

groupoids

Calculations with finite groupoids and their homomorphisms

1.81

26 November 2025

Emma J. Moore

Chris Wensley

Chris Wensley

Email: cdwensley.maths@btinternet.com

Homepage: <https://github.com/cdwensley>

Abstract

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

The package was then renamed `Gpd` and version 1.07 was released in July 2011, ready for GAP 4.5.

`Gpd` became an accepted GAP package in May 2015.

In April 2017 the package was renamed again, as `groupoids`.

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

Bug reports, comments, suggestions for additional features, and offers to implement some of these, will all be very welcome.

Please submit any issues at <https://github.com/gap-packages/groupoids/issues/> or send an email to the second author at cdwensley.maths@btinternet.com.

Copyright

© 2000–2025, Emma Moore and Chris Wensley.

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

Acknowledgements

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

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

Contents

1	Introduction	5
2	Many-object structures	6
2.1	Magmas with objects; arrows	6
2.2	Semigroups with objects	9
2.3	Monoids with objects	10
2.4	Generators of magmas with objects	11
2.5	Structures with more than one piece	12
3	Mappings of many-object structures	14
3.1	Homomorphisms of magmas with objects	14
3.2	Homomorphisms of semigroups and monoids with objects	16
3.3	Homomorphisms to more than one piece	17
3.4	Mappings defined by a function	18
4	Groupoids	19
4.1	Groupoids: their properties and attributes	19
4.2	Groupoid elements; stars; costars; homsets	25
4.3	Subgroupoids	27
4.4	Left, right and double cosets	33
4.5	Conjugation	35
4.6	Groupoids formed using isomorphisms	36
4.7	Groupoids whose objects form a monoid	38
5	Homomorphisms of Groupoids	40
5.1	Homomorphisms from a connected groupoid	40
5.2	Properties and attributes of groupoid homomorphisms	42
5.3	Special types of groupoid homomorphism	44
5.4	Homomorphisms to a connected groupoid	47
5.5	Homomorphisms to more than one piece	49
6	Automorphisms of Groupoids	51
6.1	Automorphisms of single piece groupoids	51
6.2	Matrix representations of groupoids	59
6.3	Groupoid actions	60

7	Graphs of Groups and Groupoids	63
7.1	Digraphs	63
7.2	Graphs of Groups	64
7.3	Words in a Graph of Groups and their normal forms	66
7.4	Free products with amalgamation and HNN extensions	67
7.5	GraphsOfGroupoids and their Words	70
8	Double Groupoids	74
8.1	Single piece double groupoids	74
8.2	Double groupoids with more than one piece	83
8.3	Generators of a double groupoid	84
8.4	Starting with two groupoids	85
8.5	Double groupoid homomorphisms	86
9	Technical Notes	88
9.1	Many object structures	88
9.2	Many object homomorphisms	90
10	Development History	92
10.1	Versions of the Package	92
10.2	What needs to be done next?	93
	References	94
	Index	95

Chapter 1

Introduction

Groupoids are mathematical categories in which every arrow is invertible. The `groupoids` package provides functions for the computation with groupoids and their morphisms; for graphs of groups and graphs of groupoids. The package is far from complete, and development continues.

It was used by Emma Moore in her thesis [Moo01] to calculate normal forms for *free products with amalgamation*, and for *HNN-extensions* when the initial groups have rewriting systems.

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

- the `groupoids` GitHub site: <https://github.com/gap-packages.github.io/groupoids/>.
- any GAP archive, e.g. <https://www.gap-system.org/Packages/packages.html>;

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

The information parameter `InfoGroupoids` takes default value 1 which, for the benefit of new users, causes more messages to be printed out when operations fail. When raised to a higher value, additional information is printed out.

Help is available in the usual way.

Example

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

For version 1.05 the package was completely restructured, starting with *magmas with objects* and their mappings, and building up to groupoids via semigroups with objects and monoids with objects. From version 1.07 the package includes some functions to implement constructions for automorphisms and homotopies, as described in [AW10]. More functions will be released when time permits.

Once the package is loaded, it is possible to check the installation has proceeded correctly by running the test suite of the package with the command `ReadPackage("groupoids","tst/testing.g"); .` Additional tests may be run using `ReadPackage("groupoids","tst/testextra.g"); .` (The file `"tst/testall.g"` is used for automated testing.)

You may reference this package by mentioning [BMPW02], [Moo01] and [AW10].

Additional information on *Computational Higher Dimensional Algebra* can be found in the notes on crossed modules at: <https://github.com/cdwensley/xmod-notes>.

Chapter 2

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 4.

The definitions of the standard properties of groupoids can be found in Philip Higgins’ book “Categories and Groupoids” [Hig05] (originally published in 1971, reprinted by TAC in 2005), and in Ronnie Brown’s book “Topology” [Bro88], revised and reissued as “Topology and Groupoids” [Bro06].

2.1 Magmas with objects; arrows

A *magma with objects* M consists of a set of *objects* $\text{Ob}(M)$, and a set of *arrows* $\text{Arr}(M)$ together with *tail* and *head* maps $t, h : \text{Arr}(M) \rightarrow \text{Ob}(M)$, and a *partial multiplication* $* : \text{Arr}(M) \rightarrow \text{Arr}(M)$, 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 $(a : u \rightarrow v)$.

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 $(1_u : u \rightarrow u)$ 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 $(a : u \rightarrow v)$ in a monoid with objects has *inverse* $(a^{-1} : v \rightarrow u)$ 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*.

2.1.1 MagmaWithObjects

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

The simplest construction for a magma with objects M is to take a magma m and an ordered set s , and form arrows (u, a, v) for every a in m and u, v in s . Multiplication is defined by $(u, a, v) * (v, b, w) = (u, a * b, w)$. In this package we prefer to write (u, a, v) as $(a : u \rightarrow v)$, so that the multiplication rule becomes $(a : u \rightarrow v) * (b : v \rightarrow w) = (a * b : u \rightarrow w)$.

Any finite, ordered set is in principle acceptable as the object list of M , but most of the time 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 `IsDirectProductWithCompleteDigraph`. There are other constructions implemented for groupoids.

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 );
<magma with 4 generators>
gap> SetName( m, "m" ); One(m);
fail
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) ];
[ false, false ]
gap> [ RootObject( M78 ), ObjectList( M78 ) ];
[ -8, [ -8, -7 ] ]
```

2.1.2 IsDomainWithObjects

- ▷ `IsDomainWithObjects(obj)` (Category)
- ▷ `IsMagmaWithObjects(obj)` (Category)

The output from function `MagmaWithObjects` lies in the categories `IsDomainWithObjects`, `IsMagmaWithObjects` and `CategoryCollections(IsMultiplicativeElementWithObjects)`. As composition is only partial, the output does *not* lie in the category `IsMagma`.

Example

```
gap> [ IsDomainWithObjects(M78), IsMagmaWithObjects(M78), IsMagma(M78) ];
[ true, true, false ]
```

2.1.3 Arrow

- ▷ `Arrow(mwo, elt, tail, head)` (operation)
- ▷ `MWOfArrow(arr)` (operation)
- ▷ `ElementOfArrow(arr)` (operation)
- ▷ `TailOfArrow(arr)` (operation)
- ▷ `HeadOfArrow(arr)` (operation)

Arrows in a magma with objects lie in the category `IsMultiplicativeElementWithObjects`. An attempt to multiply two arrows which do not compose results in `fail` being returned. Each arrow $arr = (a : u \rightarrow v)$ has four components. The underlying magma M may be accessed by `MWOfArrow(arr)`. Similarly, the element $a \in m$, the tail object u , and the head object v may be obtained using `ElementOfArrow(arr)`, `TailOfArrow(arr)` and `HeadOfArrow(arr)` respectively. The operation `MultiplicativeElementWithObjects` is a synonym for `Arrow` since this was used in older versions of the package.

Example

```
gap> a78 := Arrow( M78, m2, -7, -8 );
[m2 : -7 -> -8]
gap> a78 in M78;
true
gap> b87 := Arrow( M78, m4, -8, -7 );
gap> [ MWOfArrow( b87 ), ElementOfArrow( b87 ),
>      TailOfArrow( b87 ), HeadOfArrow( b87 ) ];
[ M78, m4, -8, -7 ]
gap> ba := b87*a78; ab := a78*b87;
[m4 : -8 -> -8]
[m3 : -7 -> -7]
gap> [ a78^2, ba^2, ba^3 ];
[ fail, [m1 : -8 -> -8], [m3 : -8 -> -8] ]
gap> ## this demonstrates non-associativity:
gap> [ a78*ba, ab*a78, a78*ba = ab*a78 ];
[ [m3 : -7 -> -8], [m4 : -7 -> -8], false ]
```

2.1.4 IsSinglePieceDomain

- ▷ `IsSinglePieceDomain(mwo)` (property)
- ▷ `IsSinglePiece(mwo)` (property)
- ▷ `IsDirectProductWithCompleteDigraph(mwo)` (property)
- ▷ `IsDiscreteMagmaWithObjects(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 `IsSinglePieceDomain` to describe an object with an underlying connected digraph. The property `IsSinglePiece` is a synonym for `IsSinglePieceDomain` and `IsMagmaWithObjects`. 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> IsDirectProductWithCompleteDigraph( M78 );
true
gap> IsDiscreteMagmaWithObjects( M78 );
false
```

2.2 Semigroups with objects

2.2.1 SemigroupWithObjects

- ▷ `SemigroupWithObjects(args)` (function)
- ▷ `SinglePieceSemigroupWithObjects(sgp, obs)` (operation)
- ▷ `MagmaWithSingleObject(dom, obj)` (operation)
- ▷ `IsSemigroupWithObjects(obj)` (filter)

The constructions in section 2.1 give a `SinglePieceSemigroupWithObjects` when the magma is a semigroup. In the following example we use a transformation semigroup and 3 objects.

Example

```
gap> t := Transformation( [1,1,2,3] );;
gap> s := Transformation( [2,2,3,3] );;
gap> r := Transformation( [2,3,4,4] );;
gap> 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> CategoriesOfObject( S123 );
[ "IsListOrCollection", "IsCollection", "IsExtLElement",
  "CategoryCollections(IsExtLElement)", "IsExtRElement",
  "CategoryCollections(IsExtRElement)",
  "CategoryCollections(IsMultiplicativeElement)", "IsGeneralizedDomain",
  "IsDomainWithObjects",
  "CategoryCollections(IsMultiplicativeElementWithObjects)",
  "IsMagmaWithObjects", "IsSemigroupWithObjects" ]
gap> t12 := Arrow( S123, t, -1, -2 );
[Transformation( [ 1, 1, 2, 3 ] ) : -1 -> -2]
gap> s23 := Arrow( S123, s, -2, -3 );
[Transformation( [ 2, 2, 3, 3 ] ) : -2 -> -3]
gap> r31 := Arrow( S123, r, -3, -1 );
```

```

[Transformation( [ 2, 3, 4, 4 ] ) : -3 -> -1]
gap> ts13 := t12 * s23;
[Transformation( [ 2, 2, 2, 3 ] ) : -1 -> -3]
gap> sr21 := s23 * r31;
[Transformation( [ 3, 3, 4, 4 ] ) : -2 -> -1]
gap> rt32 := r31 * t12;
[Transformation( [ 1, 2, 3, 3 ] ) : -3 -> -2]
gap> tsr1 := ts13 * r31;
[Transformation( [ 3, 3, 3 ] ) : -1 -> -1]

```

A magma, semigroup, monoid, or group M 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 M . In the example we take the transformation semigroup above, which has size 17 at the object 0.

