XMod

Crossed modules and cat1-groups in GAP

Version 2.31

17/12/2014

Murat Alp
Chris Wensley

Murat Alp Email: muratalp@nigde.edu.tr
Address: Nigde Üniversitesi,
Fen Edebiyat Fakültesi, Matematik Bölümü
Nigde, Turkey.

Chris Wensley Email: c.d.wensley@bangor.ac.uk
Homepage: http://www.bangor.ac.uk/~mas023/
Address: School of Computer Science, Bangor University,
Dean Street, Bangor, Gwynedd, LL57 1UT, U.K.

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;
- derivations of crossed modules and sections of cat1-groups;
- the actor crossed square of a crossed module.

Initial, experimental versions of the following are also included in this version:

- crossed squares and their morphisms;
- crossed modules of groupoids.

XMod was originally implemented in 1997 using the GAP3 language. when the first author was studying for a Ph.D. [Alp97] in 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.

Version 2.19 was released on 9th June 2012 for the first release of GAP 4.5.

The current version is 2.31, released 17th December 2014 for GAP 4.7.

Bug reports, suggestions and comments are, of course, welcome. Please contact the second author at `c.d.wensley@bangor.ac.uk`.

Copyright

Acknowledgements

This documentation was prepared with the GAPDoc package of Frank Lübeck and Max Neunhöffer.

The first author wishes to acknowledge support from Dumlupinar University and the Turkish government.

Contents

Chapter 1

Introduction

The XMod package provides functions for computation with

- finite crossed modules of groupsand 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;
- the actor crossed square of a crossed module;
- crossed squares and their morphisms (experimental version);
- crossed modules of groupoids (experimental version).

It is loaded with the command

Example

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

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

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

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

The current versions are 2.22 for GAP 4.6, released on January 25th 2013; and 2.31 for GAP 4.7, released on 17th December 2014.

The package may be obtained as a compressed tar file `xmod-2.231.tar.gz` by ftp from one of the sites with a GAP 4 archive, or from the Bangor Mathematics website pages using the URL: `http://www.maths.bangor.ac.uk/chda/gap4/xmod/xmod231.html`

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

The package divides into four 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 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. For the actor crossed module Act($\mathscr{X}$) of a crossed module $\mathscr{X}$ we require representations for the Whitehead group of regular derivations of $\mathscr{X}$ and for the group of automorphisms of $\mathscr{X}$. The construction also provides an inner morphism from $\mathscr{X}$ to Act($\mathscr{X}$) whose kernel is the centre of $\mathscr{X}$.

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.

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). Cat2-groups, and conversion between these and crossed squares, are also planned.

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 Gpd [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, it is possible to check the correct installation by running the test suite of the package with the following command.

Example
```
gap> ReadPackage( "xmod", "tst/testall.g" );
+ Testing examples in Chapter 2 of the XMod manual
...
```

Additional information can be found on the *Computational Higher-dimensional Discrete Algebra* website at `http://www.maths.bangor.ac.uk/chda/`.

Chapter 2

2d-groups : crossed modules and cat1-groups

2.1 Constructions for crossed modules

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

$$\textbf{XMod 1} \ : \ \partial(s^r) = r^{-1}(\partial s)r = (\partial s)^r, \qquad \textbf{XMod 2} \ : \ s_1^{\partial s_2} = s_2^{-1}s_1s_2 = {s_1}^{s_2}.$$

The kernel of ∂ is abelian.

There are a variety of constructors for crossed modules:

2.1.1 XMod

▷ `XMod(`*`args`*`)` (function)

▷ `XModByBoundaryAndAction(`*`bdy, act`*`)` (operation)

▷ `XModByTrivialAction(`*`bdy`*`)` (operation)

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

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

▷ `XModByAutomorphismGroup(`*`grp`*`)` (operation)

▷ `XModByInnerAutomorphismGroup(`*`grp`*`)` (operation)

▷ `XModByGroupOfAutomorphisms(`*`G, A`*`)` (operation)

▷ `XModByAbelianModule(`*`abgrp`*`)` (operation)

▷ `DirectProduct(`*`X1, X2`*`)` (operation)

Here are the standard constructions which these implement:

- 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 *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 *crossed abelian module* has an abelian module as source and the zero map as boundary.
- The direct product $\mathscr{X}_1 \times \mathscr{X}_2$ of two crossed modules has source $S_1 \times S_2$, range $R_1 \times R_2$ and boundary $\partial_1 \times \partial_2$, with R_1, R_2 acting trivially on S_2, S_1 respectively.

2.1.2 Source

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

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

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

2.1.3 Size

▷ `Size(X0)` (attribute)
▷ `Name(X0)` (attribute)

More familiar attributes are `Size` and `Name`, the latter formed by concatenating the names of the source and range (if these exist). An `Enumerator` function has not been implemented. The `Display` function is used to print details of 2d-groups.

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

Example
```
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:
  [ (1,2,3,4) ]
: Boundary homomorphism maps source generators to:
  [ () ]
: Action homomorphism maps range generators to automorphisms:
  (1,2,3,4) --> { 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" );
[ "IsEmpty", "IsTrivial", "IsNonTrivial", "IsFinite",
  "CanEasilyCompareElements", "CanEasilySortElements", "IsDuplicateFree",
  "IsPerm2dDomain", "IsPreXMod", "IsXMod", "IsAutomorphismGroup2dDomain" ]
gap> Print( KnownAttributesOfObject(X1), "\n" );
[ "Name", "Size", "Range", "Source", "Boundary", "AutoGroup", "XModAction" ]
```

2.1.4 SubXMod

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

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

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

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

2.2 Pre-crossed modules

2.2.1 PreXModByBoundaryAndAction

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

When axiom {\bf XMod\ 2} is *not* satisfied, the corresponding structure is known as a *pre-crossed module*.

