Gpd

Groupoids, graphs of groups, and graphs of groupoids

Version 1.22

20/11/2013

Emma Moore
Chris Wensley

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 Gpd package for GAP4 provides functions for the computation with groupoids (categories with every arrow invertible) and their morphisms; for graphs of groups, and graphs of groupoids. The most basic structure introduced is that of *magma with objects*, followed by *semigroup with objects*, then *monoid with objects* and finally *groupoid* which is a *group with objects*.

It provides normal forms for Free Products with Amalgamation and for HNN-extensions when the initial groups have rewrite systems and the subgroups have finite index.

The Gpd package was originally implemented in 2000 (as GraphGpd) when the first author was studying for a Ph.D. in Bangor.

Version 1.07 was released in July 2011, to be tested with GAP 4.5. Version 1.15 came out with the first release of GAP 4.5 in June 2012, and was submitted for official acceptance as a GAP package. The latest version is 1.22 of 20th November 2013, prepared for GAP 4.7.

Recent versions implement many of the constructions described in the paper [AW10] for automorphisms of groupoids.

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.

Contents

1 Many-object structures **5**
- 1.1 Magmas with objects . . . 5
- 1.2 Semigroups with objects . . . 7
- 1.3 Monoids with objects . . . 8
- 1.4 Structures with one or more pieces . . . 9

2 Homomorphisms of many-object structures **11**
- 2.1 Homomorphisms of magmas with objects . . . 11
- 2.2 Homomorphisms of semigroups and monoids with objects . . . 13
- 2.3 Homomorphisms to more than one piece . . . 14

3 Groupoids **15**
- 3.1 Groupoids: their properties and attributes . . . 15
- 3.2 Groupoid elements; stars; costars and homsets . . . 18
- 3.3 Subgroupoids . . . 20
- 3.4 Left, right and double cosets . . . 23
- 3.5 Conjugation . . . 24

4 Homomorphisms of Groupoids **26**
- 4.1 Homomorphisms from a connected groupoid . . . 26
- 4.2 Homomorphisms to a connected groupoid . . . 27
- 4.3 Homomorphisms with more than one piece . . . 28
- 4.4 Groupoid automorphisms . . . 29

5 Graphs of Groups and Groupoids **33**
- 5.1 Digraphs . . . 33
- 5.2 Graphs of Groups . . . 34
- 5.3 Words in a Graph of Groups and their normal forms . . . 36
- 5.4 Free products with amalgamation and HNN extensions . . . 37
- 5.5 GraphsOfGroupoids and their Words . . . 40

6 Technical Notes **42**
- 6.1 Many object structures . . . 42
- 6.2 Many object homomorphisms . . . 44

7 Development History **45**
7.1 Versions of the Package 45
7.2 What needs to be done next? 45

References **47**

Index **48**

Chapter 1

Many-object structures

The aim of this package is to provide operations for finite groupoids. A *groupoid* is constructed from a group and a set of objects. In order to provide a sequence of categories, with increasing structure, mimicing those for groups, we introduce in this chapter the notions of *magma with objects*; *semigroup with objects* and *monoid with objects*. The next chapter introduces morphisms of these structures. At a first reading of this manual, the user is advised to skip quickly through these first two chapters, and then move on to groupoids in Chapter 3.

For the definitions of the standard properties of groupoids we refer to P. Higgins' book "Categories and Groupoids" [Hig05] (originally published in 1971, reprinted by TAC in 2005), and to R. Brown's book "Topology" [Bro88], recently revised and reissued as "Topology and Groupoids" [Bro06].

1.1 Magmas with objects

A *magma with objects* M consists of a set of *objects* $\mathrm{Ob}(M)$, and a set of *arrows* $\mathrm{Arr}(M)$ together with *tail* and *head* maps $t,h : \mathrm{Arr}(M) \to \mathrm{Ob}(M)$, and a *partial multiplication* $* : \mathrm{Arr}(M) \to \mathrm{Arr}(M)$, with $a*b$ defined precisely when the head of a coincides with the tail of b. We write an arrow a with tail u and head v as $(a : u \to v)$.

When this multiplication is associative we obtain a *semigroup with objects*.

A *loop* is an arrow whose tail and head are the same object. An *identity arrow* at object u is a loop $(1_u : u \to u)$ such that $a*1_u = a$ and $1_u*b = b$ whenever u is the head of a and the tail of b. When M is a semigroup with objects and every object has an identity arrow, we obtain a *monoid with objects*, which is just the usual notion of mathematical category.

An arrow $(a : u \to v)$ in a monoid with objects has *inverse* $(a^{-1} : v \to u)$ provided $a*a^{-1} = 1_u$ and $a^{-1}*a = 1_v$. A monoid with objects in which every arrow has an inverse is a *group with objects*, usually called a *groupoid*.

1.1.1 MagmaWithObjects

▷ `MagmaWithObjects(`*`args`*`)` (function)
▷ `SinglePieceMagmaWithObjects(`*`magma, obs`*`)` (operation)
▷ `ObjectList(`*`mwo`*`)` (attribute)
▷ `RootObject(`*`mwo`*`)` (operation)

The simplest construction for a magma with objects M is to take a magma m and an ordered set s, and form arrows (u,x,v) for every x in m and u,v in s. Multiplication is defined by $(u,x,v)*(v,y,w) = (u,x*y,w)$. Any finite, ordered set is in principle acceptable as the object list of M, but we find it convenient to restrict ourselves to sets of non-positive integers.

This is the only construction implemented here for magmas, semigroups, and monoids with objects, and these all have the property `IsDirectProductWithCompleteGraph`. There are other constructions implemented for groupoids.

The output from function `MagmaWithObjects` lies in the categories `IsDomainWithObjects`, `IsMagmaWithObjects`, `CategoryCollections(IsMultiplicativeElementWithObjects)` and `IsMagma`. The *root object* of M is the first element in s.

Example

```
gap> tm := [[1,2,4,3],[1,2,4,3],[3,4,2,1],[3,4,2,1]];;
gap> Display( tm );
[ [  1,  2,  4,  3 ],
  [  1,  2,  4,  3 ],
  [  3,  4,  2,  1 ],
  [  3,  4,  2,  1 ] ]
gap> m := MagmaByMultiplicationTable( tm );;  SetName( m, "m" );
gap> m1 := MagmaElement(m,1);;  m2 := MagmaElement(m,2);;
gap> m3 := MagmaElement(m,3);;  m4 := MagmaElement(m,4);;
gap> M78 := MagmaWithObjects( m, [-8,-7] );
magma with objects :-
    magma = m
  objects = [ -8, -7 ]
gap> SetName( M78, "M78" );
gap> [ IsAssociative(M78), IsCommutative(M78), IsDomainWithObjects(M78) ];
[ false, false, true ]
gap> [ RootObject( M78 ), ObjectList( M78 ) ];
[ -8, [ -8, -7 ] ]
```

1.1.2 MultiplicativeElementWithObjects

▷ `MultiplicativeElementWithObjects(`*mwo, elt, tail, head*`)` (operation)
▷ `ElementOfArrow(`*elt*`)` (attribute)
▷ `TailOfArrow(`*elt*`)` (attribute)
▷ `HeadOfArrow(`*elt*`)` (attribute)
▷ `IsElementOfMagmaWithObjects(`*elt, mwo*`)` (operation)

Arrows in a magma with objects lie in the category `IsMultiplicativeElementWithObjects`. An attempt to multiply two arrows which do not compose resuts in `fail` being returned. Each arrow $(a : u \to v)$ has three components: a monoid *element* a; a *tail* object u; and a *head* object v. The operation `IsElementOfMagmaWithObjects` is added due to problems with writing a method for `\in`.

Example

```
gap> a78 := MultiplicativeElementWithObjects( M78, m2, -7, -8 );
[m2 : -7 -> -8]
gap> [ a78 in M78, IsElementOfMagmaWithObjects( a78, M78 ) ];
```

```
[ false, true ]
gap> b87 := MultiplicativeElementWithObjects( M78, m4, -8, -7 );;
gap> [ ElementOfArrow( b87 ), TailOfArrow( b87 ), HeadOfArrow( b87 ) ];
[ m4, -8, -7 ]
gap> ba := b87*a78;;  ab := a78*b87;;  [ ba, ab ];
[ [m4 : -8 -> -8], [m3 : -7 -> -7] ]
gap> [ a78^2, ba^2, ba^3 ];
[ fail, [m1 : -8 -> -8], [m3 : -8 -> -8] ]
```

1.1.3 IsSinglePiece

▷ IsSinglePiece(*mwo*) (property)
▷ IsDirectProductWithCompleteGraph(*mwo*) (property)
▷ IsDiscrete(*mwo*) (property)

If the partial composition is forgotten, then what remains is a digraph (usually with multiple edges and loops). Thus the notion of *connected component* may be inherited by magmas with objects from digraphs. Unfortunately the terms Component and Constituent are already in considerable use elsewhere in GAP, so (and this may change if a more suitable word is suggested) we use the term IsSinglePiece to describe a connected magma with objects. When each connected component has a single object, and there is more than one component, the magma with objects is *discrete*.

Example

```
gap> IsSinglePiece( M78 );
true
gap> IsDirectProductWithCompleteGraph( M78 );
true
gap> IsDiscrete( M78 );
false
```

1.2 Semigroups with objects

1.2.1 SemigroupWithObjects

▷ SemigroupWithObjects(*args*) (function)
▷ SinglePieceSemigroupWithObjects(*sgp, obs*) (operation)

When the magma is a semigroup the construction gives a SinglePieceSemigroupWithObjects. In the example we use a transformation semigroup and 3 objects.

Example

```
gap> t := Transformation( [1,1,2,3] );;  s := Transformation( [2,2,3,3] );;
gap> r := Transformation( [2,3,4,4] );;  sgp := Semigroup( t, s, r );;
gap> SetName( sgp, "sgp<t,s,r>" );
gap> S123 := SemigroupWithObjects( sgp, [-3,-2,-1] );
semigroup with objects :-
    magma = sgp<t,s,r>
```

```
  objects = [ -3, -2, -1 ]
gap> [ IsAssociative(S123), IsCommutative(S123) ];
[ true, false ]
gap> t12 := MultiplicativeElementWithObjects( S123, t, -1, -2 );
[Transformation( [ 1, 1, 2, 3 ] ) : -1 -> -2]
gap> s23 := MultiplicativeElementWithObjects( S123, s, -2, -3 );;
gap> r31 := MultiplicativeElementWithObjects( S123, r, -3, -1 );;
gap> ts13 := t12 * s23;  tsr1 := ts13 * r31;
[Transformation( [ 2, 2, 2, 3 ] ) : -1 -> -3]
[Transformation( [ 3, 3, 3 ] ) : -1 -> -1]
```

1.3 Monoids with objects

1.3.1 MonoidWithObjects

▷ MonoidWithObjects(*args*) (function)
▷ SinglePieceMonoidWithObjects(*mon*, *obs*) (operation)
▷ GeneratorsOfMagmaWithObjects(*mwo*) (operation)

When the semigroup is a monoid the construction gives a SinglePieceMonoidWithObjects. The example uses a finitely presented monoid with 2 generators and 2 objects.

When the construction is the direct product of a monoid and the complete graph on the objects, the generating set consists of two parts. Firstly, there is a loop at the root object for each generator of the monoid. Secondly, for each pair of objects u, v, there are arrows $(1 : u \to v)$ and $(1 : v \to u)$.

Example

```
gap> fm := FreeMonoid( 2, "f" );;
gap> em := One( fm );;
gap> gm := GeneratorsOfMonoid( fm );;
gap> mon := fm/[ [gm[1]^3,em], [gm[1]*gm[2],gm[2]] ];;
gap> M49 := MonoidWithObjects( mon, [-9,-4] );
monoid with objects :-
    magma = Monoid( [ f1, f2 ] )
  objects = [ -9, -4 ]
gap> ktpo := KnownTruePropertiesOfObject( M49 );
[ "CanEasilyCompareElements", "CanEasilySortElements", "IsDuplicateFree",
  "IsAssociative", "IsSinglePieceDomain",
  "IsDirectProductWithCompleteGraphDomain" ]
gap> genM : GeneratorsOfMagmaWithObjects( M49 );
[ [<identity ...> : -9 -> -9], [f1 : -9 -> -9], [f2 : -9 -> -9],
  [<identity ...> : -9 -> -4], [<identity ...> : -4 -> -9] ]
gap> g2:=genM[2];;
gap> g3:=genM[3];;
gap> g4:=genM[4];;
gap> g5:=genM[5];;
gap> [g5,g3,g2,g4];
[ [<identity ...> : -4 -> -9], [f2 : -9 -> -9], [f1 : -9 -> -9],
  [<identity ...> : -9 -> -4] ]
gap> g5*g3*g2*g4;
```

```
[f2*f1 : -4 -> -4]
```

1.4 Structures with one or more pieces

1.4.1 DomainWithSingleObject

▷ DomainWithSingleObject(*dom, obj*) (operation)

A magma, semigroup, monoid, or group can be made into a magma with objects by the addition of a single object. The two are algebraically isomorphic, and there is one arrow (a loop) for each element in the group. In the example we take the dihedral group of size 8 at the object 0.

Example

```
gap> d8 := Group( (1,2,3,4), (1,3) );;
gap> SetName( d8, "d8" );
gap> D0 := DomainWithSingleObject( d8, 0 );
single piece groupoid: < d8, [ 0 ] >
gap> GeneratorsOfMagmaWithInverses( D0 );
[ [(1,2,3,4) : 0 -> 0], [(1,3) : 0 -> 0] ]
gap> Size( D0 );
8
```

1.4.2 UnionOfPieces

▷ UnionOfPieces(*pieces*) (operation)
▷ Pieces(*mwo*) (attribute)

A magma with objects whose underlying digraph has two or more connected components can be constructed by taking the union of two or more connected structures. These, in turn, can be combined together. The only requirement is that all the object lists should be disjoint.

Structures S123, M49, D0 generated above have, respectively, GeneratorsOfMagma, GeneratorsOfMagmaWithOne and GeneratorsOfMagmaWithInverses. The generators of a structure with several pieces is the union of the generators of the individual pieces.

