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

\[
\mathcal{X} = \mathcal{X}_1 \leftrightarrow \mathcal{X}_2 \leftrightarrow \mathcal{X}_3 \leftrightarrow \cdots \leftrightarrow \mathcal{X}_\ell = \mathcal{X}'
\]

of length \( \ell - 1 \). Here \( \mathcal{X}_i \leftrightarrow \mathcal{X}_j \) means that either \( \mathcal{X}_i \rightarrow \mathcal{X}_j \) or \( \mathcal{X}_j \rightarrow \mathcal{X}_i \). When \( \mathcal{X}, \mathcal{X}' \) are quasi-isomorphic we write \( \mathcal{X} \simeq \mathcal{X}' \). 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 \( \mathcal{X} \) 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 \( \mathcal{C} \), as described by Ellis and Le in [EL14], is as follows. The quotient process consists of finding all normal sub-crossed modules \( N \) of the crossed module \( \mathcal{X} \) associated to \( \mathcal{C} \); constructing the quotient crossed module morphisms \( \nu : \mathcal{X} \rightarrow \mathcal{X}/N \); and converting these \( \nu \) to morphisms from \( \mathcal{C} \).

The sub-crossed module process consists of finding all sub-crossed modules \( \mathcal{I} \) of \( \mathcal{X} \) such that the inclusion \( i : \mathcal{I} \rightarrow \mathcal{X} \) is a quasi-isomorphism; then converting \( i \) to a morphism to \( \mathcal{C} \).

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 \( \mathcal{C}_1 \simeq \mathcal{C}_2 \) without either having a quasi-isomorphism reduction. In this case it is necessary to find a suitable \( \mathcal{C}_3 \) with reductions \( \mathcal{C}_3 \rightarrow \mathcal{C}_1 \) and \( \mathcal{C}_3 \rightarrow \mathcal{C}_2 \). 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)  

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)  

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)  

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 \( N \) leading to a reduction, and that there are three sub-crossed modules \( S \) 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 \( \mathcal{X}_2 = (\partial_2 : S_2 \rightarrow R_2) \) is a normal sub-precrossed module of \( \mathcal{X}_1 = (\partial_1 : S_1 \rightarrow R_1) \), then the quotient precrossed module is \( (\partial : S_2/S_1 \rightarrow R_2/R_1) \) with the induced boundary and action maps. Quotienting a precrossed module by its 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\rightarrow\text{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];
[\(\text{Group( [ f2*f4^2, f3*f4 ] )}\rightarrow\text{Group( [ f3, f4, f5 ] )}]```

40
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 \([r,q] = r^{-1}q^{-1}rq = (q^{-1})r(q^{-1})q = r^{-1}r^q\), and satisfy identities

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

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 \(\mathcal{X} = (\partial : S \to R)\) is a pre-crossed module, the first crossed module axiom requires \(\partial (r,s) = [r,\partial s]\). When \(\alpha\) is the action of \(\mathcal{X}\), the Displacement function may be used to calculate \(\langle r,s \rangle\). Displacements satisfy the following identities, where \(s,t \in S\), \(p,q,r \in R\):

\[
\langle r,s \rangle^p = \langle r^p,s^p \rangle, \quad \langle qr,s \rangle = \langle q,s \rangle^r \langle r,s \rangle, \quad \langle r,st \rangle = \langle r,t \rangle \langle s,t \rangle^r, \quad \langle r,s \rangle^{-1} = \langle r^{-1},s^{-1} \rangle.
\]

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

```gap
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> 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) ])
```

4.1.4 CommutatorSubXMod

▷ CommutatorSubXMod(X, X1, X2)

▷ CrossActionSubgroup(X, X1, X2)

When \( X_1 = (N \to Q) \) and \( X_2 = (M \to P) \) are two normal subcrossed modules of \( X = (\partial : S \to R) \), the displacements \( \langle p, n \rangle \) and \( \langle q, m \rangle \) all map by \( \partial \) into \([Q, P]\). These displacements form a normal subgroup of \( S \), called the CrossActionSubgroup. The CommutatorSubXMod \([X_1, X_2]\) has this subgroup as source and \([P, Q]\) as range, and is normal in \( X \).

Example

```gap
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 \( \mathcal{X} \) is the normal subcrossed module \( \mathcal{X}, \mathcal{X} = (\partial' : \text{Disp}(\mathcal{X}) \to [R,R]) \) where \( \partial' \) is the restriction of \( \partial \) (see page 66 of Norrie’s thesis [Nor87]).

\[
\text{Example}
\]

```gap
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 \( \mathcal{X} = (\partial : S \to R) \) is the subgroup \( \text{Fix}(\mathcal{X},T,Q) \) of elements \( t \in T \leq S \) individually fixed under the action of \( Q \leq R \).

The StabilizerSubgroupXMod(X,T,Q) for \( \mathcal{X} \) is the subgroup \( \text{Stab}(\mathcal{X},T,Q) \) of \( Q \leq R \) whose elements act trivially on the whole of \( T \leq S \) (see page 19 of Norrie’s thesis [Nor87]).

\[
\text{Example}
\]

```gap
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 \( Z(\mathcal{X}) \) of \( \mathcal{X} = (\partial : S \to R) \) has as source the fixed point subgroup \( \text{Fix}(\mathcal{X},S,R) \). The range is the intersection of the centre \( Z(R) \) with the stabilizer subgroup.

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

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

\[
\text{Example}
\]

```gap
gap> ZXn4 := CentreXMod( Xn4 );
```
```gap
gap> IdGroup( ZXn4 );
[ [ 2, 1 ], [ 4, 2 ] ]
gap> CDXn4 := Centralizer( Xn4, DXn4 );
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)  

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 $\mathcal{X}$ is the crossed square $(\mathcal{X} \Rightarrow \mathcal{X}/Z(\mathcal{X}))$ (see section 8.2).

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

### 4.1.9 IsAbelian2DimensionalGroup

▷ IsAbelian2DimensionalGroup(XO)  

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.

```
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) ];
```
4.1.10 LowerCentralSeriesOfXMod

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

Let \( \mathcal{X} \) be a subcrossed module of \( \mathcal{Y} \). A series of length \( n \) from \( \mathcal{X} \) to \( \mathcal{Y} \) has the form

\[
\mathcal{X} = \mathcal{X}_0 \triangleright \mathcal{X}_1 \triangleright \cdots \triangleright \mathcal{X}_i \triangleright \cdots \triangleright \mathcal{X}_n = \mathcal{Y} \quad (1 \leq i \leq n).
\]

The quotients \( \mathcal{F}_i = \mathcal{X}_i / \mathcal{X}_{i-1} \) are the factors of the series.
A factor \( \mathcal{F}_i \) is central if \( \mathcal{X}_{i-1} \triangleright \mathcal{X}_i \) and \( \mathcal{F}_i \) is a subcrossed module of the centre of \( \mathcal{X}_i / \mathcal{X}_{i-1} \).
A series is soluble if it has a series all of whose factors are abelian.
\( \mathcal{X} \) is nilpotent if it has a series all of whose factors are central factors of \( \mathcal{X} \).

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

\[
\mathcal{X} = \Gamma_1(\mathcal{X}) \triangleright \Gamma_2(\mathcal{X}) \triangleright \cdots \quad \text{where} \quad \Gamma_i(\mathcal{X}) = [\Gamma_{i-1}(\mathcal{X}), \mathcal{X}] .
\]

If \( \mathcal{X} \) is nilpotent, then its lower central series is its most rapidly descending central series.

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

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 IsomorphismX Mods

▷ IsomorphismX Mods(X1, X2) (operation)

The function IsomorphismX Mods computes an isomorphism \( \mu : \mathcal{X}_1 \to \mathcal{X}_2 \), 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

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 isomor-
phism 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, ($C_6 \rightarrow S_3$);
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].

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

Example
4.2 Isoclinism for groups

4.2.1 Isoclinism (for groups)

\[ \text{Isoclinism}(G, H) \] (operation)
\[ \text{AreIsoclinicDomains}(G, H) \] (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 \((\eta, \xi)\) where \( \eta : G/Z(G) \to H/Z(H) \) and \( \xi : [G, G] \to [H, H] \) such that \( c_G \ast \xi = (\eta \times \eta) \ast c_H \). Isoclinism is an equivalence relation, and all abelian groups are isoclinic to the trivial group.

Example

\[ \text{gap} \] 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]]

4.2.2 IsStemDomain (for groups)

\[ \text{IsStemDomain}(G) \] (property)
\[ \text{IsoclinicStemDomain}(G) \] (attribute)
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.

```gap
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 $\log_p |G/Z(G)(G \cap [G,G])|$ 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].