Example

```
gap> c := (11,12,13,14,15,16,17,18);;  d := (12,18)(13,17)(14,16);;
gap> d16 := Group( c, d );;
gap> sk4 := Subgroup( d16, [ c^4, d ] );;
gap> SetName( d16, "d16" );  SetName( sk4, "sk4" );
gap> bdy16 := GroupHomomorphismByImages( d16, sk4, [c,d], [c^4,d] );;
gap> h1 := GroupHomomorphismByImages( d16, d16, [c,d], [c^5,d] );;
gap> h2 := GroupHomomorphismByImages( d16, d16, [c,d], [c,c^4*d] );;
gap> aut16 := Group( [ h1, h2 ] );;
gap> act16 := GroupHomomorphismByImages( sk4, aut16, [c^4,d], [h1,h2] );;
gap> P16 := PreXModByBoundaryAndAction( bdy16 );
[d16->sk4]
```

2.2.2 PeifferSubgroup

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

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

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

Example
```
gap> P := PeifferSubgroup( P16 );
Group( [ (11,15)(12,16)(13,17)(14,18), (11,17,15,13)(12,18,16,14) ] )
gap> X16 := XModByPeifferQuotient( P16 );
[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([ (1,2), (3,4) ])
```

2.2.3 IsPermXMod

▷ `IsPermXMod(X0)` (property)
▷ `IsPcPreXMod(X0)` (property)

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

2.3 Cat1-groups and pre-cat1-groups

2.3.1 Source

▷ Source(*C*) (attribute)
▷ Range(*C*) (attribute)
▷ TailMap(*C*) (attribute)
▷ HeadMap(*C*) (attribute)
▷ RangeEmbedding(*C*) (attribute)
▷ KernelEmbedding(*C*) (attribute)
▷ Boundary(*C*) (attribute)
▷ Name(*C*) (attribute)
▷ Size(*C*) (attribute)

These are the attributes of a cat1-group $\mathcal{C}$ in this implementation.

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:

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

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`.

2.3.2 Cat1

▷ Cat1(*args*) (attribute)
▷ PreCat1ByTailHeadEmbedding(*t, h, e*) (attribute)
▷ PreCat1ByEndomorphisms(*t, h*) (attribute)
▷ PreCat1ByNormalSubgroup(*G, N*) (attribute)
▷ Cat1ByPeifferQuotient(*P*) (attribute)
▷ Reverse(*C0*) (attribute)

These are some of the constructors for pre-cat1-groups and cat1-groups.

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

Example
```
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, &lt;identity&gt; of ..., &lt;identity&gt; 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 s3c4 has generators:
  [ f1, f2, f3, f4 ]
: Range group s3 has generators:
  [ f1, f2 ]
: tail homomorphism maps source generators to:
  [ f1, f2, &lt;identity&gt; of ..., &lt;identity&gt; of ... ]
: head homomorphism maps source generators to:
  [ f1, f2, &lt;identity&gt; of ..., &lt;identity&gt; of ... ]
: range embedding maps range generators to:
  [ f1, f2 ]
: kernel has generators:
  [ f3, f4 ]
: boundary homomorphism maps generators of kernel to:
  [ &lt;identity&gt; of ..., &lt;identity&gt; of ... ]
: kernel embedding maps generators of kernel to:
  [ f3, f4 ]
: associated crossed module is [ker(t2)->s3]
gap> IsPcCat1( C2 );
true
gap> Size( C2 );
[ 24, 6 ]
```

2.3.3 Cat1OfXMod

▷ `Cat1OfXMod(X0)` (attribute)
▷ `XModOfCat1(C0)` (attribute)
▷ `PreCat1OfPreXMod(P0)` (attribute)
▷ `PreXModOfPreCat1(P0)` (attribute)

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 $\mathscr{X} = (\partial : S \to R)$ the group G is defined as the semidirect product $G = R \ltimes S$ using the action from $\mathscr{X}$, with multiplication rule

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

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 $\mathscr{C} = (e;t,h : G \to R)$, 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.

Example

```
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:
  [ &lt;identity&gt; of ..., &lt;identity&gt; of ... ]
  The automorphism group is trivial
: associated cat1-group is [s3c4=>s3]
```

2.4 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 70 (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 := 70` and `CAT1_LIST_CLASS_SIZES` are also stored. The data is read into the list `CAT1_LIST` only when this function is called.

2.4.1 Cat1Select

▷ `Cat1Select(`*`size, gpnum, num`*`)` (attribute)

The function `Cat1Select` may be used in three ways. `Cat1Select( size )` returns the names of the groups with this size, while `Cat1Select( size, gpnum )` prints a list of cat1-structures for this chosen group. `Cat1Select( size, gpnum, num )` (or just `Cat1( size, gpnum, num )`) returns the chosen cat1-group.

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

Example

```
gap> ## check the number of groups of size 18
gap> L18 := Cat1Select( 18 );
Usage:  Cat1Select( size, gpnum, num );
[ "D18", "C18", "C3 x S3", "(C3 x C3) : C2", "C6 x C3" ]
gap> ## check the number of cat1-structrues on the fourth of these
gap> Cat1Select( 18, 4 );
Usage:  Cat1Select( size, gpnum, num );
There are 4 cat1-structures for the group (C3 x C3) : C2.
[ [range gens], [tail genimages], [head genimages] ] :-
(1)  [ [ f1 ], [ f1, <identity> of ..., <identity> of ... ],
  [ f1, <identity> of ..., <identity> of ... ] ]
```

```
(2)  [ [ f1, f3 ], [ f1, <identity> of ..., f3 ],
  [ f1, <identity> of ..., f3 ] ]
(3)  [ [ f1, f3 ], [ f1, <identity> of ..., f3 ],
  [ f1, f3^2, <identity> of ... ] ]
(4)  [ [ f1, f2, f2*f3 ],  tail = head = identity mapping ]
4
gap> ## select the third of these cat1-structures
gap> C18 := Cat1( 18, 4, 3 );
[(C3 x C3) : C2=>Group( [ f1, <identity> of ..., f3 ] )]
gap> ## convert from a pc-cat1-group to a permutation cat1-group
gap> iso18 := IsomorphismPermObject( C18 );;
gap> PC18 := Image( iso18 );;
gap> Display( PC18 );

Cat1-group [..=>..] :-
: Source group has generators:
  [ (2,3)(5,6), (4,5,6), (1,2,3) ]
: Range group has generators:
  [ (2,3), (), (1,2,3) ]
: tail homomorphism maps source generators to:
  [ (2,3), (), (1,2,3) ]
: head homomorphism maps source generators to:
  [ (2,3), (1,3,2), (1,2,3) ]
: range embedding maps range generators to:
  [ (2,3)(5,6), (), (1,2,3) ]
: kernel has generators:
  [ (4,5,6) ]
: boundary homomorphism maps generators of kernel to:
  [ (1,3,2) ]
: kernel embedding maps generators of kernel to:
  [ (4,5,6) ]

gap> convert the result to the associated permutation crossed module
gap> X18 := XModByCat1( PC18 );;
gap> Display( X18 );

Crossed module X([..=>..]) :-
: Source group has generators:
  [ (4,5,6) ]
: Range group has generators:
  [ (2,3), (), (1,2,3) ]
: Boundary homomorphism maps source generators to:
  [ (1,3,2) ]
: Action homomorphism maps range generators to automorphisms:
  (2,3) --> { source gens --> [ (4,6,5) ] }
  () --> { source gens --> [ (4,5,6) ] }
  (1,2,3) --> { source gens --> [ (4,5,6) ] }
  These 3 automorphisms generate the group of automorphisms.
: associated cat1-group is [..=>..]
```

