XMod

A GAP4 Package

Crossed Modules and Cat1-groups

Version 2.006

by

Murat Alp

Dumlupinar Universitesi, Fen-Edebiyat Fakultesi, Matematik Bolumu

Merkez Kampus, Kutahya, Turkey

email: malp@dumlupinar.edu.tr

and

Chris Wensley

School of Informatics, University of Wales Bangor

Dean Street, Bangor, Gwynedd, LL57 1UT, U.K.

email: c.d.wensley@bangor.ac.uk

September 2004

Contents

1 Preface

The XMod package provides functions for the computation with finite

– crossed modules and cat1-groups, and morphisms of these structures;
– pre-crossed modules, pre-cat1-groups, and their Peiffer quotients;
– derivations of crossed modules and sections of cat1-groups; and
– the actor crossed square of a crossed module.

It is loaded with the command

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

XMod was originally implemented in 1997 using the GAP 3 language. In April 2002 the first and third parts were converted to GAP 4, the pre-structures were added, and version 2.001 was released. The final two parts, covering derivations, sections and actors, were included in the January 2004 release 2.002 for GAP 4.4.

The current version is 2.006, released on September 6th 2004.

Many of the function names have been changed during the conversion, for example `ConjugationXMod` has become `XModByNormalSubgroup`. For a list of name changes see the file `names.pdf` in the `doc` directory.

1 ► `InfoXMod` V

In order that the user has some control of the verbosity of the XMod package's functions, an *InfoClass* `InfoXMod` is provided (see Chapter 7.4 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`, e.g.

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

The following test file (from the "xmod" directory) runs all the manual commands.

```
gap> ReadTest("tst/xmod_manual.tst");
+ Testing constructions of crossed modules and cat1-groups
+ GAP4stones: 0
true
```

Please send bug reports, suggestions and other comments to the second of these e-mail addresses.

Additional information can be found on the **Computational Higher-dimensional Discrete Algebra** web site at

http://www.maths.bangor.ac.uk/chda/

2 2d-objects

2.1 Introduction

The XMod package provides functions for computation with finite **crossed modules** and **cat1-groups** and their morphisms. See the file **notes.pdf** in the **doc** directory for an introductory account of these algebraic gadgets.

The package was initially produced while Murat Alp was a Ph.D. student at Bangor (see [Alp97]).

Crossed modules and cat1-groups are special types of **2-dimensional groups** [Bro82] and are implemented as `2dObjects` having a `Source` and a `Range`.

The package divides into four parts, all of which have been converted from GAP 3 to the GAP 4.4 release.

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

The package may be obtained as a compressed file by ftp from one of the sites with a GAP 4 archive, or from the Bangor Mathematics web site:

`http://www.maths.bangor.ac.uk/chda/`

The following constructions were not in the GAP 3 version of the package: sub-2dobject functions, functions for pre-crossed modules and the Peiffer subgroup of a pre-crossed module, and the associated crossed modules. The source and range groups in these constructions are no longer required to be permutation groups.

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) and **crossed squares** and the equivalent **cat2-groups**, structures which arise as 3-dimensional groups. Examples of these are implicitly included in the fourth part, namely inclusions of normal sub-crossed modules, and the inner morphism from a crossed module to its actor.

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 crossed resolutions package `GraphGpd` [Moo01], and further work on group groupoids is planned.

2.2 Crossed modules

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.

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

$$\begin{array}{lcl} \textbf{XMod 1} & : & \partial(s^r) \;=\; r^{-1}(\partial s)r \;=\; (\partial s)^r \;, \\ \textbf{XMod 2} & : & s_1^{\partial s_2} \;=\; s_2^{-1} s_1 s_2 \;=\; {s_1}^{s_2} \;\cdot \end{array}$$

The kernel of ∂ is abelian.

There are a variety of constructors for crossed modules:

1 ▶ `XMod(` *args* `)` F
▶ `XModByBoundaryAndAction(` *bdy, act* `)` O
▶ `XModByTrivialAction(` *bdy* `)` O
▶ `XModByNormalSubgroup(` *G, N* `)` O
▶ `XModByCentralExtension(` *bdy* `)` O
▶ `XModByAutomorphismGroup(` *grp* `)` O
▶ `XModByInnerAutomorphismGroup(` *grp* `)` O
▶ `XModByGroupOfAutomorphisms(` *G, A* `)` O
▶ `XModByAbelianModule(` *abgrp* `)` O

Here are the standard constructions which these implement:

- A **conjugation crossed module** is an inclusion of a normal subgroup $S \trianglelefteq R$, where R acts on S by conjugation.
- A **central extension crossed module** has as boundary a surjection $\partial : S \to R$ with central kernel, where $r \in R$ acts on S by conjugation with $\partial^{-1}r$.
- An **automorphism crossed module** has as range a subgroup R of the automorphism group $\mathrm{Aut}(S)$ of S which contains the inner automorphism group of S. The boundary maps $s \in S$ to the inner automorphism of S by s.
- A **trivial action crossed module** $\partial : S \to R$ has $s^r = s$ for all $s \in S,\ r \in R$, the source is abelian and the image lies in the centre of the range.
- A **crossed abelian module** has an abelian module as source and the zero map as boundary.
- The direct product $\mathcal{X}_1 \times \mathcal{X}_2$ 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.

2 ▶ `Source(` *X0* `)` A
▶ `Range(` *X0* `)` A
▶ `Boundary(` *X0* `)` A
▶ `AutoGroup(` *X0* `)` A
▶ `XModAction(` *X0* `)` A

In this implementation the attributes used in the construction of a crossed module *X0* are as follows.

- `Source(X0)` and `Range(X0)`, the source S and range R of $\partial =$ `Boundary(X0)`;
- `XModAction(X0)`, a homomorphism from R to `AutoGroup(X0)`, a group of automorphisms of S.

3 ► `Size(` *X0* `)` A
► `Name(` *X0* `)` A

More familiar attributes are `Size` and `Name`, the latter formed by concatenating the names of the source and range. An `Enumerator` function has not been implemented. The `Display` function is used to print details of `2dobjects`.

Here is a simple example of an automorphism crossed module, using a cyclic group of size five.

```
gap> c5 := Group( (5,6,7,8,9) );;
gap> SetName( c5, "c5" );
gap> X1 := XModByAutomorphismGroup( c5 );
[c5 -> PAut(c5)]
gap> Display( X1 );
Crossed module [c5 -> PAut(c5)] :-
: Source group c5 has generators:
  [ (5,6,7,8,9) ]
: Range group PAut(c5) has generators:
  [ (1,2,4,3) ]
: Boundary homomorphism maps source generators to:
  [ () ]
: Action homomorphism maps range generators to automorphisms:
  (1,2,4,3) --> { source gens --> [ (5,7,9,6,8) ] }
  This automorphism generates the group of automorphisms.
gap> Size( X1 );
[ 5, 4 ]
gap> Print( RepresentationsOfObject(X1), "\n" );
[ "IsComponentObjectRep", "IsAttributeStoringRep", "IsPreXModObj" ]
gap> Print( KnownPropertiesOfObject(X1), "\n" );
[ "Is2dObject", "IsPerm2dObject", "IsPreXMod", "IsXMod",
  "IsTrivialAction2dObject", "IsAutomorphismGroup2dObject" ]
gap> Print( KnownAttributesOfObject(X1), "\n" );
[ "Name", "Size", "Range", "Source", "Boundary", "AutoGroup", "XModAction" ]
```

4 ► `SubXMod(` *X0*, *src*, *rng* `)` O
► `IdentitySubXMod(` *X0* `)` A
► `NormalSubXMods(` *X0* `)` A
► `DirectProduct(` *X1*, *X2* `)` O

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

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

These conditions ensure that $M \ltimes N$ is normal in the semidirect product $R \ltimes S$.