Example

```

gap> S0 := MagmaWithSingleObject( sgp, 0 );
semigroup with objects :-
  magma = sgp<t,s,r>
  objects = [ 0 ]
gap> t0 := Arrow( S0, t, 0, 0 );
[Transformation( [ 1, 1, 2, 3 ] ) : 0 -> 0]
gap> Size( S0 );
17

```

2.3 Monoids with objects

2.3.1 MonoidWithObjects

- ▷ MonoidWithObjects(args) (function)
- ▷ SinglePieceMonoidWithObjects(mon, obs) (operation)
- ▷ IsMonoidWithObjects(obj) (filter)

The constructions in section 2.1 give a SinglePieceMonoidWithObjects when the magma is a monoid. The example uses a finitely presented monoid with 2 generators and 2 objects.

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> [ IsAssociative(M49), IsCommutative(M49) ];
[ true, false ]
gap> ktpo := KnownTruePropertiesOfObject( M49 );;

```

```

gap> ans := [ "IsDuplicateFree", "IsAssociative", "IsSinglePieceDomain",
> "IsDirectProductWithCompleteDigraphDomain" ];;
gap> ForAll( ans, a -> ( a in ktpo ) );
true
gap> catobj := CategoriesOfObject( M49 );;
gap> ans := [ "IsListOrCollection", "IsCollection", "IsExtLElement",
> "CategoryCollections(IsExtLElement)", "IsExtRElement",
> "CategoryCollections(IsExtRElement)",
> "CategoryCollections(IsMultiplicativeElement)", "IsGeneralizedDomain",
> "IsDomainWithObjects",
> "CategoryCollections(IsMultiplicativeElementWithObjects)",
> "CategoryCollections(IsMultiplicativeElementWithObjectsAndOnes)",
> "IsMagmaWithObjects", "IsSemigroupWithObjects", "IsMonoidWithObjects" ];;
gap> ForAll( ans, a -> ( a in catobj ) );
true

```

2.4 Generators of magmas with objects

2.4.1 GeneratorsOfMagmaWithObjects

- ▷ GeneratorsOfMagmaWithObjects(*mwo*) (operation)
- ▷ GeneratorsOfSemigroupWithObjects(*swo*) (operation)
- ▷ GeneratorsOfMonoidWithObjects(*mwo*) (operation)

For a magma or semigroup with objects, the generating set consists of arrows $(g : u \rightarrow v)$ for every pair of objects u, v and every generating element for the magma or semigroup.

For a monoid with objects, the generating set consists of two parts. Firstly, there is a loop at the root object r for each generator of the monoid. Secondly, for each object u distinct from r , there are arrows $(1 : r \rightarrow u)$ and $(1 : u \rightarrow r)$. (Perhaps only one of each pair is required?) Then

$$(e : u \rightarrow v) = (1 : u \rightarrow r) * (e : r \rightarrow r) * (1 : r \rightarrow v).$$

Example

```

gap> GeneratorsOfMagmaWithObjects( M78 );
[ [m1 : -8 -> -8], [m2 : -8 -> -8], [m3 : -8 -> -8], [m4 : -8 -> -8],
  [m1 : -8 -> -7], [m2 : -8 -> -7], [m3 : -8 -> -7], [m4 : -8 -> -7],
  [m1 : -7 -> -8], [m2 : -7 -> -8], [m3 : -7 -> -8], [m4 : -7 -> -8],
  [m1 : -7 -> -7], [m2 : -7 -> -7], [m3 : -7 -> -7], [m4 : -7 -> -7] ]
gap> genS := GeneratorsOfSemigroupWithObjects( S123 );;
gap> Length( genS );
27
gap> genM := GeneratorsOfMonoidWithObjects( M49 );
[ [f1 : -9 -> -9], [f2 : -9 -> -9], [<identity ...> : -9 -> -4],
  [<identity ...> : -4 -> -9] ]
gap> g1:=genM[1];; g2:=genM[2];; g3:=genM[3];; g4:=genM[4];;
gap> [g4,g2,g1,g3];
[ [<identity ...> : -4 -> -9], [f2 : -9 -> -9], [f1 : -9 -> -9],
  [<identity ...> : -9 -> -4] ]
gap> g4*g2*g1*g3;

```

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

2.5 Structures with more than one piece

2.5.1 UnionOfPieces (for magmas with objects)

- ▷ `UnionOfPieces(pieces)` (operation)
- ▷ `Pieces(mwo)` (attribute)
- ▷ `PieceOfObject(mwo, obj)` (operation)

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. The pieces are ordered by the order of their root objects.

Example

```
gap> N1 := UnionOfPieces( M78, S123 );
magma with objects having 2 pieces :-
1: M78
2: semigroup with objects :-
   magma = sgp<t,s,r>
   objects = [ -3, -2, -1 ]
gap> ObjectList( N1 );
[ -8, -7, -3, -2, -1 ]
gap> Pieces(N1);
[ M78, semigroup with objects :-
   magma = sgp<t,s,r>
   objects = [ -3, -2, -1 ]
 ]
gap> PieceOfObject( N1, -7 );
M78
gap> N2 := UnionOfPieces( M49, S0 );
semigroup with objects having 2 pieces :-
1: monoid with objects :-
   magma = Monoid( [ f1, f2 ] )
   objects = [ -9, -4 ]
2: semigroup with objects :-
   magma = sgp<t,s,r>
   objects = [ 0 ]
gap> ObjectList( N2 );
[ -9, -4, 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: semigroup with objects :-
  magma = sgp<t,s,r>
  objects = [ 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, MagmaWithSingleObject( sgp, -2 ) ] );
fail
```

Chapter 3

Mappings 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:

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

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

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

3.1 Homomorphisms of magmas with objects

3.1.1 MagmaWithObjectsHomomorphism

▷ MagmaWithObjectsHomomorphism(<i>args</i>)	(function)
▷ HomomorphismFromSinglePiece(<i>src, rng, hom, imobs</i>)	(operation)
▷ HomomorphismToSinglePiece(<i>src, rng, images</i>)	(operation)
▷ MappingToSinglePieceData(<i>mwohom</i>)	(attribute)
▷ PiecesOfMapping(<i>mwohom</i>)	(attribute)
▷ IsomorphismNewObjects(<i>src, objlist</i>)	(operation)

There are a variety of homomorphism constructors.

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

- a magma homomorphism hom 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 := [ DirectProductElement([m1,m2]), DirectProductElement([m2,m1]),
>           DirectProductElement([m3,m4]), DirectProductElement([m4,m3]) ];;
gap> f1 := GeneralMappingByElements( m, m, tup1 );
<general mapping: m -> m >
gap> IsMagmaHomomorphism( f1 );
true
gap> hom1 := MagmaWithObjectsHomomorphism( M78, M78, f1, [-7,-8] );
magma with objects homomorphism : M78 -> M78
[ [ <mapping: m -> m >, [ -7, -8 ] ] ]
gap> [ Source( hom1 ), Range( hom1 ) ];
[ M78, M78 ]
gap> b87;
[m4 : -8 -> -7]
gap> im1 := ImageElm( hom1, b87 );
[m3 : -7 -> -8]
gap> i65 := IsomorphismNewObjects( M78, [-6,-5] );
magma with objects homomorphism : [ [ IdentityMapping( m ), [ -6, -5 ] ] ]
gap> ib87 := ImageElm( i65, b87 );
[m4 : -6 -> -5]
gap> M65 := Range( i65 );;
gap> SetName( M65, "M65" );
gap> j65 := InverseGeneralMapping( i65 );;
gap> ImagesOfObjects( j65 );
[ -8, -7 ]
gap> comp := j65 * hom1;
magma with objects homomorphism : M65 -> M78
[ [ <mapping: m -> m >, [ -7, -8 ] ] ]
gap> ImageElm( comp, ib87 );
[m3 : -7 -> -8]

```

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 := [ MappingToSinglePieceData( hom1 )[1],
> MappingToSinglePieceData( j65 )[1] ];
[ [ <mapping: m -> m >, [ -7, -8 ] ], [ IdentityMapping( m ), [ -8, -7 ] ] ]
gap> map4 := HomomorphismToSinglePiece( M4, M78, images );
magma with objects homomorphism :
[ [ <mapping: m -> m >, [ -7, -8 ] ], [ IdentityMapping( m ), [ -8, -7 ] ] ]
gap> ImageElm( map4, b87 );
[m3 : -7 -> -8]
gap> ImageElm( map4, ib87 );
[m4 : -8 -> -7]

```

3.2 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, L1, 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] ]
```


3.3 Homomorphisms to more than one piece

3.3.1 HomomorphismByUnion (for magmas with objects)

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

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

See section 5.5 for the equivalent operation with groupoids.

Example

```
gap> N4 := UnionOfPieces( [ M78, T123 ] );;
gap> h14 := HomomorphismByUnionNC( N1, N4, [ hom1, shom ] );
magma with objects homomorphism :
[ magma with objects homomorphism : M78 -> M78
  [ [ <mapping: m -> m >, [ -7, -8 ] ] ], magma with objects homomorphism :
  [ [ smap, [ -11, -12, -13 ] ] ] ]
gap> ImageElm( h14, a78 );
[m1 : -8 -> -7]
gap> ImageElm( h14, r31 );
[Transformation( [ 4, 1, 3, 3 ] ) : -11 -> -13]
```

3.3.2 IsInjectiveOnObjects

▷ `IsInjectiveOnObjects(mwohom)` (property)
 ▷ `IsSurjectiveOnObjects(mwohom)` (property)
 ▷ `IsBijectiveOnObjects(mwohom)` (property)
 ▷ `IsEndomorphismWithObjects(mwohom)` (property)
 ▷ `IsAutomorphismWithObjects(mwohom)` (property)

The meaning of these five properties is obvious.

Example

```
gap> IsInjectiveOnObjects( h14 );
true
gap> IsSurjectiveOnObjects( h14 );
true
gap> IsBijectiveOnObjects( h14 );
true
gap> IsEndomorphismWithObjects( h14 );
false
gap> IsAutomorphismWithObjects( h14 );
false
```

3.4 Mappings defined by a function

3.4.1 MappingWithObjectsByFunction

- ▷ `MappingWithObjectsByFunction(src, rng, fun, imobs)` (operation)
- ▷ `IsMappingWithObjectsByFunction(map)` (property)
- ▷ `UnderlyingFunction(map)` (attribute)

More general mappings, which need not preserve multiplication, are available using this operation. See chapter 6 for an application.

— Example —

```
gap> swap := function(a) return Arrow(M78,a![2],a![4],a![3]); end;
function( a ) ... end
gap> swapmap := MappingWithObjectsByFunction( M78, M78, swap, [-7,-8] );
magma with objects mapping by function : M78 -> M78
function: function ( a )
    return Arrow( M78, a![2], a![4], a![3] );
end

gap> a78; ImageElm( swapmap, a78 );
[m2 : -7 -> -8]
[m2 : -8 -> -7]
```

Chapter 4

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’, so we use the term *single piece*.