2.4.2 AllCat1sBasic

▷ `AllCat1sBasic(gp)` (operation)

For a group G of size greater than 70 which is reasonably straightforward this function may be used to construct a list of all cat1-group structures on G. The operation also attempts to write output to a file in the folder `xmod/lib`. (Other operations in the file `cat1data.gi` have been used to deal with the more complicated groups of size up to 70, but these are not described here.)

Van Luyen Le has a more efficient algorithm, extending the data up to groups of size 171, which is expected to appear in a future release of HAP.

Example

```
gap> gp := SmallGroup( 102, 2 );
<pc group of size 102 with 3 generators>
gap> StructureDescription( gp );
"C3 x D34"
gap> all := AllCat1sBasic( gp );
#I Edit last line of .../xmod/lib/nn.kk.out to end with ] ] ] ] ]
[ [Group( [ f1, f2, f3 ] )=>Group( [ f1, <identity> of ..., <identity> of ...
     ] )], [Group( [ f1, f2, f3 ] )=>Group( [ f1, f2, <identity> of ... ] )],
  [Group( [ f1, f2, f3 ] )=>Group( [ f1, <identity> of ..., f3 ] )],
  [Group( [ f1, f2, f3 ] )=>Group( [ f1, f2, f3 ] )] ]
gap> Display( all[3] );

Cat1-group [..=>..] :-
: Source group has generators:
  [ f1, f2, f3 ]
: Range group has generators:
  [ f1, <identity> of ..., f3 ]
: tail homomorphism maps source generators to:
  [ f1, <identity> of ..., f3 ]
: head homomorphism maps source generators to:
  [ f1, <identity> of ..., f3 ]
: range embedding maps range generators to:
  [ f1, <identity> of ..., f3 ]
: kernel has generators:
  [ f2 ]
: boundary homomorphism maps generators of kernel to:
  [ <identity> of ... ]
: kernel embedding maps generators of kernel to:
  [ f2 ]
```

Chapter 3

2d-mappings

3.1 Morphisms of 2d-groups

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

3.1.1 Source

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

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

3.2 Morphisms of pre-crossed modules

3.2.1 IsXModMorphism

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

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

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

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 $\mathrm{Aut}(\mathcal{X}_1)$.

3.2.2 IsInjective

▷ `IsInjective(`*map*`)` (property)
▷ `IsSurjective(`*map*`)` (property)

▷ IsSingleValued(*map*) (property)
▷ IsTotal(*map*) (property)
▷ IsBijective(*map*) (property)
▷ IsEndo2dMapping(*map*) (property)

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.

3.2.3 XModMorphism

▷ XModMorphism(*args*) (function)
▷ XModMorphismByHoms(*X1, X2, sigma, rho*) (operation)
▷ PreXModMorphism(*args*) (function)
▷ PreXModMorphismByHoms(*P1, P2, sigma, rho*) (operation)
▷ InclusionMorphism2dDomains(*X1, S1*) (operation)
▷ InnerAutomorphismXMod(*X1, r*) (operation)
▷ IdentityMapping(*X1*) (attribute)
▷ IsomorphismPermObject(*obj*) (function)

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 X1 constructed in the previous chapter.

Example

```
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) ]
  [ (1,2,3,4) ]
: Range = Source
: Source Homomorphism maps source generators to:
  [ (5,9,8,7,6) ]
: Range Homomorphism maps range generators to:
  [ (1,2,3,4) ]
gap> IsAutomorphism2dDomain( mor1 );
true
gap> Order( mor1 );
2
gap> RepresentationsOfObject( mor1 );
[ "IsComponentObjectRep", "IsAttributeStoringRep", "Is2dMappingRep" ]
gap> KnownPropertiesOfObject( mor1 );
[ "CanEasilyCompareElements", "CanEasilySortElements", "IsTotal",
  "IsSingleValued", "IsInjective", "IsSurjective", "RespectsMultiplication",
  "IsPreXModMorphism", "IsXModMorphism", "IsEndomorphism2dDomain",
  "IsAutomorphism2dDomain" ]
gap> KnownAttributesOfObject( mor1 );
[ "Name", "Order", "Range", "Source", "SourceHom", "RangeHom" ]
```

3.3 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

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

3.3.1 IsCat1Morphism

▷ `IsCat1Morphism(`*map*`)` (property)
▷ `IsPreCat1Morphism(`*map*`)` (property)
▷ `Cat1Morphism(`*args*`)` (function)
▷ `Cat1MorphismByHoms(`*C1, C2, gamma, rho*`)` (operation)
▷ `PreCat1Morphism(`*args*`)` (function)
▷ `PreCat1MorphismByHoms(`*P1, P2, gamma, rho*`)` (operation)
▷ `InclusionMorphism2dDomains(`*C1, S1*`)` (operation)
▷ `InnerAutomorphismCat1(`*C1, r*`)` (operation)
▷ `IdentityMapping(`*C1*`)` (attribute)
▷ `IsomorphismPermObject(`*obj*`)` (function)
▷ `SmallerDegreePerm2dDomain(`*obj*`)` (function)

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 `SmallerDegreePerm2dDomain`. Names are assigned automatically.

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

3.4 Operations on morphisms

3.4.1 CompositionMorphism

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

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.