2.3 Pre-crossed modules

1 ► `PeifferSubgroup( X0 )` A
► `PreXModByBoundaryAndAction( bdy, act )` O
► `PreXModByCentralExtension( bdy )` O
► `SubPreXMod( X0, src, rng )` O
► `XModByPeifferQuotient( pre-xmod )` A

When axiom **XMod 2** is **not** satisfied, the corresponding structure is known as a **pre-crossed module**. In this case the **Peiffer subgroup** of P of S is the subgroup of $\ker(\partial)$ generated by **Peiffer commutators**

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

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

2 ► `IsPermXMod( X0 )` P
► `IsPcPreXMod( X0 )` P

When both source and range groups are of the same type, corresponding properties are assigned to the crossed module.

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

```
gap> c := (11,12,13,14,15,16,17,18);;  d := (12,18)(13,17)(14,16);;
gap> d16 := Group( c, d );
gap> gend16 := GeneratorsOfGroup( d16 );;
gap> sk4 := Subgroup( d16, [ c^4, d ] );;
gap> gensk4 := GeneratorsOfGroup( sk4 );;
gap> SetName( d16, "d16" );  SetName( sk4, "sk4" );
gap> f16 := GroupHomomorphismByImages( d16, sk4, gend16, gensk4 );;
gap> P16 := PreXModByCentralExtension( f16 );
[d16->sk4]
gap> P := PeifferSubgroup( P16 );
Group( [ (11,17,15,13)(12,18,16,14) ] )
gap> X16 := XModByPeifferQuotient( P16 );
[d16/P->sk4]
gap> Display( X16 );
Crossed module [d16/P->sk4] :-
: Source group has generators:
  [ f1, f2 ]
: Range group has generators:
  [ (11,15)(12,16)(13,17)(14,18), (12,18)(13,17)(14,16) ]
: Boundary homomorphism maps source generators to:
  [ (12,18)(13,17)(14,16), (11,15)(12,16)(13,17)(14,18) ]
  The automorphism group is trivial
gap> iso16 := IsomorphismPermGroup( Source( X16 ) );;
gap> S16 := Image( iso16 );
Group([ (1,3)(2,4), (1,2)(3,4) ])
```

2.4 Cat1-groups and pre-cat1-groups

In this implementation a cat1-group $\mathcal{C}$ has attributes:

1 ► `Source( C )` A
► `Range( C )` A
► `Tail( C )` A
► `Head( C )` A
► `RangeEmbedding( C )` A
► `KernelEmbedding( C )` A
► `Boundary( C )` A
► `Name( C )` A
► `Size( C )` A

In [Lod82], Loday reformulated the notion of a crossed module as a cat1-group, namely a group G with a pair of homomorphisms $t, h : G \to G$ having a common image R and satisfying certain axioms. We find it convenient to define a cat1-group $\mathcal{C} = (e; t, h : G \to R)$ as having source group G, range group R, and three homomorphisms: two surjections $t, h : G \to R$ and an embedding $e : R \to G$ satisfying:

$$\begin{array}{lcl} \textbf{Cat 1} & : & te \; = \; he \; = \; \mathrm{id}_R \;, \\ \textbf{Cat 2} & : & [\ker t, \ker h] \; = \; \{1_G\} \;. \end{array}$$

It follows that $teh = h$, $het = t$, $tet = t$, $heh = h$.

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

Here are some of the constructors for pre-cat1-groups and cat1-groups:

2 ► `Cat1( args )` F
► `PreCat1ByTailHeadEmbedding( t, h, e )` O
► `PreCat1ByEndomorphisms( t, h )` O
► `PreCat1ByNormalSubgroup( G, N )` O
► `Cat1ByPeifferQuotient( P )` O
► `Reverse( C0 )` A

The following listing shows an example of a cat1-group of pc-groups:

```
gap> s3 := SymmetricGroup(IsPcGroup,3);;
gap> gens3 := GeneratorsOfGroup(s3);
[ f1, f2 ]
gap> pc4 := CyclicGroup(4);;
gap> SetName(s3,"s3");  SetName( pc4, "pc4" );
gap> s3c4 := DirectProduct( s3, pc4 );;
gap> SetName( s3c4, "s3c4" );
gap> gens3c4 := GeneratorsOfGroup( s3c4 );
[ f1, f2, f3, f4 ]
gap> a := gens3[1];;  b := gens3[2];;  one := One(s3);;
gap> t2 := GroupHomomorphismByImages( s3c4, s3, gens3c4, [a,b,one,one] );
[ f1, f2, f3, f4 ] -> [ f1, f2, <identity> of ..., <identity> of ... ]
gap> e2 := Embedding( s3c4, 1 );
[ f1, f2 ] -> [ f1, f2 ]
gap> C2 := Cat1( t2, t2, e2 );
[s3c4=>s3]
```

```
gap> Display( C2 );
Cat1-group [s3c4=>s3] :-
: source group has generators:
  [ f1, f2, f3, f4 ]
:  range group has generators:
  [ f1, f2 ]
: tail homomorphism maps source generators to:
  [ f1, f2, <identity> of ..., <identity> of ... ]
: head homomorphism maps source generators to:
  [ f1, f2, <identity> of ..., <identity> of ... ]
: range embedding maps range generators to:
  [ f1, f2 ]
: kernel has generators:
  [ f3, f4 ]
: boundary homomorphism maps generators of kernel to:
  [ <identity> of ..., <identity> of ... ]
: kernel embedding maps generators of kernel to:
  [ f3, f4 ]
gap> IsPcCat1( C2 );
true
gap> Size( C2 );
[ 24, 6 ]
```

3 ► `Cat1OfXMod(` *X0* `)` A
► `XModOfCat1(` *C0* `)` A
► `PreCat1OfPreXMod(` *P0* `)` A
► `PreXModOfPreCat1(` *P0* `)` A

The category of crossed modules is equivalent to the category of cat1-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 $G = R \ltimes S$ using the action from $\mathcal{X}$, with multiplication rule

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

The structural morphisms are given by

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

On the other hand, starting with a cat1-group $\mathcal{C} = (e; t, h : G \to R)$, we define $S = \ker t$, the range R remains unchanged, and $\partial = h|_S$. The action of R on S is conjugation in G via the embedding of R in G.

```
gap> SetName( Kernel(t2), "ker(t2)" );;
gap> X2 := XModOfCat1( C2 );
[Group( [ f3, f4 ] )->s3]
gap> Display( X2 );
Crossed module [ker(t2)->s3] :-
: Source group has generators:
  [ f3, f4 ]
: Range group s3 has generators:
  [ f1, f2 ]
: Boundary homomorphism maps source generators to:
  [ <identity> of ..., <identity> of ... ]
  The automorphism group is trivial
: associated cat1-group is [s3c4=>s3]
```

2.5 Selection of a small cat1-group

The `Cat1` function may also be used to select a cat1-group from a data file. All cat1-structures on groups of size up to 47 are stored in a list in file `cat1data.g`. Global variables `CAT1_LIST_MAX_SIZE := 47` and `CAT1_LIST_CLASS_SIZES` are also stored. The XMod2 version of the database orders the groups of size up to 47 according to the GAP4 numbering of small groups. The data is read into the list `CAT1_LIST` only when this function is called.