The simplest form for a *single piece groupoid* is the direct product of a group and a complete digraph, and so is determined by a set of *objects* $\text{obs} = \Omega$ (the least of which is the *root object*), and a *root group* $\text{grp} = G$. Then the elements of the groupoid are *arrows* $(g : o_1 \rightarrow o_2)$, stored as triples $[g, o_1, o_2]$, where $g \in G$ and $o_1, o_2 \in \Omega$. The objects will generally be chosen to be consecutive negative integers, but any suitable ordered set is acceptable, and ‘consecutive’ is not a requirement. The root group will usually be taken to be a permutation group, but pc-groups, fp-groups and matrix groups are also supported.

A *group* may be considered as 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`.

The underlying digraph of a single piece groupoid is a regular, complete digraph on the object set Ω with $|G|$ arrows from any one object to any other object. It will be convenient to specify a set of *rays* consisting of $|\Omega|$ arrows $(r_i : o_1 \rightarrow o_i)$, where o_1 is the root object and r_1 is the identity in G . In the simplest examples all the r_i will be identity elements, but other rays are useful when forming subgroupoids (see `SubgroupoidWithRays` (4.3.4)).

A groupoid is *homogeneous* if it has two or 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 are used in the `XMod` package as the source of a 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].

4.1 Groupoids: their properties and attributes

4.1.1 SinglePieceGroupoid

- ▷ `SinglePieceGroupoid(grp, obs)` (operation)
- ▷ `Groupoid(args)` (function)
- ▷ `MagmaWithSingleObject(gp, obj)` (operation)
- ▷ `IsGroupoid(mwo)` (Category)

The simplest construction of a groupoid is as the direct product of a group and a complete digraph. Such a groupoid will be called a *standard groupoid*. Many subgroupoids of such a groupoid do not have this simple form, and will be considered in section 4.3. The global function `Groupoid` will normally find the appropriate constructor to call, the options being:

- the object group and a set of objects;
- a group being converted to a groupoid and a single object;
- a list of groupoids which have already been constructed (see 4.1.4).

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> a4 := Group( (1,2,3), (2,3,4) );;
gap> d8 := Group( (5,6,7,8), (5,7) );;
gap> SetName( a4, "a4" ); SetName( d8, "d8" );
gap> Ga4 := SinglePieceGroupoid( a4, [-15 .. -11] );
single piece groupoid: < a4, [ -15 .. -11 ] >
gap> Gd8 := Groupoid( d8, [-9,-8,-7] );
single piece groupoid: < d8, [ -9, -8, -7 ] >
gap> c6 := Group( (11,12,13)(14,15) );;
gap> SetName( c6, "c6" );
gap> Gc6 := MagmaWithSingleObject( c6, -10 );
single piece groupoid: < c6, [ -10 ] >
gap> IsGroupoid( Gc6 );
true
gap> SetName( Ga4, "Ga4" ); SetName( Gd8, "Gd8" ); SetName( Gc6, "Gc6" );
```

More operations for constructing groupoids are described in the following subsections:

- Homogeneous groupoids (see 4.1.5);
- Direct products of groupoids (see 4.1.6);
- A variety of subgroupoid constructions in section 4.3;
- Groupoids formed using group isomorphisms in section 4.6;
- Groupoids whose objects form a monoid in section 4.7.

4.1.2 ObjectList (for groupoids)

▷ <code>ObjectList(gpd)</code>	(attribute)
▷ <code>RootObject(gpd)</code>	(attribute)
▷ <code>RootGroup(gpd)</code>	(attribute)
▷ <code>ObjectGroup(gpd, obj)</code>	(operation)

The `ObjectList` of a groupoid is the sorted list of its objects. The `RootObject` in a single-piece groupoid is the object with the least label. A *loop* is an arrow of the form $(g : o \rightarrow o)$, and the loops at a particular object o form a group, the `ObjectGroup` at o . The `RootGroup` is the `ObjectGroup` at the `RootObject`.

In the example, the groupoids `Gf2c6` and `Gabc` illustrate that the objects need not be integers.

Example

```
gap> ObjectList( Ga4 );
[ -15 .. -11 ]
gap> f2 := FreeGroup(2);
gap> Gf2d8 := Groupoid( d8, GeneratorsOfGroup(f2) );
single piece groupoid: < d8, [ f1, f2 ] >
gap> Arrow( Gf2d8, (6,8), f2.1, f2.2 );
[(6,8) : f1 -> f2]
gap> Gabc := Groupoid( c6, [ "a", "b", "c" ] );
single piece groupoid: < c6, [ "a", "b", "c" ] >
gap> Arrow( Gabc, (14,15), "c", "b" );
[(14,15) : c -> b]
```

4.1.3 IsPermGroupoid

- | | |
|--------------------------------------|------------|
| ▷ <code>IsPermGroupoid(gpd)</code> | (property) |
| ▷ <code>IsPcGroupoid(gpd)</code> | (property) |
| ▷ <code>IsFpGroupoid(gpd)</code> | (property) |
| ▷ <code>IsMatrixGroupoid(gpd)</code> | (property) |
| ▷ <code>IsFreeGroupoid(gpd)</code> | (property) |

A groupoid is a permutation groupoid if all its pieces have permutation root 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 and also a free groupoid, `Gq8` is a pc-groupoid, and `Gsl43` is a matrix groupoid. See section 6.2 for matrix representations of groupoids.

Example

```
gap> f2 := FreeGroup( 2 );
gap> Gf2 := Groupoid( f2, -20 );
gap> SetName( f2, "f2" ); SetName( Gf2, "Gf2" );
gap> q8 := QuaternionGroup( 8 );
gap> genq8 := GeneratorsOfGroup( q8 );
gap> x := genq8[1]; y := genq8[2];
gap> Gq8 := Groupoid( q8, [ -19, -18, -17 ] );
gap> SetName( q8, "q8" ); SetName( Gq8, "Gq8" );
gap> sl43 := SpecialLinearGroup( 4, 3 );
gap> Gsl43 := SinglePieceGroupoid( sl43, [-23,-22,-21] );
gap> SetName( sl43, "sl43" ); SetName( Gsl43, "Gsl43" );
gap> [ IsMatrixGroupoid( Gsl43 ), IsFpGroupoid( Gf2 ), IsFreeGroupoid( Gf2 ),
>     IsPcGroupoid( Gq8 ), IsPermGroupoid( Ga4 ) ];
[ true, true, true, true, true ]
```

4.1.4 UnionOfPieces (for groupoids)

- ▷ `UnionOfPieces(pieces)` (operation)
- ▷ `Pieces(gpd)` (attribute)
- ▷ `Size(gpd)` (attribute)
- ▷ `ReplaceOnePieceInUnion(U, old_piece, new_piece)` (operation)

When a groupoid consists of two or more pieces, we require their object lists to be disjoint. The operation `UnionOfPieces` and the attribute `Pieces`, introduced in section 2.5, are 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 arrows. For a single piece groupoid, this is the product of the size of the group with the square of the number of objects. For a non-connected groupoid, the size is the sum of the sizes of its pieces.

One of the pieces in a groupoid may be replaced by an alternative piece using the operation `ReplaceOnePieceInUnion`. The *old_piece* may be either the *position* of the piece to be replaced, or one of the pieces in `U`. The objects in the new piece may or may not overlap the objects in the piece being removed -- we just require that the object lists in the new union are disjoint.

Example

```
gap> U3 := UnionOfPieces( [ Ga4, Gc6, Gd8 ] );
gap> Display( U3 );
groupoid with 3 pieces:
< objects: [ -15 .. -11 ]
   group: a4 = <[ (1,2,3), (2,3,4) ]> >
< objects: [ -10 ]
   group: c6 = <[ (11,12,13)(14,15) ]> >
< objects: [ -9, -8, -7 ]
   group: d8 = <[ (5,6,7,8), (5,7) ]> >
gap> Pieces( U3 );
[ Ga4, Gc6, Gd8 ]
gap> ObjectList( U3 );
[ -15, -14, -13, -12, -11, -10, -9, -8, -7 ]
gap> [ Size(Ga4), Size(Gd8), Size(Gc6), Size(U3) ];
[ 300, 72, 6, 378 ]
gap> U2 := Groupoid( [ Gf2, Gq8 ] );
gap> [ Size(Gf2), Size(Gq8), Size(U2) ];
[ infinity, 72, infinity ]
gap> U5 := UnionOfPieces( [ U3, U2 ] );
groupoid with 5 pieces:
[ Gf2, Gq8, Ga4, Gc6, Gd8 ]
gap> V3 := ReplaceOnePieceInUnion( U3, Gd8, Gq8 );
groupoid with 3 pieces:
[ Gq8, Ga4, Gc6 ]
gap> ObjectList( V3 );
[ -19, -18, -17, -15, -14, -13, -12, -11, -10 ]
```

4.1.5 HomogeneousGroupoid

- ▷ `HomogeneousGroupoid(gpd, oblist)` (operation)
- ▷ `PieceIsomorphisms(hgpd)` (attribute)
- ▷ `HomogeneousDiscreteGroupoid(gp, obs)` (operation)

Special functions are provided for the case where a groupoid has more than one connected component, and when these components are identical except for their object sets. Such groupoids are said to be *homogeneous*.

The operation `HomogeneousGroupoid` is used when the components each contain more than one object. The arguments consist of a single piece groupoid `gpd` and a list of lists of objects `oblist`, each of whose lists has the same length as the object list `obs` of `gpd`. Note that `gpd` is *not* included as one of the pieces in the output unless `obs` is included as one of the lists in `oblist`.

The `PieceIsomorphisms` of a homogeneous groupoid are isomorphisms from the first piece to each of the others. See Chapter 5 for details of groupoid isomorphisms.

The operation `HomogeneousDiscreteGroupoid` is used when the components each have a single object. In this case the first argument is just a group -- the root group for each component. These groupoids are used in the `XMod` package as the source of many crossed modules of groupoids.

Both types of groupoid have the property `IsHomogeneousDomainWithObjects`. In the latter case a separate representation `IsHomogeneousDiscreteGroupoidRep` is used.

Example

```
gap> HGd8 := HomogeneousGroupoid( Gd8,
> [ [-39,-38,-37], [-36,-35,-34], [-33,-32,-31] ] );
homogeneous groupoid with 3 pieces:
1: single piece groupoid: < d8, [ -39, -38, -37 ] >
2: single piece groupoid: < d8, [ -36, -35, -34 ] >
3: single piece groupoid: < d8, [ -33, -32, -31 ] >
gap> Size( HGd8 );    ## 8x3x3 + 8x3x3 + 8x3x3
216
gap> PieceIsomorphisms( HGd8 );
[ groupoid homomorphism :
  [ [ [(5,6,7,8) : -39 -> -39], [(5,7) : -39 -> -39], [() : -39 -> -38],
    [() : -39 -> -37] ],
    [ [(5,6,7,8) : -36 -> -36], [(5,7) : -36 -> -36], [() : -36 -> -35],
    [() : -36 -> -34] ] ], groupoid homomorphism :
  [ [ [(5,6,7,8) : -39 -> -39], [(5,7) : -39 -> -39], [() : -39 -> -38],
    [() : -39 -> -37] ],
    [ [(5,6,7,8) : -33 -> -33], [(5,7) : -33 -> -33], [() : -33 -> -32],
    [() : -33 -> -31] ] ] ]
gap> HDc6 := HomogeneousDiscreteGroupoid( c6, [-27..-24] );
homogeneous, discrete groupoid: < c6, [ -27 .. -24 ] >
gap> Size( HDc6 );    ## 6x4
24
gap> RepresentationsOfObject( Gd8 );
[ "IsComponentObjectRep", "IsAttributeStoringRep", "IsMW0SinglePieceRep" ]
gap> RepresentationsOfObject( HGd8 );
[ "IsComponentObjectRep", "IsAttributeStoringRep", "IsPiecesRep" ]
gap> RepresentationsOfObject( HDc6 );
[ "IsComponentObjectRep", "IsAttributeStoringRep",
  "IsHomogeneousDiscreteGroupoidRep" ]
```