Example

```
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 := SubXMod( X16, c4, d8 );
[c4->d8]
gap> IsSubXMod( X16, X8 );
true
gap> inc8 := InclusionMorphism2dDomains( 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
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)
 ] )]
```

3.4.2 Kernel

▷ `Kernel(map)` (operation)

▷ `Kernel2dMapping(map)` (attribute)

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

Example

```
gap> c2 := Group( (19,20) );;
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)
 ] )]
```

Chapter 4

Derivations and Sections

4.1 Whitehead Multiplication

4.1.1 IsDerivation

▷ `IsDerivation(map)` (property)
▷ `IsSection(map)` (property)
▷ `IsUp2dMapping(map)` (property)

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 $\chi : R \to S$, with *Whitehead multiplication* $\star$ (on the *right*) satisfying:

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

The zero map is the identity for this composition. Invertible elements in the monoid are called *regular*. The Whitehead group of $\mathcal{X}$ 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:

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

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

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

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

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

4.1.2 DerivationByImages

▷ `DerivationByImages(X0, ims)` (operation)

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 {\bf 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`.

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 := 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) ] )
```

4.1.3 SectionByImages

▷ `SectionByImages(C, ims)` (operation)
▷ `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 `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 GAP3 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`.

Example
```
gap> xi := SectionByDerivation( chi );
SectionByImages( s3, g18, [ (4,5,6), (2,3)(5,6) ], [ (1,2,3), (1,2)(4,6) ] )
```

4.2 Whitehead Groups and Monoids

4.2.1 RegularDerivations

▷ `RegularDerivations(`*X0*`)` (attribute)
▷ `AllDerivations(`*X0*`)` (attribute)
▷ `RegularSections(`*C0*`)` (attribute)
▷ `AllSections(`*C0*`)` (attribute)
▷ `ImagesList(`*obj*`)` (attribute)
▷ `ImagesTable(`*obj*`)` (attribute)

There are two functions to determine the elements of the Whitehead group and the Whitehead monoid of $\mathscr{X}0$, 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`.

Example
```
gap> all3 := AllDerivations( X3 );;
gap> imall3 := ImagesList( all3 );;
gap> PrintListOneItemPerLine( 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> PrintListOneItemPerLine( 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 ]
  ]
```

4.2.2 CompositeDerivation

▷ `CompositeDerivation(`*`chi1, chi2`*`)` (operation)
▷ `UpImagePositions(`*`chi`*`)` (attribute)
▷ `UpGeneratorImages(`*`chi`*`)` (attribute)
▷ `CompositeSection(`*`xi1, xi2`*`)` (operation)

The Whitehead product $\chi_1 \star \chi_2$ is implemented as `CompositeDerivation( <chi1>, <chi2> )`. The composite of two sections is just the composite of homomorphisms.

Example
```
gap> reg3 := RegularDerivations( X3 );;
gap> imder3 := ImagesList( reg3 );
[ [ (), () ], [ (), (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> 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 := UpImagePositions( chi4 );
[ 1, 3, 2, 1, 3, 2 ]
gap> im5 := UpImagePositions( 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 := UpImagePositions( chi45 );
[ 1, 1, 1, 2, 2, 2 ]
gap> pos := Position( imder3, UpGeneratorImages( chi45 ) );
2
```

4.2.3 WhiteheadGroupTable

▷ `WhiteheadGroupTable(`*`X0`*`)` (attribute)
▷ `WhiteheadMonoidTable(`*`X0`*`)` (attribute)
▷ `WhiteheadPermGroup(`*`X0`*`)` (attribute)
▷ `WhiteheadTransMonoid(`*`X0`*`)` (attribute)

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

Example

```
gap> wgt3 := WhiteheadGroupTable( X3 );;
gap> PrintListOneItemPerLine( 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 );;
gap> PrintListOneItemPerLine( 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 );
<transformation monoid on 9 pts with 3 generators>
gap> GeneratorsOfMonoid( wtm3 );
[ Transformation( [ 2, 3, 1, 5, 6, 4, 8, 9, 7 ] ),
  Transformation( [ 4, 4, 4, 4, 4, 4, 4, 4, 4 ] ),
  Transformation( [ 7, 9, 8, 4, 6, 5, 1, 3, 2 ] ) ]
```

Chapter 5

Actors of 2d-groups

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 $\mathcal{X}$.

5.1.1 WhiteheadXMod

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

An automorphism (σ,ρ) of `X` acts on the Whitehead monoid by $\chi^{(\sigma,\rho)} = \sigma \circ \chi \circ \rho^{-1}$, and this 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.

Example

```
gap> WGX3 := WhiteheadPermGroup( X3 );
Group([ (1,2,3)(4,5,6), (1,4)(2,6)(3,5) ])
gap> APX3 := AutomorphismPermGroup( X3 );
Group([ (5,7,6), (1,2)(3,4)(6,7) ])
```

```
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:
  [ (1,3,2)(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 );;
gap> Display( LX3 );
Crossed module Lue[c3->s3] :-
: Source group has generators:
  [ (1,2,3)(4,6,5) ]
: Range group has generators:
  [ (5,7,6), (1,2)(3,4)(6,7) ]
: Boundary homomorphism maps source generators to:
  [ (5,7,6) ]
: Action homomorphism maps range generators to automorphisms:
  (5,7,6) --> { source gens --> [ (1,2,3)(4,6,5) ] }
  (1,2)(3,4)(6,7) --> { source gens --> [ (1,3,2)(4,5,6) ] }
  These 2 automorphisms generate the group of automorphisms.
gap> NX3 := NorrieXMod( X3 );;
gap> Display( NX3 );
Crossed module Norrie[c3->s3] :-
: Source group has generators:
  [ (4,5,6), (2,3)(5,6) ]
: Range group has generators:
  [ (5,7,6), (1,2)(3,4)(6,7) ]
: Boundary homomorphism maps source generators to:
  [ (5,6,7), (1,2)(3,4)(6,7) ]
: Action homomorphism maps range generators to automorphisms:
  (5,7,6) --> { source gens --> [ (4,5,6), (2,3)(4,5) ] }
  (1,2)(3,4)(6,7) --> { source gens --> [ (4,6,5), (2,3)(5,6) ] }
  These 2 automorphisms generate the group of automorphisms.
gap> AX3 := ActorXMod( X3 );;  Display( AX3);
Crossed module Actor[c3->s3] :-
: Source group has generators:
  [ (1,2,3)(4,5,6), (1,4)(2,6)(3,5) ]
: Range group has generators:
  [ (5,7,6), (1,2)(3,4)(6,7) ]
: Boundary homomorphism maps source generators to:
  [ (5,6,7), (1,2)(3,4)(6,7) ]
: Action homomorphism maps range generators to automorphisms:
  (5,7,6) --> { source gens --> [ (1,2,3)(4,5,6), (1,5)(2,4)(3,6) ] }
  (1,2)(3,4)(6,7) --> { source gens --> [ (1,3,2)(4,6,5), (1,4)(2,6)(3,5) ] }
  These 2 automorphisms generate the group of automorphisms.
```

5.1.2 XModCentre

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

Pairs of boundaries or identity mappings provide six morphisms of crossed modules. In particular, the boundaries of $\mathscr{W}(\mathscr{X})$ and $\mathscr{N}(\mathscr{X})$ form the *inner morphism* of $\mathscr{X}$, mapping source elements to principal derivations and range elements to inner automorphisms. The image of $\mathscr{X}$ under this morphism is the *inner actor* of $\mathscr{X}$, while the kernel is the *centre* of $\mathscr{X}$. In the example which follows, using the crossed module `(X3 : c3 -> s3)` from Chapter 4, the inner morphism is an inclusion of crossed modules.

Example
```
gap> IAX3 := InnerActorXMod( X3 );;
gap> Display( IAX3 );
Crossed module InnerActor[c3->s3] :-
: Source group has generators:
  [ (1,3,2)(4,6,5) ]
: 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 --> [ (1,3,2)(4,6,5) ] }
  (1,2)(3,4)(6,7) --> { source gens --> [ (1,2,3)(4,5,6) ] }
  These 2 automorphisms generate the group of automorphisms.