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

```
gap> CAT1_LIST_CLASS_SIZES[ 18 ];
5
gap> Cat1( 18 );
#I  Loading cat1-group data into CAT1_LIST
Usage:  Cat1( size, gpnum, num )
[ "d18", "c18", "s3c3", "c3^2Xc2", "c6c3" ]
gap> Cat1( 18, 4 );
There are 4 cat1-structures for the group c3^2 X c2.
[ [range gens], source & range names, [tail genimages], [head genimages] ] :-
[ [ (1,2,3), (4,5,6), (2,3)(5,6) ],  tail = head = identity mapping ]
[ [ (2,3)(5,6) ], "c3^2", "c2", [ (), (), (2,3)(5,6) ],
  [ (), (), (2,3)(5,6) ] ]
[ [ (4,5,6), (2,3)(5,6) ], "c3", "s3", [ (), (4,5,6), (2,3)(5,6) ],
  [ (), (4,5,6), (2,3)(5,6) ] ]
[ [ (4,5,6), (2,3)(5,6) ], "c3", "s3", [ (4,5,6), (4,5,6), (2,3)(5,6) ],
  [ (), (4,5,6), (2,3)(5,6) ] ]
Usage:  Cat1( size, gpnum, num )
Group has generators [ (1,2,3), (4,5,6), (2,3)(5,6) ]

gap> C4 := Cat1( 18, 4, 4 );
[c3^2 X c2=>s3]
gap> Display( C4 );
Cat1-group [c3^2 X c2=>s3] :-
: source group has generators:
  [ (1,2,3), (4,5,6), (2,3)(5,6) ]
:  range group has generators:
  [ (4,5,6), (4,5,6), (2,3)(5,6) ]
: tail homomorphism maps source generators to:
  [ (4,5,6), (4,5,6), (2,3)(5,6) ]
: head homomorphism maps source generators to:
  [ (), (4,5,6), (2,3)(5,6) ]
: range embedding maps range generators to:
  [ (4,5,6), (4,5,6), (2,3)(5,6) ]
: kernel has generators:
  [ ( 1, 2, 3)( 4, 6, 5) ]
: boundary homomorphism maps generators of kernel to:
  [ (4,6,5) ]
: kernel embedding maps generators of kernel to:
  [ (1,2,3)(4,6,5) ]

gap> XC4 := XModOfCat1( C4 );
[Group( [ ( 1, 2, 3)( 4, 6, 5) ] )->s3]
```

3 2d-mappings

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

1 ► `Source( map )` A
► `Range( map )` A
► `SourceHom( map )` A
► `RangeHom( map )` A

Morphisms of `2dObjects` are implemented as `2dMappings`. These have a pair of 2d-objects 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.

3.1 Morphisms of pre-crossed modules

1 ► `IsPreXModMorphism( map )` P
► `IsXModMorphism( map )` P
► `IsPreCat1Morphism( map )` P
► `IsCat1Morphism( map )` P

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

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

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

2 ► `IsInjective( map )` P
► `IsSurjective( map )` P
► `IsSingleValued( map )` P
► `IsTotal( map )` P
► `IsBijective( map )` P
► `IsEndomorphism2dObject( map )` P
► `IsAutomorphism2dObject( map )` P

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

Constructors for morphisms of pre-crossed and crossed modules include:

3 ► `PreXModMorphism(` *args* `)` F
► `XModMorphism(` *args* `)` F
► `PreXModMorphismByHoms(` *P1, P2, sigma, rho* `)` O
► `XModMorphismByHoms(` *X1, X2, sigma, rho* `)` O
► `InclusionMorphism2dObjects(` *X1, S1* `)` O
► `InnerAutomorphismXMod(` *X1, r* `)` O
► `IdentityMapping(` *X1* `)` A
► `IsomorphismPermObject(` *obj* `)` F

In the following example we construct a simple automorphism of the crossed module `X1` constructed in the previous chapter.

```
gap> sigma1 := GroupHomomorphismByImages( c5, c5, [ (5,6,7,8,9) ]
        [ (5,9,8,7,6) ] );;
gap> rho1 := IdentityMapping( Range( X1 ) );
IdentityMapping( PAut(c5) )
gap> mor1 := XModMorphism( X1, X1, sigma1, rho1 );
[[c5->PAut(c5))] => [c5->PAut(c5))]]
gap> Display( mor1 );
Morphism of crossed modules :-
: Source = [c5->PAut(c5))] with generating sets:
  [ (5,6,7,8,9) ]
  [ (1,2,4,3) ]
: Range = Source
: Source Homomorphism maps source generators to:
  [ (5,9,8,7,6) ]
: Range Homomorphism maps range generators to:
  [ (1,2,4,3) ]
gap> IsAutomorphism2dObject( mor1 );
true
gap> Print( RepresentationsOfObject(mor1), "\n" );
[ "IsComponentObjectRep", "IsAttributeStoringRep", "Is2dMappingRep" ]
gap> Print( KnownPropertiesOfObject(mor1), "\n" );
[ "IsTotal", "IsSingleValued", "IsInjective", "IsSurjective", "Is2dMapping",
  "IsPreXModMorphism", "IsXModMorphism", "IsEndomorphism2dObject",
  "IsAutomorphism2dObject" ]
gap> Print( KnownAttributesOfObject(mor1), "\n" );
[ "Name", "Range", "Source", "SourceHom", "RangeHom" ]
```

3.2 Morphisms of pre-cat1-groups

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

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

1 ▶ `PreCat1Morphism(` *args* `)` F
▶ `Cat1Morphism(` *args* `)` F
▶ `PreCat1MorphismByHoms(` *P1, P2, gamma, rho* `)` O
▶ `Cat1MorphismByHoms(` *C1, C2, gamma, rho* `)` O
▶ `InclusionMorphism2dObjects(` *C1, S1* `)` O
▶ `InnerAutomorphismCat1(` *C1, r* `)` O
▶ `IdentityMapping(` *C1* `)` A
▶ `IsomorphismPermObject(` *obj* `)` F
▶ `SmallerDegreePerm2dObject(` *obj* `)` F