```

gap> ktpo := KnownTruePropertiesOfObject( HDc6 );;
gap> ans :=
> [ "IsDuplicateFree", "IsAssociative", "IsCommutative",
>   "IsDiscreteDomainWithObjects", "IsHomogeneousDomainWithObjects" ];;
gap> ForAll( ans, a -> ( a in ktpo ) );
true

```

4.1.6 DirectProductOp

- ▷ DirectProductOp(list, gpd) (operation)
- ▷ Projection(gpd, pos) (operation)
- ▷ Embedding(gpd, pos) (operation)

The direct product of groupoids G, H has as root group the direct product of the root groups in G and H and as object list the cartesian product of their object lists. As usual with DirectProductOp the two parameters are a list of groupoids followed by the first entry in the list.

Operations Projection and Embedding are as for direct product of groups. See Chapter 5 for details of groupoid homomorphisms.

Example

```

gap> prod := DirectProductOp( [Gd8,Gc6], Gd8 );
single piece groupoid: < Group( [ (1,2,3,4), (1,3), (5,6,7)(8,9) ] ),
[ [ -9, -10 ], [ -8, -10 ], [ -7, -10 ] ] >
gap> Embedding( prod, 2 );
groupoid homomorphism :
[ [ [(11,12,13)(14,15) : -10 -> -10] ],
  [ [(5,6,7)(8,9) : [ -9, -10 ] -> [ -9, -10 ]] ] ]
gap> ## note that the first embedding has not yet been created
gap> DirectProductInfo( prod );
rec( embeddings := [ , groupoid homomorphism :
  [ [ [(11,12,13)(14,15) : -10 -> -10] ],
    [ [(5,6,7)(8,9) : [ -9, -10 ] -> [ -9, -10 ]] ] ] ], first := Gd8,
  groupoids := [ Gd8, Gc6 ], groups := [ d8, c6 ],
  objectlists := [ [ -9, -8, -7 ], [ -10 ] ], projections := [ ] )
gap> Projection( prod, 1 );
groupoid homomorphism :
[ [ [(1,2,3,4) : [ -9, -10 ] -> [ -9, -10 ]],
  [(1,3) : [ -9, -10 ] -> [ -9, -10 ]],
  [(5,6,7)(8,9) : [ -9, -10 ] -> [ -9, -10 ]],
  [()] : [ -9, -10 ] -> [ -8, -10 ]], [()] : [ -9, -10 ] -> [ -7, -10 ] ] ],
  [ [(5,6,7,8) : -9 -> -9], [(5,7) : -9 -> -9], [()] : -9 -> -9],
  [()] : -9 -> -8], [()] : -9 -> -7 ] ]

```


4.2 Groupoid elements; stars; costars; homsets

4.2.1 GroupoidElement

▷ GroupoidElement(<i>gpd</i> , <i>elt</i> , <i>tail</i> , <i>head</i>)	(operation)
▷ MWOfArrow(<i>arr</i>)	(operation)
▷ ElementOfArrow(<i>arr</i>)	(operation)
▷ TailOfArrow(<i>arr</i>)	(operation)
▷ HeadOfArrow(<i>arr</i>)	(operation)
▷ IsGroupoidElement(<i>arr</i>)	(Category)

The operation `GroupoidElement` is a synonym for the operation `Arrow`, as described in subsection 2.1.3. To recapitulate, an arrow *e* has four components: the groupoid `MWOfArrow(e)` to which it belongs; a group element, `ElementOfArrow(e)`; the tail (source) object, `TailOfArrow(e)`; and the head (target) object, `HeadOfArrow(e)`. These four components may be accessed using $\{e![i] \mid 1 \leq i \leq 4\}$.

Arrows have a *partial composition*: two arrows may be multiplied when the head of the first coincides with the tail of the second. If an attempt is made to multiply arrows where this condition does not hold, then the value `fail` is returned.

Example

```
gap> e1 := GroupoidElement( Gd8, (5,6,7,8), -9, -8 );
[(5,6,7,8) : -9 -> -8]
gap> e2 := Arrow( Gd8, (5,7), -8, -7 );
[(5,7) : -8 -> -7]
gap> [ MWOfArrow( e1 ), ElementOfArrow( e1 ),
>      TailOfArrow( e1 ), HeadOfArrow( e1 ) ];
[ Gd8, (5,6,7,8), -9, -8 ]
gap> [ e1![1], e1![2], e1![3], e1![4] ];
[ Gd8, (5,6,7,8), -9, -8 ]
gap> IsGroupoidElement( e1 );
true
gap> e1e2 := e1*e2;
[(5,6)(7,8) : -9 -> -7]
gap> SetInfoLevel( InfoGroupoids, 1 );;
gap> e2*e1;
#I head of the first arrow <> tail of the second
fail
gap> SetInfoLevel( InfoGroupoids, 0 );;
gap> e3 := Arrow( Gd8, (6,8), -7, -9 );;
gap> loop := e1e2*e3;
[(5,8,7,6) : -9 -> -9]
gap> loop^2;
[(5,7)(6,8) : -9 -> -9]
```

4.2.2 IdentityArrow

▷ IdentityArrow(<i>gpd</i> , <i>obj</i>)	(operation)
--	-------------

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

Example

```
gap> i8 := IdentityArrow( Gd8, -8 );
[() : -8 -> -8]
gap> [ e1*i8, i8*e1, e1^-1 ];
[ [(5,6,7,8) : -9 -> -8], fail, [(5,8,7,6) : -8 -> -9] ]
```

4.2.3 Order

▷ Order(arr)

(attribute)

A groupoid element is a *loop* when the tail and head coincide. In this case the order of the arrow is defined to be the order of its group element.

Example

```
gap> [ i8, loop ];
[ [() : -8 -> -8], [(5,8,7,6) : -9 -> -9] ]
gap> [ Order( i8 ), Order(loop) ];
[ 1, 4 ]
```

4.2.4 ObjectStar

▷ ObjectStar(gpd, obj)

(operation)

▷ ObjectCostar(gpd, obj)

(operation)

▷ Homset(gpd, tail, head)

(operation)

The *star* at obj is the set of arrows which have obj as tail, while the *costar* is the set of arrows which have obj as head. The *homset* from obj_1 to obj_2 is the set of arrows with the specified tail and head, and so is bijective with the elements of the object groups. Indeed, $\text{Homset}(G, o, o)$ is the object group at o . Thus every star and every costar is a union of homsets. The identity arrow 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 unnecessarily 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 vertex group d8>
gap> Size( star9 );
24
gap> ## print the elements in star9 from 19 to 24
gap> iter := Iterator( star9 );
gap> for i in [1..18] do a := NextIterator( iter ); od;
gap> for i in [19..24] do Print( i, " : ", NextIterator( iter ), "\n" ); od;
19 : [(5,6,7,8) : -9 -> -9]
20 : [(5,6,7,8) : -9 -> -8]
```

```

21 : [(5,6,7,8) : -9 -> -7]
22 : [(5,6)(7,8) : -9 -> -9]
23 : [(5,6)(7,8) : -9 -> -8]
24 : [(5,6)(7,8) : -9 -> -7]
gap> costar12 := ObjectCostar( Ga4, -12 );
<costar at -12 with vertex group a4>
gap> Size( costar12 );
60
gap> Elements( q8 );
[ <identity> of ..., x, y, y2, x*y, x*y2, y*y2, x*y*y2 ]
gap> hsetq8 := Homset( Gq8, -18, -17 );
<homset -18 -> -17 with head group q8>
gap> Perform( hsetq8, Display );
[<identity> of ... : -18 -> -17]
[x : -18 -> -17]
[y : -18 -> -17]
[y2 : -18 -> -17]
[x*y : -18 -> -17]
[x*y2 : -18 -> -17]
[y*y2 : -18 -> -17]
[x*y*y2 : -18 -> -17]

```

4.2.5 Groupoids with objects in common

Users are advised to select disjoint object sets when creating more than one groupoid. When groupoids G_1, G_2 do have an object o in common, if $a_1 = (g_1 : o_1 \rightarrow o) \in G_1$ and $a_2 = (g_2 : o \rightarrow o_2) \in G_2$ then any attempt to compose $a_1 * a_2$ will not succeed.

Example

```

gap> c5 := Group( (11,12,13,14,15) );;
gap> Gc5 := Groupoid( c5, [ -11, -10, -9 ] );;
gap> e3;
[(6,8) : -7 -> -9]
gap> e5 := Arrow( Gc5, (11,13,15,12,14), -9, -11 );
[(11,13,15,12,14) : -9 -> -11]
gap> SetInfoLevel( InfoGroupoids, 1 );;
gap> e3 * e5;
#I arrows not in the same groupoid or subgroupoid
fail
gap> SetInfoLevel( InfoGroupoids, 0 );;

```

4.3 Subgroupoids

4.3.1 Subgroupoid

- ▷ Subgroupoid(*args*) (function)
- ▷ IsSubgroupoid(*gpd*, *sgpd*) (operation)
- ▷ IsWideSubgroupoid(*gpd*, *sgpd*) (operation)

▷ `IsFullSubgroupoid(gpd, sgpd)` (operation)

Let S be a *subgroupoid* of a groupoid G . Then the object set of S is a subset of the objects of G , and the object groups are subgroups of the object groups in G . S is *wide* in G if both groupoids have the same object set. S is *full* if, for any two of its objects, the Homset is the same as that in G . The arrows of S form a subset of those of G , closed under multiplication and with tails and heads in the chosen object set.

There are a variety of constructors for a subgroupoid of a standard groupoid, as described in the following sections. The global function `Subgroupoid` should call the operation appropriate to the parameters provided.

4.3.2 SubgroupoidByObjects

▷ `SubgroupoidByObjects(gpd, obs)` (operation)
 ▷ `SubgroupoidBySubgroup(gpd, sgp)` (attribute)

The `SubgroupoidByObjects` of a groupoid `gpd` on a subset `obs` of its objects contains all the arrows of `gpd` with tail and head in `obs`.

The `SubgroupoidBySubgroup` of a connected groupoid `gpd` determined by a subgroup `sgp` of the root group is the wide subgroupoid with root group `sgp` and containing the rays of `gpd`.