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:
  [ (1,3,2)(4,6,5) ]
: 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( () )]
```

Chapter 6

Induced Constructions

6.1 Induced crossed modules

6.1.1 InducedXMod

▷ `InducedXMod(args)` (function)
▷ `InducedCat1(args)` (function)
▷ `IsInducedXMod(xmod)` (property)
▷ `IsInducedCat1(cat1)` (property)
▷ `MorphismOfInducedXMod(xmod)` (attribute)

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

Data for the cases of algebraic interest is provided by a conjugation crossed module $\mathscr{X} = (\partial : S \to R)$ and a homomorphism ι from R to a third group Q. The output from the calculation is a crossed module $\iota_*\mathscr{X} = (\delta : \iota_* S \to Q)$ together with a morphism of crossed modules $\mathscr{X} \to \iota_*\mathscr{X}$. When ι is a surjection with kernel K then $\iota_* S = [S,K]$ (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 $\mathscr{X}$ the normal inclusion crossed module of `a4` in `s4`, and for ι the surjection from `s4` to `s3` with kernel `k4`. The induced crossed module is isomorphic to `X3`.

Example

```
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> Display( indX4 );

Crossed module [a4/ker->s3] :-
: Source group a4/ker has generators:
  [ (1,3,2), (1,2,3) ]
: Range group s3 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->s3]]
```

For a second, injective example we take for $\mathscr{X}$ 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. (The groups in `indX8` vary from run to run due to random methods.)

Example

```
gap> incd8 := RangeHom( inc8 );;
gap> [ Source(incd8), Range(incd8), IsInjective(incd8) ];
[ d8, d16, true ]
gap> indX8 := InducedXMod( X8, incd8 );
#I induced group has Size: 16
#I factor 2 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)`.

Example

```
gap> s3b := Subgroup( s4, [ (2,3), (3,4) ] );;
gap> SetName( s3b, "s3b" );
gap> indX3 := InducedXMod( s4, s3b, s3b );
#I induced group has Size: 48
i*([s3b->s3b])
gap> isoX3 := IsomorphismGroups( Source( indX3 ), GeneralLinearGroup(2,3) );
[ (2,3)(4,7)(5,6), (1,2,3)(4,8,7), (2,4,5)(3,6,7) ] ->
[ [ [ 0*Z(3), Z(3)^0 ], [ Z(3)^0, 0*Z(3) ] ],
  [ [ Z(3), Z(3) ], [ Z(3)^0, 0*Z(3) ] ],
  [ [ Z(3), Z(3)^0 ], [ Z(3), 0*Z(3) ] ] ]
```

6.1.2 AllInducedXMods

▷ `AllInducedXMods(Q)` (operation)

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`.

Chapter 7

Crossed squares and their morphisms

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

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

- Groups S_J for each of the four subsets $J \subseteq [2]$;
- a commutative diagram of group homomorphisms:

$$\ddot{\partial}_1 : S_{[2]} \to S_{\{2\}}, \quad \ddot{\partial}_2 : S_{[2]} \to S_{\{1\}}, \quad \dot{\partial}_1 : S_{\{1\}} \to S_{\emptyset}, \quad \dot{\partial}_2 : S_{\{2\}} \to S_{\emptyset};$$

- actions of $S_{\emptyset}$ on $S_{\{1\}}, S_{\{2\}}$ and $S_{[2]}$ which determine actions of $S_{\{1\}}$ on $S_{\{2\}}$ and $S_{[2]}$ via $\dot{\partial}_1$ and actions of $S_{\{2\}}$ on $S_{\{1\}}$ and $S_{[2]}$ via $\dot{\partial}_2$˜;
- a function $\boxtimes : S_{\{1\}} \times S_{\{2\}} \to S_{[2]}$.

The following axioms must be satisfied for all $l \in S_{[2]},\ m, m_1, m_2 \in S_{\{1\}},\ n, n_1, n_2 \in S_{\{2\}},\ p \in S_{\emptyset}$:

- the homomorphisms $\ddot{\partial}_1, \ddot{\partial}_2$ preserve the action of $S_{\emptyset}$˜;
- each of

$$\ddot{\mathscr{S}}_1 = (\ddot{\partial}_1 : S_{[2]} \to S_{\{2\}}), \ddot{\mathscr{S}}_2 = (\ddot{\partial}_2 : S_{[2]} \to S_{\{1\}}), \dot{\mathscr{S}}_1 = (\dot{\partial}_1 : S_{\{1\}} \to S_{\emptyset}), \dot{\mathscr{S}}_2 = (\dot{\partial}_2 : S_{\{2\}} \to S_{\emptyset}),$$

  and the diagonal

$$\mathscr{S}_{12} = (\partial_{12} := \dot{\partial}_1 \ddot{\partial}_2 = \dot{\partial}_2 \ddot{\partial}_1 : S_{[2]} \to S_{\emptyset})$$

  are crossed modules (with actions via $S_{\emptyset}$);
- $\boxtimes$ is a *crossed pairing*:
  - $(m_1 m_2 \boxtimes n) \;=\; (m_1 \boxtimes n)^{m_2}\, (m_2 \boxtimes n)$,
  - $(m \boxtimes n_1 n_2) \;=\; (m \boxtimes n_2)\, (m \boxtimes n_1)^{n_2}$,
  - $(m \boxtimes n)^p \;=\; (m^p \boxtimes n^p)$;
- $\ddot{\partial}_1 (m \boxtimes n) \;=\; (n^{-1})^m\, n$ \quad \mbox{and} \quad $\ddot{\partial}_2 (m \boxtimes n) \;=\; m^{-1}\, m^n$,

- $(m \boxtimes \ddot{\partial}_1 l) \;=\; (l^{-1})^m\, l$ \quad \mbox{and} \quad $(\ddot{\partial}_2 l \boxtimes n) \;=\; l^{-1}\, l^n$.