The global function `IsomorphismPermObject` calls `IsomorphismPermPreCat1` which constructs a morphism whose `SourceHom` and `RangeHom` are calculated using `IsomorphismPermGroup` on the source and range. Similarly `SmallerDegreePermutationRepresentation` is used on the two groups to obtain `SmallerDegreePerm2dObject`. Names are assigned automatically.

```
gap> iso2 := IsomorphismPermObject( C2 );
[[s3c4=>s3] => [Ps3c4=>Ps3]]
gap> Display( iso2 );
Morphism of cat1-groups :-
: Source = [s3c4=>s3] with generating sets:
  [ f1, f2, f3, f4 ]
  [ f1, f2 ]
:  Range = [Ps3c4=>Ps3] with generating sets:
  [ ( 5, 9)( 6,10)( 7,11)( 8,12), ( 1, 5, 9)( 2, 6,10)( 3, 7,11)( 4, 8,12),
  ( 1, 3, 2, 4)( 5, 7, 6, 8)( 9,11,10,12), ( 1, 2)( 3, 4)( 5, 6)( 7, 8)( 9,10)
    (11,12) ]
  [ (2,3), (1,2,3) ]
: Source Homomorphism maps source generators to:
  [ ( 5, 9)( 6,10)( 7,11)( 8,12), ( 1, 5, 9)( 2, 6,10)( 3, 7,11)( 4, 8,12),
  ( 1, 3, 2, 4)( 5, 7, 6, 8)( 9,11,10,12), ( 1, 2)( 3, 4)( 5, 6)( 7, 8)( 9,10)
    (11,12) ]
: Range Homomorphism maps range generators to:
  [ (2,3), (1,2,3) ]
```

3.3 Operations on morphisms

1 ▶ `Order(` *auto* `)` A
▶ `CompositionMorphism(` *map2, map1* `)` O

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

```
gap> Order( mor1 );
2
gap> GeneratorsOfGroup( d16 );
[ (11,12,13,14,15,16,17,18), (12,18)(13,17)(14,16) ]
gap> d8 := Subgroup( d16, [ c^2, d ] );;
gap> c4 := Subgroup( d8, [ c^2 ] );;
gap> SetName( d8, "d8" );  SetName( c4, "c4" );
gap> X16 := XModByNormalSubgroup( d16, d8 );
[d8->d16]
gap> X8 := XModByNormalSubgroup( d8, c4 );
[c4->d8]
```

```
gap> IsSubXMod( X16, X8 );
true
gap> inc8 := InclusionMorphism2dObjects( X16, X8 );
[[c4->d8] => [d8->d16]]
gap> rho := GroupHomomorphismByImages( d16, d16, [c,d], [c,d^(c^2)] );;
gap> sigma := GroupHomomorphismByImages( d8, d8, [c^2,d], [c^2,d^(c^2)] );;
gap> mor := XModMorphismByHoms( X16, X16, sigma, rho );
[[d8->d16] => [d8->d16]]
gap> comp := inc8 * mor;
[[c4->d8] => [d8->d16]]
gap> comp = CompositionMorphism( mor, inc8 );
true
```

2 ► `Kernel(` *map* `)` O

► `Kernel2dMapping(` *map* `)` A

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.

```
gap> c2 := Group( (19,20) );;
gap> i2 := Subgroup( c2, [()] );;
gap> X9 := XModByNormalSubgroup( c2, i2 );;
gap> sigma9 := GroupHomomorphismByImages( c4, i2, [c^2], [()] );;
gap> rho9 := GroupHomomorphismByImages( d8, c2, [c^2,d], [(),(19,20)] );;
gap> mor9 := XModMorphism( X8, X9, sigma9, rho9 );
[[c4->d8] => [..]]
gap> K9 := Kernel( mor9 );
[Group( [ (11,13,15,17)(12,14,16,18) ] )->Group( [ (11,13,15,17)(12,14,16,18)
 ] )]
```

4 Derivations and Sections

4.1 Whitehead Multiplication

1 ▶ `IsUp2dMapping(` *map* `)` P
▶ `IsDerivation(` *chi* `)` P
▶ `IsSection(` *chi* `)` P

The Whitehead monoid $\mathrm{Der}(\mathcal{X})$ 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 $R \to S$, with Whitehead multiplication $\star$ (on the **right**) satisfying:

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

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 $\mathrm{Der}(\mathcal{X})$. 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 $\mathcal{C} = (e; t, h : G \to R)$ is the set of group homomorphisms $\xi : R \to G$, with Whitehead multiplication $\star$, (on the **right**) satisfying:

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

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

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

$$\xi r \quad = \quad (er)(e\chi r) \quad = \quad (r, \chi r)$$

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

2 ▶ `Object2d(` *chi* `)` A
▶ `GeneratorImages(` *chi* `)` A
▶ `DerivationByImages(` *X0*`,` *ims* `)` O

Derivations are stored like group homomorphisms by specifying the images of a generating set. Images of the remaining elements may then be obtained using axiom **Der 1**. The function `IsDerivation` is automatically called to check that this procedure is well-defined.

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`.

```
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 := Cat1( t, h, e );
[g18=>s3]
gap> SetName( Kernel(t), "c3" );;
gap> X3 := XModOfCat1( C3 );;
gap> Display( X3 );
Crossed module [c3->s3] :-
: Source group has generators:
  [ ( 1, 2, 3)( 4, 6, 5) ]
: Range group has generators:
  [ (4,5,6), (2,3)(5,6) ]
: Boundary homomorphism maps source generators to:
  [ (4,6,5) ]
: Action homomorphism maps range generators to automorphisms:
  (4,5,6) --> { source gens --> [ (1,2,3)(4,6,5) ] }
  (2,3)(5,6) --> { source gens --> [ (1,3,2)(4,5,6) ] }
  These 2 automorphisms generate the group of automorphisms.
: associated cat1-group is [g18=>s3]

gap> imchi := [ (1,2,3)(4,6,5), (1,2,3)(4,6,5) ];;
gap> chi := DerivationByImages( X3, imchi );
DerivationByImages( s3, c3, [ (4,5,6), (2,3)(5,6) ],
[ (1,2,3)(4,6,5), (1,2,3)(4,6,5) ] )
```

3 ▶ `SectionByImages(` *C, ims* `)` O
▶ `SectionByDerivation(` *chi* `)` O
▶ `DerivationBySection(` *xi* `)` O

Sections **are** group homomorphisms, so do not need a special representation. Operations `SectionByDerivation` and `DerivationBySection` convert derivations to sections, and vice-versa, calling `Cat1OfXMod` and `XModOfCat1` automatically.

Two strategies for calculating derivations and sections are implemented, see [AW00]. The default method for `AllDerivations` 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 the GAP 3 version of this package, the default method for `AllSections(` C `)` was to compute all endomorphisms on the range group R of C as possibilities for the composite $h\xi$. A backtrack method then found possible images for such a section. In the current version the derivations of the associated crossed module are calculated, and these are all converted to sections using `SectionByDerivation`.

```
gap> xi := SectionByDerivation( chi );
[ (4,5,6), (2,3)(5,6) ] -> [ (1,2,3), (1,2)(4,6) ]
```

4.2 Whitehead Groups and Monoids

1 ► `RegularDerivations(` *X0* `)` A
► `AllDerivations(` *X0* `)` A
► `RegularSections(` *C0* `)` A
► `AllSections(` *C0* `)` A
► `ImagesList(` *obj* `)` A
► `ImagesTable(` *obj* `)` A

There are two functions to determine the elements of the Whitehead group and the Whitehead monoid of $\mathcal{X}_0$, namely `RegularDerivations` and `AllDerivations`. (The functions `RegularSections` and `AllSections` perform corresponding tasks for a cat1-group.)