```gap
gap> IsoclinicMiddleLength( G );
1
gap> IsoclinicRank( G );
4
```
4.3 Isoclinism for crossed modules

4.3.1 Isoclinism (for crossed modules)

▷ Isoclinism(\(X_0, Y_0\)) (operation)
▷ AreIsoclinicDomains(\(X_0, Y_0\)) (operation)

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

Example

\begin{verbatim}
gap> C8 := Cat1Group( 16, 8, 2 );;
gap> X8 := XMod(C8); IdGroup( X8 );
[ [ 8, 1 ], [ 2, 1 ] ]
gap> C9 := Cat1Group( 32, 9, 2 );
[(C8 x C2) : C2 => Group( [ f2, f2 ] )]
gap> X9 := XMod( C9 ); IdGroup( X9 );
[ [ 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 ... ] ) -> Group( [ f2, f2 ] )]],
[Group( [ f3 ] ) -> Group( <identity> of ... )] => [Group( [ f3 ] ) -> Group( <identity> of ... )]]
\end{verbatim}

4.3.2 IsStemDomain (for crossed modules of groups)

▷ IsStemDomain(\(X_0\)) (property)
▷ IsoclinicStemDomain(\(X_0\)) (attribute)

A crossed module \(\mathcal{X}\) is a stem crossed module if \(Z(\mathcal{X}) \leq [\mathcal{X}, \mathcal{X}]\). 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

\begin{verbatim}
gap> IsStemDomain(X8);
true
gap> IsStemDomain(X9);
false
\end{verbatim}
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.

```gap
gap> IsoclinicMiddleLength(X0);
[ 1, 0 ]
gap> IsoclinicRank(X0);
[ 3, 1 ]
```
Chapter 5

Whitehead group of a crossed module

5.1 Derivations and Sections

The Whitehead monoid $\text{Der}(\mathcal{X})$ of $\mathcal{X}$ 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 $\ast$ (on the right) satisfying:

\[
\begin{align*}
\text{Der 1} : \chi(qr) &= (\chi q)^\ast (\chi r), \\
\text{Der 2} : (\chi_1 \ast \chi_2)(r) &= (\chi_2 r)(\chi_1 r)(\chi_2 \partial \chi_1 r).
\end{align*}
\]

The zero map is the identity for this composition. Invertible elements in the monoid are called regular.

The Whitehead group of $\mathcal{X}$ is the group of regular derivations in $\text{Der}(\mathcal{X})$. In the next chapter the actor of $\mathcal{X}$ is defined as a crossed module whose source and range are permutation representations of the Whitehead group and the automorphism group of $\mathcal{X}$.

The construction for cat1-groups equivalent to the derivation of a crossed module is the section.

The monoid of sections of $C = (e; t, h : G \to R)$ is the set of group homomorphisms $\xi : R \to G$, with Whitehead multiplication $\ast$ (on the right) satisfying:

\[
\begin{align*}
\text{Sect 1} : t \circ \xi &= \text{id}_R, \\
\text{Sect 2} : (\xi_1 \ast \xi_2)(r) &= (\xi_2 h \xi_1 r)^{-1}(\xi_2 h \xi_1 r) = (\xi_2 h \xi_1 r)(eh \xi_1 r)^{-1}(\xi_1 r).
\end{align*}
\]

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

If $e$ denotes the inclusion of $S = \ker t$ in $G$ then $\partial = he : S \to R$ and

\[\xi r = (er)(e\chi r)\]

which equals $(r, \chi r) \in R \ltimes S$,

determines a section $\xi$ of $C$ in terms of the corresponding derivation $\chi$ of $\mathcal{X}$, and conversely.

5.1.1 DerivationByImages

- DerivationByImages($X_0$, $ims$) (operation)
- IsDerivation(map) (property)
- IsUp2DimensionalMapping(chi) (property)
- UpGeneratorImages(chi) (attribute)
- UpImagePositions(chi) (attribute)
- DerivationImage(chi, $r$) (operation)
A derivation $\chi$ is stored like a group homomorphisms by specifying the images of the generating set $\text{StrongGeneratorsStabChain}(\text{StabChain}(R))$ of the range $R$. This set of images is stored as the attribute $\text{UpGeneratorImages}$ of $\chi$. The function $\text{IsDerivation}$ is automatically called to check that this procedure is well-defined.

Images of the remaining elements may be obtained using axiom $\text{Der 1}$. $\text{UpImagePositions}(\chi)$ is the list of the images under $\chi$ of $\text{Elements}(R)$ and $\text{DerivationImage}(\chi, r)$ returns $\chi r$.

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 $\chi_1$ maps $c3$ to the identity and the other 3 elements to $\langle 1, 2, 3 \rangle \langle 4, 6, 5 \rangle$.

```
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

$\text{PrincipalDerivation}(X0, s)$ (operation)

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

```
Example

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

▷ SectionByHomomorphism\((C, \text{hom})\) (operation)
▷ IsSection\((\text{hom})\) (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\(<\text{C}>\) 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

\begin{verbatim}
gap> hom2 := GroupHomomorphismByImages( s3, g18, [ (4,5,6), (2,3)(5,6) ], [ (1,3,2)(4,5,6), (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) ] )
\end{verbatim}

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

\begin{verbatim}
gap> IdentityDerivation( X3 );
DerivationByImages( s3, c3, [ (4,5,6), (2,3)(5,6) ], [ (), () ] )
\end{verbatim}
5.1.5 WhiteheadProduct

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

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 $\chi$ is the smallest power of $\chi$, using this product, equal to the IdentityDerivation (5.1.4).

Example

```gap
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 $\text{Der}(\mathcal{X})$ of $\mathcal{X}$ is the monoid of all derivations from $R$ to $S$. Monoids of derivations have representation $\text{IsMonoidOfUp2Dimensional1MappingsObj}$. 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
gap> all3 := AllDerivations( X3 );
monoid of derivations with images list:
[ (), () ]
[ (), (1,3,2)(4,5,6) ]
```
\[
\begin{array}{l}
[ (1,2,3)(4,5,6), (1,3,2)(4,5,6) ] \\
[ (1,2,3)(4,6,5), (1,3,2)(4,5,6) ] \\
[ (1,3,2)(4,5,6), (1,2,3)(4,6,5) ] \\
[ (1,2,3)(4,6,5), (1,3,2)(4,5,6) ] \\
[ (1,2,3)(4,6,5), (1,2,3)(4,6,5) ] \\
\text{gap> DerivationClass( all3 );} \\
\text{"all"} \\
\text{gap> Perform( ImagesTable( all3 ), Display );} \\
[ 1, 1, 1, 1, 1, 1 ] \\
[ 1, 1, 1, 3, 3, 3 ] \\
[ 1, 1, 2, 2, 2 ] \\
[ 1, 3, 2, 1, 3, 2 ] \\
[ 1, 3, 2, 3, 2, 1 ] \\
[ 1, 3, 2, 2, 1, 3 ] \\
[ 1, 2, 3, 1, 2, 3 ] \\
[ 1, 2, 3, 3, 2, 1 ] \\
[ 1, 2, 3, 2, 3, 1 ] \\
\text{gap> wmt3 := WhiteheadMonoidTable( X3 );}; \\
\text{gap> Perform( wmt3, Display );} \\
[ 1, 2, 3, 4, 5, 6, 7, 8, 9 ] \\
[ 2, 3, 1, 5, 6, 4, 8, 9, 7 ] \\
[ 3, 1, 2, 6, 4, 5, 9, 7, 8 ] \\
[ 4, 6, 5, 1, 3, 2, 7, 9, 8 ] \\
[ 5, 4, 6, 2, 1, 3, 8, 7, 9 ] \\
[ 6, 5, 4, 3, 2, 1, 9, 8, 7 ] \\
[ 7, 7, 7, 7, 7, 7, 7, 7, 7 ] \\
[ 8, 8, 8, 8, 8, 8, 8, 8, 8 ] \\
[ 9, 9, 9, 9, 9, 9, 9, 9, 9 ] \\
\text{gap> wtm3 := WhiteheadTransformationMonoid( X3 );}; \\
<\text{transformation monoid of degree 9 with 3 generators}> \\
\text{gap> GeneratorsOfMonoid( wtm3 );} \\
[ \text{Transformation( [ 2, 3, 1, 5, 6, 4, 8, 9, 7 ] )}, \text{Transformation( [ 4, 6, 5, 1, 3, 2, 7, 9, 8 ] )}, \text{Transformation( [ 7, 7, 7, 7, 7, 7, 7, 7, 7 ] )} ]
\end{array}
\]

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 \(X_3\) just six are regular. The associated group is isomorphic to the symmetric group \(S_3\).
**5.2.3 PrincipalDerivations**

▷ PrincipalDerivations(X0) (attribute)

The principal derivations form a subgroup of the Whitehead group.

**Example**