Example

```
gap> N1 := UnionOfPieces( [ M78, S123 ] );;  ObjectList( N1 );
[ -8, -7, -3, -2, -1 ]
gap> N2 := UnionOfPieces( [ M49, D0 ] );;  Pieces( N2 );
[ monoid with objects :-
        magma = Monoid( [ f1, f2 ] )
      objects = [ -9, -4 ]
    , single piece groupoid: < d8, [ 0 ] > ]
gap> N3 := UnionOfPieces( [ N1, N2 ] );
magma with objects having 4 pieces :-
1: monoid with objects :-
    magma = Monoid( [ f1, f2 ] )
  objects = [ -9, -4 ]
```

```
2: M78
3: semigroup with objects :-
    magma = sgp<t,s,r>
  objects = [ -3, -2, -1 ]
4: single piece groupoid: < d8, [ 0 ] >
gap> ObjectList( N3 );
[ -9, -8, -7, -4, -3, -2, -1, 0 ]
gap> Length( GeneratorsOfMagmaWithObjects( N3 ) );
50
gap> ## the next command returns fail since the object sets are not disjoint:
gap> N4 := UnionOfPieces( [ S123, DomainWithSingleObject( d8, -2 ) ] );
fail
```

Chapter 2

Homomorphisms of many-object structures

A *homomorphism* f from a magma with objects M to a magma with objects N consists of

- a map f_O from the objects of M to those of N,
- a map f_A from the arrows of M to those of N.

The map f_A is required to be compatible with the tail and head maps and to preserve multiplication:

$$f_A(a : u \to v) * f_A(b : v \to w) \;=\; f_A(a * b : u \to w)$$

with tail $f_O(u)$ and head $f_O(w)$.

When M is a monoid or group, the map f_A is required to preserve object identities and inverses.

2.1 Homomorphisms of magmas with objects

2.1.1 MagmaWithObjectsHomomorphism

▷ `MagmaWithObjectsHomomorphism(`*`args`*`)` (function)
▷ `HomomorphismFromSinglePiece(`*`src, rng, hom, imobs`*`)` (operation)
▷ `HomomorphismToSinglePiece(`*`src, rng, images`*`)` (operation)
▷ `PieceImages(`*`mwohom`*`)` (attribute)
▷ `HomsOfMapping(`*`mwohom`*`)` (attribute)
▷ `PiecesOfMapping(`*`mwohom`*`)` (attribute)
▷ `IsomorphismNewObjects(`*`src, objlist`*`)` (operation)

There are a variety of homomorphism constructors.

The simplest construction gives a homomorphism $M \to N$ with both M and N connected. It is implemented as `IsMappingToSinglePieceRep` with attributes `Source`, `Range` and `PieceImages`. The operation requires the following information:

- a magma homomorphism `f` from the underlying magma of M to the underlying magma of N,
- a list `imobs` of the images of the objects of M.

In the first example we construct endomappings of m and M78.

Example

```
gap> tup1 := [Tuple([m1,m2]), Tuple([m2,m1]), Tuple([m3,m4]), Tuple([m4,m3])];
gap> f1 := GeneralMappingByElements( m, m, tup1 );
gap> IsMagmaHomomorphism( f1 );
true
gap> hom1 := MagmaWithObjectsHomomorphism( M78, M78, f1, [-8,-7] );;
gap> Display( hom1 );
homomorphism to single piece magma: M78 -> M78
magma hom: <mapping: m -> m >, object map: [ -8, -7 ] -> [ -8, -7 ]
gap> [ Source( hom1 ), Range( hom1 ) ];
[ M78, M78 ]
gap> b87;
[m4 : -8 -> -7]
gap> im1 := ImageElm( hom1, b87 );
[m3 : -8 -> -7]
gap> i56 := IsomorphismNewObjects( M78, [-5,-6] );
magma with objects homomorphism :
[ [ IdentityMapping( m ), [ -5, -6 ] ] ]
gap> M65 := Range( i56);;
gap> SetName( M65, "M65" );
gap> j56 := InverseGeneralMapping( i56 );;
gap> ImagesOfObjects( j56 );
[ -7, -8 ]
gap> im56 := ImageElm( i56, b87 );
[m4 : -5 -> -6]
gap> comp := j56 * hom1;
magma with objects homomorphism : M65 -> M78
[ [ <mapping: m -> m >, [ -7, -8 ] ] ]
gap> ImageElm( comp, im56 );
[m3 : -8 -> -7]
```

A homomorphism *to* a connected magma with objects may have a source with several pieces, and so is a union of homomorphisms *from* single pieces.

Example

```
gap> M4 := UnionOfPieces( [ M78, M65 ] );;
gap> images := [ PieceImages( hom1 )[1], PieceImages( j56 )[1] ];
[ [ <mapping: m -> m >, [ -8, -7 ] ], [ IdentityMapping( m ), [ -7, -8 ] ] ]
gap> map4 := HomomorphismToSinglePiece( M4, M78, images );
magma with objects homomorphism :
[ [ <mapping: m -> m >, [ -8, -7 ] ], [ IdentityMapping( m ), [ -7, -8 ] ] ]
gap> ImageElm( map4, b87 );
[m3 : -8 -> -7]
gap> ImageElm( map4, im56 );
[m4 : -8 -> -7]
```

2.2 Homomorphisms of semigroups and monoids with objects

The next example exhibits a homomorphism between transformation semigroups with objects.

Example

```
gap> t2 := Transformation( [2,2,4,1] );;
gap> s2 := Transformation( [1,1,4,4] );;
gap> r2 := Transformation( [4,1,3,3] );;
gap> sgp2 := Semigroup( [ t2, s2, r2 ] );;
gap> SetName( sgp2, "sgp<t2,s2,r2>" );
gap> ##  apparently no method for transformation semigroups available for:
gap> ##  nat := NaturalHomomorphismByGenerators( sgp, sgp2 );  so we use:
gap> ##  in the function flip below t is a transformation on [1..n]
gap> flip := function( t )
>     local i, j, k, L, L2, n;
>     n := DegreeOfTransformation( t );
>     L := ImageListOfTransformation( t );
>     if IsOddInt(n) then n:=n+1; L1:=Concatenation(L,[n]);
>                    else L1:=L; fi;
>     L2 := ShallowCopy( L1 );
>     for i in [1..n] do
>         if IsOddInt(i) then j:=i+1; else j:=i-1; fi;
>         k := L1[j];
>         if IsOddInt(k) then L2[i]:=k+1; else L2[i]:=k-1; fi;
>     od;
>     return( Transformation( L2 ) );
> end;;
gap> smap := MappingByFunction( sgp, sgp2, flip );;
gap> ok := RespectsMultiplication( smap );
true
gap> [ t, Image( smap, t ) ];
[ Transformation( [ 1, 1, 2, 3 ] ), Transformation( [ 2, 2, 4, 1 ] ) ]
gap> [ s, Image( smap, s ) ];
[ Transformation( [ 2, 2, 3, 3 ] ), Transformation( [ 1, 1, 4, 4 ] ) ]
gap> [ r, Image( smap, r ) ];
[ Transformation( [ 2, 3, 4, 4 ] ), Transformation( [ 4, 1, 3, 3 ] ) ]
gap> SetName( smap, "smap" );
gap> T123 := SemigroupWithObjects( sgp2, [-13,-12,-11] );;
gap> shom := MagmaWithObjectsHomomorphism( S123, T123, smap, [-11,-12,-13] );;
gap> it12 := ImageElm( shom, t12 );;  [ t12, it12 ];
[ [Transformation( [ 1, 1, 2, 3 ] ) : -1 -> -2],
  [Transformation( [ 2, 2, 4, 1 ] ) : -13 -> -12] ]
gap> is23 := ImageElm( shom, s23 );;  [ s23, is23 ];
[ [Transformation( [ 2, 2, 3, 3 ] ) : -2 -> -3],
  [Transformation( [ 1, 1, 4, 4 ] ) : -12 -> -11] ]
gap> ir31 := ImageElm( shom, r31 );;  [ r31, ir31 ];
[ [Transformation( [ 2, 3, 4, 4 ] ) : -3 -> -1],
  [Transformation( [ 4, 1, 3, 3 ] ) : -11 -> -13] ]
```

2.3 Homomorphisms to more than one piece

2.3.1 HomomorphismByUnion

▷ HomomorphismByUnion(*src, rng, homs*) (operation)
▷ IsInjectiveOnObjects(*mwohom*) (property)
▷ IsSurjectiveOnObjects(*mwohom*) (property)
▷ IsBijectiveOnObjects(*mwohom*) (property)
▷ IsEndomorphismWithObjects(*mwohom*) (property)
▷ IsAutomorphismWithObjects(*mwohom*) (property)

When $f : M \to N$ and N has more than one connected component, then f is a union of homomorphisms, one for each piece in the range.

Example

```
gap> N4 := UnionOfPieces( [ M78, T123 ] );
magma with objects having 2 pieces :-
1: semigroup with objects :-
    magma = sgp<t2,s2,r2>
  objects = [ -13, -12, -11 ]
2: M78
gap> h14 := HomomorphismByUnionNC( N1, N4, [ hom1, shom ] );
magma with objects homomorphism :
[ magma with objects homomorphism : M78 -> M78
    [ [ <mapping: m -> m >, [ -8, -7 ] ] ], magma with objects homomorphism :
    [ [ smap, [ -11, -12, -13 ] ] ] ]
gap> IsInjectiveOnObjects( h14 );
true
gap> IsSurjectiveOnObjects( h14 );
true
gap> IsBijectiveOnObjects( h14 );
true
gap> ImageElm( h14, t12 );
[Transformation( [ 2, 2, 4, 1 ] ) : -13 -> -12]
gap> h45 := IsomorphismNewObjects( N4, [ [-103,-102,-101], [-108,-107] ] );
magma with objects homomorphism :
[ magma with objects homomorphism :
    [ [ IdentityMapping( m ), [ -108, -107 ] ] ],
  magma with objects homomorphism :
    [ [ IdentityMapping( sgp<t2,s2,r2> ), [ -103, -102, -101 ] ] ] ]
gap> N5 := Range( h45 );;  SetName( N5, "N5" );
gap> h15 := h14 * h45;
magma with objects homomorphism :
[ magma with objects homomorphism : [ [ <mapping: m -> m >, [ -108, -107 ] ] ]
    , magma with objects homomorphism : [ [ smap, [ -101, -102, -103 ] ] ] ]
gap> ImageElm( h15, t12 );
[Transformation( [ 2, 2, 4, 1 ] ) : -103 -> -102]
```

Chapter 3

Groupoids

A *groupoid* is a (mathematical) category in which every element is invertible. It consists of a set of *pieces*, each of which is a connected groupoid. (The usual terminology is 'connected component', but in GAP 'component' is used for 'record component'.)

A *single piece groupoid* is determined by a set of *objects* `obs` and an *object group* `grp`. The objects of a single piece groupoid are generally chosen to be consecutive negative integers, but any suitable ordered set is acceptable, and 'consecutive' is not a requirement. The object groups will usually be taken to be permutation groups, but pc-groups and fp-groups are also supported.

A *group* is a single piece groupoid with one object.

A *groupoid* is a set of one or more single piece groupoids, its *pieces*, and is represented as `IsGroupoidRep`, with attribute `PiecesOfGroupoid`.

A groupoid is *homogeneous* if it has two of more isomorphic pieces, with identical groups. The special case of *homogeneous, discrete* groupoids, where each piece has a single object, is given its own representation. These groupoids are used in the XMod package as the source of many crossed modules of groupoids.

For the definitions of the standard properties of groupoids we refer to R. Brown's book "Topology" [Bro88], recently revised and reissued as "Topology and Groupoids" [Bro06].

3.1 Groupoids: their properties and attributes

3.1.1 SinglePieceGroupoid

▷ `SinglePieceGroupoid(`*grp, obs*`)` (operation)
▷ `Groupoid(`*args*`)` (function)
▷ `RootObject(`*gpd*`)` (operation)

As for magmas with objects, the simplest construction of a groupoid is as the direct product of a group and a complete graph. Some subgroupoids of such a groupoid do not have this simple form, and will be considered in a later section. As usual, the `RootObject` is the object with least label.

The global function `Groupoid` will normally find the appropriate constructor to call, the options being:

- the object group, a list of objects;
- a group being converted to a groupoid, a single object;

- a list of groupoids which have already been constructed.

Methods for ViewObj, PrintObj and Display are provided for groupoids and the other types of object in this package. Users are advised to supply names for all the groups and groupoids they construct.

Example

```
gap> s4 := Group( (1,2,3,4), (3,4) );;
gap> d8 := Subgroup( s4, [ (1,2,3,4), (1,3) ] );;
gap> SetName( s4, "s4" );  SetName( d8, "d8" );
gap> Gs4 := SinglePieceGroupoid( s4, [-15 .. -11] );
single piece groupoid: < s4, [ -15 .. -11 ] >
gap> Gd8 := Groupoid( d8, [-9,-8,-7] );
single piece groupoid: < d8, [ -9, -8, -7 ] >
gap> c6 := Group( (5,6,7)(8,9) );;
gap> SetName( c6, "c6" );
gap> Gc6 := DomainWithSingleObject( c6, -6 );
single piece groupoid: < c6, [ -6 ] >
gap> SetName( Gs4, "Gs4" );  SetName( Gd8, "Gd8" );  SetName( Gc6, "Gc6" );
```

3.1.2 IsPermGroupoid

▷ IsPermGroupoid(*gpd*) (property)
▷ IsPcGroupoid(*gpd*) (property)
▷ IsFpGroupoid(*gpd*) (property)

A groupoid is a permutation groupoid if all its pieces have permutation groups. Most of the examples in this chapter are permutation groupoids, but in principle any type of group known to GAP may be used. In the following example Gf2 is an fp-groupoid, while Gq8 is a pc-groupoid.