Using our example `X3` we find that there are just nine derivations, six of them regular, and the associated group is isomorphic to `s3`.

```
gap> all3 := AllDerivations( X3 );;
gap> imall3 := ImagesList( all3 );; Display( imall3 );
[ [ (), () ],
  [ (), ( 1, 2, 3)( 4, 6, 5) ],
  [ (), ( 1, 3, 2)( 4, 5, 6) ],
  [ ( 1, 2, 3)( 4, 6, 5), () ],
  [ ( 1, 2, 3)( 4, 6, 5), ( 1, 2, 3)( 4, 6, 5) ],
  [ ( 1, 2, 3)( 4, 6, 5), ( 1, 3, 2)( 4, 5, 6) ],
  [ ( 1, 3, 2)( 4, 5, 6), () ],
  [ ( 1, 3, 2)( 4, 5, 6), ( 1, 2, 3)( 4, 6, 5) ],
  [ ( 1, 3, 2)( 4, 5, 6), ( 1, 3, 2)( 4, 5, 6) ]
  ]
gap> KnownAttributesOfObject( all3 );
[ "Object2d", "ImagesList", "AllOrRegular", "ImagesTable" ]
gap> Display( ImagesTable( all3 ) );
[ [  1,  1,  1,  1,  1,  1 ],
  [  1,  1,  1,  2,  2,  2 ],
  [  1,  1,  1,  3,  3,  3 ],
  [  1,  2,  3,  1,  2,  3 ],
  [  1,  2,  3,  2,  3,  1 ],
  [  1,  2,  3,  3,  1,  2 ],
  [  1,  3,  2,  1,  3,  2 ],
  [  1,  3,  2,  2,  1,  3 ],
  [  1,  3,  2,  3,  2,  1 ] ]
```

2 ► `CompositeDerivation(` *chi1* `,` *chi2* `)` O
► `ImagePositions(` *chi* `)` A

The Whitehead multiplication $\chi_1 \star \chi_2$ is implemented as `CompositeDerivation(` *chi1* `,` *chi2* `)`.

```
gap> reg3 := RegularDerivations( X3 );;
gap> imder3 := ImagesList( reg3 );;
gap> chi4 := DerivationByImages( X3, imder3[4] );
DerivationByImages( s3, c3, [ (4,5,6), (2,3)(5,6) ],
[ ( 1, 3, 2)( 4, 5, 6), () ] )
gap> chi5 := DerivationByImages( X3, imder3[5] );
DerivationByImages( s3, c3, [ (4,5,6), (2,3)(5,6) ],
[ ( 1, 3, 2)( 4, 5, 6), ( 1, 2, 3)( 4, 6, 5) ] )
gap> im4 := ImagePositions( chi4 );
```

```
[ 1, 3, 2, 1, 3, 2 ]
gap> im5 := ImagePositions( chi5 );
[ 1, 3, 2, 2, 1, 3 ]
gap> chi45 := chi4 * chi5;
DerivationByImages( s3, c3, [ (4,5,6), (2,3)(5,6) ],
[ (), ( 1, 2, 3)( 4, 6, 5) ] )
gap> im45 := ImagePositions( chi45 );
[ 1, 1, 1, 2, 2, 2 ]
gap> pos := Position( imder3, GeneratorImages( chi45 ) );
2
```

3 ► `WhiteheadGroupTable(` *X0* `)` A
► `WhiteheadMonoidTable(` *X0* `)` A
► `WhiteheadPermGroup(` *X0* `)` A
► `WhiteheadTransMonoid(` *X0* `)` A

Multiplication tables for the Whitehead group or monoid enable the construction of permutation or transformation representations.

```
gap> wgt3 := WhiteheadGroupTable( X3 );; Display( wgt3 );
[ [  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) ])
gap> wmt3 := WhiteheadMonoidTable( X3 );; Display( wmt3 );
[ [  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,  4,  4,  4,  4,  4,  4,  4,  4 ],
  [  5,  5,  5,  5,  5,  5,  5,  5,  5 ],
  [  6,  6,  6,  6,  6,  6,  6,  6,  6 ],
  [  7,  9,  8,  4,  6,  5,  1,  3,  2 ],
  [  8,  7,  9,  5,  4,  6,  2,  1,  3 ],
  [  9,  8,  7,  6,  5,  4,  3,  2,  1 ] ]
gap> wtm3 := WhiteheadTransMonoid( X3 );
Monoid( [ Transformation( [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ] ),
  Transformation( [ 2, 3, 1, 5, 6, 4, 8, 9, 7 ] ),
  Transformation( [ 3, 1, 2, 6, 4, 5, 9, 7, 8 ] ),
  Transformation( [ 4, 4, 4, 4, 4, 4, 4, 4, 4 ] ),
  Transformation( [ 5, 5, 5, 5, 5, 5, 5, 5, 5 ] ),
  Transformation( [ 6, 6, 6, 6, 6, 6, 6, 6, 6 ] ),
  Transformation( [ 7, 9, 8, 4, 6, 5, 1, 3, 2 ] ),
  Transformation( [ 8, 7, 9, 5, 4, 6, 2, 1, 3 ] ),
  Transformation( [ 9, 8, 7, 6, 5, 4, 3, 2, 1 ] ) ], ... )
```

5 Actors of 2d-objects

5.1 Actor of a Crossed Module

The **actor** of $\mathcal{X}$ is a crossed module $(\Delta : \mathcal{W}(\mathcal{X}) \to \mathrm{Aut}(\mathcal{X}))$ 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 W of the Whitehead group of regular derivations, and the range is a permutation representation A of the automorphism group $\mathrm{Aut}(\mathcal{X})$ of $\mathcal{X}$.

1 ▶ `AutomorphismPermGroup(` *xmod* `)` A
▶ `WhiteheadXMod(` *xmod* `)` A
▶ `LueXMod(` *xmod* `)` A
▶ `NorrieXMod(` *xmod* `)` A
▶ `ActorXMod(` *xmod* `)` A

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

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

2 ▶ `InnerActor(` *xmod* `)` A
▶ `InnerMorphism(` *xmod* `)` A
▶ `Centre(` *xmod* `)` A