```gap
gap> PDX3 := PrincipalDerivations( 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) ])
```

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

6.1.1 AutomorphismPermGroup

\begin{verbatim}
\end{verbatim}

The automorphisms \((\sigma, \rho)\) of \( \mathcal{X} \) form a group \( \text{Aut}(\mathcal{X}) \) of crossed module isomorphisms. The function AutomorphismPermGroup finds a set of GeneratingAutomorphisms for \( \text{Aut}(\mathcal{X}) \), and then constructs a permutation representation of this group, which is used as the range of the actor crossed module of \( \mathcal{X} \). The individual automorphisms can be constructed from the permutation group using the function PermAutomorphismAsXModMorphism. The example below uses the crossed module \( \mathcal{X}_3 = [c_3 -> s_3] \) constructed in section 5.1.

Example

\begin{verbatim}
\end{verbatim}

\begin{verbatim}
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) ]
\end{verbatim}
6.1.2 WhiteheadXMod

\[\text{WhiteheadXMod}(xmod)\]
\[\text{LueXMod}(xmod)\]
\[\text{NorrieXMod}(xmod)\]
\[\text{ActorXMod}(xmod)\]

An automorphism \((\sigma, \rho)\) 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)).

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

Example

\begin{verbatim}
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:
\end{verbatim}
The main methods for these operations are written for permutation crossed modules. For other crossed modules an isomorphism to a permutation crossed module is found first, and then the main method is applied to the image. In the example the crossed module \( \text{XAq8} \) is the automorphism crossed module of the quaternion group.

\begin{verbatim}
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" ]
\end{verbatim}
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 \( \mathcal{H}(\mathcal{X}) \) and \( \mathcal{N}(\mathcal{X}) \) form the inner morphism of \( \mathcal{X} \), mapping source elements to principal derivations and range elements to inner automorphisms. The image of \( \mathcal{X} \) under this morphism is the inner actor of \( \mathcal{X} \), while the kernel is the centre of \( \mathcal{X} \). In the example which follows, the inner morphism of \( X_3=(c_3\to s_3) \), 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.

```gap
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 $X_1 = (\partial_1 : S_1 \to R)$ and $X_2 = (\partial_2 : S_2 \to R)$. See below for the case of three or more crossed modules.

The source $S_2$ of $X_2$ acts on $S_1$ via $\partial_2$ and the action of $X_1$, so we can form a precrossed module $(\partial': S_1 \times S_2 \to R)$ where $\partial'(s_1, s_2) = (\partial_1 s_1)(\partial_2 s_2)$. The action of this precrossed module is the diagonal action $(s_1, s_2)\cdot r = (s_1\cdot r, s_2)$. Factoring out by the Peiffer subgroup, we obtain the coproduct crossed module $X_1 \circ X_2$.

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.

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

61
Given a list of more than two crossed modules with a common range \( R \), then an iterated coproduct is formed:
\[
\bigcirc [\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_n] = \mathcal{X}_1 \circ (\mathcal{X}_2 \circ (\ldots (\mathcal{X}_{n-1} \circ \mathcal{X}_n)\ldots)).
\]
The embeddings field of the CoproductInfo of the resulting crossed module \( \mathcal{Y} \) contains the \( n \) morphisms \( \varepsilon_i : \mathcal{X}_i \to \mathcal{Y} \) (\( 1 \leq i \leq n \)).

Example

```gap
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

A morphism of crossed modules \((\sigma, \rho) : \mathcal{X}_1 \to \mathcal{X}_2\) factors uniquely through an induced crossed module \(\rho' \circ S_1 \to \mathcal{R}_2\). Similarly, a morphism of cat1-groups factors through an induced cat1-group. Calculation of induced crossed modules of \(\mathcal{X}\) also provides an algebraic means of determining the homotopy 2-type of homotopy pushouts of the classifying space of \(\mathcal{X}\). 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 $\text{XMod}$ and $\text{Cat1}$.

Data for the cases of algebraic interest is provided by a crossed module $\mathcal{X} = (\partial : S \to R)$ and a homomorphism $\iota : R \to Q$. The output from the calculation is a crossed module $\iota_* \mathcal{X} = (\delta : \iota_* S \to Q)$ together with a morphism of crossed modules $\mathcal{Y} \to \iota_* \mathcal{X}$. When $\iota$ 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 $\iota$ is neither a surjection nor an inclusion, $\iota$ 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 $\mathcal{X}$ the normal inclusion crossed module of $a_4$ in $s_4$, and for $\iota$ the surjection from $s_4$ to $s_3$ with kernel $k_4$. The induced crossed module is isomorphic to $X_3 = [c_3 \to s_3]$.

```
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 $\mathcal{X}$ the automorphism crossed module $\text{XAq8}$ of $\text{CoproductXMod}(7.1.1)$, and for $\iota$ an inclusion of $s_4b$ in $s_5$. The resulting source group is $\text{SL}(2,5)$.

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

---

Example
gap> s5 := Group( (1,2,3,4,5), (4,5) );;
gap> SetName( s5, "s5" );
gap> inc45 := InclusionMappingGroups( s5, s4 );;
gap> iota45 := iso4 * inc45;;

indXAq8 := InducedXMod( XAq8, iota45 );
i*(XAq8)

Size2d( indXAq8 );
[ 120, 120 ]

StructureDescription( indXAq8 );
[ "SL(2,5)", "S5" ]

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

gap> s3c := Subgroup( s4, [ (2,3), (3,4) ] );;
gap> SetName( s3c, "s3c" );

gap> indXs3c := InducedXMod( s4, s3c, s3c );
i*(s3c->s3c)

StructureDescription( indXs3c );
[ "GL(2,3)", "S4" ]

7.2.2 AllInducedX Mods

AllInducedX Mods(Q)

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

all := AllInducedX Mods( 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 $\mathcal{S}$ 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:

$$
\begin{array}{cccccc}
\tilde{\partial}_1 : S_{[2]} & \rightarrow & S_{\{1\}} \\
\tilde{\partial}_2 : S_{[2]} & \rightarrow & S_{\{2\}} \\
\hat{\partial}_1 : S_{\{1\}} & \rightarrow & S_\emptyset \\
\hat{\partial}_2 : S_{\{2\}} & \rightarrow & S_\emptyset
\end{array}
$$

(again we often write $\kappa = \tilde{\partial}_1$, $\lambda = \tilde{\partial}_2$, $\mu = \hat{\partial}_2$ and $\nu = \hat{\partial}_1$);

- 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 $\hat{\partial}_1$ and actions of $S_{\{2\}}$ on $S_{\{1\}}$ and $S_{[2]}$ via $\hat{\partial}_2$;

- a function $\boxtimes : S_{\{1\}} \times S_{\{2\}} \rightarrow S_{[2]}$.

Here is a picture of the situation:

$$
\begin{array}{ccc}
S_{[2]} & \xrightarrow{\tilde{\partial}_1} & S_{\{2\}} \\
\downarrow \tilde{\partial}_2 & & \downarrow \hat{\partial}_1 \\
S_{\{1\}} & \xrightarrow{\hat{\partial}_2} & S_\emptyset
\end{array}
\quad
\begin{array}{ccc}
L & \xrightarrow{\kappa} & M \\
\downarrow \lambda & & \downarrow \mu \\
N & \xrightarrow{\nu} & P
\end{array}
$$
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 \( \kappa, \lambda \) preserve the action of \( P \).
- Each of the upper, left-hand, right-hand and lower sides of the square,
  \[ \mathcal{J}_1 = (\kappa : L \to M), \quad \mathcal{J}_2 = (\lambda : L \to N), \quad \mathcal{J}_2 = (\mu : M \to P), \quad \mathcal{J}_1 = (\nu : N \to P), \]
  and the diagonal
  \[ \mathcal{J}_{12} = (\partial_{12} := \mu \circ \kappa = \nu \circ \lambda : L \to P) \]
  are crossed modules (with actions via \( P \)).
  These will be called the \textit{up, left, right, down} and \textit{diagonal} crossed modules of \( \mathcal{J} \).
- \( \boxtimes \) is a \textit{crossed pairing}:
  \begin{align*}
  &- (n_1 n_2 \boxtimes m) = (n_1 \boxtimes m)^{m_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).
  \end{align*}
- \( \kappa (n \boxtimes m) = (m^{-1})^n m \) and \( \lambda (n \boxtimes m) = n^{-1} m^n \).
- \( (n \boxtimes \kappa l) = (l^{-1})^n l \) 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
\[ n_1 (m^n) = n_1 \mu (m^n) = n_1 n^{-1} (\mu m)^n = ((n_1 n^{-1})^n)^n. \]

(A \textit{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 \textit{Computing homotopy types using crossed \( n \)-cubes of groups} into right actions – but this definition is not yet completely understood!)