Note that the actions of $S_{\{1\}}$ on $S_{\{2\}}$ and $S_{\{2\}}$ on $S_{\{1\}}$ via $S_\emptyset$ are compatible since

$$m_1{}^{(n^m)} \;=\; m_1{}^{\dot{\partial}_2(n^m)} \;=\; m_1{}^{m^{-1}(\dot{\partial}_2 n)m} \;=\; ((m_1{}^{m^{-1}})^n)^m.$$

7.1 Constructions for crossed squares

Analogously to the data structure used for crossed modules, crossed squares are implemented as `3d-groups`. When times allows, cat2-groups will also be implemented, with conversion between the two types of structure. Some standard constructions of crossed squares are listed below. At present, a limited number of constructions are implemented. Morphisms of crossed squares have also been implemented, though there is a lot still to do.

7.1.1 XSq

▷ `XSq(`*args*`)` (function)
▷ `XSqByNormalSubgroups(`*P, N, M, L*`)` (operation)
▷ `ActorXSq(`*X0*`)` (operation)
▷ `Transpose3dGroup(`*S0*`)` (attribute)
▷ `Name(`*S0*`)` (attribute)

Here are some standard examples of crossed squares.

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

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

  is a crossed square. This construction is implemented as `XSqByNormalSubgroups(P,N,M,L);`.

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

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

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

- The *transpose* of $\mathscr{S}$ is the crossed square $\tilde{\mathscr{S}}$ obtained by interchanging $S_{\{1\}}$ with $S_{\{2\}}$, $\ddot{\partial}_1$ with $\ddot{\partial}_2$, and $\dot{\partial}_1$ with $\dot{\partial}_2$. The crossed pairing is given by

$$\tilde{\boxtimes} \;:\; S_{\{2\}} \times S_{\{1\}} \to S_{[2]}, \quad (n,m) \;\mapsto\; n \tilde{\boxtimes} m := (m \boxtimes n)^{-1}\,.$$

Example

```
gap> c := (11,12,13,14,15,16);;
gap> d := (12,16)(13,15);;
gap> cd := c*d;;
gap> d12 := Group( [ c, d ] );;
gap> s3a := Subgroup( d12, [ c^2, d ] );;
gap> s3b := Subgroup( d12, [ c^2, cd ] );;
gap> c3 := Subgroup( d12, [ c^2 ] );;
gap> SetName( d12, "d12");  SetName( s3a, "s3a" );
gap> SetName( s3b, "s3b" );  SetName( c3, "c3" );
gap> XSconj := XSqByNormalSubgroups( d12, s3b, s3a, c3 );
[  c3 -> s3b ]
[  |      |  ]
[ s3a -> d12 ]
gap> Name( XSconj );
"[c3->s3b,s3a->d12]"
gap> XStrans := Transpose3dGroup( XSconj );
[  c3 -> s3a ]
[  |      |  ]
[ s3b -> d12 ]
gap> X12 := XModByNormalSubgroup( d12, s3a );
[s3a->d12]
gap> XSact := ActorXSq( X12 );
crossed square with:
      up = Whitehead[s3a->d12]
    left = [s3a->d12]
    down = Norrie[s3a->d12]
   right = Actor[s3a->d12]
```

7.1.2 IsXSq

▷ `IsXSq(`*obj*`)` (property)
▷ `Is3dObject(`*obj*`)` (property)
▷ `IsPerm3dObject(`*obj*`)` (property)
▷ `IsPc3dObject(`*obj*`)` (property)
▷ `IsFp3dObject(`*obj*`)` (property)
▷ `IsPreXSq(`*obj*`)` (property)

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

7.1.3 Up2dGroup

▷ `Up2dGroup(`*XS*`)` (attribute)
▷ `Left2dGroup(`*XS*`)` (attribute)
▷ `Down2dGroup(`*XS*`)` (attribute)
▷ `Right2dGroup(`*XS*`)` (attribute)
▷ `DiagonalAction(`*XS*`)` (attribute)
▷ `XPair(`*XS*`)` (attribute)
▷ `ImageElmXPair(`*XS*, *pair*`)` (operation)

In this implementation the attributes used in the construction of a crossed square `XS` are the four crossed modules (2d-groups) on the sides of the square; the diagonal action of P on L, and the crossed pairing.

The GAP development team have suggested that crossed pairings should be implemented as a special case of `BinaryMappings` – a structure which does not yet exist in GAP. As a temporary measure, crossed pairings have been implemented using `Mapping2ArgumentsByFunction`.

Example

```
gap> Up2dGroup( XSconj );
[c3->s3b]
gap> Right2dGroup( XSact );
Actor[s3a->d12]
gap> xpconj := XPair( XSconj );;
gap> ImageElmXPair( xpconj, [ (12,16)(13,15), (11,16)(12,15)(13,14) ] );
(11,15,13)(12,16,14)
gap> diag := DiagonalAction( XSact );
[ (2,3)(5,6), (1,2)(4,6) ] ->
[ [ (11,13,15)(12,14,16), (12,16)(13,15) ] -> [ (11,15,13)(12,16,14),
      (12,16)(13,15) ], [ (11,13,15)(12,14,16), (12,16)(13,15) ] ->
    [ (11,15,13)(12,16,14), (11,13)(14,16) ] ]
```

7.2 Morphisms of crossed squares

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

7.2.1 Source

▷ `Source(`*map*`)` (attribute)
▷ `Range(`*map*`)` (attribute)
▷ `Up2dMorphism(`*map*`)` (attribute)
▷ `Left2dMorphism(`*map*`)` (attribute)
▷ `Down2dMorphism(`*map*`)` (attribute)
▷ `Right2dMorphism(`*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.