Pairs of boundaries or identity mappings provide six morphisms of crossed modules. In particular, the boundaries of $\mathcal{W}(\mathcal{X})$ 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, using the crossed module `(X3 : c3 -> s3)` from the previous chapter, the inner morphism is an inclusion of crossed modules.

```
gap> X3;
[c3->s3]]
gap> WGX3 := WhiteheadPermGroup( X3 );
Group( [ (1,2,3)(4,5,6), (1,4)(2,6)(3,5) ] )
gap> APX3 := AutomorphismPermGroup( X3 );
Group( [ (3,4,5), (1,2)(4,5) ] )
gap> WX3 := WhiteheadXMod( X3 );;  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:
  [ (1,3,2)(4,6,5) ]
: 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 );
Lue[c3->s3]
gap> NX3 := NorrieXMod( X3 );
Norrie[c3->s3]
gap> AX3 := ActorXMod( X3 );;  Display( AX3);
Crossed module Actor[c3->s3] :-
: Source group has generators:
  [ (1,2,3)(4,5,6), (1,4)(2,6)(3,5) ]
: Range group has generators:
  [ (3,4,5), (1,2)(4,5) ]
: Boundary homomorphism maps source generators to:
  [ (3,5,4), (1,2)(4,5) ]
: Action homomorphism maps range generators to automorphisms:
  (3,4,5) --> { source gens --> [ (1,2,3)(4,5,6), (1,5)(2,4)(3,6) ] }
  (1,2)(4,5) --> { source gens --> [ (1,3,2)(4,6,5), (1,4)(2,6)(3,5) ] }
  These 2 automorphisms generate the group of automorphisms.
gap> IAX3 := InnerActorXMod( X3 );;  Display( IAX3 );
Crossed module InnerActor[c3->s3] :-
: Source group has generators:
  [ (1,3,2)(4,6,5) ]
: Range group has generators:
  [ (3,5,4), (1,2)(4,5) ]
: Boundary homomorphism maps source generators to:
  [ (3,4,5) ]
: Action homomorphism maps range generators to automorphisms:
  (3,5,4) --> { source gens --> [ (1,3,2)(4,6,5) ] }
  (1,2)(4,5) --> { source gens --> [ (1,2,3)(4,5,6) ] }
  These 2 automorphisms generate the group of automorphisms.
gap> IMX3 := InnerMorphism( X3 );; 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) ]
  [ (3,4,5), (1,2)(4,5) ]
: Source Homomorphism maps source generators to:
  [ (1,3,2)(4,6,5) ]
: Range Homomorphism maps range generators to:
  [ (3,5,4), (1,2)(4,5) ]
gap> Centre( X3 );
[Group( () )->Group( () )]
```

6 Induced Constructions

6.1 Induced crossed modules

1 ► `IsInducedXMod( xmod )` P
► `IsInducedCat1( cat1 )` P
► `InducedXMod( args )` F
► `InducedCat1( args )` F
► `MorphismOfInducedXMod( xmod )` A

A morphism of crossed modules $(\sigma, \rho) : \mathcal{X}_1 \to \mathcal{X}_2$ factors uniquely through an induced crossed module $\rho_*\mathcal{X}_1 = (\delta : \rho_* S_1 \to R_2)$. Similarly, a morphism of cat1-groups factors through an induced cat1-group. Calculation of induced crossed modules of $\mathcal{X}$ 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 **XMod** and **Cat1**.

Data for the cases of algebraic interest is provided by a conjugation crossed module $\mathcal{X} = (\partial : S \to R)$ and a homomorphism ι from R to a third group 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 $\mathcal{X} \to \iota_*\mathcal{X}$. When ι is a surjection with kernel K then $\iota_* S = [S, K]$ (see [BH78]). When ι is an inclusion the induced crossed module may be calculated using a copower construction [BW95] or, in the case when R is normal in Q, as a coproduct of crossed modules ([BW96], but not yet implemented). When ι is neither a surjection nor an inclusion, ι is written 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 `a4` in `s4`, and for ι the surjection from `s4` to `s3` with kernel `k4`. The induced crossed module is isomorphic to `X3`.

```
gap> s4gens := [ (1,2), (2,3), (3,4) ];;
gap> s4 := Group( s4gens );; SetName(s4,"s4");
gap> a4gens := [ (1,2,3), (2,3,4) ];;
gap> a4 := Subgroup( s4, a4gens );;  SetName( a4, "a4" );
gap> s3 := Group( (5,6),(6,7) );;  SetName( s3, "s3" );
gap> epi := GroupHomomorphismByImages( s4, s3, s4gens, [(5,6),(6,7),(5,6)] );;
gap> X4 := XModByNormalSubgroup( s4, a4 );;
gap> indX4 := SurjectiveInducedXMod( X4, epi );
[a4/ker->s3]
gap> morX4 := MorphismOfInducedXMod( indX4 );
[[a4->s4] => [a4/ker->s3]]
```

For a second, injective example we take for $\mathcal{X}$ the conjugation crossed module $(\partial : c4 \to d8)$ of Chapter 3, and for ι the inclusion `incd8` of `d8` in `d16`. The induced crossed module has $c4 \times c4$ as source.

```
gap> incd8 := RangeHom( inc8 );;
gap> [ Source(incd8), Range(incd8), IsInjective(incd8) ];
[ d8, d16, true ]
gap> indX8 := InducedXMod( X8, incd8 );
#I Simplified presentation for induced group :-
<presentation with 2 gens and 3 rels of total length 12>
#I  generators: [ f11, f14 ]
#I  relators:
#I  1.  4  [ 1, 1, 1, 1 ]
#I  2.  4  [ 2, 2, 2, 2 ]
#I  3.  4  [ 2, -1, -2, 1 ]
#I induced group has Size: 16
#I factor 1 is abelian  with invariants: [ 4, 4 ]
i*([c4->d8])
gap> Display( indX8 );
Crossed module i*([c4->d8]) :-
: Source group has generators:
  [ ( 1, 2, 6, 3)( 4, 7,12, 9)( 5, 8,13,10)(11,14,16,15),
  ( 1, 4,11, 5)( 2, 7,14, 8)( 3, 9,15,10)( 6,12,16,13) ]
: Range group d16 has generators:
  [ (11,12,13,14,15,16,17,18), (12,18)(13,17)(14,16) ]
: Boundary homomorphism maps source generators to:
  [ (11,13,15,17)(12,14,16,18), (11,17,15,13)(12,18,16,14) ]
: Action homomorphism maps range generators to automorphisms:
  (11,12,13,14,15,16,17,18) --> { source gens -->
[ ( 1, 5,11, 4)( 2, 8,14, 7)( 3,10,15, 9)( 6,13,16,12),
  ( 1, 3, 6, 2)( 4, 9,12, 7)( 5,10,13, 8)(11,15,16,14) ] }
  (12,18)(13,17)(14,16) --> { source gens -->
[ ( 1, 3, 6, 2)( 4, 9,12, 7)( 5,10,13, 8)(11,15,16,14),
  ( 1, 5,11, 4)( 2, 8,14, 7)( 3,10,15, 9)( 6,13,16,12) ] }
  These 2 automorphisms generate the group of automorphisms.
gap> morX8 := MorphismOfInducedXMod( indX8 );
[[c4->d8] => i*([c4->d8])]
gap> Display( morX8 );
Morphism of crossed modules :-
: Source = [c4->d8] with generating sets:
  [ (11,13,15,17)(12,14,16,18) ]
  [ (11,13,15,17)(12,14,16,18), (12,18)(13,17)(14,16) ]
:  Range = i*([c4->d8]) with generating sets:
  [ ( 1, 2, 6, 3)( 4, 7,12, 9)( 5, 8,13,10)(11,14,16,15),
  ( 1, 4,11, 5)( 2, 7,14, 8)( 3, 9,15,10)( 6,12,16,13) ]
  [ (11,12,13,14,15,16,17,18), (12,18)(13,17)(14,16) ]
: Source Homomorphism maps source generators to:
  [ ( 1, 2, 6, 3)( 4, 7,12, 9)( 5, 8,13,10)(11,14,16,15) ]
: Range Homomorphism maps range generators to:
  [ (11,13,15,17)(12,14,16,18), (12,18)(13,17)(14,16) ]
```