A \textit{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 \( S_B \) on each \( S_A \); and hence actions of \( S_B \) on \( S_A \) via \( \partial_B \) 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 cat²-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.

```
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\" );
exp([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( () )]
```

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.
8.2.3 \textbf{CrossedSquareByNormalSubgroups}
\begin{itemize}
\item \textbf{CrossedSquareByNormalSubgroups}(L, M, N, P) \hspace{2cm} \text{(operation)}
\item \textbf{CrossedPairingByCommutators}(N, M, L) \hspace{2cm} \text{(operation)}
\end{itemize}

If \( L, M, N \) are normal subgroups of a group \( P \), and \([M,N] \leq L \leq M \cap N \), 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

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

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

\begin{example}
\begin{verbatim}
gap> Size3d( PXS );
[ 2, 2, 2, 1 ]
\end{verbatim}
\end{example}

8.2.4 \textbf{CrossedSquareByNormalSubXMod}
\begin{itemize}
\item \textbf{CrossedSquareByNormalSubXMod}(X0, X1) \hspace{2cm} \text{(operation)}
\item \textbf{CrossedPairingBySingleXModAction}(X0, X1) \hspace{2cm} \text{(operation)}
\end{itemize}

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

\[ \bowtie : R_1 \times S_0 \to S_1, \quad (r_1,s_0) \mapsto (s_0^{-1})r_1s_0. \]

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

\begin{example}
\begin{verbatim}
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)
\end{verbatim}
\end{example}
8.2.5 ActorCrossedSquare

\begin{itemize}
\item \texttt{ActorCrossedSquare(X0)}
\item \texttt{CrossedPairingByDerivations(X0)}
\end{itemize}

The actor $\mathcal{A}(\mathcal{X}_0)$ of a crossed module $\mathcal{X}_0$ has been described in Chapter 5 (see \texttt{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 \texttt{ActorCrossedSquare(X0)}.

\begin{example}
\begin{verbatim}
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)
\end{verbatim}
\end{example}

8.2.6 CrossedSquareByAutomorphismGroup

\begin{itemize}
\item \texttt{CrossedSquareByAutomorphismGroup(G)}
\item \texttt{CrossedPairingByConjugators(G)}
\end{itemize}

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

**Example**

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

\( \triangleright \) CrossedSquareByPullback(\( X1, X2 \) ) (operation)

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

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

The second example below uses the central extension crossed module \( X_{12} = (D_{12} \rightarrow S_3) \) which was constructed in subsection (XModByCentralExtension (2.1.5)), with pullback group \( D_{12} \times C_2 \).

**Example**

```gap

gap> dn := Down2DimensionalGroup( XSconj );;
gap> rt := Right2DimensionalGroup( XSconj );;
gap> XSP := CrossedSquareByPullback( dn, rt );
[ (d10b x d20 d10a) -\rightarrow d10a ]
| |
[ d10b -\rightarrow 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> xpi12 := CrossedPairing( XS12 );;

gap> ImageElmCrossedPairing( xpi12, [ (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 $\mathcal{X} = (\partial : S \to R)$ let $Q$ be the image of $\partial$. Then $\partial = \partial' \circ \iota$ where $\partial' : S \to Q$ and $\iota$ is the inclusion of $Q$ in $R$. The diagonal of the square is then the initial $\mathcal{X}$, and the crossed pairing is given by commutators of preimages.

A particular case is when $S$ is an $R$-module $A$ and $\partial$ is the zero map.

\begin{equation}
\begin{array}{c}
S \\
\downarrow \partial' \\
Q \\
\downarrow \iota \\
R
\end{array}
\begin{array}{c}
A \\
\downarrow 0 \\
0 \\
\downarrow 1 \\
R
\end{array}
\end{equation}

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
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)

The transpose of a crossed square $S$ is the crossed square $\tilde{S}$ obtained by interchanging $M$ with $N$, $\kappa$ with $\lambda$, and $\nu$ with $\mu$. The crossed pairing is given by

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

Example

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

8.2.11 CentralQuotient (for crossed modules)

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

• the left crossed module is $\mathcal{X}$;
• the right crossed module is the quotient $\mathcal{X}/Z(\mathcal{X})$ (see CentreXMod (4.1.7));
• the up and down homomorphisms are the natural homomorphisms onto the quotient groups;
• the crossed pairing $\triangleright : (R \times F) \rightarrow S$, where $F = \text{Fix}(\mathcal{X}, S, R)$, is the displacement element $\triangleright(r,Fs) = \langle r,s \rangle = (s^{-1})^r s$ (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
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(S0) (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 $\{M,N\} \rightarrow L$ (see the next subsection). The diagonal crossed module $(L \rightarrow P)$ is an additional attribute.

```gap
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 );;
```
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.

```
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
```

The second example shows how to construct a crossed pairing.

```
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)
```

According to the text, the code snippet demonstrates how to define and use crossed pairings in GAP, a computational system for discrete algebra. The examples illustrate the implementation of crossed pairings using a mapping function and the ImageElmCrossedPairing function. The properties of 3D groups are also discussed, emphasizing the concept of symmetry in these structures.
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 $\text{mor}$ between two pre-crossed squares $\mathcal{S}_1$ and $\mathcal{S}_2$ consists of four crossed module morphisms $\text{Up2DimensionalMorphism}(\text{mor})$, mapping the $\text{Up2DimensionalGroup}$ of $\mathcal{S}_1$ to that of $\mathcal{S}_2$, $\text{Left2DimensionalMorphism}(\text{mor})$, $\text{Right2DimensionalMorphism}(\text{mor})$ and $\text{Down2DimensionalMorphism}(\text{mor})$. These four morphisms are required to commute with the four boundary maps and to preserve the rest of the structure. The current version of $\text{IsCrossedSquareMorphism}$ does not perform all the required checks.

Example

```gap
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
```

Known properties of object `autoconj`:
- CanEasyCompareElements
- CanEasySortElements
- 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 \( \mathcal{C} = (C_2, C_1, C_0) \) comprises four groups (one for each of the subsets of \([2]\)) and 15 homomorphisms, as shown in the following diagram:

\[
\begin{array}{c}
\begin{tikzpicture}
\node[align=center] (A) at (0,0) {\( C_2 \)};
\node[align=center] (B) at (3,0) {\( C_2 \)};
\node[align=center] (C) at (0,-3) {\( C_1 \)};
\node[align=center] (D) at (3,-3) {\( C_0 \)};
\node[align=center] (E) at (1.5,-1.5) {\( \mathcal{C} \)};
\draw[->] (A) -- (B);
\draw[->] (B) -- (C);
\draw[->] (C) -- (D);
\draw[->] (D) -- (A);
\draw[->] (A) -- (E);
\draw[->] (B) -- (E);
\draw[->] (C) -- (E);
\draw[->] (D) -- (E);
\node at (0,-1.5) {\( \text{other homomorphisms} \)};
\end{tikzpicture}
\end{array}
\]

The following axioms are satisfied by these homomorphisms:

- the four sides of the square (up, left, right, down) are cat\(^1\)-groups, denoted \( \mathcal{C}_1^1, \mathcal{C}_2^1, \mathcal{C}_1^2, \mathcal{C}_2^2 \);
- \( \bar{t}_1 \circ \bar{h}_1 = \bar{h}_2 \circ \bar{i}_1, \bar{t}_2 \circ \bar{h}_1 = \bar{h}_1 \circ \bar{i}_2, \bar{e}_1 \circ \bar{i}_2 = \bar{t}_2 \circ \bar{e}_1, \bar{e}_2 \circ \bar{i}_1 = \bar{t}_1 \circ \bar{e}_2, \bar{e}_1 \circ \bar{h}_2 = \bar{h}_2 \circ \bar{e}_1, \bar{e}_2 \circ \bar{h}_1 = \bar{h}_1 \circ \bar{e}_2; \)
- \( \bar{t}_1 \circ \bar{t}_2 = \bar{t}_2 \circ \bar{i}_1 = \bar{t}_2, \bar{h}_1 \circ \bar{h}_2 = \bar{h}_2 \circ \bar{h}_1 = \bar{h}_2, \bar{e}_1 \circ \bar{e}_2 = \bar{e}_2 \circ \bar{e}_1 = \bar{e}_2, \) making the diagonal a pre-cat\(^1\)-group \( (e_2 \circ i_2, h_2 \circ h_2 : C_2 \to C_0) \).

It follows from these identities that \( (\bar{t}_1, \bar{i}_1), (\bar{h}_1, \bar{h}_1) \) and \( (\bar{e}_1, \bar{e}_1) \) 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 \( \mathcal{C} \) 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 \( 1 \leq i \leq 2 \),

- \( (t_i \circ e_i) r = r, (h_i \circ e_i) r = r, \forall r \in R_2 \setminus \{ i \}, \quad [\ker t_i, \ker h_i] = 1, \)
- \( (e_1 \circ t_1) \circ (e_2 \circ t_2) = (e_2 \circ t_2) \circ (e_1 \circ t_1), \quad (e_1 \circ h_1) \circ (e_2 \circ h_2) = (e_2 \circ h_2) \circ (e_1 \circ h_1), \)
- \( (e_1 \circ t_1) \circ (e_2 \circ h_2) = (e_2 \circ h_2) \circ (e_1 \circ t_1), \quad (e_2 \circ t_2) \circ (e_1 \circ h_1) = (e_1 \circ h_1) \circ (e_2 \circ t_2). \)

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

\( (t \circ e) \) and \( (h \circ e) \) are the identity mapping on \( \mathcal{C}_2 \), \quad [\ker t, \ker h] = \{ 1_{e_1} \}, \)
where \( \ker t = (\ker \dot t, \ker \ddot t) \), 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 cat1-groups - or with a single argument which is a crossed square. The operation *PreCat2GroupByPreCat1Groups* has five arguments - the up, left, right, down and diagonal cat1-groups.

The two cat2-groups \( C_{2a} \), \( C_{2b} \) 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) ] )]
```
8.4.2 DirectProduct