Example

```
gap> Ha4 := SubgroupoidByObjects( Ga4, [-14,-13,-12] );
single piece groupoid: < a4, [ -14, -13, -12 ] >
gap> SetName( Ha4, "Ha4" );
gap> IsSubgroupoid( Ga4, Ha4 );
true
gap> c3a := Subgroup( a4, [ (1,2,3) ] );
gap> SetName( c3a, "c3a" );
gap> Hc3a := SubgroupoidBySubgroup( Ha4, c3a );
single piece groupoid: < c3a, [ -14, -13, -12 ] >
gap> [ IsWideSubgroupoid( Ga4, Ha4 ), IsWideSubgroupoid( Ha4, Hc3a ) ];
[ false, true ]
gap> [ IsFullSubgroupoid( Ga4, Ha4 ), IsFullSubgroupoid( Ha4, Hc3a ) ];
[ true, false ]
```

4.3.3 ParentList

▷ `ParentList(dwo)` (attribute)

When S is constructed as a groupoid of G then G is assigned as the Parent of S . The `ParentList` of S is the concatenation of the parent list of G (or `[]` if none exists) and G .

Example

```
gap> ParentList( Hc3a );
[ Ga4, Ha4 ]
```

4.3.4 SubgroupoidWithRays

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

(operation)

If groupoid G is of type `IsDirectProductWithCompleteDigraph` 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 arrow $(r : q \rightarrow p)$ for each of the objects p . The chosen loop arrow at q must be the identity arrow. These n arrows are called the *ray arrows* of the subgroupoid. The arrows in the homset from p to p' have the form $r^{-1}xr'$ where r, r' are the rays from q to p, p' respectively, and $x \in h$.

The operation `RayArrowsOfGroupoid` returns a list of arrows, one for each object, while the operation `RaysOfGroupoid` returns the list of group elements in these arrows.

Note that it is also possible to construct a subgroupoid with rays of a subgroupoid with rays.

In the following example we construct a subgroupoid `Gk4` of the groupoid `Ga4`, and then a second subgroupoid `Gc2`. The initial standard groupoid `Ga4` is set as the parent for both `Gk4` and `Gc2`.

Example

```
gap> k4 := Subgroup( a4, [ (1,2)(3,4), (1,3)(2,4) ] );;
gap> SetName( k4, "k4" );
gap> Gk4 := SubgroupoidWithRays( Ga4, k4,
> [ (), (1,2,3), (1,2,4), (1,3,4), (2,3,4) ] );;
single piece groupoid with rays: < k4, [ -15 .. -11 ],
[ (), (1,2,3), (1,2,4), (1,3,4), (2,3,4) ] >
gap> SetName( Gk4, "Gk4" );
gap> RayArrowsOfGroupoid( Ga4 );
[ [() : -15 -> -15], [(1,2,3) : -15 -> -14], [(1,2,4) : -15 -> -13], [(1,3,4) : -15 -> -12],
  [(2,3,4) : -15 -> -11] ]
gap> RayArrowsOfGroupoid( Gk4 );
[ [() : -15 -> -15], [(1,2,3) : -15 -> -14], [(1,2,4) : -15 -> -13],
  [(1,3,4) : -15 -> -12], [(2,3,4) : -15 -> -11] ]
gap> RaysOfGroupoid( Gk4 );
[ (), (1,2,3), (1,2,4), (1,3,4), (2,3,4) ]
gap> IsDirectProductWithCompleteDigraph( Gk4 );
false
gap> ObjectGroup( Gk4, -14 );
Group([ (1,4)(2,3), (1,2)(3,4) ])
gap> c2 := Subgroup( k4, [ (1,4)(2,3) ] );; SetName( c2, "c2" );
gap> Gc2 := Subgroupoid( Gk4, c2, [ (), (1,3,4), (1,4,3), (1,2,3), (1,3,2) ] );;
single piece groupoid with rays: < c2, [ -15 .. -11 ],
[ (), (1,3,4), (1,4,3), (1,2,3), (1,3,2) ] >
```

4.3.5 SubgroupoidByPieces

▷ SubgroupoidByPieces(*gpd*, *pieces*)

(operation)

The most general way to construct a subgroupoid is to use the operation `SubgroupoidByPieces`. Its two parameters are a groupoid and a list of *pieces*, each piece being specified either 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, or as a list `[sgp,obs,rays]` when a set of rays is required. In the example both types of piece are used.

Example

```
gap> Display( Ga4 );
perm single piece groupoid: Ga4
  objects: [ -15 .. -11 ]
    group: a4 = <[ (1,2,3), (2,3,4) ]>
gap> c3b := Subgroup( a4, [ (1,2,4) ] );
gap> SetName( c3b, "c3b" );
gap> pieces := [ [ c3a, [-14] ], [ c3b, [-13,-12], [(),(1,4)(2,3)] ] ];
gap> Jc3 := Subgroupoid( Ha4, pieces );
gap> SetName( Jc3, "Jc3" );
gap> Display( Jc3 );
groupoid with 2 pieces:
< objects: [ -14 ]
  group: c3a = <[ (1,2,3) ]> >
<
  objects: [ -13, -12 ]
  parent gpd: single piece groupoid: < a4, [ -13, -12 ] >
  root group: c3b = <[ (1,2,4) ]>
    rays: [ (), (1,4)(2,3) ]
gap> [ Parent( Jc3 ), IsWideSubgroupoid( Ha4, Jc3 ) ];
[ Ha4, true ]
gap> pJc3 := Pieces( Jc3 );
gap> Jc3a := pJc3[1]; SetName( Jc3a, "Jc3a" );
gap> Jc3b := pJc3[2]; SetName( Jc3b, "Jc3b" );
gap> U2;
groupoid with 2 pieces:
[ Gf2, Gq8 ]
gap> genf2b := List( GeneratorsOfGroup(f2), g -> g^2 );
[ f1^2, f2^2 ]
gap> f2b := Subgroup( f2, genf2b );
gap> SetName( f2b, "f2b" );
gap> JU2 := SubgroupoidByPieces( U2, [ [f2b,-20]], [q8,-17]] );
groupoid with 2 pieces:
1: single piece groupoid: < f2b, [ -20 ] >
2: single piece groupoid: < q8, [ -17 ] >
gap> [ IsWideSubgroupoid(U2,JU2), IsSubgroupoid(Gf2,Groupoid(f2b,-20))] ];
[ false, true ]
gap> pJU2 := Pieces( JU2 );
gap> SetName( pJU2[1], "JU2a" ); SetName( pJU2[2], "JU2b" );
```

4.3.6 PiecePositions

▷ PiecePositions(*gpd*, *sgpd*)

(operation)

When G is a groupoid with a number of pieces and H is a subgroupoid of G , it is useful to know for each piece of H the piece of G of which it is a subgroupoid. The inclusion mapping of H in G will be described in subsection InclusionMappingGroupoids (5.3.1).

Example

```

gap> T1 := UnionOfPieces( [Ha4,U2] );; Pieces( T1 );
[ Gf2, Gq8, Ha4 ]
gap> T2 := UnionOfPieces( [Jc3,JU2] );; Pieces( T2 );
[ JU2a, JU2b, Jc3a, Jc3b ]
gap> PiecePositions( T1, T2 );
[ 1, 2, 3, 3 ]
gap> InclusionMappingGroupoids( T1, T2 );
groupoid homomorphism from several pieces :
groupoid homomorphism : JU2a -> Gf2
[ [ [ [f1^2 : -20 -> -20], [f2^2 : -20 -> -20] ],
    [ [f1^2 : -20 -> -20], [f2^2 : -20 -> -20] ] ] ]
groupoid homomorphism : JU2b -> Gq8
[ [ [ [x : -17 -> -17], [y : -17 -> -17], [y2 : -17 -> -17] ],
    [ [x : -17 -> -17], [y : -17 -> -17], [y2 : -17 -> -17] ] ] ]
groupoid homomorphism :
[ [ [ [(1,2,3) : -14 -> -14] ], [ [(1,2,3) : -14 -> -14] ] ],
  [ [ [(1,2,4) : -13 -> -13], [(1,4)(2,3) : -13 -> -12] ],
    [ [(1,2,4) : -13 -> -13], [(1,4)(2,3) : -13 -> -12] ] ] ]

```

4.3.7 FullTrivialSubgroupoid

- ▷ FullTrivialSubgroupoid(*gpd*) (attribute)
- ▷ DiscreteTrivialSubgroupoid(*gpd*) (attribute)

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.) In the example $\text{id}(G)$ denotes the identity subgroup of G .

Example

```

gap> Ic3 := FullTrivialSubgroupoid( Jc3 );
groupoid with 2 pieces:
1: single piece groupoid: < id(c3a), [ -14 ] >
2: single piece groupoid: < id(c3b), [ -13, -12 ] >
gap> ParentList( Ic3 );
[ Ga4, Ha4, Jc3 ]
gap> DiscreteTrivialSubgroupoid( Gd8 );
homogeneous, discrete groupoid: < id(d8), [ -9, -8, -7 ] >

```

4.3.8 DiscreteSubgroupoid

- ▷ DiscreteSubgroupoid(*gpd*, *sgps*, *obs*) (operation)
- ▷ HomogeneousDiscreteSubgroupoid(*gpd*, *gp*, *obs*) (operation)
- ▷ MaximalDiscreteSubgroupoid(*gpd*) (attribute)

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*.

Example

```

gap> U3;
groupoid with 3 pieces:
[ Ga4, Gc6, Gd8 ]
gap> c4 := Subgroup( d8, [ (5,6,7,8) ] );; SetName( c4, "c4" );
gap> DiscreteSubgroupoid( U3, [ c3a, c3b, c6, c4 ], [-15,-13,-10,-7] );
groupoid with 4 pieces:
1: single piece groupoid: < c3a, [ -15 ] >
2: single piece groupoid: < c3b, [ -13 ] >
3: single piece groupoid: < c6, [ -10 ] >
4: single piece groupoid: < c4, [ -7 ] >
gap> HomogeneousDiscreteSubgroupoid( Ga4, k4, [-15,-13,-11] );
homogeneous, discrete groupoid: < a4, [ -15, -13, -11 ] >
gap> MaximalDiscreteSubgroupoid( Jc3 );
groupoid with 3 pieces:
1: single piece groupoid: < c3a, [ -14 ] >
2: single piece groupoid: < c3b, [ -13 ] >
3: single piece groupoid: < Group( [ (1,4,3) ] ), [ -12 ] >

```

4.3.9 SinglePieceSubgroupoidByGenerators

▷ SinglePieceSubgroupoidByGenerators(parent, gens)

(operation)

A set of arrows 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 together with 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 digraph, should be provided. This is the case needed for ConjugateGroupoid in section 4.5.2. Other cases will be added as time permits.

Example

```

gap> a1 := Arrow( Gk4, (1,2)(3,4), -15, -15 );;
gap> a2 := Arrow( Gk4, (1,3,2), -15, -13 );;
gap> a3 := Arrow( Gk4, (2,3,4), -15, -11 );;
gap> SinglePieceSubgroupoidByGenerators( Gk4, [a1,a2,a3] );
single piece groupoid with rays: < Group( [ (1,2)(3,4) ] ), [ -15, -13, -11 ],
[ (), (1,3,2), (2,3,4) ] >

```

4.3.10 SinglePieceGroupoidWithRays

▷ SinglePieceGroupoidWithRays(gp, obs, rays)

(operation)

A more general way of constructing a groupoid is to specify a group, a set of objects, and a set of rays. The rays must be composable with the elements of the group.

This construction is used in the XMod package to construct the group-groupoid which corresponds to a crossed module or cat^2 -group.