For a third example we take the identity mapping on `s3` as boundary, and the inclusion of `s3` in `s4` as `iota`. The induced group is a general linear group `GL(2,3)`.

```
gap> s3b := Subgroup( s4, [ (2,3), (3,4) ] );;  SetName( s3b, "s3b" );
gap> indX3 := InducedXMod( s4, s3b, s3b );
#I Simplified presentation for induced group :-
<presentation with 2 gens and 4 rels of total length 33>
#I  generators: [ f11, f112 ]
#I  relators:
#I  1.  2  [ 1, 1 ]
#I  2.  3  [ 2, 2, 2 ]
#I  3.  12  [ 1, -2, 1, 2, 1, 2, 1, -2, 1, 2, 1, 2 ]
#I  4.  16  [ -2, 1, -2, 1, -2, 1, -2, 1, -2, 1, -2, 1, -2, 1, -2, 1 ]
#I induced group has Size: 48
#I IdGroup = [ [  48,  29 ] ]
i*([s3b->s3b])
gap> isoX3 := IsomorphismGroups( Source( indX3 ), GeneralLinearGroup(2,3) );
[ (1,2)(4,5)(6,8), (2,3,4)(5,6,7) ] ->
[ [ [ Z(3)^0, 0*Z(3) ], [ Z(3), Z(3) ] ],
  [ [ Z(3)^0, Z(3)^0 ], [ 0*Z(3), Z(3)^0 ] ] ]
```

2 ► `AllInducedXMods(` Q `)` O

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

7 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.

7.1 Inclusion and Restriction Mappings

1 ► `InclusionMappingGroups(` *G*, *H* `)` O
► `RestrictionMappingGroups(` *hom*, *src*, *rng* `)` O
► `MappingToOne(` *G*, *H* `)` O

The first set of utilities concerns inclusion and restriction mappings. Restriction may apply to both the source and the range of the map. The map `incd8` is the inclusion of `d8` in `d16` used in Section 3.3.

```
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> resd8 := RestrictionMappingGroups( incd8, c4, imd8 );;
gap> Source( res8 );  Range( res8 );
c4
Group([ (11,13,15,17)(12,14,16,18), (11,18)(12,17)(13,16)(14,15) ])
gap> MappingToOne( c4, imd8 );
[ (11,13,15,17)(12,14,16,18) ] -> [ () ]
```

7.2 Endomorphism Classes and Automorphisms

1 ► `EndomorphismClasses(` *grp*, *case* `)` F
► `EndoClassNaturalHom(` *class* `)` A
► `EndoClassIsomorphism(` *class* `)` A
► `EndoClassConjugators(` *class* `)` A
► `AutoGroup(` *class* `)` A

The monoid of endomorphisms of a group is used when calculating the monoid of derivations of a crossed module and when determining all the cat1-structures on a group.

An endomorphism ϵ of R with image H' is determined by

- a normal subgroup N of R and a permutation representation $\theta : R/N \to Q$ of the quotient, giving a projection $\theta \circ \nu : R \to Q$, where $\nu : R \to R/N$ is the natural homomorphism;
- an automorphism α of Q;
- a subgroup H' in a conjugacy class $[H]$ of subgroups of R isomorphic to Q having representative H, an isomorphism $\phi : Q \cong H$, and a conjugating element $c \in R$ such that $H^c = H'$.

Then ϵ takes values

$$\epsilon r \;=\; (\phi\alpha\theta\nu\, r)^c \;.$$

Endomorphisms are placed in the same class if they have the same choice of N and $[H]$, and so the number of endomorphisms is

$$|\mathrm{End}(R)| \;=\; \sum_{\text{classes}} |\mathrm{Aut}(Q)| \cdot |[H]| \;.$$

The function `EndomorphismClasses(` *grp*`,` *case* `)` may be called in three ways:

- case 1 includes automorphisms and the zero map,
- case 2 excludes automorphisms and the zero map,
- case 3 is when `N` intersects `H` trivially.

```
gap> end8 := EndomorphismClasses( d8, 1 );;
gap> Length( end8 );
13
gap> e4 := end8[4];
<enumerator>
gap> EndoClassNaturalHom( e4 );
GroupHomomorphismByImages( d8, Group( [ f1 ] ),
[ (11,13,15,17)(12,14,16,18), (12,18)(13,17)(14,16) ], [ f1, f1 ] )
gap> EndoClassIsomorphism( e4 );
Pcgs([ f1 ]) -> [ (11,13)(14,18)(15,17) ]
gap> EndoClassConjugators( e4 );
[ (), (12,18)(13,17)(14,16) ]
gap> AutoGroup( e4 );
Group( [ Pcgs([ f1 ]) -> [ f1 ] ] )
gap> L := List( end8, e -> Length(EndoClassConjugators(e)) * Size(AutoGroup(e)) );
[ 8, 1, 2, 2, 1, 2, 2, 1, 2, 2, 6, 6, 1 ]
gap> Sum( L );
36
```

2 ► `InnerAutomorphismsByNormalSubgroup(` *G*`,` *N* `)` O

► `IsGroupOfAutomorphisms(` *A* `)` P

Inner automorphisms of a group `G` by the elements of a normal subgroup `N` are calculated with the first of these functions, usually with `G` = `N`.

```
gap> autd8 := AutomorphismGroup( d8 );;
gap> innd8 := InnerAutomorphismsByNormalSubgroup( d8, d8 );;
gap> GeneratorsOfGroup( innd8 );
[ InnerAutomorphism( d8, (11,13,15,17)(12,14,16,18) ),
  InnerAutomorphism( d8, (12,18)(13,17)(14,16) ) ]
gap> IsGroupOfAutomorphisms( innd8 );
true
```

7.3 Abelian Modules

1 ► `IsAbelianModule(` *obj* `)` P
► `AbelianModuleGroup(` *obj* `)` A
► `AbelianModuleAction(` *obj* `)` A
► `AbelianModuleObject(` *grp*, *act* `)` O

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

```
gap> x := (6,7)(8,9);;  y := (6,8)(7,9);;  z := (6,9)(7,8);;
gap> k4 := Group( x, y );  SetName( k4, "k4" );
gap> s3 := Group( (1,2), (2,3) );;  SetName( s3, "s3" );
gap> alpha := GroupHomomorphismByImages( k4, k4, [x,y], [y,x] );
gap> beta := GroupHomomorphismByImages( k4, k4, [x,y], [x,z] );
gap> aut := Group( alpha, beta );
gap> act := GroupHomomorphismByImages( s3, aut, [(1,2),(2,3)], [alpha,beta] );
gap> abmod := AbelianModuleObject( k4, act );
<enumerator>
gap> Xabmod := XModByAbelianModule( abmod );
[k4->s3]
```

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

7.4 Distinct and Common Representatives

1 ► `DistinctRepresentatives(` *list* `)` O
► `CommonRepresentatives(` *list1*, *list2* `)` O
► `CommonTransversal(` *grp*, *subgrp* `)` O
► `IsCommonTransversal(` *grp*, *subgrp*, *list* `)` O

The final set of utilities deal with lists of subsets of $[1 \ldots n]$ and construct systems of distinct and common representatives using simple, non-recursive, combinatorial algorithms.