Example

```
gap> f2 := FreeGroup( 2 );;
gap> Gf2 := Groupoid( f2, -22 );;
gap> SetName( f2, "f2" );  SetName( Gf2, "Gf2" );
gap> q8 := SmallGroup( 8, 4 );;
gap> Gq8 := Groupoid( q8, [ -28, -27 ] );;
gap> SetName( q8, "q8" );  SetName( Gq8, "Gq8" );
gap> [ IsFpGroupoid( Gf2 ), IsPcGroupoid( Gq8 ), IsPermGroupoid( Gs4 ) ];
[ true, true, true ]
```

3.1.3 UnionOfPieces

▷ UnionOfPieces(*pieces*) (operation)
▷ ReplaceOnePieceInUnion(*gpd, old_piece, new_piece*) (operation)
▷ Size(*gpd*) (attribute)

When a groupoid consists of two or more pieces, we require their object lists to be disjoint. UnionOfPieces from section 1.4 is also used for groupoids. The pieces are sorted by the least object

in their object lists. The `ObjectList` is the sorted concatenation of the objects in the pieces. The `Size` of a groupoid is the number of its elements which, for a single piece groupoid, is the product of the size of the group with the square of the number of objects.

Example

```
gap> U3 := UnionOfPieces( [ Gs4, Gd8, Gc6 ] );;
gap> SetName( U3, "Gs4+Gd8+Gc6" );
gap> Display( U3 );
groupoid with 3 pieces:
< objects: [ -15 .. -11 ]
    group: s4 = <[ (1,2,3,4), (3,4) ]> >
< objects: [ -9, -8, -7 ]
    group: d8 = <[ (1,2,3,4), (1,3) ]> >
< objects: [ -6 ]
    group: c6 = <[ (5,6,7)(8,9) ]> >
gap> Pieces( U3 );
[ Gs4, Gd8, Gc6 ]
gap> ObjectList( U3 );
[ -15, -14, -13, -12, -11, -9, -8, -7, -6 ]
gap> U2 := Groupoid( [ Gf2, Gq8 ] );;
gap> [ Size(Gs4), Size(Gd8), Size(Gc6), Size(U3), Size(U2) ];
[ 600, 72, 6, 678, infinity ]
gap> U5 := UnionOfPieces( [ U3, Gf2, Gq8 ] );
groupoid with 5 pieces:
[ Gq8, Gf2, Gs4, Gd8, Gc6 ]
gap> Display( U5 );
groupoid with 5 pieces:
< objects: [ -28, -27 ]
    group: q8 = <[ f1, f2, f3 ]> >
< objects: [ -22 ]
    group: f2 = <[ f1, f2 ]> >
< objects: [ -15, -14, -13, -12, -11 ]
    group: s4 = <[ (1,2,3,4), (3,4) ]> >
< objects: [ -9, -8, -7 ]
    group: d8 = <[ (1,2,3,4), (1,3) ]> >
< objects: [ -6 ]
    group: c6 = <[ (5,6,7)(8,9) ]> >
gap> V5 := Groupoid( [ U2, U3 ] );
groupoid with 5 pieces:
[ Gq8, Gf2, Gs4, Gd8, Gc6 ]
gap> U5 = V5;
true
gap> Rs4 := Groupoid( s4, [-30,-29] );;
gap> SetName( Rs4, "Rs4" );
gap> W5 := ReplaceOnePieceInUnion( U5, Gs4, Rs4 );
groupoid with 5 pieces:
[ Rs4, Gq8, Gf2, Gd8, Gc6 ]
```

3.1.4 HomogeneousGroupoid

▷ HomogeneousGroupoid(*gpd, oblist*) (operation)
▷ HomogeneousDiscreteGroupoid(*gp, obs*) (operation)

Special functions are provided for the case where a groupoid consists of identical connected components. We call these groupoids *homogeneous*. The operation HomogeneousGroupoid is used when the components each contain more than one object, while the operation HomogeneousDiscreteGroupoid is used when the components each have a single object. Both types of groupoid have the property IsHomogeneousDomainWithObjects, and in the latter case a separate representation IsHomogeneousDiscreteGroupoidRep is used.

Example

```
gap> Hd8 := HomogeneousGroupoid( Gd8, [ [-12,-11,-10], [-16,-15,-14] ] );
homogeneous groupoid with 2 pieces:
1:  single piece groupoid: < d8, [ -16, -15, -14 ] >
2:  single piece groupoid: < d8, [ -12, -11, -10 ] >
gap> IsHomogeneousDomainWithObjects(Hd8);
true
gap> Hc6 := HomogeneousDiscreteGroupoid( c6, [-7..-4] );
homogeneous, discrete groupoid: < c6, [ -7 .. -4 ] >
gap> RepresentationsOfObject(Gd8);
[ "IsComponentObjectRep", "IsAttributeStoringRep", "IsMWOSinglePieceRep" ]
gap> RepresentationsOfObject(Hd8);
[ "IsComponentObjectRep", "IsAttributeStoringRep", "IsPiecesRep" ]
gap> RepresentationsOfObject(Hc6);
[ "IsComponentObjectRep", "IsAttributeStoringRep",
  "IsHomogeneousDiscreteGroupoidRep" ]
gap> KnownTruePropertiesOfObject(Hc6);
[ "CanEasilyCompareElements", "CanEasilySortElements", "IsDuplicateFree",
  "IsAssociative", "IsCommutative", "IsDiscreteDomainWithObjects",
  "IsHomogeneousDomainWithObjects" ]
```

3.2 Groupoid elements; stars; costars and homsets

3.2.1 GroupoidElement

▷ GroupoidElement(*gpd, elt, tail, head*) (operation)
▷ IsElementOfGroupoid(*elt*) (property)
▷ ElementOfArrow(*elt*) (operation)
▷ TailOfArrow(*elt*) (operation)
▷ HeadOfArrow(*elt*) (operation)

A *groupoid element* e is a triple consisting of a group element, ElementOfArrow(e) or e![1]; the tail (source) object, TailOfArrow(e) or e![2]; and the head (target) object, HeadOfArrow(e) or e![3]. Note that GroupoidElement is essentially a synonym, in the groupoid case, for MultiplicativeElementWithObjects.

As for elements in any magma with objects, groupoid elements have a *partial composition*: two elements may be multiplied when the head of the first coincides with the tail of the second.

Example

```
gap> e1 := GroupoidElement( Gd8, (1,2,3,4), -9, -8 );
[(1,2,3,4) : -9 -> -8]
gap> e2 := GroupoidElement( Gd8, (1,3), -8, -7 );;
gap> Print( [ Arrowelt(e2), Arrowtail(e2), Arrowhead(e2) ], "\n" );
[ (1,3), -8, -7 ]
gap> prod := e1*e2;
[(1,2)(3,4) : -9 -> -7]
gap> e3 := GroupoidElement( Gd8, (2,4), -7, -9 );;
gap> cycle := prod*e3;
[(1,4,3,2) : -9 -> -9]
gap> cycle^2;
[(1,3)(2,4) : -9 -> -9]
gap> Order(cycle);
4
```

3.2.2 IdentityElement

▷ `IdentityElement(`*gpd, obj*`)` (operation)

The identity groupoid element 1_o of G at object o is $(e : o \to o)$ where e is the identity element in the object group. The inverse e^{-1} of $e = (c : p \to q)$ is $(c^{-1} : q \to p)$, so that $e * e^{-1} = 1_p$ and $e^{-1} * e = 1_q$.

Example

```
gap> i8 := IdentityElement( Gd8, -8 );
[() : -8 -> -8]
gap> [ e1*i8, i8*e1, e1^-1];
[ [(1,2,3,4) : -9 -> -8], fail, [(1,4,3,2) : -8 -> -9] ]
```

3.2.3 ObjectStar

▷ `ObjectStar(`*gpd, obj*`)` (operation)
▷ `ObjectCostar(`*gpd, obj*`)` (operation)
▷ `Homset(`*gpd, tail, head*`)` (operation)

The *star* at `obj` is the set of groupoid elements which have `obj` as tail, while the *costar* is the set of elements which have `obj` as head. The *homset* from `obj1` to `obj2` is the set of elements with the specified tail and head, and so is bijective with the elements of the group. Thus every star and every costar is a union of homsets. The identity element at an object is a left identity for the star and a right identity for the costar at that object.

In order not to create unneccessary long lists, these operations return objects of type `IsHomsetCosetsRep` for which an `Iterator` is provided. (An `Enumerator` is not yet implemented.)

Example

```
gap> star9 := ObjectStar( Gd8, -9 );
<star at [ -9 ] with group d8>
gap> Size( star9 );
24
gap> for e in star9 do
>      if ( Order( e![1] ) = 4 ) then Print( e, "\n" ); fi;
>    od;
[(1,4,3,2) : -9 -> -9]
[(1,4,3,2) : -9 -> -8]
[(1,4,3,2) : -9 -> -7]
[(1,2,3,4) : -9 -> -9]
[(1,2,3,4) : -9 -> -8]
[(1,2,3,4) : -9 -> -7]
gap> costar6 := ObjectCostar( Gc6, -6 );
<costar at [ -6 ] with group c6>
gap> Size( costar6 );
6
gap> hsetq8 := Homset( Gq8, -28, -27 );
<homset -28 -> -27 with group q8>
gap> for e in hsetq8 do Print(e,"\n"); od;
[<identity> of ... : -28 -> -27]
[f3 : -28 -> -27]
[f2 : -28 -> -27]
[f2*f3 : -28 -> -27]
[f1 : -28 -> -27]
[f1*f3 : -28 -> -27]
[f1*f2 : -28 -> -27]
[f1*f2*f3 : -28 -> -27]
```

3.3 Subgroupoids

3.3.1 Subgroupoid

▷ Subgroupoid(*args*) (function)
▷ SubgroupoidByPieces(*gpd, obhoms*) (operation)
▷ IsSubgroupoid(*gpd, sgpd*) (operation)
▷ FullSubgroupoid(*gpd, obs*) (operation)
▷ MaximalDiscreteSubgroupoid(*gpd*) (attribute)
▷ DiscreteSubgroupoid(*gpd, sgps, obs*) (operation)
▷ FullTrivialSubgroupoid(*gpd*) (attribute)
▷ DiscreteTrivialSubgroupoid(*gpd*) (attribute)
▷ IsWide(*gpd, sgpd*) (operation)

A *subgroupoid* sgpd of gpd has as objects some subset of the objects of gpd. It is *wide* if all the objects are included. It is *full* if, for any two objects in sgpd, the Homset is the same as that in gpd. The elements of sgpd are a subset of those of gpd, closed under multiplication and with tail and head in the chosen object set.

There are a variety of constructors for a subgroupoid of a groupoid, and the most general is the operation `SubgroupoidByPieces`. Its two parameters are a groupoid and a list of pieces, each piece being specified as a list `[sgp,obs]`, where `sgp` is a subgroup of the root group in that piece, and `obs` is a subset of the objects in that piece. The `FullSubgroupoid` of a groupoid `gpd` on a subset `obs` of its objects contains all the elements of `gpd` with tail and head in `obs`. A subgroupoid is *discrete* if it is a union of groups. The `MaximalDiscreteSubgroupoid` of `gpd` is the union of all the single-object full subgroupoids of `gpd`. A *trivial subgroupoid* has trivial object groups, but need not be discrete. A single piece trivial groupoid is sometimes called a *tree groupoid*. (The term *identity subgroupoid* was used in versions up to 1.14.) The global function `Subgroupoid` should call the appropriate operation.

Example

```
gap> c4 := Subgroup( d8, [ (1,2,3,4) ] );;
gap> k4 := Subgroup( d8, [ (1,2)(3,4), (1,3)(2,4) ] );;
gap> SetName( c4, "c4" );  SetName( k4, "k4" );
gap> Ud8 := Subgroupoid( Gd8, [ [ k4, [-9] ], [ c4, [-8,-7] ] ] );;
gap> SetName( Ud8, "Ud8" );
gap> Display( Ud8 );
groupoid with 2 pieces:
< objects: [ -9 ]
    group: k4 = <[ (1,2)(3,4), (1,3)(2,4) ]> >
< objects: [ -8, -7 ]
    group: c4 = <[ (1,2,3,4) ]> >
gap> [ Parent( Ud8 ), IsWide( Gd8, Ud8 ) ];
[ Gd8, true ]
gap> genf2b := List( GeneratorsOfGroup(f2), g -> g^2 );
[ f1^2, f2^2 ]
gap> f2b := Subgroup( f2, genf2b );;
gap> SubgroupoidByPieces( U2, [ [q8,[-27]], [f2b,[-22]] ] );
groupoid with 2 pieces:
1:  single piece groupoid: < q8, [ -27 ] >
2:  single piece groupoid: < Group( [ f1^2, f2^2 ] ), [ -22 ] >
gap> IsSubgroupoid( Gf2, Groupoid( f2b, [-22] ) );
true
gap> FullSubgroupoid( U3, [-7,-6] );
groupoid with 2 pieces:
1:  single piece groupoid: < d8, [ -7 ] >
2:  single piece groupoid: < c6, [ -6 ] >
gap> DiscreteSubgroupoid( U3, [ c4, k4 ], [-9,-7] );
groupoid with 2 pieces:
1:  single piece groupoid: < c4, [ -9 ] >
2:  single piece groupoid: < k4, [ -7 ] >
gap> FullTrivialSubgroupoid( Ud8 );
groupoid with 2 pieces:
1:  single piece groupoid: < id(k4), [ -9 ] >
2:  single piece groupoid: < id(c4), [ -8, -7 ] >
gap> MaximalDiscreteSubgroupoid(U2);
groupoid with 3 pieces:
1:  single piece groupoid: < q8, [ -28 ] >
2:  single piece groupoid: < q8, [ -27 ] >
3:  single piece groupoid: < f2, [ -22 ] >
```

3.3.2 SubgroupoidWithRays

▷ SubgroupoidWithRays(*gpd, sgp, rays*) (operation)
▷ RaysOfGroupoid(*gpd*) (operation)

If groupoid G is of type IsDirectProductWithCompleteGraph with group g and n objects, then a typical wide subgroupoid H of G is formed by choosing a subgroup h of g to be the object group at the root object q, and an element $r : q \to p$ for each of the objects p. The chosen loop element at q must be the identity element. These n elements are called the *rays* of the subgroupoid. The elements in the homset from p to p' have the form $r^{-1}xr'$ for all rays r, r' and all x in h.