Example

```
gap> G3 := SinglePieceGroupoidWithRays( k4, [-15,-13,-11], [(),(1,2,4),(2,3,4)] );
single piece groupoid with rays: < k4, [ -15, -13, -11 ],
[ (), (1,2,4), (2,3,4) ] >
gap> IsSubgroupoid( Gk4, G3 );
true
```

4.4 Left, right and double cosets

4.4.1 RightCoset

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

If U is a subgroupoid of G , the *right cosets* Ug of U in G are the equivalence classes for the relation on the arrows of G where g_1 is related to g_2 if and only if $g_2 = u * g_1$ for some arrow 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 (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 arrow 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 and right cosets. The operation `LeftCosetRepresentativesFromObject(G, U, obj)` is used in Chapter 7, 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.

In the example the representative for the right coset `re4` is the seventeenth one in the printed list `rcra4`, namely `[():-12->-12]`.

Example

```
gap> e6 := Arrow( Ha4, (2,4,3), -13, -12 );;;
gap> re6 := RightCoset( Ha4, Jc3, e6 );
<right coset of Jc3b with representative [(2,4,3) : -13 -> -12]>
gap> Perform( re6, Display );
[(2,4,3) : -13 -> -12]
[(1,3,4) : -12 -> -12]
[(1,3,2) : -13 -> -12]
```

```

[(1,4,3) : -12 -> -12]
[(1,4)(2,3) : -13 -> -12]
[() : -12 -> -12]
gap> rcra4 := RightCosetRepresentatives( Ha4, Jc3 );
[ [() : -14 -> -14], [(1,2)(3,4) : -14 -> -14], [(1,3)(2,4) : -14 -> -14],
  [(1,4)(2,3) : -14 -> -14], [() : -14 -> -13], [(1,2)(3,4) : -14 -> -13],
  [(1,3)(2,4) : -14 -> -13], [(1,4)(2,3) : -14 -> -13], [() : -14 -> -12],
  [(1,2)(3,4) : -14 -> -12], [(1,3)(2,4) : -14 -> -12],
  [(1,4)(2,3) : -14 -> -12], [() : -13 -> -13], [(1,2)(3,4) : -13 -> -13],
  [(1,3)(2,4) : -13 -> -13], [(1,4)(2,3) : -13 -> -13], [() : -12 -> -12],
  [(1,2)(3,4) : -12 -> -12], [(1,3)(2,4) : -12 -> -12],
  [(1,4)(2,3) : -12 -> -12], [() : -13 -> -14], [(1,2)(3,4) : -13 -> -14],
  [(1,3)(2,4) : -13 -> -14], [(1,4)(2,3) : -13 -> -14] ]
gap> le6 := LeftCoset( Ha4, Jc3, e6 );
<left coset of Jc3b with representative [(2,4,3) : -13 -> -12]>
gap> Perform( le6, Display );
[(1,4,2) : -13 -> -13]
[(2,4,3) : -13 -> -12]
[() : -13 -> -13]
[(1,4)(2,3) : -13 -> -12]
[(1,2,4) : -13 -> -13]
[(1,3,2) : -13 -> -12]
gap> lcra4 := LeftCosetRepresentatives( Ha4, Jc3 );
[ [() : -14 -> -14], [(1,2)(3,4) : -14 -> -14], [(1,3)(2,4) : -14 -> -14],
  [(1,4)(2,3) : -14 -> -14], [() : -13 -> -14], [(1,2)(3,4) : -13 -> -14],
  [(1,3)(2,4) : -13 -> -14], [(1,4)(2,3) : -13 -> -14], [() : -12 -> -14],
  [(1,2)(3,4) : -12 -> -14], [(1,3)(2,4) : -12 -> -14],
  [(1,4)(2,3) : -12 -> -14], [() : -13 -> -13], [(1,2)(3,4) : -13 -> -13],
  [(1,3)(2,4) : -13 -> -13], [(1,4)(2,3) : -13 -> -13], [() : -12 -> -12],
  [(1,2)(3,4) : -12 -> -12], [(1,3)(2,4) : -12 -> -12],
  [(1,4)(2,3) : -12 -> -12], [() : -14 -> -13], [(1,2)(3,4) : -14 -> -13],
  [(1,3)(2,4) : -14 -> -13], [(1,4)(2,3) : -14 -> -13] ]
gap> lcr11 := LeftCosetRepresentativesFromObject( Ha4, Jc3, -12 );
[ [() : -12 -> -14], [(1,2)(3,4) : -12 -> -14], [(1,3)(2,4) : -12 -> -14],
  [(1,4)(2,3) : -12 -> -14], [() : -12 -> -12], [(1,2)(3,4) : -12 -> -12],
  [(1,3)(2,4) : -12 -> -12], [(1,4)(2,3) : -12 -> -12] ]
gap> de6 := DoubleCoset( Ha4, Jc3, Jc3, e6 );
<double coset of [ Jc3b, Jc3b ] with representative [(2,4,3) : -13 -> -12]>
gap> Perform( de6, Display );
[(1,4,2) : -13 -> -13]
[(2,4,3) : -13 -> -12]
[(1,2,3) : -12 -> -13]
[(1,3,4) : -12 -> -12]
[(1,2,4) : -13 -> -13]
[(1,3,2) : -13 -> -12]
[(2,3,4) : -12 -> -13]
[(1,4,3) : -12 -> -12]
[() : -13 -> -13]
[(1,4)(2,3) : -13 -> -12]
[(1,4)(2,3) : -12 -> -13]
[() : -12 -> -12]
gap> dcra4 := DoubleCosetRepresentatives( Ha4, Jc3, Jc3 );

```

```
[ [() : -14 -> -14], [(1,2)(3,4) : -14 -> -14], [() : -14 -> -13],
  [(1,2)(3,4) : -14 -> -13], [() : -13 -> -14], [(1,2)(3,4) : -13 -> -14],
  [() : -13 -> -13], [(1,2)(3,4) : -13 -> -13] ]
```

4.5 Conjugation

4.5.1 \backslash^\sim

▷ $\backslash^\sim(a)$

(operation)

Conjugation by an arrow $a = (c : p \rightarrow q)$ is the groupoid inner automorphism (see chapter 6) defined as follows. There are two cases to consider. In the case $p \neq q$,

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

In the case $p = q$,

- all the objects are fixed;
- loops at p are conjugated by c , so $(b : p \rightarrow p) \mapsto (b^c : p \rightarrow p)$;
- the rest of the costar and star at p are permuted,

$$(b : r \rightarrow p) \mapsto (bc : r \rightarrow p) \quad \text{and} \quad (b : p \rightarrow r) \mapsto (c^{-1}b : p \rightarrow r);$$

- the remaining arrows are unchanged.

The details of this construction may be found in section 3.2 of [AW10].

See section GroupoidInnerAutomorphism (6.1.2) where conjugation is used to construct groupoid automorphisms.

Example

```
gap> p := Arrow( Gd8, (5,7), -9, -9 );;
gap> q := Arrow( Gd8, (5,6,7,8), -8, -9 );;
gap> r := Arrow( Gd8, (5,6)(7,8), -9, -7 );;
gap> s := Arrow( Gd8, (5,6,7,8), -7, -8 );;
gap> ## conjugation with elements p, q, and r in Gd8:
gap> p^q;
[(6,8) : -8 -> -8]
gap> p^r;
[(6,8) : -7 -> -7]
```

```

gap> q^p;
[( ) : -8 -> -9]
gap> q^r;
[(6,8) : -8 -> -7]
gap> r^p;
[(5,8,7,6) : -9 -> -7]
gap> r^q;
[(6,8) : -8 -> -7]
gap> s^p;
[(5,6,7,8) : -7 -> -8]
gap> s^q;
[(5,7)(6,8) : -7 -> -9]
gap> s^r;
[(5,7) : -9 -> -8]

```

4.5.2 ConjugateGroupoid

▷ ConjugateGroupoid(gpd, e)

(operation)

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

Example

```

gap> a5 := Arrow( Ga4, (1,2,3), -13, -12 );
[(1,2,3) : -13 -> -12]
gap> ConjugateGroupoid( Ga4, Gk4, a5 );
single piece groupoid with rays: < Group( [ (1,2)(3,4), (1,3)(2,4) ] ),
[ -15, -14, -13, -12, -11 ], [ ( ), (1,2,3), (1,2)(3,4), (1,3)(2,4), (2,3,4) ] >

```

4.6 Groupoids formed using isomorphisms

Here we describe an alternative way of constructing a connected groupoid.

Object groups in a connected groupoid are isomorphic, so we may use a collection of isomorphisms to form a groupoid. Let G_1, G_2, \dots, G_n be isomorphic groups and, for $2 \leq i \leq n$, let $\mu_i : G_1 \rightarrow G_i$ be isomorphisms. Then $\mu_{ij} = \mu_i^{-1} * \mu_j$ is an isomorphism from G_i to G_j . If we take $\{u_1, \dots, u_n\}$ to be our set of objects, with G_i the object group at u_i , we may consider the arrows in the groupoid to have the form $[[g_i, g_j] : u_i \rightarrow u_j]$ where $g_i \in G_i$ and $g_j = \mu_{ij}(g_i) \in G_j$. The product of $[[g_i, g_j] : u_i \rightarrow u_j]$ and $[[g'_j, g_k] : u_j \rightarrow u_k]$ is $[[\mu_{ij}^{-1}(g_j g'_j), \mu_{jk}(g_j g'_j)] : u_i \rightarrow u_k]$.

4.6.1 GroupoidByIsomorphisms

▷ GroupoidByIsomorphisms(gp, obs, isos)

(operation)

▷ IsGroupoidByIsomorphisms(gpd)

(property)

The operation GroupoidByIsomorphisms takes a group G_1 as root group; a set of n objects; and a set of n isomorphisms from the root group, where the first isomorphism should be the identity mapping

on $G1$. The output is a single piece groupoid of type `IsGroupoidByIsomorphisms`. Its rays have the form $[One(G1), One(Gi)]$ where G_i is the image of the i -th isomorphism.

In the example we first take three permutation groups isomorphic to the symmetric group S_3 . There follows an isomorphic groupoid whose object groups are a permutation group; a pc-group and an fp-group.