7.2.2 IsXSqMorphism

▷ `IsXSqMorphism(`*map*`)` (property)
▷ `IsPreXSqMorphism(`*map*`)` (property)
▷ `IsBijective(`*mor*`)` (property)
▷ `IsEndomorphism3dObject(`*mor*`)` (property)
▷ `IsAutomorphism3dObject(`*mor*`)` (property)

A morphism `mor` between two pre-crossed squares $\mathcal{S}_1$ and $\mathcal{S}_2$ consists of four crossed module morphisms `Up2dMorphism( mor )`, mapping the `Up2dGroup` of $\mathcal{S}_1$ to that of $\mathcal{S}_2$, `Left2dMorphism( mor )`, `Down2dMorphism( mor )` and `Right2dMorphism( 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 `IsXSqMorphism` does not perform all the required checks.

Example

```
gap> ad12 := GroupHomomorphismByImages( d12, d12, [c,d], [c,d^c] );;
gap> as3a := GroupHomomorphismByImages( s3a, s3a, [c^2,d], [c^2,d^c] );;
gap> as3b := GroupHomomorphismByImages( s3b, s3b, [c^2,cd], [c^2,cd^c] );;
gap> idc3 := IdentityMapping( c3 );;
gap> upconj := Up2dGroup( XSconj );;
gap> leftconj := Left2dGroup( XSconj );;
gap> downconj := Down2dGroup( XSconj );;
gap> rightconj := Right2dGroup( XSconj );;
gap> up := XModMorphismByHoms( upconj, upconj, idc3, as3b );
[[c3->s3b] => [c3->s3b]]
gap> left := XModMorphismByHoms( leftconj, leftconj, idc3, as3a );
[[c3->s3a] => [c3->s3a]]
gap> down := XModMorphismByHoms( downconj, downconj, as3a, ad12 );
[[s3a->d12] => [s3a->d12]]
gap> right := XModMorphismByHoms( rightconj, rightconj, as3b, ad12 );
[[s3b->d12] => [s3b->d12]]
gap> autoconj := XSqMorphism( XSconj, XSconj, up, left, down, right );;
gap> ord := Order( autoconj );;
gap> Display( autoconj );
Morphism of crossed squares :-
:    Source = [c3->s3b,s3a->d12]
:     Range = [c3->s3b,s3a->d12]
:     order = 3
:    up-left: [ [ (11,13,15)(12,14,16) ], [ (11,13,15)(12,14,16) ] ]
:   up-right: [ [ (11,13,15)(12,14,16), (11,16)(12,15)(13,14) ],
  [ (11,13,15)(12,14,16), (11,12)(13,16)(14,15) ] ]
:  down-left: [ [ (11,13,15)(12,14,16), (12,16)(13,15) ],
  [ (11,13,15)(12,14,16), (11,13)(14,16) ] ]
: down-right: [ [ (11,12,13,14,15,16), (12,16)(13,15) ],
  [ (11,12,13,14,15,16), (11,13)(14,16) ] ]
gap> KnownPropertiesOfObject( autoconj );
[ "IsTotal", "IsSingleValued", "IsInjective", "IsSurjective", "Is3dMapping",
  "IsPreXSqMorphism", "IsXSqMorphism", "IsEndomorphism3dObject" ]
gap> IsAutomorphism3dObject( autoconj );
true
```

Chapter 8

Crossed modules of groupoids

8.1 Constructions for crossed modules of groupoids

A typical example of a crossed module over a groupoid has as range a connected groupoid which is a direct product of a group and a complete graph, and as source a totally disconnected groupoid, with the same objects. The boundary morphism is constant on objects. For details and other references see [AW10].

8.1.1 PreXModWithObjectsObj

▷ `PreXModWithObjectsObj(`*obs, bdy, act*`)` (operation)
▷ `DiscreteNormalPreXModWithObjects(`*gpd, gp*`)` (operation)

The next stage of development of this package will be to implement constuctions of crossed modules over groupoids. This will require further developments in the Gpd package. The following example shows what has been achieved in an earlier version, but which fails in GAP 4.7.

Example

```
gap> d8 := Group( (1,2,3,4), (1,3) );;
gap> SetName( d8, "d8" );
gap> Gd8 := SinglePieceGroupoid( d8, [-9,-8,-7] );;
gap> Display( Gd8 );
single piece groupoid:
  objects: [ -9, -8, -7 ]
    group: d8 = <[ (1,2,3,4), (1,3) ]>
gap> k4 := Subgroup( d8, [ (1,2)(3,4), (1,3)(2,4) ] );;
gap> PX0 := DiscreteNormalPreXModWithObjects( Gd8, k4 );
homogeneous, discrete groupoid with:
  group: Group( [ (1,2)(3,4), (1,3)(2,4) ] ) >
objects: [ -9, -8, -7 ]
#I  now need to be able to test:   ok := IsXMod( PM );
<magma>
gap> Source(PX0);
perm homogeneous, discrete groupoid: < Group( [ (1,2)(3,4), (1,3)(2,4) ] ),
[ -9, -8, -7 ] >
```

Chapter 9

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.

9.1 Printing Lists

9.1.1 PrintListOneItemPerLine

▷ `PrintListOneItemPerLine(`*L*`)` (operation)

This function is used in Chapter 4 when printing image lists of derivations and sections.

Example
```
gap> L := [ [1,2,3,4], true, [ (1,2), (2,3) ] ];;
gap> PrintListOneItemPerLine( L );
[ [ 1, 2, 3, 4 ],
  true,
  [ (1,2), (2,3) ]
  ]
```

9.2 Inclusion and Restriction Mappings

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

9.2.1 InclusionMappingGroups

▷ `InclusionMappingGroups(`*G, H*`)` (operation)
▷ `RestrictionMappingGroups(`*hom, src, rng*`)` (operation)
▷ `MappingToOne(`*G, H*`)` (operation)

This 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 3.4.

Example
```
gap> Print( incd8, "\n" );
[ (11,13,15,17)(12,14,16,18), (11,18)(12,17)(13,16)(14,15) ] ->
[ (11,13,15,17)(12,14,16,18), (11,18)(12,17)(13,16)(14,15) ]
gap> imd8 := Image( incd8 );;
gap> resd8 := RestrictionMappingGroups( incd8, c4, imd8 );;
gap> Source( resd8 );  Range( resd8 );
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) ] -> [ () ]
```

9.2.2 InnerAutomorphismByNormalSubgroup

▷ `InnerAutomorphismByNormalSubgroup(`*G, N*`)` (operation)
▷ `IsGroupOfAutomorphisms(`*A*`)` (property)

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`.