When L is a set of n subsets of $[1 \ldots n]$ and the Hall condition is satisfied (the union of any k subsets has at least k elements), a set of distinct representatives exists.

When J, K are both lists of n sets, the function `CommonRepresentatives` returns two lists: the set of representatives, and a permutation of the subsets of the second list. It may also be used to provide a common transversal for sets of left and right cosets of a subgroup H of a group G, although a greedy algorithm is usually quicker.

```
gap> J := [ [1,2,3], [3,4], [3,4], [1,2,4] ];;
gap> DistinctRepresentatives( J );
[ 1, 3, 4, 2 ]
gap> K := [ [3,4], [1,2], [2,3], [2,3,4] ];;
gap> CommonRepresentatives( J, K );
[ [ 3, 3, 3, 1 ], [ 1, 3, 4, 2 ] ]
gap> CommonTransversal( d16, c4 );
[ (), (12,18)(13,17)(14,16), (11,12,13,14,15,16,17,18),
  (11,12)(13,18)(14,17)(15,16) ]
gap> IsCommonTransversal( d16, c4, [ (), c, d, c*d ] );
true
```

8 Development History

This chapter, which is intended to contain 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.

Version 1

The inspiration for this package was the need, in the mid-1990's, to calculate induced crossed modules (see [BW1,BW2,BW3]). 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.)

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

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 names `2dObject` for (pre-)crossed modules and (pre-)cat1-groups, and `2dMapping` for the various types of morphism. This allows `3dObject` to be used in future 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.

Version 2.001

This was the first version of XMod for GAP 4, completed in April 2002 in a rush to catch 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`. During the ten days prior to the release, the main changes made were:

- Generic name `UpMapping` chosen for derivations and sections (now changed to `Up2dMapping`).
- File names changed to `obj2.gd`, `map2.gi`, `up2.tex`, etc.
- Added alternative methods for `IsomorphismPermGroup` for `2dObjects`. (Strange terminology here! Will probably define global functions `IsomorphismPermObject`, `IsomorphismFpObject`, etc. in a later version which will call `IsomorphismPermGroup`, `IsomorphismPerm2dObject` or whatever is appropriate.)
- Sorted a problem with fixing the generating set for R when used to define derivations. The (old) code uses an fp-group version of R and checks that all the relators map by *chi* to *1*. Unfortunately, `IsomorphismFpGroup` sometimes permutes the order of the R-generators, with unfortunate effects. The fix is to use `IsomorphismFpGroupByGenerators` which returns the images of the generators specified in the function call. We have also used `genR := StrongGeneratorsStabChain( StabChain( rng ) );` throughout to specify the generators of R.
- Operation `XModMorphism` renamed as `XModMorphismByHoms`, and a new global function `XModMorphism` introduced (and ditto for other `2dMappings`).

- Now using $\chi_1 \star \chi_2$ for Whitehead multiplication **on the right**, with CompositeDerivation still giving multiplication **on the left**. This means that the second axiom for derivations and for sections has changed – see Chapter 4.

During the period May 20th - May 27th 2002 converted induce.g to induce.gd and induce.gi (later renamed induce2.gd, induce2.gi), at least as regards induced crossed modules — induced cat1-groups still to come. This involved the following.

- Converted combinatorial functions – DistinctRepresentatives, CommonRepresentatives, CommonTransversal and IsCommonTransversal.
- Converted Tietze modification functions TzCommutatorPair, TzPartition and FactorsPresentation.
- As suggested above, introduced global functions IsomorphismPermObject, IsomorphismFpObject, IsomorphismPcObject which call IsomorphismPermGroup etc. when the object is a group. Added functions IsomorphismPermPreXMod, IsomorphismPermPreCat1, etc. to be called when the object is a 2d-object.
- Added IsomorphismXModByNormalSubgroup which applies when the boundary of the xmod is injective.
- Added PreXModIsomorphismByIsomorphisms (a similar PreCat1IsomorphismByIsomorphisms will be needed) where the data consists of an xmod, an isomorphism of the source, and an isomorphism of the range.
- Changed RModule to AbelianModule.

8.1 Versions 2.002 – 2.005

Version 2.002 was prepared for the 4.4 release at the end of January 2004, and so required a **PackageInfo.g** file.

Version 2.003 of February 28th 2004 just fixed some file protections.

Version 2.004 of April 14th 2004 gave a new email address for Murat Alp and added the Cat1Select functionality of version 1 to the Cat1 function (the loading mechanism was revised in version 2.006).

Version 2.005 of April 16th 2004 moved the example files from **tst/test_i.g** to **examples/example_i.g**, and converted **testmanual.g** to a proper test file **tst/xmod_manual.tst**.

Changes made include the following.

- Replaced OperationHomomorphism by ActionHomomorphism – a general GAP 4.4 change.
- Finished replacing RModule by AbelianModule.
- Renamed UpMapping as Up2dMapping.
- Added MappingGeneratorsImages and InverseGeneralMapping for a 2dMapping.
- Converted the actor crossed module functions from the 3.4.4 version, including AutomorphismPermGroup for a crossed module, WhiteheadXMod, NorrieXMod, LueXMod, ActorXMod, Centre of a crossed module, InnerMorphism and InnerActorXMod.
- Added SmallerDegreePermPreXMod after discovering SmallerDegreePermutationRepresentation in the library.

8.2 Version 2.006

This version contains changes made between May 4th and September 6th 2004.

- Changed morphism functions to return fail when invalid data is supplied, rather than calling Error.
- Fixed a bug in XmodByGroupOfAutomorphisms.

8.3 What needs doing next?

- Speed up the calculation of Whitehead groups.
- Complete the conversion from Version 1 of the calculation of sections using `EndoClasses`.
- Add basic functions for `3dObjects`: the actor of a crossed module is a typical example of a crossed square.
- Add interaction with IdRel, XRes, `ntp`.
- Need `InverseGeneralMapping` for morphisms.
- Need more features for `FpXMods`, `PcXMods`, etc.
- Implement actions of a crossed module.
- Implement `FreeXMods`.
- Implement an operation `Isomorphism2dObjects`.

Bibliography

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

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

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

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

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

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

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

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

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

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

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

[Nor87] K. J. Norrie. *Crossed modules and analogues of group theorems.* Ph.D. thesis, King's College, University of London, 1987.

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

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

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

Index

This index covers only this manual. A page number in *italics* refers to a whole section which is devoted to the indexed subject. Keywords are sorted with case and spaces ignored, e.g., "PermutationCharacter" comes before "permutation group".

V

W

X