In the following example a subgroupoid with rays is to be constructed on four of the five objects. It is therefore necessary to construct the full subgroupoid on these four objects first.

It is also possible to construct a subgroupoid with rays of a subgroupoid with rays.

Note that the function Ancestor provides an iteration of Parent.

Example

```
gap> Hs4 := FullSubgroupoid( Gs4, [-14,-13,-12] );;
gap> SetName( Hs4, "Hs4" );
gap> Hd8a := SubgroupoidWithRays( Hs4, d8, [(),(2,3),(3,4)] );
single piece groupoid with rays: < d8, [ -14, -13, -12 ], [ (), (2,3), (3,4)
 ] >
gap> hs1413 := Homset( Hd8a, -14, -13 );
<homset -14 -> -13 with group d8>
gap> for e in hs1413 do  Print(e,", "); od;  Print( "\n");
[(2,3) : -14 -> -13], [(1,2,4,3) : -14 -> -13], [(1,4,2) : -14 -> -13], [
(1,3,4) : -14 -> -13], [(2,4,3) : -14 -> -13], [(1,2,3) : -14 -> -13], [
(1,4) : -14 -> -13], [(1,3,4,2) : -14 -> -13],
gap> Hd8b := SubgroupoidWithRays( Hs4, d8, [(),(1,2,3),(1,2,4)] );
single piece groupoid with rays: < d8, [ -14, -13, -12 ],
[ (), (1,2,3), (1,2,4) ] >
gap> Hd8a = Hd8b;
true
gap> RaysOfGroupoid( Hd8b );
[ (), (1,2,3), (1,2,4) ]
gap> Parent( Hd8a );
Hs4
gap> Ancestor( Hd8a );
Gs4
gap> Fd8a := FullSubgroupoid( Hd8a, [-14,-13]);
single piece groupoid with rays: < d8, [ -14, -13 ], [ (), (2,3) ] >
gap> Fd8b := FullSubgroupoid( Hd8a, [-13,-12]);
single piece groupoid with rays: < Group( [ (1,3,2,4), (1,2) ] ),
[ -13, -12 ], [ (), (2,4,3) ] >
gap> Fd8a := FullSubgroupoid( Hd8a, [-13,-12] );
single piece groupoid with rays: < Group( [ (1,3,2,4), (1,2) ] ),
[ -13, -12 ], [ (), (2,4,3) ] >
gap> Kd8a := SubgroupoidWithRays( Fd8a, k4, [ (), (1,3) ] );
single piece groupoid with rays: < k4, [ -13, -12 ], [ (), (1,3) ] >
```

3.4 Left, right and double cosets

3.4.1 RightCoset

▷ RightCoset(*G, U, elt*) (operation)
▷ RightCosetRepresentatives(*G, U*) (operation)
▷ LeftCoset(*G, U, elt*) (operation)
▷ LeftCosetRepresentatives(*G, U*) (operation)
▷ LeftCosetRepresentativesFromObject(*G, U, obj*) (operation)
▷ DoubleCoset(*G, U, elt, V*) (operation)
▷ DoubleCosetRepresentatives(*G, U, V*) (operation)

If U is a wide subgroupoid of G, the *right cosets* Ug of U in G are the equivalence classes for the relation on the elements of G where $g1$ is related to $g2$ if and only if $g2 = u * g1$ for some element u of U. The right coset containing g is written Ug. These right cosets partition the costars of G and, in particular, the costar $U1\_o$ of U at object o, so that (unlike groups) U is itself a coset only when G has a single object.

The *right coset representatives* for U in G form a list containing one groupoid element for each coset where, in a particular piece of U, the group element chosen is the right coset representative of the group of U in the group of G.

Similarly, the *left cosets* gU refine the stars of G, while *double cosets* are unions of left cosets and right cosets. The operation LeftCosetRepresentativesFromObject(G, U, obj) is used in Chapter 4, and returns only those representatives which have tail at obj.

As with stars and homsets, these cosets are implemented with representation IsHomsetCosetsRep and provided with an iterator. Note that, when U has more than one piece, cosets may have differing lengths.

Example

```
gap> re2 := RightCoset( Gd8, Ud8, e2 );
RightCoset(single piece groupoid: < c4, [ -8, -7 ] >,[(1,3) : -8 -> -7])
gap> for x in re2 do Print( x, "\n" ); od;
[(1,3) : -8 -> -7]
[(1,3) : -7 -> -7]
[(2,4) : -8 -> -7]
[(2,4) : -7 -> -7]
[(1,4)(2,3) : -8 -> -7]
[(1,4)(2,3) : -7 -> -7]
[(1,2)(3,4) : -8 -> -7]
[(1,2)(3,4) : -7 -> -7]
gap> rcrd8 := RightCosetRepresentatives( Gd8, Ud8 );
[ [() : -9 -> -9], [() : -9 -> -8], [() : -9 -> -7], [(2,4) : -9 -> -9],
  [(2,4) : -9 -> -8], [(2,4) : -9 -> -7], [() : -8 -> -9], [() : -8 -> -8],
  [() : -8 -> -7], [(2,4) : -8 -> -9], [(2,4) : -8 -> -8], [(2,4) : -8 -> -7]
 ]
gap> lcr7 := LeftCosetRepresentativesFromObject( Gd8, Ud8, -7 );
[ [() : -7 -> -9], [(2,4) : -7 -> -9], [() : -7 -> -8], [(2,4) : -7 -> -8] ]
```

3.5 Conjugation

3.5.1 ConjugateGroupoidElement

▷ `ConjugateGroupoidElement(e1, e)` (operation)

When $e = (c : p \to q)$ conjugation by e is the groupoid automorphism defined as follows. In the case $p \neq q$,

- objects p, q are interchanged, and the remaining objects are fixed;
- loops at p, q: $(b : p \to p) \mapsto (b^c : q \to q)$ and $(b : q \to q) \mapsto (b^{c^{-1}} : p \to p)$;
- arrows between p and q: $(b : p \to q) \mapsto (c^{-1}bc^{-1} : q \to p)$ and $(b : q \to p) \mapsto (cbc : p \to q)$;
- costar at p, q: $(b : r \to p) \mapsto (bc : r \to q)$ and $(b : r \to q) \mapsto (bc^{-1} : r \to p)$;
- star at p, q: $(b : p \to r) \mapsto (c^{-1}b :\to q)$ and $(b : q \to r) \mapsto (cb : p \to r)$;
- the remaining arrows are unchanged.

In the case $p = q$ all the objects are fixed; loops at p are conjugated by c, $(b : p \to p) \mapsto (b^c : p \to p)$; the rest of the costar and star at p are permuted, $(b : r \to p) \mapsto (bc : r \to p)$ and $(b : p \to r) \mapsto (c^{-1}b : p \to r)$; the remaining arrows are unchanged.

The details of this construction may be found in [AW10].

(Note that it is more desirable to use the command `e1^e2`, but it has not yet been possible to get this to work!)

Example

```
gap> x := GroupoidElement( Gd8, (1,3), -9, -9 );;
gap> y := GroupoidElement( Gd8, (1,2,3,4), -8, -9 );;
gap> z := GroupoidElement( Gd8, (1,2)(3,4), -9, -7 );;
gap> w := GroupoidElement( Gd8, (1,2,3,4), -7, -8 );;
gap> ##  conjugation with elements x, y, and z in Gd8:
gap> ConjugateGroupoidElement(x,y);
[(2,4) : -8 -> -8]
gap> ConjugateGroupoidElement(x,z);
[(2,4) : -7 -> -7]
gap> ConjugateGroupoidElement(y,x);
[() : -8 -> -9]
gap> ConjugateGroupoidElement(y,z);
[(2,4) : -8 -> -7]
gap> ConjugateGroupoidElement(z,x);
[(1,4,3,2) : -9 -> -7]
gap> ConjugateGroupoidElement(z,y);
[(2,4) : -8 -> -7]
gap> ConjugateGroupoidElement(w,y);
[(1,3)(2,4) : -7 -> -9]
gap> ConjugateGroupoidElement(w,z);
[(1,3) : -9 -> -8]
```

3.5.2 SinglePieceGroupoidByGenerators

▷ SinglePieceGroupoidByGenerators(*parent, gens*) (operation)

A set of groupoid elements generates a groupoid by taking all possible products and inverses. So far, the only implementation is for the case of loops generating a group at an object o andf a set of rays from o, where o is *not* the least object. A suitably large supergroupoid, which must be a direct product with a complete graph, should be provided. This is the case needed for ConjugateGroupoid in the following section. Other cases will be added as time permits.

Example

```
gap> u := GroupoidElement( Gs4, (1,2,3), -15, -13 );
[(1,2,3) : -15 -> -13]
gap> gensa := GeneratorsOfGroupoid( Hd8a );
[ [(1,2,3,4) : -14 -> -14], [(1,3) : -14 -> -14], [(2,3) : -14 -> -13],
  [(3,4) : -14 -> -12] ]
gap> imsa := List( gensa, g -> ConjugateGroupoidElement( g, u ) );
[ [(1,2,3,4) : -14 -> -14], [(1,3) : -14 -> -14], [(1,3) : -14 -> -15],
  [(3,4) : -14 -> -12] ]
gap> C := SinglePieceGroupoidByGenerators( Gs4, imsa );
single piece groupoid with rays: < Group( [ (1,4,3,2), (1,3) ] ),
[ -15, -14, -12 ], [ (), (1,3), (1,4,3) ] >
```

3.5.3 ConjugateGroupoid

▷ ConjugateGroupoid(*gpd, e*) (operation)

When H is a subgroupoid of a groupoid G and e is an element of G, then the conjugate of H by e is the subgroupoid generated by the conjugates of the generators of H.

Example

```
gap> ConjugateGroupoid( Hd8a, u^-1 );
single piece groupoid with rays: < Group( [ (1,4,3,2), (1,3) ] ),
[ -15, -14, -12 ], [ (), (1,3), (1,4,3) ] >
```

More examples of all these operations may be found in the example file gpd/examples/gpd.g.

Chapter 4

Homomorphisms of Groupoids

A *homomorphism m* from a groupoid G to a groupoid H consists of a map from the objects of G to those of H together with a map from the elements of G to those of H which is compatible with tail and head and which preserves multiplication:

$$m(g1 : o1 \to o2) * m(g2 : o2 \to o3) \;=\; m(g1 * g2 : o1 \to o3).$$

Note that when a homomorphism is not injective on objects, the image of the source need not be a subgroupoid of the range. A simple example of this is given by a homomorphism from the two-object groupoid with trivial group to the free group $\langle a \rangle$ on one generator, when the image is $[1, a^n, a^{-n}]$ for some $n > 0$.

4.1 Homomorphisms from a connected groupoid

4.1.1 GroupoidHomomorphismFromSinglePiece

▷ `GroupoidHomomorphismFromSinglePiece(`*`src, rng, hom, imobs, imrays`*`)` (operation)
▷ `GroupoidHomomorphismByGroupHom(`*`src, rng, hom`*`)` (operation)
▷ `GroupoidHomomorphism(`*`args`*`)` (function)
▷ `InclusionMappingGroupoids(`*`gpd, sgpd`*`)` (operation)
▷ `RootObjectHomomorphism(`*`gpdhom`*`)` (attribute)

As usual, there are various homomorphism operations. The basic construction is a homomorphism $G \to H$ with G the direct product of a group and a complete graph. The homomorphism has attributes `Source`, `Range`, `ImagesOfObjects`, `PieceImages` and `RootObjectHomomorphism`. The input data consists of the source; the range; and

- a homomorphism `hom` from the root group of G to that of H;
- a list `imobs` of the images of the objects of G;
- a list `imrays` of the images of the rays of G.

Example

```
gap> gend12 := [ (15,16,17,18,19,20), (15,20)(16,19)(17,18) ];;
gap> d12 := Group( gend12 );;
gap> Gd12 := Groupoid( d12, [-37,-36,-35,-34] );;
```

```
gap> SetName( d12, "d12" );
gap> SetName( Gd12, "Gd12" );
gap> s3 := Subgroup( d12, [ (15,17,19)(16,18,20), (15,20)(16,19)(17,18) ] );;
gap> Gs3 := SubgroupoidByPieces( Gd12, [ [ s3, [-36,-35,-34] ] ] );;
gap> SetName( s3, "s3" );
gap> SetName( Gs3, "Gs3" );
gap> gend8 := GeneratorsOfGroup( d8 );;
gap> imhd8 := [ ( ), (15,20)(16,19)(17,18) ];;
gap> hd8 := GroupHomomorphismByImages( d8, s3, gend8, imhd8 );;
gap> homd8 := GroupoidHomomorphismByGroupHom( Gd8, Gs3, hd8 );
groupoid homomorphism : Gd8 -> Gs3
[ [ GroupHomomorphismByImages( d8, s3, [ (1,2,3,4), (1,3) ],
        [ (), (15,20)(16,19)(17,18) ] ), [ -36, -35, -34 ], [ (), (), () ] ] ]
gap> e2; ImageElm( homd8, e2 );
[(1,3) : -8 -> -7]
[(15,20)(16,19)(17,18) : -35 -> -34]
gap> incGs3 := InclusionMappingGroupoids( Gd12, Gs3 );;
gap> ihomd8 := homd8 * incGs3;;
gap> IsBijectiveOnObjects( ihomd8 );
false
gap> Display( ihomd8 );
 groupoid mapping: [ Gd8 ] -> [ Gd12 ]
root homomorphism: [ [ (1,2,3,4), (1,3) ], [ (), (15,20)(16,19)(17,18) ] ]
images of objects: [ -36, -35, -34 ]
   images of rays: [ (), (), () ]
```

4.2 Homomorphisms to a connected groupoid

4.2.1 HomomorphismToSinglePiece

▷ HomomorphismToSinglePiece(*src*, *rng*, *pieces*) (operation)

When G is made up of two or more pieces, all of which get mapped to a connected groupoid, we have a *homomorphism to a single piece*. The third input parameter in this case is a list of the PieceImages of the individual homomorphisms *from* the single pieces. See section 2.1 for the corresponding operation on homomorphisms of magmas with objects.