Example

```
gap> s3a := Group( (1,2), (2,3) );;
gap> s3b := Group( (4,6,8)(5,7,9), (4,9)(5,8)(6,7) );;
gap> s3c := Group( (4,6,8)(5,7,9), (5,9)(6,8) );;
gap> SetName( s3a, "s3a" );;
gap> SetName( s3b, "s3b" );;
gap> SetName( s3c, "s3c" );;
gap> ida := IdentityMapping( s3a );;
gap> isoab := IsomorphismGroups( s3a, s3b );;
gap> isoac := IsomorphismGroups( s3a, s3c );;
gap> isos1 := [ ida, isoab, isoac ];;
gap> G1 := GroupoidByIsomorphisms( s3a, [-3,-2,-1], isos1 );;
gap> gens1 := GeneratorsOfGroupoid( G1 );
[[ [(1,2), (1,2)] : -3 -> -3], [(2,3), (2,3)] : -3 -> -3],
  [( ), ( ) ] : -3 -> -2], [( ), ( ) ] : -3 -> -1] ]
gap> x1 := ImageElm( isos1[2], (1,2) );;
gap> a1 := Arrow( G1, [ (1,2), x1 ], -3, -2 );
[[ (1,2), (4,5)(6,9)(7,8) ] : -3 -> -2]
gap> a1^-1;
[[ (4,5)(6,9)(7,8), (1,2) ] : -2 -> -3]
gap> y1 := ImageElm( isos1[2], (2,3) );;
gap> z1 := ImageElm( isos1[3], (2,3) );;
gap> b1 := Arrow( G1, [ y1, z1 ], -2, -1 );
[[ (4,9)(5,8)(6,7), (5,9)(6,8) ] : -2 -> -1]
gap> c1 := a1*b1;
[[ (1,3,2), (4,8,6)(5,9,7) ] : -3 -> -1]

gap> isopc := IsomorphismPcGroup( s3a );;
gap> s3p := Image( isopc );;
gap> f2 := FreeGroup( 2 );;
gap> s3f := f2/[ f2.1^3, f2.2^2, (f2.1*f2.2)^2 ];;
gap> isofp := GroupHomomorphismByImages(s3a,s3f,[(1,2,3),(2,3)], [s3f.1,s3f.2]);;
gap> isos2 := [ ida, isopc, isofp ];;
gap> G2 := GroupoidByIsomorphisms( s3a, [-6,-5,-4], isos2 );;
gap> gens2 := GeneratorsOfGroupoid( G2 );
[[ [(1,2), (1,2)] : -6 -> -6], [(2,3), (2,3)] : -6 -> -6],
  [( ), <identity> of ... ] : -6 -> -5], [( ), <identity ...> ] : -6 -> -4]
]
gap> x2 := ImageElm( isos2[2], (1,2) );;
gap> a2 := Arrow( G2, [ (1,2), x2 ], -6, -5 );
[[ (1,2), f1*f2 ] : -6 -> -5]
gap> a2^-1;
[[ f1*f2, (1,2) ] : -5 -> -6]
gap> y2 := ImageElm( isos2[2], (2,3) );;
gap> z2 := ImageElm( isos2[3], (2,3) );;
gap> b2 := Arrow( G2, [ y2, z2 ], -5, -4 );
```

```
[[ f1, f2^-1 ] : -5 -> -4]
gap> c2 := a2*b2;
[[ (1,3,2), f1^2 ] : -6 -> -4]
```

4.7 Groupoids whose objects form a monoid

Let M be a monoid with G its maximal subgroup. We may form a groupoid with the elements of M as its objects and with arrows $t \rightarrow t * g$ for all $t \in M$ and $g \in G$.

4.7.1 GroupoidWithMonoidObjects

▷ `GroupoidWithMonoidObjects(gp)` (operation)

When M is a group, $G = M$ and the groupoid so constructed is a single piece which represents the regular representation of G . The ray from 1 to g is just g since $1 * g = g$.

Example

```
gap> d8 := Group( (5,6,7,8), (5,7) );;
gap> SetName( d8, "d8" );;
gap> ed8 := Elements( d8 );;
gap> Rd8 := GroupoidWithMonoidObjects( d8 );
single piece groupoid: < d8, [ (), (6,8), (5,6)(7,8), (5,6,7,8), (5,7),
(5,7)(6,8), (5,8,7,6), (5,8)(6,7) ] >
gap> Homset( Rd8, (6,8), (5,7) );
<homset (6,8) -> (5,7) with head group d8>
gap> Display( last );
homset (6,8) -> (5,7) with elements:
[() : (6,8) -> (5,7)]
[(6,8) : (6,8) -> (5,7)]
[(5,7)(6,8) : (6,8) -> (5,7)]
[(5,7) : (6,8) -> (5,7)]
[(5,8,7,6) : (6,8) -> (5,7)]
[(5,8)(6,7) : (6,8) -> (5,7)]
[(5,6,7,8) : (6,8) -> (5,7)]
[(5,6)(7,8) : (6,8) -> (5,7)]
```

When M is a monoid, rather than a group, this construction produces several components. One of these has as objects the elements of the group G of the monoid.

As a simple example we take a monoid M of size 13 generated by 2 transformations of degree 4. The groupoid has 8 components, of which 3 (the first, second and fourth) have a single object and group C_2 generated by `Transformation([1,2,4,3])`, while 5 have two objects and trivial group.

Example

```
gap> M13 := Monoid( Transformation( [1,1,2,3] ), Transformation( [1,2,4,3] ) );
<transformation monoid of degree 4 with 2 generators>
gap> Size( M13 );
13
```

```

gap> SetName( M13, "M13" );
gap> gpd13 := GroupoidWithMonoidObjects( M13 );
groupoid with 8 pieces:
1:  single piece groupoid: < gp(M13), [ Transformation( [ 1, 1, 1, 1 ] ) ] >
2:  single piece groupoid: < gp(M13), [ Transformation( [ 1, 1, 1, 2 ] ) ] >
3:  single piece groupoid: < gp(M13), [ Transformation( [ 1, 1, 1, 3 ] ),
    Transformation( [ 1, 1, 1 ] ) ] >
4:  single piece groupoid: < gp(M13), [ Transformation( [ 1, 1, 2, 1 ] ) ] >
5:  single piece groupoid: < gp(M13), [ Transformation( [ 1, 1, 2, 3 ] ),
    Transformation( [ 1, 1, 2 ] ) ] >
6:  single piece groupoid: < gp(M13), [ Transformation( [ 1, 1, 3, 1 ] ),
    Transformation( [ 1, 1, 4, 1 ] ) ] >
7:  single piece groupoid: < gp(M13), [ Transformation( [ 1, 1, 3, 2 ] ),
    Transformation( [ 1, 1, 4, 2 ] ) ] >
8:  single piece groupoid: < gp(M13),
[ IdentityTransformation, Transformation( [ 1, 2, 4, 3 ] ) ] >
gap> IsGroupoidWithMonoidObjects( gpd13 );
true
gap> obs13 := ObjectList( gpd13 );
gap> hs := Homset( gpd13, obs13[3], obs13[4] );
gap> Display( hs );
homset Transformation( [ 1, 1, 1, 3 ] ) -> Transformation( [ 1, 1, 1 ] )
  with elements:
[IdentityTransformation : Transformation( [ 1, 1, 1, 3 ] ) ->
Transformation( [ 1, 1, 1 ] )]
[Transformation( [ 1, 2, 4, 3 ] ) : Transformation( [ 1, 1, 1, 3 ] ) ->
Transformation( [ 1, 1, 1 ] )]

```

Chapter 5

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:

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

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, four-element 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 $n > 0$.

A variety of homomorphism operations are available.

- The basic construction is a homomorphism $\phi : G \rightarrow H$ from a connected groupoid G to a connected groupoid H , constructed using `GroupoidHomomorphismFromSinglePiece`, (see 5.1).
- Since more than one connected groupoid may be mapped to the same range, we then have the operation `GroupoidHomomorphismToSinglePiece`, (see 5.4).
- The third case arises when both source and range are unions of connected groupoids, in which case `HomomorphismByUnion` is called, (see 5.5).
- Fourthly, there is an additional operation for the case where the source is homogeneous and discrete, `GroupoidHomomorphismFromHomogeneousDiscrete`, (see 5.4.2).
- Finally, there are special operations for inclusion mappings, restricted mappings (see 5.3). and groupoid automorphisms (see 6).

5.1 Homomorphisms from a connected groupoid

5.1.1 `GroupoidHomomorphismFromSinglePiece`

- ▷ `GroupoidHomomorphismFromSinglePiece(src, rng, gens, images)` (operation)
- ▷ `GroupoidHomomorphism(args)` (function)
- ▷ `IsGroupoidHomomorphism(mwohom)` (Category)

The simplest groupoid homomorphism is a mapping $\phi : G \rightarrow H$ from a connected groupoid G to a connected groupoid H . There are two equivalent sets of input data which may be used. Both require the Source G and the Range H . The first then requires:

- the set of generating arrows, `genG = GeneratorsOfGroupoid(G)`;
- a list of image arrows `imphi` in H .

This may be implemented by the call `GroupoidHomomorphismFromSinglePiece(G,H,genG,imphi)`, and the data is stored in the attribute `MappingGeneratorsImages`. Alternatively, use the global function `GroupoidHomomorphism` with the same four parameters.

The alternative input data consists of:

- a homomorphism `rhom` from the root group of G to the group at the image object in H ;
- a list `imobs` of the images of the objects of G ;
- a list `imrays` of the elements in the images of the rays of G , so that the image $\phi(r_i : o_1 \rightarrow o_i)$ of the i -th ray is `(imrays[i]:imobs[1]→imobs[i])`.

This data is stored in the attribute `MappingToSinglePieceData`.

So an alternative way to construct this homomorphism of groupoids is to make a call of the form `GroupoidHomomorphism(G,H,rhom,imobs,imrays)`.

In the following example the same homomorphism is constructed using both methods.

Example

```
gap> Kk4 := SubgroupoidWithRays( Ha4, k4, [ (), (1,3,4), (1,4)(2,3) ] );
gap> SetName( Kk4, "Kk4" );
gap> gen1 := GeneratorsOfGroupoid( Gd8 );
[ [(5,6,7,8) : -9 -> -9], [(5,7) : -9 -> -9], [() : -9 -> -8],
  [() : -9 -> -7] ]
gap> gen2 := GeneratorsOfGroupoid( Kk4 );
[ [(1,2)(3,4) : -14 -> -14], [(1,3)(2,4) : -14 -> -14],
  [(1,3,4) : -14 -> -13], [(1,4)(2,3) : -14 -> -12] ]
gap> images := [ gen2[1]*gen2[2], gen2[1]^2, gen2[3], gen2[4] ];
[ [(1,4)(2,3) : -14 -> -14], [() : -14 -> -14], [(1,3,4) : -14 -> -13],
  [(1,4)(2,3) : -14 -> -12] ]
gap> hom8 := GroupoidHomomorphismFromSinglePiece( Gd8, Kk4, gen1, images );
groupoid homomorphism : Gd8 -> Kk4
[ [ [(5,6,7,8) : -9 -> -9], [(5,7) : -9 -> -9], [() : -9 -> -8],
    [() : -9 -> -7] ],
  [ [(1,4)(2,3) : -14 -> -14], [() : -14 -> -14], [(1,3,4) : -14 -> -13],
    [(1,4)(2,3) : -14 -> -12] ] ]
gap> gend8 := GeneratorsOfGroup( d8 );
gap> imh := [ (1,4)(2,3), () ];
gap> h := GroupHomomorphismByImages( d8, a4, gend8, imh );
[ (5,6,7,8), (5,7) ] -> [ (1,4)(2,3), () ]
gap> hom9 := GroupoidHomomorphism( Gd8, Kk4, h, [-14,-13,-12],
> [ (), (1,3,4), (1,4)(2,3) ] );
gap> hom8 = hom9;
true
gap> e1 := Arrow( Gd8, (5,6,7,8), -7, -8 );
```

```
gap> ImageElm( hom8, e1 );
[(1,3,4) : -12 -> -13]
gap> IsGroupoidHomomorphism( hom8 );
true
```

5.2 Properties and attributes of groupoid homomorphisms

5.2.1 Properties of a groupoid homomorphism

The properties listed in subsection 3.3 for homomorphisms of magmas with objects also apply to groupoid homomorphisms.