▷ DirectProduct(C2a, C2b) (operation)

The direct product $G_1 \times G_2$ has as its four up, left, right and down cat1-groups the direct products of those in $G_1$ and $G_2$. The embeddings and projections are constructed automatically, and placed in the DirectProductInfo attribute, together with the two objects $G_1$ and $G_2$.

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) ],
    ( 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) ],
    ( 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)

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

**Example**

```gap
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

A (pre-)cat$^2$-group morphism $\mu : C = (G,R,Q,P) \to C' = (G',R',Q',P')$ 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 $(\xi,\pi)$ 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 $C$.

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

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

**Example**

```gap
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 );
gap> upmor := Cat1GroupMorphism( up, up, gamma, rho );;
gap> ltmor := Cat1GroupMorphism( left, left, gamma, xi );;
```
XMod

8.4.6 Cat2GroupOfCrossedSquare

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

These functions provide for conversion between crossed squares and cat²-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> mor2 := Cat2GroupMorphismByCat1GroupMorphisms( C2a, C2a, upmor, ltmor );;
gap> mor1 = mor2;
true

8.4.7 Subdiagonal2DimensionalGroup

▷ Subdiagonal2DimensionalGroup(obj) (attribute)

The diagonal of a crossed square is always a crossed module, but the diagonal of a cat²-group need only be a pre-cat¹-group. There is, however, a sub-cat¹-group of this diagonal which, in the case of a cat²-group constructed from a crossed square, is \( (P \ltimes L \Rightarrow P) \). (The name of this operation is very provisional.)

Example

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

\begin{verbatim}
gap> G8 := Group( (1,2), (3,4), (5,6) );;
\end{verbatim}
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) ]
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), (), () ], [ (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) ]
8.5.3 AllCat2GroupsMatrix

The operation \texttt{AllCat2GroupsMatrix}(G) constructs a symmetric matrix \( M \) with rows and columns labelled by the cat\(^1\)-groups \( C_i \) on \( G \), where \( M_{ij} = 1 \) 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 \texttt{AllCat2GroupsNumber}(G) returns the number \( n \) of these cat\(^2\)-groups.

```
gap> AllCat2GroupsMatrix(d12);;
number of cat2-groups found = 41
1.....1.1
.1.....1.1
..1....1.1
...1....1.1
.....1....1
1...1...1.1
.1...1...1.1
..11....11
111...111.1
...11111.11
1111111111

gap> AllCat2GroupsNumber(d12);
41
```

8.5.4 AllCat2GroupsIterator

\texttt{AllCat2GroupsIterator}(G) \hspace{1cm} \texttt{AllCat2Groups}(G) \hspace{1cm} \texttt{AllCat2GroupsUpToIsomorphism}(G)
The iterator $\text{AllCat2GroupsIterator}(G)$ iterates through all the cat$^2$-groups with source $G$. The operation $\text{AllCat2Groups}(G)$ 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 $\text{AllCat2GroupsUpToIsomorphism}(G)$ returns representatives of the isomorphism classes of these subgroups. The operation $\text{AllCat2GroupFamilies}(G)$ returns a list of lists. The $k$-th list contains the positions of the cat$^2$-groups in $\text{AllCat2Groups}(G)$ which are isomorphic to the $k$-th representative. So, for $d_{12}$, 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 $\text{CatnGroupNumbers}(G).\text{cat2}$ is the number of cat$^2$-groups on $G$, while $\text{CatnGroupNumbers}(G).\text{iso2}$ is the number of isomorphism classes of these cat$^2$-groups. Also $\text{CatnGroupNumbers}(G).\text{symm}$ is the number of cat$^2$-groups whose Up2DimensionalGroup is the same as the Left2DimensionalGroup, while $\text{CatnGroupNumbers}(G).\text{siso}$ is the number of isomorphism classes of these symmetric cat$^2$-groups.

Provided that $\text{CatnGroupLists}(G).\text{omit}$ is not set to true, sorted lists of generating pairs, and of the classes they belong to, are added to the record $\text{CatnGroupLists}$. For example $[5,7]$ in these lists for $d_{12}$ 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
gap> AllCat2GroupsNumber( d12 );
41
gap> reps2 := AllCat2GroupsUpToIsomorphism( d12 );;

gap> Length( reps2 );
10

gap> List( reps2, C -> StructureDescription( C ) );
[ "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> 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 ] ] ]
```
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.

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($\partial$) 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 cat²-groups or a trio of cat¹-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 cat¹-groups for `d12` generate a cat³-group.

#### Example

```gap
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 cat³-group are stored as these attributes.

#### Example

```gap
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 \texttt{CatnGroupLists(G).pairs} has been obtained we may seek all triples \([i, j, k]\) 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 \texttt{AllCat3GroupTriples}

\[\begin{array}{l}
\triangleright \texttt{AllCat3GroupTriples(G)} \\
\triangleright \texttt{AllCat3GroupsNumber(G)} \\
\triangleright \texttt{AllCat3Groups(G)}
\end{array}\]

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

As yet there is no operation \texttt{AllCat3GroupsUpToIsomorphism(G)}.

\begin{verbatim}
Example

\texttt{gap> triples := AllCat3GroupTriples( d12 );;}
\texttt{gap> CatnGroupNumbers( d12 ).cat3;}
\texttt{94}
\texttt{gap> triples[46];}
\texttt{[ 5, 7, 11 ]}
\texttt{gap> all1 := AllCat1Groups( d12 );;}
\texttt{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) ] )]
\end{verbatim}

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,

\[\mathcal{C}_{A,i} = (e_{A,i}; t_{A,i}, h_{A,i} : G_A \rightarrow G_{A\setminus\{i\}}), \quad \text{for all} \quad A \subseteq [n], \ i \in A,\]

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 \(\mathcal{C}_{A,\{i,j\}}\) 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 $t_{A,B}, h_{A,B} : G_A \rightarrow G_A \setminus B$ and $e_{A,B} : G_A \setminus B \rightarrow G_A$ 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 $\mathcal{C}$ consists of $2^n$ groups $G_A$, one for each subset $A$ of $[n]$, and $3n$ homomorphisms

$$t_{[0],i}, h_{[0],i} : G_{[0]} \rightarrow G_{[0] \setminus [i]}, e_{[0],i} : G_{[0] \setminus [i]} \rightarrow G_{[0]},$$

satisfying the following axioms for all $1 \leq i \leq n$.