In the following example the source has three pieces, and one of the component homomorphisms is an IdentityMapping .

Example

```
gap> hc6 := GroupHomomorphismByImages( c6, s3,
>            [(5,6,7)(8,9)], [(15,16)(17,20)(18,19)] );;
gap> Fs3 := FullSubgroupoid( Gs3, [ -35 ] );;
gap> SetName( Fs3, "Fs3" );
gap> homc6 := GroupoidHomomorphism( Gc6, Fs3, hc6 );;
gap> incFs3 := InclusionMappingGroupoids( Gs3, Fs3 );;
gap> ihomc6 := homc6 * incFs3;
groupoid homomorphism : Gc6 -> Gs3
[ [ GroupHomomorphismByImages( c6, s3, [ (5,6,7)(8,9) ],
```

```
        [ (15,16)(17,20)(18,19) ] ), [ -35 ], [ () ] ] ]
gap> idGs3 := IdentityMapping( Gs3 );;
gap> V3 := ReplaceOnePieceInUnion( U3, Gs4, Gs3 );
groupoid with 3 pieces:
[ Gs3, Gd8, Gc6 ]
gap> images3 := [ PieceImages( idGs3 )[1],
>                 PieceImages( homd8 )[1],
>                 PieceImages( ihomc6 )[1] ];;
gap> homV3 := HomomorphismToSinglePiece( V3, Gs3, images3 );;
gap> Display( homV3 );
homomorphism to single piece magma with pieces:
(1): [ Gs3 ] -> [ Gs3 ]
magma mapping: [ [ (15,17,19)(16,18,20), (15,20)(16,19)(17,18) ],
  [ (15,17,19)(16,18,20), (15,20)(16,19)(17,18) ] ]
   object map: [ -36, -35, -34 ] -> [ -36, -35, -34 ]
(2): [ Gd8 ] -> [ Gs3 ]
magma mapping: [ [ (1,2,3,4), (1,3) ], [ (), (15,20)(16,19)(17,18) ] ]
   object map: [ -9, -8, -7 ] -> [ -36, -35, -34 ]
(3): [ Gc6 ] -> [ Gs3 ]
magma mapping: [ [ (5,6,7)(8,9) ], [ (15,16)(17,20)(18,19) ] ]
   object map: [ -6 ] -> [ -35 ]
```

4.3 Homomorphisms with more than one piece

4.3.1 HomomorphismByUnion

▷ HomomorphismByUnion(*src*, *rng*, *homs*) (operation)

As in section 2.3, when the range H has more than one connected component, a homomorphism is a union of homomorphisms, one for each piece.

Example

```
gap> isoq8 := IsomorphismNewObjects( Gq8, [-38,-37] );
groupoid homomorphism :
[
  [ IdentityMapping( q8 ), [ -38, -37 ],
      [ <identity> of ..., <identity> of ... ] ] ]
gap> Gq8b := Range( isoq8 );;
gap> SetName( Gq8b, "Gq8b" );
gap> V4 := UnionOfPieces( [ V3, Gq8 ] );
groupoid with 4 pieces:
[ Gs3, Gq8, Gd8, Gc6 ]
gap> SetName( V4, "V4" );
gap> Vs3q8b := UnionOfPieces( [ Gs3, Gq8b ] );
gap> SetName( Vs3q8b, "Vs3q8b" );
gap> hom4 := HomomorphismByUnion( V4, Vs3q8b, [ homV3, isoq8 ] );;
gap> PiecesOfMapping( hom4 );
[ groupoid homomorphism : Gq8 -> Gq8b
    [ [ IdentityMapping( q8 ), [ -38, -37 ],
          [ <identity> of ..., <identity> of ... ] ] ],
```

```
  groupoid homomorphism :
    [ [ IdentityMapping( s3 ), [ -36, -35, -34 ], [ (), (), () ] ],
      [ GroupHomomorphismByImages( d8, s3, [ (1,2,3,4), (1,3) ],
            [ (), (15,20)(16,19)(17,18) ] ), [ -36, -35, -34 ],
          [ (), (), () ] ],
      [ GroupHomomorphismByImages( c6, s3, [ (5,6,7)(8,9) ],
            [ (15,16)(17,20)(18,19) ] ), [ -35 ], [ () ] ] ] ]
```

4.4 Groupoid automorphisms

4.4.1 GroupoidAutomorphismByObjectPerm

▷ `GroupoidAutomorphismByObjectPerm(`*gpd, imobs*`)` (operation)
▷ `GroupoidAutomorphismByGroupAuto(`*gpd, gpauto*`)` (operation)
▷ `GroupoidAutomorphismByRayImages(`*gpd, imrays*`)` (operation)

We first describe automorphisms a of a groupoid G where G is the direct product of a group g and a complete graph. The group of automorphisms is generated by three types of automorphism:

- a permutation of the n objects;
- an automorphism of the root group g;
- a choice of image for each ray: $a(1 : o_1 \to o_i) = (g_i : o_1 \to o_i)$ for $i \neq 1$.

Example

```
gap> a4 := Subgroup( s4, [(1,2,3),(2,3,4)] );;
gap> SetName( a4, "a4" );
gap> gensa4 := GeneratorsOfGroup( a4 );;
gap> Ga4 := SubgroupoidByPieces( Gs4, [ [a4, [-15,-13,-11]] ] );
single piece groupoid: < a4, [ -15, -13, -11 ] >
gap> SetName( Ga4, "Ga4" );
gap> aut1 := GroupoidAutomorphismByObjectPerm( Ga4, [-13,-11,-15] );;
gap> Display( aut1 );
 groupoid mapping: [ Ga4 ] -> [ Ga4 ]
root homomorphism: [ [ (1,2,3), (2,3,4) ], [ (1,2,3), (2,3,4) ] ]
images of objects: [ -13, -11, -15 ]
   images of rays: [ (), (), () ]
gap> h2 := GroupHomomorphismByImages( a4, a4, gensa4, [(2,3,4), (1,3,4)] );;
gap> aut2 := GroupoidAutomorphismByGroupAuto( Ga4, h2 );;
gap> Display( aut2 );
 groupoid mapping: [ Ga4 ] -> [ Ga4 ]
root homomorphism: [ [ (1,2,3), (2,3,4) ], [ (2,3,4), (1,3,4) ] ]
images of objects: [ -15, -13, -11 ]
   images of rays: [ (), (), () ]
gap> im3 := [(), (1,3,2), (2,4,3)];;
gap> aut3 := GroupoidAutomorphismByRayImages( Ga4, im3 );;
gap> Display( aut3 );
 groupoid mapping: [ Ga4 ] -> [ Ga4 ]
root homomorphism: [ [ (1,2,3), (2,3,4) ], [ (1,2,3), (2,3,4) ] ]
```

```
images of objects: [ -15, -13, -11 ]
   images of rays: [ (), (1,3,2), (2,4,3) ]
gap> aut123 := aut1*aut2*aut3;;
gap> Display( aut123 );
 groupoid mapping: [ Ga4 ] -> [ Ga4 ]
root homomorphism: [ [ (1,2,3), (2,3,4) ], [ (2,3,4), (1,3,4) ] ]
images of objects: [ -13, -11, -15 ]
   images of rays: [ (), (1,4,3), (1,2,3) ]
gap> inv123 := InverseGeneralMapping( aut123 );;
gap> Display( inv123 );
 groupoid mapping: [ Ga4 ] -> [ Ga4 ]
root homomorphism: [ [ (2,3,4), (1,3,4) ], [ (1,2,3), (2,3,4) ] ]
images of objects: [ -11, -15, -13 ]
   images of rays: [ (), (1,2,4), (1,3,4) ]
gap> id123 := aut123 * inv123;;
gap> id123 = IdentityMapping( Ga4 );
true
```

The `AutomorphismGroup` of G is isomorphic to the quotient of $S_n \times A \times g^n$ by a subgroup isomorphic to g, where A is the automorphism group of g and S_n is the symmetric group on the n objects. This is one of the main topics in [AW10].

The current implementation is experimental, producing a *nice monomorphism* from the automorphism group to a pc-group, if one is available. However `ImageElm` at present only works on generating elements.

Example

```
gap> AGa4 := AutomorphismGroup( Ga4 );
<group with 10 generators>
gap> NGa4 := NiceObject( AGa4 );
Group([ f6, f3, f11*f12, f12, f2, f1, f4*f9, f4^2, f5*f9*f10*f11*f12, f5^2 ])
gap> MGa4 := NiceMonomorphism( AGa4 );;
gap> Size( AGa4 );
20736
gap> SetName( AGa4, "AGa4" );
gap> SetName( NGa4, "NGa4" );
gap> Print( MGa4, "\n" );
GroupHomomorphismByImages( AGa4, Group( [ f1, f2, f3, f4, f5, f6, f7, f8, f9,
  f10, f11, f12 ] ), [ magma with objects homomorphism : Ga4 -> Ga4
    [ [ InnerAutomorphism( a4, (2,4,3) ), [ -15, -13, -11 ], [ (), (), () ] ]
     ]
    , magma with objects homomorphism : Ga4 -> Ga4
    [ [ ConjugatorAutomorphism( a4, (3,4) ), [ -15, -13, -11 ],
          [ (), (), () ] ] ]
    , magma with objects homomorphism : Ga4 -> Ga4
    [ [ InnerAutomorphism( a4, (1,2)(3,4) ), [ -15, -13, -11 ],
          [ (), (), () ] ] ]
    , magma with objects homomorphism : Ga4 -> Ga4
    [ [ InnerAutomorphism( a4, (1,4)(2,3) ), [ -15, -13, -11 ],
          [ (), (), () ] ] ]
    , magma with objects homomorphism : Ga4 -> Ga4
    [ [ GroupHomomorphismByImages( a4, a4, [ (1,2,3), (2,3,4) ],
```

```
            [ (1,2,3), (2,3,4) ] ), [ -13, -11, -15 ], [ (), (), () ] ] ]
    , magma with objects homomorphism : Ga4 -> Ga4
    [ [ GroupHomomorphismByImages( a4, a4, [ (1,2,3), (2,3,4) ],
            [ (1,2,3), (2,3,4) ] ), [ -13, -15, -11 ], [ (), (), () ] ] ]
    , magma with objects homomorphism : Ga4 -> Ga4
    [ [ IdentityMapping( a4 ), [ -15, -13, -11 ], [ (), (1,2,3), () ] ] ]
    , magma with objects homomorphism : Ga4 -> Ga4
    [ [ IdentityMapping( a4 ), [ -15, -13, -11 ], [ (), (2,3,4), () ] ] ]
    , magma with objects homomorphism : Ga4 -> Ga4
    [ [ IdentityMapping( a4 ), [ -15, -13, -11 ], [ (), (), (1,2,3) ] ] ]
    , magma with objects homomorphism : Ga4 -> Ga4
    [ [ IdentityMapping( a4 ), [ -15, -13, -11 ], [ (), (), (2,3,4) ] ] ]
     ], [ f6, f3, f11*f12, f12, f2, f1, f4*f9, f4^2, f5*f9*f10*f11*f12, f5^2
 ] )
gap> ##  Now do some tests!
gap> mgi := MappingGeneratorsImages( MGa4 );;
gap> autgen := mgi[1];;
gap> pcgen := mgi[2];;
gap> ngen := Length( autgen );;
gap> ForAll( [1..ngen], i -> Order(autgen[i]) = Order(pcgen[i]) );
true
```

4.4.2 GroupoidAutomorphismByGroupAutos

▷ `GroupoidAutomorphismByGroupAutos(`*gpd, auts*`)` (operation)

Homogeneous, discrete groupoids are the second type of groupoid for which a method is provided for `AutomorphismGroup( gpd )`. This is used in the XMod package for constructing crossed modules of groupoids. The two types of generating automorphism are `GroupoidAutomorphismByGroupAutos`, which requires a list of group automorphisms, one for each object group, and `GroupoidAutomorphismByObjectPerm`, which permutes the objects.