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

9.3 Abelian Modules

9.3.1 AbelianModuleObject

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

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

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

Example
```
gap> x := (6,7)(8,9);;  y := (6,8)(7,9);;  z := (6,9)(7,8);;
gap> 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 );;
gap> Xabmod := XModByAbelianModule( abmod );
[k4->s3]
gap> Display( Xabmod );

Crossed module [k4->s3] :-
: Source group k4 has generators:
  [ (6,7)(8,9), (6,8)(7,9) ]
: Range group s3 has generators:
  [ (1,2), (2,3) ]
: Boundary homomorphism maps source generators to:
  [ (), () ]
: Action homomorphism maps range generators to automorphisms:
  (1,2) --> { source gens --> [ (6,8)(7,9), (6,7)(8,9) ] }
  (2,3) --> { source gens --> [ (6,7)(8,9), (6,9)(7,8) ] }
  These 2 automorphisms generate the group of automorphisms.
```

9.4 Distinct and Common Representatives

9.4.1 DistinctRepresentatives

▷ DistinctRepresentatives(*list*) (operation)
▷ CommonRepresentatives(*list*) (operation)
▷ CommonTransversal(*grp*, *subgrp*) (operation)
▷ IsCommonTransversal(*grp*, *subgrp*, *list*) (operation)

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

When L is a set of n subsets of $[1\dots n]$ 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.

Example

```
gap> J := [ [1,2,3], [3,4], [3,4], [1,2,4] ];
[ [ 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] ];
[ [ 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) ]
```

Chapter 10

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.)

10.1 Changes from version to version

10.1.1 Version 1 for GAP 3

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

10.1.2 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.

10.1.3 Version 2.001 for GAP 4

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`.

10.1.4 Induced crossed modules

During the period May 20th - May 27th 2002 converted `induce.g` to `induce.gd` and `induce.gi` (later renamed `gp2ind.gd`, `gp2ind.gi`), at least as regards induced crossed modules. (Induced cat1-groups may be convereted one day.) For details, see the file `CHANGES`.

10.1.5 Versions 2.002 – 2.006

Version 2.002 was prepared for the 4.4 release at the end of January 2004.

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

Version 2.004 of April 14th 2004 added the `Cat1Select` functionality of version 1 to the `Cat1` function (see also version 2.007).

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`.

A significant change was the conversion of 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`.

10.1.6 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` for crossed modules, and introduced a special method for conjugation crossed modules.
- Substantial revisons made to `XModByCentralExtension`, `NorrieXMod`, `LueXMod`, `ActorXMod`, and `InclusionInducedXModByCopower`.
- Reintroduced the `Cat1Select` operation.
- Version 2.010, of October 2007, was timed to coincide with the release of GAP 4.4.10, and included a change of filenames; correct file protection codes; and an improvement to `AutomorphismPermGroup` for crossed modules.

10.2 Versions for GAP 4.5, 4.6 and 4.7

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.
- The package webpage has moved along with the whole of the Bangor Maths website: `http://www.maths.bangor.ac.uk/chda/`.
- A GNU General Public License declaration has been added.

10.2.1 AllCat1s

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

The `AllCat1s` function was reintroduced from the GAP3 version, and renamed as the operation `AllCat1sBasic`.

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

10.2.2 Version 2.31

This latest version was released on 17th December 2014, following a change in the package website location. A `Bitbucket` repository for the package was also created.

10.3 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 Gpd 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 $\mathrm{Act}\, L = (\iota : L \to \mathrm{Aut}\, L)$ splits to form the square with $(\iota_1 : L \to \mathrm{Inn}\, L)$ on two sides, and $(\iota_2 : \mathrm{Inn}\, L \to \mathrm{Aut}\, L)$ on the other two sides, where ι_1 maps $l \in L$ to the inner automorphism $\beta_l : L \to L,\ l' \mapsto l^{-1}l'l$, and ι_2 is the inclusion of Inn L in Aut L. The actions are standard, and the crossed pairing is
  $$\boxtimes \ : \ \mathrm{Inn}\, L \times \mathrm{Inn}\, L \to L, \quad (\beta_l, \beta_{l'}) \ \mapsto \ [l, l'] \ .$$

References

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

[AW00] M. Alp and C. D. Wensley. Enumeration of cat1-groups of low order. *Int. J. Algebra and Computation*, 10:407–424, 2000. 5, 22

[AW10] M. Alp and C.D. Wensley. Automorphisms and homotopies of groupoids and crossed modules. *Applied Categorical Structures*, 18:473–495, 2010. 37

[BH78] R. Brown and P. J. Higgins. On the connection between the second relative homotopy group and some related spaces. *Proc. London Math. Soc.*, 36:193–212, 1978. 5, 29

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

[BL87] R. Brown and J.-L. Loday. Van kampen theorems for diagram of spaces. *Topology*, 26:311–335, 1987. 32

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

[BW95] R. Brown and C. D. Wensley. On finite induced crossed modules, and the homotopy 2-type of mapping cones. *Theory and Applications of Categories*, 1:54–71, 1995. 5, 29, 42

[BW96] R. Brown and C. D. Wensley. Computing crossed modules induced by an inclusion of a normal subgroup, with applications to homotopy 2-types. *Theory and Applications of Categories*, 2:3–16, 1996. 5, 29, 42

[BW03] R. Brown and C. D. Wensley. Computation and homotopical applications of induced crossed modules. *J. Symbolic Computation*, 35:59–72, 2003. 42

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

[ES87] G. Ellis and R. Steiner. Higher dimensional crossed modules and the homotopy groups of (n+1)-ads. *J. Pure and Appl. Algebra*, 46:117–136, 1987. 32

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

[Lod82] J. L. Loday. Spaces with finitely many non-trivial homotopy groups. *J. App. Algebra*, 24:179–202, 1982. 5, 11

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

[Nor87] K. J. Norrie. *Crossed modules and analogues of group theorems*. Ph.D. thesis, King's College, University of London, 1987. 5, 26

[Nor90] K. J. Norrie. Actions and automorphisms of crossed modules. *Bull. Soc. Math. France*, 118:129–146, 1990. 5, 26

[Whi48] J. H. C. Whitehead. On operators in relative homotopy groups. *Ann. of Math.*, 49:610–640, 1948. 5, 21

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

Index