- the $G_{[n,i]} = (e_{[n],i}; t_{[n],i}; h_{[n],i} : G_{[n]} \rightarrow G_{[n] \setminus [i]})$ 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 $\mathcal{C}$ consists of two cat$^{(n-1)}$-groups:

- $\mathcal{A}$ with groups $G_A, A \subseteq [n-1]$, and homomorphisms $t_{A,i}, h_{A,i}, e_{A,i},$
- $\mathcal{B}$ with groups $H_B, B \subseteq [n-1]$, and homomorphisms $t_{B,i}, h_{B,i}, e_{B,i}$, and
- cat$^{(n-1)}$-morphisms $t, h : \mathcal{A} \rightarrow \mathcal{B}$ and $e : \mathcal{B} \rightarrow \mathcal{A}$ subject to the following conditions:

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

### 9.3.1 PreCatnGroup

- PreCatnGroup($L$) \hspace{1cm} (operation)
- CatnGroup($L$) \hspace{1cm} (operation)

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

#### Example

```gap
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)

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 $D_X^4$ has discrete source, while the crossed module $C_X^4$ has connected source. These are groupoid equivalents of $\text{XModByNormalSubgroup} \,(2.1.2)$.

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 );;
\begin{verbatim}
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 );;
\end{verbatim}

\subsection*{10.1.2 IsXModWithObjects}
\begin{itemize}
  \item IsXModWithObjects(\texttt{pxmod}) (property)
  \item IsPreXModWithObjects(\texttt{pxmod}) (property)
  \item IsDirectProductWithCompleteDigraphDomain(\texttt{pxmod}) (property)
\end{itemize}

The precrossed module $\text{DX4}$ belongs to the category \texttt{Is2DimensionalGroupWithObjects} and is, of course, a crossed module.

\begin{verbatim}
gap> Set( KnownPropertiesOfObject( DX4 ) );
[ "CanEasilyCompareElements", "CanEasilySortElements", "IsAssociative",
  "IsDirectProductWithCompleteDigraphDomain", "IsDuplicateFree",
  "IsGeneratorsOfSemigroup", "IsPreXModWithObjects", "IsSinglePieceDomain",
\end{verbatim}

\subsection*{10.1.3 IsPermPreXModWithObjects}
\begin{itemize}
  \item IsPermPreXModWithObjects(\texttt{pxmod}) (property)
  \item IsPcPreXModWithObjects(\texttt{pxmod}) (property)
  \item IsFpPreXModWithObjects(\texttt{pxmod}) (property)
\end{itemize}

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

\begin{verbatim}
gap> IsPermPreXModWithObjects( CX4 );
true
gap> IsPcPreXModWithObjects( CX4 );
false
\end{verbatim}

\subsection*{10.1.4 Root2dGroup}
\begin{itemize}
  \item Root2dGroup(\texttt{pxmod}) (attribute)
  \item XModAction(\texttt{pxmod}) (attribute)
\end{itemize}
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 );
[ [ [ (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 horizontal and two vertical, and there is a horizontal 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 $\Omega$. Let $\square$ be the set of squares with objects from $\Omega$ at each corner; plus two vertical arrows and two horizontal arrows from $\text{Arr}(G)$. Further, let $\mathcal{P} = (\partial : S \rightarrow R)$ 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:

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 horizontal composition, as illustrated by

\[
\begin{array}{cccc}
\text{u}_1 & \mathbf{a}_1 & \text{u}_2 & \mathbf{a}_2 & \text{u}_3 \\
& m_1 & & e_1 & m_2 & f_1 = d_1 & m_1 m_2 & f_1 \\
& v_1 & b_1 & & v_2 & b_2 & & v_3 \\
& & m_3 & e_2 & & & w_1 & c_1 & & w_2 \\
& & v_1 & b_1 & & & & & & \\
\end{array}
\]

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:

\[
\partial (m_1^{b_2} m_2) = \partial (m_1^{b_2}) (\partial m_2) = b_2^{-1}(b_1^{-1} d_1^{-1} a_1 e_1) b_2 (b_2^{-1} e_1^{-1} a_2 f_1) = (b_1 b_2)^{-1} d_1^{-1} (a_1 a_2) f_1.
\]

Similarly, vertical composition is illustrated by

\[
\begin{array}{cccc}
\text{u}_1 & \mathbf{a}_1 & \text{u}_2 & \mathbf{a}_1 \text{a}_2 & \text{u}_3 \\
& m_1 & & e_1 & m_2 & f_2 = d_1 & m_1 m_2 & f_2 \\
& v_1 & b_1 & & v_2 & b_1 & & v_3 \\
& & & & w_1 & c_1 & & w_2 \\
& & & & v_1 & b_1 & & v_3 \\
\end{array}
\]

Again the boundary condition is satisfied:

\[
\partial (m_3^{c_2} m_1^{a_2}) = (\partial m_3) \partial (m_1^{a_2}) = (c_1^{-1} d_2^{-1} b_1 e_2) e_2^{-1} (b_1^{-1} d_1^{-1} a_1 e_1) e_2 = c_1^{-1} (d_1 d_2)^{-1} a_1 (e_1 e_2).
\]

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

\[
\begin{array}{cccc}
\text{u}_1 & \mathbf{a}_1 & \text{u}_2 & \mathbf{a}_2 & \text{u}_3 \\
& m_1 & & e_1 & m_2 & f_1 = d_1 & m_1 m_2 & f_1 \\
& v_1 & b_1 & & v_2 & b_1 & & v_3 \\
& & & & w_1 & c_1 & & w_2 \\
& & & & v_1 & b_1 & & v_3 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{u}_1 & \mathbf{a}_1 \text{a}_2 & \text{u}_3 \\
& m_1 m_2 (m_1^{b_2} m_2)^{c_2} & f_1 f_2 = d_1 & m_1 m_2 (m_1^{b_2} m_2)^{c_2} & f_1 f_2 \\
& v_1 & c_1 & & w_3 \\
& & & & & & & \\
\end{array}
\]
where
\[ \begin{align*}
& m^2_3 m_4 (m_1^2 m_2)^{f_2} = (m_3 m_1^2)^c_2 m_4 m_2^{f_2} = (c_1 c_2)^{-1}(a_1 a_2)^{-1}(a_1 a_2)(f_1 f_2).
\end{align*} \]

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 \([-6..-1]\) and, for our pre-crossed module, the \( X_{12} \), isomorphic to \( (D_{12} \rightarrow S_3) \), constructed using \texttt{XModByCentralExtension} (2.1.5). The source of \( X_{12} \) 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.

\[
\begin{array}{cccc}
-6 & (7,8) & -5 & (8,9) \\
(7,9) & gh & (8,9) & g^2 & (7,8) \\
-1 & (7,9,8) & -3 & (7,9) & -4 \\
(8,9) & g & (7,9,8) & h & (8,9) \\
-2 & (7,9) & -2 & (7,8) & -3
\end{array}
\]

Example

\[
\begin{array}{cccc}
-6 & (7,8,9,13,14,15,16) & -5 & (8,9,10,11) \\
(7,9,10) & gh & (8,9,10) & (9,10) \\
-1 & (7,9,10,11,12,13,14,15,16) & -3 & (7,9,10,11,12,13,14,15,16) \\
(8,9,10) & g & (7,9,10,11,12,13,14,15,16) & h & (8,9,10) \\
-2 & (7,9,10,11,12,13,14,15,16) & -2 & (7,9,10,11,12,13,14,15,16) & -3
\end{array}
\]

\[
\begin{array}{cccc}
\text{Crossed module } X_{12} :-
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{Source group } d_{12} \text{ has generators:} \\
\text{Range group } s_3 \text{ has generators:} \\
\text{Boundary homomorphism maps source generators to:} \\
\text{Action homomorphism maps range generators to automorphisms:}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{These 2 automorphisms generate the group of automorphisms.}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{Example}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{Example}
\end{array}
\end{array}
\]
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 \(B_0\) to \(D_0\), and we check that the same square is produced in each case.

```gap
g := (1,2,3,4);; h := (1,3);;
gend8 := [ g, h ];;
d8 := Group( gend8 );;
SetName( d8, "d8" );
Gd8 := Groupoid( d8, [-9..-7] );;
SetName( Gd8, "Gd8" );
B0 := SinglePieceBasicDoubleGroupoid( Gd8 );;
SetName( B0, "B0\!" );
Gd8
B0!.objects;
[ -9 .. -7 ]
a0 := Arrow(Gd8,(),-9,-7);;
b0 := Arrow(Gd8,(2,4),-9,-8);;
d0 := Arrow(Gd8,g,-9,-9);;
e0 := Arrow(Gd8,(1,3),-7,-8);;
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]
D0 := EnhancedBasicDoubleGroupoid( B0 );;
D0!.prexmod;
[d8->d8]
bsq0 := SquareOfArrows( D0, bdy0, a0, d0, e0, b0 );
true
```

### 11.3 Commutative double groupoids

A double groupoid square

\[
\begin{array}{ccc}
  u_1 & \xrightarrow{a_1} & u_2 \\
  d_1 & \downarrow & c_1 \\
  v_1 & \xleftarrow{b_1} & v_2
\end{array}
\]

is commutative if \(a_1 c_1 = d_1 b_1\), 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 horizontally 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 \( Gd_8 \) and the pre-crossed module \( Q_{16} \) of section 2.3. We introduce a new right arrow to construct a square which commutes.