Example

```
gap> Hs3 := HomogeneousDiscreteGroupoid( s3, [ -13..-10] );
homogeneous, discrete groupoid: < s3, [ -13 .. -10 ] >
gap> aut4 := GroupoidAutomorphismByObjectPerm( Hs3, [-12,-10,-11,-13] );
morphism from a homogeneous discrete groupoid:
[ -13, -12, -11, -10 ] -> [ -12, -10, -11, -13 ]
object homomorphisms:
IdentityMapping( s3 )
IdentityMapping( s3 )
IdentityMapping( s3 )
IdentityMapping( s3 )

gap> gens3 := GeneratorsOfGroup( s3 );;
gap> g1 := gens3[1];;
gap> g2 := gens3[2];;
gap> b1 := GroupHomomorphismByImages( s3, s3, gens3, [ g1, g2^g1 ] );;
gap> b2 := GroupHomomorphismByImages( s3, s3, gens3, [ g1^g2, g2 ] );;
gap> b3 := GroupHomomorphismByImages( s3, s3, gens3, [ g1^g2, g2^(g1*g2) ] );;
```

```
gap> b4 := GroupHomomorphismByImages( s3, s3, gens3, [ g1^(g2*g1), g2^g1 ] );;
gap> aut5 := GroupoidAutomorphismByGroupAutos( Hs3, [b1,b2,b3,b4] );
morphism from a homogeneous discrete groupoid:
[ -13, -12, -11, -10 ] -> [ -13, -12, -11, -10 ]
object homomorphisms:
GroupHomomorphismByImages( s3, s3,
[ (15,17,19)(16,18,20), (15,20)(16,19)(17,18) ],
[ (15,17,19)(16,18,20), (15,18)(16,17)(19,20) ] )
GroupHomomorphismByImages( s3, s3,
[ (15,17,19)(16,18,20), (15,20)(16,19)(17,18) ],
[ (15,19,17)(16,20,18), (15,20)(16,19)(17,18) ] )
GroupHomomorphismByImages( s3, s3,
[ (15,17,19)(16,18,20), (15,20)(16,19)(17,18) ],
[ (15,19,17)(16,20,18), (15,16)(17,20)(18,19) ] )
GroupHomomorphismByImages( s3, s3,
[ (15,17,19)(16,18,20), (15,20)(16,19)(17,18) ],
[ (15,19,17)(16,20,18), (15,18)(16,17)(19,20) ] )

gap> AHs3 := AutomorphismGroup( Hs3 );;  Size( AHs3 );
31104
gap> for z in GeneratorsOfGroup(AHs3) do Print(z); od;
morphism from a homogeneous discrete groupoid:
[ -13, -12, -11, -10 ] -> [ -13, -12, -11, -10 ]
object homomorphisms:
InnerAutomorphism( s3, (15,20)(16,19)(17,18) )
IdentityMapping( s3 )
IdentityMapping( s3 )
IdentityMapping( s3 )
morphism from a homogeneous discrete groupoid:
[ -13, -12, -11, -10 ] -> [ -13, -12, -11, -10 ]
object homomorphisms:
InnerAutomorphism( s3, (15,19,17)(16,20,18) )
IdentityMapping( s3 )
IdentityMapping( s3 )
IdentityMapping( s3 )
morphism from a homogeneous discrete groupoid:
[ -13, -12, -11, -10 ] -> [ -12, -11, -10, -13 ]
object homomorphisms:
IdentityMapping( s3 )
IdentityMapping( s3 )
IdentityMapping( s3 )
IdentityMapping( s3 )
morphism from a homogeneous discrete groupoid:
[ -13, -12, -11, -10 ] -> [ -12, -13, -11, -10 ]
object homomorphisms:
IdentityMapping( s3 )
IdentityMapping( s3 )
IdentityMapping( s3 )
IdentityMapping( s3 )
```

Chapter 5

Graphs of Groups and Groupoids

This package was originally designed to implement *graphs of groups*, a notion introduced by Serre in [Ser80]. It was only when this was extended to *graphs of groupoids* that the functions for groupoids, described in the previous chapters, were required. The methods described here are based on Philip Higgins' paper [Hig76]. For further details see Chapter 2 of [Moo01]. Since a graph of groups involves a directed graph, with a group associated to each vertex and arc, we first define digraphs with edges weighted by the generators of a free group.

5.1 Digraphs

5.1.1 FpWeightedDigraph

▷ `FpWeightedDigraph(`*verts, arcs*`)` (attribute)
▷ `IsFpWeightedDigraph(`*dig*`)` (attribute)
▷ `InvolutoryArcs(`*dig*`)` (attribute)

A *weighted digraph* is a record with two components: *vertices*, which are usually taken to be positive integers (to distinguish them from the objects in a groupoid); and *arcs*, which take the form of 3-element lists `[weight,tail,head]`. The *tail* and *head* are the two vertices of the arc. The *weight* is taken to be an element of a finitely presented group, so as to produce digraphs of type `IsFpWeightedDigraph`.

Example

```
gap> V1 := [ 5, 6 ];;
gap> fg1 := FreeGroup( "y" );;
gap> y := fg1.1;;
gap> A1 := [ [ y, 5, 6 ], [ y^-1, 6, 5 ] ];
gap> D1 := FpWeightedDigraph( fg1, V1, A1 );
weighted digraph with vertices: [ 5, 6 ]
and arcs: [ [ y, 5, 6 ], [ y^-1, 6, 5 ] ]
gap> inv1 := InvolutoryArcs( D1 );
[ 2, 1 ]
```

The example illustrates the fact that we require arcs to be defined in involutory pairs, as though they were inverse elements in a groupoid. We may in future decide just to give `[y,5,6]` as the data and

get the function to construct the reverse edge. The attribute `InvolutoryArcs` returns a list of the positions of each inverse arc in the list of arcs. In the second example the graph is a complete digraph on three vertices.

Example

```
gap> fg3 := FreeGroup( 3, "z" );;
gap> z1 := fg3.1;;  z2 := fg3.2;;  z3 := fg3.3;;
gap> V3 := [ 7, 8, 9 ];;
gap> A3 := [[z1,7,8],[z2,8,9],[z3,9,7],[z1^-1,8,7],[z2^-1,9,8],[z3^-1,7,9]];;
gap> D3 := FpWeightedDigraph( fg3, V3, A3 );
weighted digraph with vertices: [ 7, 8, 9 ]
and arcs: [ [ z1, 7, 8 ], [ z2, 8, 9 ], [ z3, 9, 7 ], [ z1^-1, 8, 7 ],
  [ z2^-1, 9, 8 ], [ z3^-1, 7, 9 ] ]
[gap> inv3 := InvolutoryArcs( D3 );
[ 4, 5, 6, 1, 2, 3 ]
```

5.2 Graphs of Groups

5.2.1 GraphOfGroups

▷ `GraphOfGroups(`*dig, gps, isos*`)` (operation)
▷ `DigraphOfGraphOfGroups(`*gg*`)` (attribute)
▷ `GroupsOfGraphOfGroups(`*gg*`)` (attribute)
▷ `IsomorphismsOfGraphOfGroups(`*gg*`)` (attribute)

A graph of groups is traditionally defined as consisting of:

- a digraph with involutory pairs of arcs;
- a *vertex group* associated to each vertex;
- a group associated to each pair of arcs;
- an injective homomorphism from each arc group to the group at the head of the arc.

We have found it more convenient to associate to each arc:

- a subgroup of the vertex group at the tail;
- a subgroup of the vertex group at the head;
- an isomorphism between these subgroups, such that each involutory pair of arcs determines inverse isomorphisms.

These two viewpoints are clearly equivalent.

In this implementation we require that all subgroups are of finite index in the vertex groups.

The three attributes provide a means of calling the three items of data in the construction of a graph of groups.

We shall be representing free products with amalgamation of groups and HNN extensions of groups, so we take as our first example the trefoil group with generators a, b and relation $a^3 = b^2$. For this we take digraph `D1` above with an infinite cyclic group at each vertex, generated by a and b respectively. The two subgroups will be generated by a^3 and b^2 with the obvious isomorphisms.

Example

```
gap> ## free vertex group at 5
gap> fa := FreeGroup( "a" );;
gap> a := fa.1;;
gap> SetName( fa, "fa" );
gap> hy := Subgroup( fa, [a^3] );;
gap> SetName( hy, "hy" );
gap> ## free vertex group at 6
gap> fb := FreeGroup( "b" );;
gap> b := fb.1;;
gap> SetName( fb, "fb" );
gap> hybar := Subgroup( fb, [b^2] );;
gap> SetName( hybar, "hybar" );
gap> ## isomorphisms between subgroups
gap> homy := GroupHomomorphismByImagesNC( hy, hybar, [a^3], [b^2] );;
gap> homybar := GroupHomomorphismByImagesNC( hybar, hy, [b^2], [a^3] );;
gap> ## defining graph of groups G1
gap> G1 := GraphOfGroups( D1, [fa,fb], [homy,homybar] );
Graph of Groups: 2 vertices; 2 arcs; groups [ fa, fb ]
gap> Display( G1 );
Graph of Groups with :-
    vertices: [ 5, 6 ]
        arcs: [ [ y, 5, 6 ], [ y^-1, 6, 5 ] ]
      groups: [ fa, fb ]
isomorphisms: [ [ [ a^3 ], [ b^2 ] ], [ [ b^2 ], [ a^3 ] ] ]
```

5.2.2 IsGraphOfFpGroups

▷ IsGraphOfFpGroups(*gg*) (property)
▷ IsGraphOfPcGroups(*gg*) (property)
▷ IsGraphOfPermGroups(*gg*) (property)

This is a list of properties to be expected of a graph of groups. In principle any type of group known to GAP may be used as vertex groups, though these types are not normally mixed in a single structure.

Example

```
gap> IsGraphOfFpGroups( G1 );
true
gap> IsomorphismsOfGraphOfGroups( G1 );
[ GroupHomomorphismByImages( hy, hybar, [ a^3 ], [ b^2 ] ),
  GroupHomomorphismByImages( hybar, hy, [ b^2 ], [ a^3 ] ) ]
```

5.2.3 RightTransversalsOfGraphOfGroups

▷ RightTransversalsOfGraphOfGroups(*gg*) (attribute)
▷ LeftTransversalsOfGraphOfGroups(*gg*) (attribute)

Computation with graph of groups words will require, for each arc subgroup ha, a set of representatives for the left cosets of ha in the tail vertex group. As already pointed out, we require subgroups of finite index. Since GAP prefers to provide right cosets, we obtain the right representatives first, and then invert them.

When the vertex groups are of type FpGroup we shall require normal forms for these groups, so we assume that such vertex groups are provided with Knuth Bendix rewriting systems using functions from the main GAP library, (e.g. IsomorphismFpSemigroup).

Example

```
gap> RTG1 := RightTransversalsOfGraphOfGroups( G1 );
[ [ <identity ...>, a, a^2 ], [ <identity ...>, b ] ]
gap> LTG1 := LeftTransversalsOfGraphOfGroups( G1 );
[ [ <identity ...>, a^-1, a^-2 ], [ <identity ...>, b^-1 ] ]
```

5.3 Words in a Graph of Groups and their normal forms

5.3.1 GraphOfGroupsWord

▷ GraphOfGroupsWord(*gg, tv, list*) (operation)
▷ IsGraphOfGroupsWord(*w*) (property)
▷ GraphOfGroupsOfWord(*w*) (attribute)
▷ WordOfGraphOfGroupsWord(*w*) (attribute)
▷ GGTail(*w*) (attribute)
▷ GGHead(*w*) (attribute)

If G is a graph of groups with underlying digraph D, the following groupoids may be considered. First there is the free groupoid or path groupoid on D. Since we want each involutory pair of arcs to represent inverse elements in the groupoid, we quotient out by the relations y\^{}-1 = ybar to obtain PG(D). Secondly, there is the discrete groupoid VG(D), namely the union of all the vertex groups. Since these two groupoids have the same object set (the vertices of D) we can form A(G), the free product of PG(D) and VG(D) amalgamated over the vertices. For further details of this universal groupoid construction see [Moo01]. (Note that these groupoids are not implemented in this package.)

An element of A(G) is a graph of groups word which may be represented by a list of the form $w = [g_1, y_1, g_2, y_2, \ldots, g_n, y_n, g_{n+1}]$. Here each y_i is an arc of D; the head of y_{i-1} is a vertex v_i which is also the tail of y_i; and g_i is an element of the vertex group at v_i.

The attributes GGTail and GGHead are *temporary* names for the tail and head of a graph of groups word, and are likely to be replaced in future versions.

So a graph of groups word requires as data the graph of groups; the tail vertex for the word; and a list of arcs and group elements. We may specify each arc by its position in the list of arcs.

In the following example, where gw1 is a word in the trefoil graph of groups, the y_i are specified by their positions in A1. Both arcs are traversed twice, so the resulting word is a loop at vertex 5.

Example

```
gap> L1 := [ a^7, 1, b^-6, 2, a^-11, 1, b^9, 2, a^7 ];;
gap> gw1 := GraphOfGroupsWord( G1, 5, L1 );
```

```
(5)a^7.y.b^-6.y^-1.a^-11.y.b^9.y^-1.a^7(5)
gap> IsGraphOfGroupsWord( gw1 );
true
gap> [ GGTail(gw1), GGHead(gw1) ];
[ 5, 5 ]
gap> GraphOfGroupsOfWord(gw1);
Graph of Groups: 2 vertices; 2 arcs; groups [ fa, fb ]
gap> WordOfGraphOfGroupsWord( gw1 );
[ a^7, 1, b^-6, 2, a^-11, 1, b^9, 2, a^7 ]
```

5.3.2 ReducedGraphOfGroupsWord

▷ `ReducedGraphOfGroupsWord(w)` (operation)
▷ `IsReducedGraphOfGroupsWord(w)` (property)

A graph of groups word may be reduced in two ways, to give a normal form. Firstly, if part of the word has the form `[yi, identity, yibar]` then this subword may be omitted. This is known as a length reduction. Secondly there are coset reductions. Working from the left-hand end of the word, subwords of the form $[g_i, y_i, g_{i+1}]$ are replaced by $[t_i, y_i, m_i(h_i) * g_{i+1}]$ where $g_i = t_i * h_i$ is the unique factorisation of g_i as a left coset representative times an element of the arc subgroup, and m_i is the isomorphism associated to y_i. Thus we may consider a coset reduction as passing a subgroup element along an arc. The resulting normal form (if no length reductions have taken place) is then $[t_1, y_1, t_2, y_2, ..., t_n, y_n, k]$ for some k in the head group of y_n. For further details see Section 2.2 of [Moo01].

The reduction of the word `gw1` in our example includes one length reduction. The four stages of the reduction are as follows:

$$a^7b^{-6}a^{-11}b^9a^7 \mapsto a^{-2}b^0a^{-11}b^9a^7 \mapsto a^{-13}b^9a^7 \mapsto a^{-1}b^{-8}b^9a^7 \mapsto a^{-1}b^{-1}a^{10}.$$

Example

```
gap> nw1 := ReducedGraphOfGroupsWord( gw1 );
(5)a^-1.y.b^-1.y^-1.a^10(5)
```

5.4 Free products with amalgamation and HNN extensions

5.4.1 FreeProductWithAmalgamation

▷ `FreeProductWithAmalgamation(gp1, gp2, iso)` (operation)
▷ `IsFpaGroup(fpa)` (property)
▷ `GraphOfGroupsRewritingSystem(fpa)` (attribute)
▷ `NormalFormGGRWS(fpa, word)` (attribute)

As we have seen with the trefoil group example, graphs of groups can be used to obtain a normal form for free products with amalgamation $G_1 *_H G_2$ when G_1, G_2 both have rewrite systems, and H is of finite index in both G_1 and G_2.

When gp1 and gp2 are fp-groups, the operation FreeProductWithAmalgamation constructs the required fp-group. When the two groups are permutation groups, the IsomorphismFpGroup operation is called on both gp1 and gp2, and the resulting isomorphism is transported to one between the two new subgroups.

The attribute GraphOfGroupsRewritingSystem of fpa is the graph of groups which has underlying digraph D1, with two vertices and two arcs; the two groups as vertex groups; and the specified isomorphisms on the arcs. Despite the name, graphs of groups constructed in this way *do not* belong to the category IsRewritingSystem. This anomaly may be dealt with when time permits.

The example below shows a computation in the the free product of the symmetric s3 and the alternating a4, amalgamated over a cyclic subgroup c3.

Example

```
gap> ## set up the first group s3 and a subgroup c3=<a1>
gap> fg2 := FreeGroup( 2, "a" );;
gap> rel1 := [ fg2.1^3, fg2.2^2, (fg2.1*fg2.2)^2 ];;
gap> s3 := fg2/rel1;;
gap> gs3 := GeneratorsOfGroup(s3);;
gap> SetName( s3, "s3" );
gap> a1 := gs3[1];;  a2 := gs3[2];;
gap> H1 := Subgroup(s3,[a1]);;
gap> ## then the second group a4 and subgroup c3=<b1>
gap> f2 := FreeGroup( 2, "b" );;
gap> rel2 := [ f2.1^3, f2.2^3, (f2.1*f2.2)^2 ];;
gap> a4 := f2/rel2;;
gap> ga4 := GeneratorsOfGroup(a4);;
gap> SetName( a4, "a4" );
gap> b1 := ga4[1];  b2 := ga4[2];;
gap> H2 := Subgroup(a4,[b1]);;
gap> ## form the isomorphism and the fpa group
gap> iso := GroupHomomorphismByImages(H1,H2,[a1],[b1]);;
gap> fpa := FreeProductWithAmalgamation( s3, a4, iso );
<fp group on the generators [ fa1, fa2, fa3, fa4 ]>
gap> RelatorsOfFpGroup( fpa );
[ fa1^3, fa2^2, (fa1*fa2)^2, fa3^3, fa4^3, (fa3*fa4)^2, fa1*fa3^-1 ]
gap> gg1 := GraphOfGroupsRewritingSystem( fpa );;
gap> Display( gg1 );
Graph of Groups with :-
    vertices: [ 5, 6 ]
        arcs: [ [ y, 5, 6 ], [ y^-1, 6, 5 ] ]
      groups: [ s3, a4 ]
isomorphisms: [ [ [ a1 ], [ b1 ] ], [ [ b1 ], [ a1 ] ] ]
gap> LeftTransversalsOfGraphOfGroups( gg1 );
[ [ <identity ..>, a2^-1 ], [ <identity ..>, b2^-1, b1^-1*b2^-1, b1*b2^-1 ] ]
gap> ## choose a word in fpa and find its normal form
gap> gfpa := GeneratorsOfGraphOfGroups( fpa );;
gap> w2 := (gfpa[1]*gfpa[2]*gfpa[3]^gfpa[4])^3;
(fa1*fa2*fa4^-1*fa3*fa4)^3
gap> n2 := NormalFormGGRWS( fpa, w2 );
fa2*fa3*(fa4^-1*fa2)^2*fa3^-1*fa4*fa3^-1
```

5.4.2 HnnExtension

▷ HnnExtension(*gp, iso*) (operation)
▷ IsHnnGroup(*hnn*) (property)

For *HNN extensions*, the appropriate graph of groups has underlying digraph with just one vertex and one pair of loops, weighted with FpGroup generators z, z^{-1}. There is one vertex group G, two isomorphic subgroups H1,H2 of G, with the isomorphism and its inverse on the loops. The presentation of the extension has one more generator than that of G and corresponds to the generator z.

The functions GraphOfGroupsRewritingSystem and NormalFormGGRWS may be applied to hnn-groups as well as to fpa-groups.

In the example we take G=a4 and the two subgroups are cyclic groups of order 3.

Example

```
gap> H3 := Subgroup(a4,[b2]);;
gap> i23 := GroupHomomorphismByImages( H2, H3, [b1], [b2] );;
gap> hnn := HnnExtension( a4, i23 );
<fp group on the generators [ fe1, fe2, fe3 ]>
gap> phnn := PresentationFpGroup( hnn );;
gap> TzPrint( phnn );
#I  generators: [ fe1, fe2, fe3 ]
#I  relators:
#I  1.  3  [ 1, 1, 1 ]
#I  2.  3  [ 2, 2, 2 ]
#I  3.  4  [ 1, 2, 1, 2 ]
#I  4.  4  [ -3, 1, 3, -2 ]
gap> gg2 := GraphOfGroupsRewritingSystem( hnn );
Graph of Groups: 1 vertices; 2 arcs; groups [ a4 ]
gap> LeftTransversalsOfGraphOfGroups( gg2 );
[ [ <identity ...>, b2^-1, b1^-1*b2^-1, b1*b2^-1 ],
  [ <identity ...>, b1^-1, b1, b2^-1*b1 ] ]
gap> gh := GeneratorsOfGroup( hnn );;
gap> w3 := (gh[1]^gh[2])*gh[3]^-1*(gh[1]*gh[3]*gh[2]^2)^2*gh[3]*gh[2];
fe2^-1*fe1*fe2*fe3^-1*(fe1*fe3*fe2^2)^2*fe3*fe2
gap> n3 := NormalFormGGRWS( hnn, w3 );
(fe2*fe1*fe3)^2
```

Both fpa-groups and hnn-groups are provided with a record attribute, FpaInfo(fpa) and HnnInfo(hnn) respectively, storing the groups and isomorphisms involved in their construction.

Example

```
gap> fpainfo := FpaInfo( fpa );
rec( groups := [ s3, a4 ], positions := [ [ 1, 2 ], [ 3, 4 ] ],
  isomorphism := [ a1 ] -> [ b1 ] )
gap> hnninfo := HnnInfo( hnn );
rec( group := a4, isomorphism := [ b1 ] -> [ b2 ] )
```

5.5 GraphsOfGroupoids and their Words

5.5.1 GraphOfGroupoids

▷ GraphOfGroupoids(*dig, gpds, subgpds, isos*) (operation)
▷ IsGraphOfPermGroupoids(*gg*) (property)
▷ IsGraphOfFpGroupoids(*gg*) (property)
▷ GroupoidsOfGraphOfGroupoids(*gg*) (attribute)
▷ DigraphOfGraphOfGroupoids(*gg*) (attribute)
▷ SubgroupoidsOfGraphOfGroupoids(*gg*) (attribute)
▷ IsomorphismsOfGraphOfGroupoids(*gg*) (attribute)
▷ RightTransversalsOfGraphOfGroupoids(*gg*) (attribute)
▷ LefvtTransversalsOfGraphOfGroupoids(*gg*) (attribute)

Graphs of groups generalise naturally to graphs of groupoids, forming the class IsGraphOfGroupoids. There is now a groupoid at each vertex and the isomorphism on an arc identifies wide subgroupoids at the tail and at the head. Since all subgroupoids are wide, every groupoid in a connected constituent of the graph has the same number of objects, but there is no requirement that the object sets are all the same.

The example below generalises the trefoil group example in subsection 4.4.1, taking at each vertex of D1 a two-object groupoid with a free group on one generator, and full subgroupoids with groups $\langle a^3 \rangle$ and $\langle b^2 \rangle$.

Example

```
gap> Gfa := SinglePieceGroupoid( fa, [-2,-1] );;
gap> ofa := One( fa );;
gap> SetName( Gfa, "Gfa" );
gap> Uhy := Subgroupoid( Gfa, [ [[-2,-1], hy ] ] );;
gap> SetName( Uhy, "Uhy" );
gap> Gfb := SinglePieceGroupoid( fb, [-4,-3] );;
gap> ofb := One( fb );;
gap> SetName( Gfb, "Gfb" );
gap> Uhybar := Subgroupoid( Gfb, [ [[-4,-3], hybar ] ] );;
gap> SetName( Uhybar, "Uhybar" );
gap> mory := GroupoidMappingOfSinglePieces(
gap>      Uhy, Uhybar, homy, [-4,-3], [ofb,ofb] );;
gap> morybar := GroupoidMappingOfSinglePieces(
gap>      Uhybar, Uhy, homybar, [-2,-1], [ofa,ofa] );;
gap> gg3 := GraphOfGroupoids( D1, [Gfa,Gfb], [Uhy,Uhybar], [mory,morybar] );;
gap> Display( gg3 );
Graph of Groupoids with :-
    vertices: [ 5, 6 ]
        arcs: [ [ y, 5, 6 ], [ y^-1, 6, 5 ] ]
   groupoids:
fp single piece groupoid: Gfa
  objects: [ -2, -1 ]
    group: fa = <[ a ]>
fp single piece groupoid: Gfb
  objects: [ -4, -3 ]
    group: fb = <[ b ]>
subgroupoids: single piece groupoid: Uhy
```

```
  objects: [ -2, -1 ]
    group: hy = <[ a^3 ]>
single piece groupoid: Uhybar
  objects: [ -4, -3 ]
    group: hybar = <[ b^2 ]>
isomorphisms: [ groupoid homomorphism : Uhy -> Uhybar
    [ [ GroupHomomorphismByImages( hy, hybar, [ a^3 ], [ b^2 ] ), [ -4, -3 ],
          [ <identity ...>, <identity ...> ] ] ],
  groupoid homomorphism : Uhybar -> Uhy
    [ [ GroupHomomorphismByImages( hybar, hy, [ b^2 ], [ a^3 ] ), [ -2, -1 ],
          [ <identity ...>, <identity ...> ] ] ] ]
```

5.5.2 GraphOfGroupoidsWord

▷ GraphOfGroupoidsWord(*gg, tv, list*) (operation)
▷ IsGraphOfGroupoidsWord(*w*) (property)
▷ GraphOfGroupoidsOfWord(*w*) (attribute)
▷ WordOfGraphOfGroupoidsWord(*w*) (attribute)
▷ ReducedGraphOfGroupoidsWord(*w*) (operation)
▷ IsReducedGraphOfGroupoidsWord(*w*) (property)

Having produced the graph of groupoids gg3, we may construct left coset representatives; choose a graph of groupoids word; and reduce this to normal form. Compare the nw3 below with the normal form nw1 in subsection 4.3.2.

Example

```
gap> f1 := GroupoidElement( Gfa, a^7, -1, -2);;
gap> f2 := GroupoidElement( Gfb, b^-6, -4, -4 );;
gap> f3 := GroupoidElement( Gfa, a^-11, -2, -1 );;
gap> f4 := GroupoidElement( Gfb, b^9, -3, -4 );;
gap> f5 := GroupoidElement( Gfa, a^7, -2, -1 );;
gap> L3 := [ f1, 1, f2, 2, f3, 1, f4, 2, f5 ];
[ [a^7 : -1 -> -2], 1, [b^-6 : -4 -> -4], 2, [a^-11 : -2 -> -1], 1,
  [b^9 : -3 -> -4], 2, [a^7 : -2 -> -1] ]
gap> gw3 := GraphOfGroupoidsWord( gg3, 5, L3);
(5)[a^7 : -1 -> -2].y.[b^-6 : -4 -> -4].y^-1.[a^-11 : -2 -> -1].y.[b^9 :
-3 -> -4].y^-1.[a^7 : -2 -> -1](5)
gap> nw3 := ReducedGraphOfGroupoidsWord( gw3 );
(5)[a^-1 : -1 -> -2].y.[b^-1 : -4 -> -4].y^-1.[a^10 : -2 -> -1](5)
```

More examples of these operations may be found in the example file gpd/examples/ggraph.g.

Chapter 6

Technical Notes

This short chapter is included for the benefit of anyone wishing to implement some other variety of many-object structures, for example *ringoids*, which are rings with many objects; *Lie groupoids*, which are Lie groups with many objects; and so on.

6.1 Many object structures

Structures with many objects, and their elements, are defined in a manner similar to the single object case. For elements we have:

- `DeclareCategory( "IsMultiplicativeElementWithObjects", IsMultiplicativeElement );`
- `DeclareCategory( "IsMultiplicativeElementWithObjectsAndOnes", IsMultiplicativeElementWithObjects );`
- `DeclareCategory( "IsMultiplicativeElementWithObjectsAndInversesIfNonzero", IsMultiplicativeElementWithObjectsAndOnes );`
- `DeclareCategory( "IsGroupoidElement", IsMultiplicativeElementWithObjectsAndInversesIfNonzero );`

as well as various category collections. For the various structures we have:

- `DeclareCategory( "IsDomainWithObjects", IsDomain );`
- `DeclareCategory( "IsMagmaWithObjects", IsDomainWithObjects and IsMagma and IsMultiplicativeElementWithObjectsCollection );`
- `DeclareCategory( "IsMagmaWithObjectsAndOnes", IsMagmaWithObjects and IsMultiplicativeElementWithObjectsAndOnesCollection );`
- `DeclareCategory( "IsMagmaWithObjectsAndInversesIfNonzero", IsMagmaWithObjectsAndOnes and IsMultiplicativeElementWithObjectsAndInversesIfNonzeroCollection );`
- `DeclareCategory( "IsGroupoid", IsMagmaWithObjectsAndInversesIfNonzero and IsGroupoidElementCollection );`

Some of the groupoids constructed earlier are the single piece Gd8 and the five component union U5:

Example

```
gap> CategoriesOfObject( Gd8 );
[ "IsListOrCollection", "IsCollection", "IsExtLElement",
  "CategoryCollections(IsExtLElement)", "IsExtRElement",
  "CategoryCollections(IsExtRElement)",
  "CategoryCollections(IsMultiplicativeElement)", "IsGeneralizedDomain",
  "IsMagma", "IsDomainWithObjects",
  "CategoryCollections(IsMultiplicativeElementWithObjects)",
  "CategoryCollections(IsMultiplicativeElementWithObjectsAndOnes)",
  "CategoryCollections(IsMultiplicativeElementWithObjectsAndInversesIfNonzero)\
", "CategoryCollections(IsGroupoidElement)", "IsMagmaWithObjects",
  "IsMagmaWithObjectsAndOnes", "IsMagmaWithObjectsAndInversesIfNonzero",
  "IsGroupoid" ]
gap> FamilyObj( Gd8 );
NewFamily( "GroupoidFamily", [ 2275 ], [ 51, 52, 77, 78, 79, 80, 90, 91, 114,
  115, 117, 118, 121, 202, 427, 2245, 2256, 2260, 2264, 2268, 2271, 2273,
  2274, 2275 ] )
gap> KnownAttributesOfObject( Gd8 );
[ "Name", "Size", "ParentAttr", "GeneratorsOfMagmaWithInverses",
  "ObjectList", "Pieces" ]
gap> KnownPropertiesOfObject( Gd8 );
[ "IsEmpty", "IsTrivial", "IsNonTrivial", "IsFinite",
  "CanEasilyCompareElements", "CanEasilySortElements", "IsDuplicateFree",
  "IsAssociative", "IsCommutative", "IsSinglePieceDomain",
  "IsDirectProductWithCompleteGraphDomain" ]
gap> RepresentationsOfObject( Gd8 );
[ "IsComponentObjectRep", "IsAttributeStoringRep", "IsMWOSinglePieceRep" ]
gap> RepresentationsOfObject( U5 );
[ "IsComponentObjectRep", "IsAttributeStoringRep", "IsPiecesRep" ]
```

Similarly, for elements, we have:

Example

```
gap> CategoriesOfObject(a78);
[ "IsExtLElement", "IsExtRElement", "IsMultiplicativeElement",
  "IsMultiplicativeElementWithObjects" ]
gap> FamilyObj( a78 );
NewFamily( "MultiplicativeElementWithObjectsFamily", [ 2255 ],
[ 77, 78, 79, 80, 114, 117, 120, 2255 ] )
gap> CategoriesOfObject(e2);
[ "IsExtLElement", "IsExtRElement", "IsMultiplicativeElement",
  "IsMultiplicativeElementWithObjects",
  "IsMultiplicativeElementWithObjectsAndOnes",
  "IsMultiplicativeElementWithObjectsAndInversesIfNonzero",
  "IsGroupoidElement" ]
gap> FamilyObj( e2 );
NewFamily( "GroupoidElementFamily", [ 2267 ],
[ 77, 78, 79, 80, 114, 117, 120, 2255, 2259, 2263, 2267 ] )
```

6.2 Many object homomorphisms

Homomorphisms of structures with many objects have a similar heirarchy.

- `DeclareCategory( "IsGeneralMappingWithObjects", IsGeneralMapping );`
- `DeclareSynonymAttr( "IsMagmaWithObjectsGeneralMapping", IsGeneralMappingWithObjects and RespectsMultiplication );`
- `DeclareSynonymAttr( "IsMagmaWithObjectsHomomorphism", IsMagmaWithObjectsGeneralMapping and IsMapping );`
- `DeclareCategory( "IsGroupoidHomomorphism", IsMagmaWithObjectsHomomorphism );`

Two forms of representation are used: for mappings to a single piece; and for unions of such mappings:

- `DeclareRepresentation( "IsMappingToSinglePieceRep", IsMagmaWithObjectsHomomorphism and IsAttributeStoringRep and IsGeneralMapping, [ "Source", "Range", "PieceImages" ] );`
- `DeclareRepresentation( "IsMappingWithObjectsRep", IsMagmaWithObjectsHomomorphism and IsAttributeStoringRep and IsGeneralMapping, [ "Source", "Range", "PiecesOfMapping" ] );`

In previous chapters, `hom1` was an endofunction on `M78`; `homd8` was a homomorphism from `Gd8` to `Gs3`; and `aut3` was an automorphism of `Ga4`. All homomorphisms have family `GeneralMappingWithObjectsFamily`. Perhaps it would be better to have separate families for each structure?

Example

```
gap> FamilyObj(hom1);
NewFamily( "GeneralMappingWithObjectsFamily", [ 2279 ],
[ 77, 78, 79, 80, 114, 117, 120, 124, 128, 147, 338, 2279 ] )
gap> KnownAttributesOfObject( hom1 );
[ "Range", "Source", "PieceImages" ]
gap> KnownPropertiesOfObject( hom1 );
[ "CanEasilyCompareElements", "CanEasilySortElements", "IsTotal",
  "IsSingleValued", "RespectsMultiplication", "IsGeneralMappingToSinglePiece",
  "IsGeneralMappingFromSinglePiece", "IsInjectiveOnObjects",
  "IsSurjectiveOnObjects" ]
gap> CategoriesOfObject( homd8 );
[ "IsExtLElement", "IsExtRElement", "IsMultiplicativeElement",
  "IsMultiplicativeElementWithOne", "IsMultiplicativeElementWithInverse",
  "IsAssociativeElement", "IsGeneralMapping", "IsGeneralMappingWithObjects",
  "IsGroupoidHomomorphism" ]
gap> KnownAttributesOfObject( homd8 );
[ "Range", "Source", "PieceImages", "ImagesOfObjects", "ImagesOfRays",
  "ObjectTransformationOfGroupoidHomomorphism", "RootObjectHomomorphism" ]
gap> KnownAttributesOfObject( aut3 );
[ "Range", "Source", "PieceImages", "ImagesOfObjects", "ImagesOfRays",
  "ObjectTransformationOfGroupoidHomomorphism", "RootObjectHomomorphism" ]
```

Chapter 7

Development History

7.1 Versions of the Package

The first version, GraphGpd 1.001, formed part of Emma Moore's thesis [Moo01] in December 2000, but was not made generally available.

Version 1.002 of GraphGpd was prepared to run under GAP 4.4 in January 2004; was submitted to the GAP council to be considered as an accepted package; but suggestions from the referee were not followed up.

In April 2006 the manual was converted to GAPDoc format. Variables `Star`, `Costar` and `CoveringGroup` were found to conflict with usage in other packages, and were renamed `VertexStar`, `VertexCostar` and `CoveringGroupOfGroupoid` respectively. Similarly, the `Vertices` and `Arcs` of an `FpWeightedDigraph` were changed from attributes to record components.

In the spring of 2006 the package was extensively rewritten and renamed Gpd. Version 1.01 was submitted as a deposited package in June 2006. Version 1.03, of October 2007, fixed some file protections, and introduced the test file `gpd_manual.tst`.

Version 1.05, of November 2008, was released when the website at Bangor changed.

Since then, the package has been rewritten again, introducing magmas with objects and their mappings. Functions to implement constructions contained in [AW10] have been added, but this is ongoing work.

Versions 1.09 to 1.15 were prepared for the anticipated release of GAP 4.5 in June 2012.

The latest version is 1.22 of 20th November 2013, for GAP 4.7.

7.2 What needs to be done next?

Computationally, there are three types of connected groupoid:

- those with identical object groups,
- those with object groups conjugate in some supergroup,
- those with object groups which are simply isomorphic.

GraphGpd attempted to implement the second case, while Gpd 1.01 and 1.03 considered only the first case, and Gpd 1.05 extended 1.03 to the second case.

The third case has not yet been considered for implementation, and there does not appear to be much need to do so.

Here are some other immediate requirements:

- more work on automorphism groups of groupoids;
- normal subgroupoids and quotient groupoids;
- more methods for morphisms of groupoids, particularly when the range is not connected;
- `ImageElm` and `ImagesSource` for the cases of groupoid morphisms not yet covered;
- `Enumerator` for `IsHomsetCosetsRep`;
- free groupoid on a graph;
- methods for `FreeProductWithAmalgamation` and `HnnEntension` for pc-groups;
- convert `GraphOfGroupsRewritingSystem` to the category `IsRewritingSystem`;
- in XMod, implement crossed modules over groupoids (a start has been made).

References

[AW10] M. Alp and C.D. Wensley. Automorphisms and homotopies of groupoids and crossed modules. *Applied Categorical Structures*, 18:473–495, 2010. 2, 24, 30, 45

[Bro88] Ronald Brown. *Topology: a geometric account of general topology, homotopy types, and the fundamental groupoid*. Ellis Horwood, Chichester, 1988. 5, 15

[Bro06] Ronald Brown. *Topology and groupoids*. www.groupoids.org, Deganwy, 2006. 5, 15

[Hig76] Philip Higgins. The fundamental groupoid of a graph of groups. *J. London Math. Soc.*, 13:145–149, 1976. 33

[Hig05] Philip Higgins. *Categories and Groupoids*. Reprints in Theory and Applications of Categories, 2005. 5

[Moo01] Emma Moore. *Graphs of groups: word computations and free crossed resolutions*. Ph.D. thesis, University of Wales, Bangor, 2001. 33, 36, 37, 45

[Ser80] J. Serre. *Trees*. Springer-Verlag, Berlin, 1980. 33

Index

* for groupoid elements, 19
\^{} for groupoid elements, 24
\^{} for groupoids, 25

Ancestor, 22
AutomorphismGroup, 30

ConjugateGroupoid, 25
ConjugateGroupoidElement, 24

DigraphOfGraphOfGroupoids, 40
DigraphOfGraphOfGroups, 34
DiscreteSubgroupoid, 20
DiscreteTrivialSubgroupoid, 20
DomainWithSingleObject, 9
DoubleCoset, 23
DoubleCosetRepresentatives, 23

ElementOfArrow, 6, 18

FpWeightedDigraph, 33
FreeProductWithAmalgamation, 37
FullSubgroupoid, 20
FullTrivialSubgroupoid, 20

GeneratorsOfMagmaWithObjects, 8
GGHead, 36
GGTail, 36
GraphOfGroupoids, 40
GraphOfGroupoidsOfWord, 41
GraphOfGroupoidsWord, 41
GraphOfGroups, 34
GraphOfGroupsOfWord, 36
GraphOfGroupsRewritingSystem, 37
GraphOfGroupsWord, 36
Groupoid, 15
GroupoidAutomorphismByGroupAuto, 29
GroupoidAutomorphismByGroupAutos, 31
GroupoidAutomorphismByObjectPerm, 29
GroupoidAutomorphismByRayImages, 29
GroupoidElement, 18
GroupoidHomomorphism, 26
GroupoidHomomorphismByGroupHom, 26
GroupoidHomomorphismFromSinglePiece, 26
GroupoidsOfGraphOfGroupoids, 40
GroupsOfGraphOfGroups, 34

HeadOfArrow, 6, 18
HnnExtension, 39
HomogeneousDiscreteGroupoid, 18
HomogeneousGroupoid, 18
HomomorphismByUnion, 14, 28
HomomorphismFromSinglePiece, 11
HomomorphismToSinglePiece, 11, 27
Homset, 19
HomsOfMapping, 11

identity subgroupoid, 21
IdentityElement, 19
IdentityMapping, 27
InclusionMappingGroupoids, 26
InvolutoryArcs, 33
IsAutomorphismWithObjects, 14
IsBijectiveOnObjects, 14
IsDirectProductWithCompleteGraph, 7
IsDiscrete, 7
IsDomainWithObjects, 6
IsElementOfGroupoid, 18
IsElementOfMagmaWithObjects, 6
IsEndomorphismWithObjects, 14
IsFpaGroup, 37
IsFpGroupoid, 16
IsFpWeightedDigraph, 33
IsGraphOfFpGroupoids, 40
IsGraphOfFpGroups, 35
IsGraphOfGroupoidsWord, 41
IsGraphOfGroupsWord, 36
IsGraphOfPcGroups, 35
IsGraphOfPermGroupoids, 40
IsGraphOfPermGroups, 35

IsHnnGroup, 39
IsHomogeneousDomainWithObjects, 18
IsHomogeneousDiscreteGroupoidRep, 18
IsInjectiveOnObjects, 14
IsMagmaWithObjects, 6
IsMappingToSinglePieceRep, 11
IsMultiplicativeElementWithObjects, 6
IsomorphismNewObjects, 11
IsomorphismsOfGraphOfGroupoids, 40
IsomorphismsOfGraphOfGroups, 34
IsPcGroupoid, 16
IsPermGroupoid, 16
IsReducedGraphOfGroupoidsWord, 41
IsReducedGraphOfGroupsWord, 37
IsSinglePiece, 7
IsSubgroupoid, 20
IsSurjectiveOnObjects, 14
IsWide, 20

LeftCoset, 23
LeftCosetRepresentatives, 23
LeftCosetRepresentativesFromObject, 23
LeftTransversalsOfGraphOfGroups, 35
LefvtTransversalsOfGraphOfGroupoids, 40
License, 2

MagmaWithObjects, 5
MagmaWithObjectsHomomorphism, 11
MaximalDiscreteSubgroupoid, 20
MonoidWithObjects, 8
MultiplicativeElementWithObjects, 6

NormalFormGGRWS, 37

ObjectCostar, 19
ObjectList, 5
ObjectList
    for groupoids, 17
ObjectStar, 19

PieceImages, 11
Pieces, 9
PiecesOfMapping, 11

Range, 11
rays, 22
RaysOfGroupoid, 22
ReducedGraphOfGroupoidsWord, 41
ReducedGraphOfGroupsWord, 37
ReplaceOnePieceInUnion, 16
RightCoset, 23
RightCosetRepresentatives, 23
RightTransversalsOfGraphOfGroupoids, 40
RightTransversalsOfGraphOfGroups, 35
RootObject, 5, 15
RootObjectHomomorphism, 26

SemigroupWithObjects, 7
SinglePieceGroupoid, 15
SinglePieceGroupoidByGenerators, 25
SinglePieceMagmaWithObjects, 5
SinglePieceMonoidWithObjects, 8
SinglePieceSemigroupWithObjects, 7
Size, 16
Source, 11
Subgroupoid, 20
SubgroupoidByPieces, 20
SubgroupoidsOfGraphOfGroupoids, 40
SubgroupoidWithRays, 22

TailOfArrow, 6, 18
tree groupoid, 21
trivial subgroupoid, 21

UnionOfPieces, 9, 16

WordOfGraphOfGroupoidsWord, 41
WordOfGraphOfGroupsWord, 36