```
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 $\mathcal{M} = (\partial : M \to P)$ be a crossed module of groups and let $X = B\mathcal{M}$ be the classifying space of $\mathcal{M}$. 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 $L\mathcal{M}[a] = (\partial_a : M \to P(a))$ 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 $\text{cokernel}(\partial_a)$, and $\pi_2(LX, a) \cong \pi_2(X, *)^{\partial_a}$, the elements of $\pi_2(X, *)$ fixed under the action of $\partial_a$, the class of $a$ in $\pi_1(X, *)$.

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

12.1.1 LoopClasses

- LoopClasses($\mathcal{M}$) (operation)
- LoopsXMod($\mathcal{M}, a$) (operation)
- AllLoopsXMod($\mathcal{M}$) (operation)

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

The operation LoopsXMod($\mathcal{M}, a$) calculates the crossed module $L\mathcal{M}[a]$ described in the theorem.
The operation $\text{AllLoopsXMod}(M)$ 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 $P = \text{PSL}(3,2)$ 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.

```gap
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

\( \text{▷ SmallCat1Group}(n, i, j) \) (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.

```
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

\( \text{▷ CatOneGroupToXMod}(C) \) (operation)
\( \text{▷ Cat1GroupToHAP}(C) \) (operation)

These two functions convert between the two alternative implementations.

```
gap> C12 := CatOneGroupToXMod( H12 );
[Group( [ f1, f2, f3 ] )=>Group( [ f1, f2, <identity> of ... ] )]
```
13.1.3 \textbf{IdCat1Group}

\begin{verbatim}
gap> IdCat1Group( H18 );
[ 18, 4, 4 ]
gap> IdCat1Group( C18 );
[ 18, 4, 4 ]
\end{verbatim}

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

Example

\begin{verbatim}
gap> IdCatOneGroup( H18 );
[ 18, 4, 4 ]
gap> IdCat1Group( C18 );
[ 18, 4, 4 ]
\end{verbatim}
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 )
▷ MappingToOne( G, H )

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.

```
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(\textit{G}, \textit{N}) (operation)

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

\begin{verbatim}
  gap> autd8 := AutomorphismGroup( d8 );;
  gap> innd8 := InnerAutomorphismsByNormalSubgroup( d8, d8 );;
  gap> GeneratorsOfGroup( innd8 );
  [ ^(1,2,3,4), ^(1,3) ]
\end{verbatim}

14.1.3 IsGroupOfAutomorphisms

▷ IsGroupOfAutomorphisms(\textit{A}) (property)

Tests whether the elements of a group are automorphisms.

\begin{verbatim}
  gap> IsGroupOfAutomorphisms( innd8 );
  true
\end{verbatim}

14.2 Abelian Modules

14.2.1 AbelianModuleObject

▷ AbelianModuleObject(\textit{grp}, \textit{act}) (operation)
▷ IsAbelianModule(\textit{obj}) (property)
▷ AbelianModuleGroup(\textit{obj}) (attribute)
▷ AbelianModuleAction(\textit{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 \textit{Xabmod} is isomorphic to the output from XModByAutomorphismGroup( k4 );.

\begin{verbatim}
  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 );
\end{verbatim}
Crossed module \([k4a\rightarrow s3a]\) :-

: 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:
\[(1,2) \rightarrow \{\text{source gens} \rightarrow [(6,8)(7,9), (6,7)(8,9)]\}\]
\[(2,3) \rightarrow \{\text{source gens} \rightarrow [(6,7)(8,9), (6,9)(7,8)]\}\]

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 revisions 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 ExtRep0f0b of words in the generators.
The \textsc{AllCat1s} function was reintroduced from the GAP3 version, and renamed as the operation \textsc{AllCat1DataGroupsBasic}.

In version 2.25 the data in \texttt{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.

\subsection*{15.2.2 Versions 2.43 - 2.56}

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

\begin{itemize}
\item The material on isoclinism in Chapter 4 was added.
\item The package webpage has moved to \url{https://github.com/cdwensley}.
\item A GitHub repository was started at: \url{https://github.com/gap-packages/xmod}.
\item The section on \textit{Distinct and Common Representatives} was moved to the \texttt{Utils} package.
\end{itemize}

\subsection*{15.2.3 Version 2.61}

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

\subsection*{15.2.4 Versions 2.63 - 2.74}

\begin{itemize}
\item Added an implementation of crossed modules of groupoids.
\item Lots more work on crossed squares and cat2-groups.
\item Added an implementation of group groupoids.
\end{itemize}

\subsection*{15.2.5 Versions 2.75 - 2.85}

\begin{itemize}
\item Added conversion functions between \texttt{XMod} and \texttt{Hap} and a new chapter in the manual about these functions.
\item Added functions for quasi-isomorphisms.
\end{itemize}

\subsection*{15.2.6 Versions 2.86 - 2.91}

\begin{itemize}
\item Added attributes \texttt{Size2d} for 2d-objects and \texttt{Size3d} for 3d-objects since lists are inappropriate values for the standard function \texttt{Size}.
\item Added \texttt{PreXModWithTrivialRange} and started work on double groupoids.
\end{itemize}
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 $\text{Act } L = (t : L \to \text{Aut } L)$ splits to form the square with $(t_1 : L \to \text{Inn } L)$ on two sides, and $(t_2 : \text{Inn } L \to \text{Aut } L)$ on the other two sides, where $t_1$ maps $l \in L$ to the inner automorphism $\beta_l : L \to L, \ l' \mapsto l^{-1}l'l$, and $t_2$ is the inclusion of $\text{Inn } L$ in $\text{Aut } L$. The actions are standard, and the crossed pairing is
    \[
    \boxtimes : \text{Inn } L \times \text{Inn } L \to L, \quad (\beta_l, \beta_{l'}) \mapsto [l, l'].
    \]
- Improve the interaction with the HAP package.
- Implement cat1-groups with objects.
- Lots more work on double groupoids.
References


[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


Index

2d-domain, 8
2d-group, 8
2d-mapping, 32
2dimensional-domain with objects, 92
3d-domain, 66
3d-group, 66
3d-mapping, 76
4d-domain, 88
4d-group, 88

abelian module, 106
AbelianModuleAction, 106
AbelianModuleGroup, 106
AbelianModuleObject, 106
actor, 57
ActorCrossedSquare, 70
ActorXMod, 58
AllCat1Groups, 26
AllCat1GroupsIterator, 26
AllCat1GroupsMatrix, 25
AllCat1GroupsUpToIsomorphism, 26
AllCat1GroupsWithImage, 24
AllCat1GroupsWithImageIterator, 24
AllCat1GroupsWithImageNumber, 24
AllCat1GroupsWithImageUpToIsomorphism, 24
AllCat2GroupFamilies, 86
AllCat2Groups, 85
AllCat2GroupsIterator, 85
AllCat2GroupsMatrix, 85
AllCat2GroupsUpToIsomorphism, 85
AllCat2GroupsWithFixedUp, 84
AllCat2GroupsWithFixedUpAndLeftRange, 84
AllCat2GroupsWithImages, 83
AllCat2GroupsWithImagesIterator, 83
AllCat2GroupsWithImagesNumber, 83
AllCat2GroupsWithImagesUpToIsomorphism, 83
AllCat3Groups, 90
AllCat3GroupsNumber, 90
AllCat3GroupTriples, 90
AllDerivations, 54
AllInducedXMods, 64
AllLoopsXMod, 101
AllSections, 56
AllStemGroupFamilies, 48
AllStemGroupIds, 48
AllXMods, 46
AllXModsUpToIsomorphism, 46
AllXModsWithGroups, 46
AreIsoclinicDomains
  for crossed modules of groups, 49
  for groups, 47
AutoGroup, 12
AutomorphismPermGroup, 57
Back3DimensionalGroup, 89
Boundary
  for cat 1 -groups, 18
  for crossed modules, 12

cat 1 -group, 17
cat 2 -group, 78
cat 3 -group, 88
cat n -group, 90
Cat1Group, 17
Cat1GroupByPeifferQuotient, 20
Cat1GroupMorphism, 35
Cat1GroupMorphismByGroupHomomorphisms, 35
Cat1GroupMorphismOfXModMorphism, 35
Cat1GroupOfXMod, 23
Cat1GroupToHAP, 103
Cat1Select, 27
CAT1_LIST_MAX_SIZE, 27
CAT1_LIST_NUMBERS, 27
Cat2Group, 79
for precat1-morphisms, 35
IdentitySection, 53
IdGroup
  for 2d-groups, 28
  for crossed modules, 13
ImageElmCrossedPairing, 75
ImageElmXModAction, 12
ImagesList, 55
ImagesTable, 54
inclusion mapping, 105
InclusionMappingGroups, 105
InclusionMorphism2DimensionalDomains
  for cat1-groups, 35
  for crossed modules, 33
InclusionMorphismHigherDimensionalDomains, 78
induced cat1-groups, 64
induced crossed module, 62
InducedCat1Group, 64
InducedCat1GroupByFreeProduct, 64
InducedXMod, 62
InducedXModByCopower, 62
InducedXModBySurjection, 62
InfoXMod, 6
InitCatnGroupRecords, 26
InnerActorXMod, 60
InnerAutomorphismCat1, 35
InnerAutomorphismsByNormalSubgroup, 106
InnerAutomorphismXMod, 33
InnerMorphism, 60
IntersectionSubXMods, 41
Is2DimensionalDomain, 14
Is2DimensionalGroup, 14
Is2DimensionalGroupWithObjects, 93
Is3dObject, 74
IsAbelian2DimensionalGroup, 44
IsAbelian3DimensionalGroup, 75
IsAbelianModule, 106
IsAbelianModule2DimensionalGroup, 14
IsAspherical2DimensionalGroup, 44
IsAutomorphism3dObject, 76
IsAutomorphismGroup2DimensionalGroup, 14
IsAutomorphismGroup3DimensionalGroup, 75
IsBijective, 76
  for pre-xmod morphisms, 33
IsCat1Group, 22
IsCat1GroupMorphism, 35
IsCat2Group, 79
IsCat3Group, 89
IsCentralExtension2DimensionalGroup, 14
IsCentralExtension3DimensionalGroup, 75
IsCrossedSquare, 74
IsCrossedSquareMorphism, 76
IsDerivation, 51
IsDirectProductWithCompleteDigraphDomain, 93
IsEndo2DimensionalMapping, 33
IsEndomorphism3dObject, 76
IsFaithful2DimensionalGroup, 44
IsFp2DimensionalGroup, 14
IsFp3dObject, 74
IsFpPreXModWithObjects, 93
IsGroupOfAutomorphisms, 106
IsIdentityCat1Group, 22
IsInducedXMod, 62
IsInjective
  for pre-xmod morphisms, 33
IsMonoidOfUp2DimensionalMappingsObj, 54
IsNilpotent2DimensionalGroup, 45
IsNormal for crossed modules, 14
IsNormal3DimensionalGroup, 75
IsNormalSubgroup2DimensionalGroup, 14
IsoclinicMiddleLength
  for crossed modules of groups, 50
  for groups, 48
IsoclinicRank
  for crossed modules of groups, 50
  for groups, 48
IsoclinicStemDomain
  for crossed modules of groups, 49
  for groups, 47
Isoclinitism
  for crossed modules, 49
  for groups, 47
IsomorphicPreCat1GroupWithIdentityEmbedding, 23
IsomorphismByIsomorphisms, 34
IsomorphismClassRepresentatives2dGroups, 46
IsomorphismFp2DimensionalGroup
  for pre-cat1 morphisms, 36
Isomorphism\(\text{Pc2DimensionalGroup}\) for pre-cat1 morphisms, 36
for pre-xmod morphisms, 34
Isomorphism\(\text{Perm2DimensionalGroup}\) for pre-cat1 morphisms, 36
for pre-xmod morphisms, 34
Isomorphism\(\text{PermObject}\), 36
Isomorphism\(\text{ToPreCat1GroupWithIdentityEmbedding}\), 23
Isomorphism\(\text{XMods}\), 45
Is\(\text{Pc2DimensionalGroup}\), 14
Is\(\text{Pc3dObject}\), 74
Is\(\text{Perm2DimensionalGroup}\), 14
Is\(\text{Perm3dObject}\), 74
Is\(\text{PermPreXModWithObjects}\), 93
Is\(\text{PreCat1GroupMorphism}\), 35
Is\(\text{PreCat1GroupWithIdentityEmbedding}\), 23
Is\(\text{PreCrossedSquare}\), 74
Is\(\text{PreCrossedSquareMorphism}\), 76
Is\(\text{PreXCat1Group}\), 22
Is\(\text{PreXMod}\), 14
Is\(\text{PreXModMorphism}\), 32
Is\(\text{PreXModWithObjects}\), 93
Is\(\text{Section}\), 53
Is\(\text{SimplyConnected2DimensionalGroup}\), 44
Is\(\text{SingleValued}\)
for pre-xmod morphisms, 33
Is\(\text{StemDomain}\)
for pre-xmod morphisms, 33
Is\(\text{Sub2DimensionalGroup}\), 29
Is\(\text{SubCat1Group}\), 28
Is\(\text{SubPreCat1Group}\), 28
Is\(\text{SubXMod}\), 28
Is\(\text{Surjective}\)
for pre-xmod morphisms, 33
Is\(\text{Symmetric3DimensionalGroup}\), 75
Is\(\text{Total}\)
for pre-xmod morphisms, 33
Is\(\text{TrivialAction2DimensionalGroup}\), 14
Is\(\text{TrivialAction3DimensionalGroup}\), 75
Is\(\text{Up2DimensionalMapping}\), 51
Is\(\text{XMod}\), 14
Is\(\text{XModMorphism}\), 32
Is\(\text{XModWithObjects}\), 93
Kernel
for 2d-mappings, 37
Kernel\(\text{2DimensionalMapping}\), 37
Kernel\(\text{CokernelXMod}\), 15
Kernel\(\text{Embedding}\), 18
Left\(\text{2DimensionalGroup}\), 74
Left\(\text{2DimensionalMorphism}\), 76
Left\(\text{3DimensionalGroup}\), 89
loop space, 101
LoopClasses, 101
Loops\(\text{XMod}\), 101
Lower\(\text{CentralSeriesOfXMod}\), 45
Norrie\(\text{XMod}\), 58
Mapping\(\text{2ArgumentsByFunction}\), 75
Mapping\(\text{ToOne}\), 105
morphism, 32
morphism of 2d-group, 32
morphism of 3d-group, 76
Morphism\(\text{OfInducedXMod}\), 62
Morphism\(\text{OfPullback}\)
for a crossed module by pullback, 34
Name, 74
for cat1-groups, 18
for crossed modules, 13
Natural\(\text{MorphismByNormalSubPreXMod}\), 40
Nilpotency\(\text{Class2DimensionalGroup}\), 45
Normalizer, 43
Normal\(\text{SubXMods}\), 15
Norrie\(\text{XMod}\), 58
Object\(\text{List}\), 93
operations on morphisms, 37
order of a 2d-automorphism, 33
Peiffer subgroup, 16
Peiffer\(\text{Subgroup}\), 17
Perm\(\text{AutomorphismAsXModMorphism}\), 57
pre-crossed module, 16
Pre\(\text{Cat1Group}\), 17
Pre\(\text{Cat1GroupByTailHeadEmbedding}\), 17
Pre\(\text{Cat1GroupMorphism}\), 35
Pre\(\text{Cat1GroupMorphismByGroupHomomorphisms}\), 35
Whitehead multiplication, 51
WhiteheadGroupTable, 55
WhiteheadMonoidTable, 54
WhiteheadOrder, 54
WhiteheadPermGroup, 55
WhiteheadProduct, 54
WhiteheadTransformationMonoid, 54
WhiteheadXMod, 58

XMod, 8
XModAction
  for crossed modules of groupoids, 93
  for crossed modules of groups, 12
XModByAbelianModule, 11
XModByAutomorphismGroup, 9
XModByBoundaryAndAction, 8
XModByCentralExtension, 10
XModByGroupOfAutomorphisms, 9
XModByInnerAutomorphismGroup, 9
XModByNormalSubgroup, 9
XModByPeifferQuotient, 17
XModByPullback, 11
XModByTrivialAction, 9
XModCentre, 60
XModMorphism, 33
XModMorphismByGroupHomomorphisms, 33
XModMorphismOfCat1GroupMorphism, 35
XModOfCat1Group, 23