Example

```
gap> [ IsInjectiveOnObjects( hom8 ), IsSurjectiveOnObjects( hom8 ) ];
[ true, true ]
gap> [ IsInjective( hom8 ), IsSurjective( hom8 ) ];
[ false, false ]
gap> ad8 := GroupHomomorphismByImages( d8, d8,
>      [ (5,6,7,8), (5,7) ], [ (5,8,7,6), (6,8) ] );
gap> md8 := GroupoidHomomorphism( Gd8, Gd8, ad8,
>      [-7,-9,-8], [(),(5,7),(6,8)] );
groupoid homomorphism : Gd8 -> Gd8
[ [ [(5,6,7,8) : -9 -> -9], [(5,7) : -9 -> -9], [() : -9 -> -8],
  [() : -9 -> -7] ],
  [ [(5,8,7,6) : -7 -> -7], [(6,8) : -7 -> -7], [(5,7) : -7 -> -9],
  [(6,8) : -7 -> -8] ] ]
gap> IsBijectiveOnObjects( md8 );
true
gap> [ IsInjective( md8 ), IsSurjective( md8 ) ];
[ true, true ]
gap> [ IsEndomorphismWithObjects( md8 ), IsAutomorphismWithObjects( md8 ) ];
[ true, true ]
```

5.2.2 Attributes of a groupoid homomorphism

The attributes of a groupoid homomorphism `mor` from a single piece groupoid cover both forms of construction defined above.

- `S = Source(mor)` is the source groupoid of the homomorphism;
- `R = Range(mor)` is the range groupoid of the homomorphism;
- `RootGroupHomomorphism(mor)` is the group homomorphism from the root group of `S` to the group at the image object in `R` of the root object in `S`;
- `ImagesOfObjects(mor)` is the list of objects in `R` which are the images of the objects in `S`;
- `ImageElementsOfRays(mor)` is the list of group elements in those arrows in `R` which are the images of the rays in `S`;

- `MappingGeneratorsImages(mor)` is the two element list containing the list of generators in S and the list of their images in R ;
- `MappingToSinglePieceData(mor)` is a list with three elements: the root group homomorphism; the images of the objects; and the images of the rays.

For other types of homomorphism the attributes are very similar.

The function `ObjectGroupHomomorphism`, though an operation, is included in this section for convenience.

5.2.3 RootGroupHomomorphism

▷ `RootGroupHomomorphism(hom)` (attribute)

This is the group homomorphism from the root group of the source groupoid to the group at the image object in the range groupoid of the root object in the source.

5.2.4 ImagesOfObjects

▷ `ImagesOfObjects(hom)` (attribute)

This is the list of objects in the range groupoid which are the images of the objects in the source.

5.2.5 ImageElementsOfRays

▷ `ImageElementsOfRays(hom)` (attribute)

This is the list of group elements in those arrows in the range groupoid which are the images of the rays in the source.

Example

```
gap> RootGroupHomomorphism( hom8 );
[ (5,6,7,8), (5,7) ] -> [ (1,4)(2,3), () ]
gap> ImagesOfObjects( hom8 );
[ -14, -13, -12 ]
gap> ImageElementsOfRays( hom8 );
[ (), (1,3,4), (1,4)(2,3) ]
```

5.2.6 MappingToSinglePieceData (for groupoids)

▷ `MappingToSinglePieceData(map)` (attribute)

As mentioned earlier, this attribute stores the root group homomorphism; a list of the images of the objects; and a list of the elements in the images of the rays.

Example

```
gap> MappingGeneratorsImages( hom8 );
[ [ [(5,6,7,8) : -9 -> -9], [(5,7) : -9 -> -9], [() : -9 -> -8],
```

```

      [ ( ) : -9 -> -7 ] ],
    [ [ (1,4)(2,3) : -14 -> -14], [ ( ) : -14 -> -14], [(1,3,4) : -14 -> -13],
      [(1,4)(2,3) : -14 -> -12] ] ]
gap> MappingToSinglePieceData( hom8 );
[ [ [ (5,6,7,8), (5,7) ] -> [ (1,4)(2,3), ( ) ], [ -14, -13, -12 ],
  [ ( ), (1,3,4), (1,4)(2,3) ] ] ]

```

5.2.7 ObjectGroupHomomorphism

▷ `ObjectGroupHomomorphism(gpdhom, obj)` (operation)

For a given groupoid homomorphism, this operation gives the group homomorphism from an object group of the source to the object group at the image object in the range.

Example

```

gap> ObjectGroupHomomorphism( hom8, -8 );
[ (5,6,7,8), (5,7) ] -> [ (1,3)(2,4), ( ) ]

```

5.3 Special types of groupoid homomorphism

In this section we mention inclusion mappings of subgroupoids; and mappings restricted to a source subgroupoid. We also discuss various types of isomorphism: to a different set of objects; to a permutation groupoid; to a pc-groupoid.

5.3.1 InclusionMappingGroupoids

▷ `InclusionMappingGroupoids(gpd, sgpd)` (operation)

The operation `InclusionMappingGroupoids(gpd, sgpd)` returns the inclusion homomorphism from the subgroupoid `sgpd` to `gpd`.

Example

```

gap> incKk4 := InclusionMappingGroupoids( Ha4, Kk4 );
groupoid homomorphism : Kk4 -> Ha4
[ [ [(1,2)(3,4) : -14 -> -14], [(1,3)(2,4) : -14 -> -14],
  [(1,3,4) : -14 -> -13], [(1,4)(2,3) : -14 -> -12] ],
  [ [(1,2)(3,4) : -14 -> -14], [(1,3)(2,4) : -14 -> -14],
  [(1,3,4) : -14 -> -13], [(1,4)(2,3) : -14 -> -12] ] ]

```

For another example, refer back to subsection `PiecePositions` (4.3.6).

5.3.2 RestrictedMappingGroupoids

- ▷ `RestrictedMappingGroupoids(mor, sgpd)` (operation)
- ▷ `ParentMappingGroupoids(mor)` (attribute)

The operation `RestrictedMappingGroupoids(mor, sgpd)` returns the restriction of the homomorphism `mor` to the subgroupoid `sgpd` of its source. The range is usually set to the `ImagesSource` of the restriction. The restriction is assigned the attribute `ParentMappingGroupoids` with value `mor` (or that of `mor` if one exists). For another example see section 6.2.

Example

```
gap> Gc4 := Subgroupoid( Gd8, c4 ); SetName( Gc4, "Gc4" );
single piece groupoid: < c4, [ -9, -8, -7 ] >
gap> res4 := RestrictedMappingGroupoids( hom8, Gc4 );
groupoid homomorphism :
[ [ [(5,6,7,8) : -9 -> -9], [( ) : -9 -> -8], [( ) : -9 -> -7] ],
  [ [(1,4)(2,3) : -14 -> -14], [(1,3,4) : -14 -> -13],
    [(1,4)(2,3) : -14 -> -12] ] ]
gap> ParentMappingGroupoids( res4 ) = hom8;
true
```

5.3.3 IsomorphismNewObjects (for groupoids)

- ▷ `IsomorphismNewObjects(src, objlist)` (operation)

The operation `IsomorphismNewObjects(gpd, obs)` returns the isomorphism from a groupoid `gpd` to a groupoid with the same object group and ray elements but with a different set `obs` of objects.

We then compute the composite homomorphism, `mor8 : Gd8 -> Kk4 -> Ha4 -> Ga4`.

Example

```
gap> isoHa4 := IsomorphismNewObjects( Ha4, [-30,-29,-28] );
groupoid homomorphism :
[ [ [(1,2,3) : -14 -> -14], [(2,3,4) : -14 -> -14], [( ) : -14 -> -13],
  [( ) : -14 -> -12] ],
  [ [(1,2,3) : -30 -> -30], [(2,3,4) : -30 -> -30], [( ) : -30 -> -29],
    [( ) : -30 -> -28] ] ]
gap> Ka4 := Range( isoHa4 ); SetName( Ka4, "Ka4" );
single piece groupoid: < a4, [ -30, -29, -28 ] >
gap> IsSubgroupoid( Gk4, Kk4 );
true
gap> incHa4 := InclusionMappingGroupoids( Ga4, Ha4 );
gap> mor8 := hom8 * incKk4 * incHa4;
groupoid homomorphism : Gd8 -> Ga4
[ [ [(5,6,7,8) : -9 -> -9], [(5,7) : -9 -> -9], [( ) : -9 -> -8],
  [( ) : -9 -> -7] ],
  [ [(1,4)(2,3) : -14 -> -14], [( ) : -14 -> -14], [(1,3,4) : -14 -> -13],
    [(1,4)(2,3) : -14 -> -12] ] ]
gap> ImageElm( mor8, e1 );
[(1,3,4) : -12 -> -13]
```

5.3.4 IsomorphismStandardGroupoid

▷ `IsomorphismStandardGroupoid(gpd, obs)`

(operation)

The operation `IsomorphismStandardGroupoid(gpd, obs)` returns the isomorphism from a groupoid with rays to the groupoid of type `IsDirectProductWithCompleteDigraphDomain` on the given set `obs` of objects. `Gk4`, a subgroupoid of `Ga4`, was our first example of a groupoid with rays (see `SubgroupoidWithRays` (4.3.4)), and a standard isomorphic copy is formed here.

Example

```
gap> isoGk4 := IsomorphismStandardGroupoid( Gk4, [-45..-41] );
groupoid homomorphism :
[ [ [(1,2)(3,4) : -15 -> -15], [(1,3)(2,4) : -15 -> -15],
    [(1,2,3) : -15 -> -14], [(1,2,4) : -15 -> -13], [(1,3,4) : -15 -> -12],
    [(2,3,4) : -15 -> -11] ],
  [ [(1,2)(3,4) : -45 -> -45], [(1,3)(2,4) : -45 -> -45], [() : -45 -> -44],
    [() : -45 -> -43], [() : -45 -> -42], [() : -45 -> -41] ] ]
gap> G2k4 := Image( isoGk4 ); SetName( G2k4, "G2k4" );
single piece groupoid: < k4, [ -45 .. -41 ] >
gap> e5 := Arrow( Gk4, (1,2,4) , -13, -12 );
[(1,2,4) : -13 -> -12]
gap> ImageElm( isoGk4, e5 );
[(1,3)(2,4) : -43 -> -42]
gap> invGk4 := InverseGeneralMapping( isoGk4 );
groupoid homomorphism :
[ [ [(1,2)(3,4) : -45 -> -45], [(1,3)(2,4) : -45 -> -45], [() : -45 -> -44],
    [() : -45 -> -43], [() : -45 -> -42], [() : -45 -> -41] ],
  [ [(1,2)(3,4) : -15 -> -15], [(1,3)(2,4) : -15 -> -15],
    [(1,2,3) : -15 -> -14], [(1,2,4) : -15 -> -13], [(1,3,4) : -15 -> -12],
    [(2,3,4) : -15 -> -11] ] ]
```

This operation may also be used to provide a standard form for groupoids of type `IsGroupoidByIsomorphisms` as described in subsection `GroupoidByIsomorphisms` (4.6.1).

Example

```
gap> G2;
single piece groupoid with rays: < s3a, [ -6, -5, -4 ],
[ [ (), () ], [ (), <identity> of ... ], [ (), <identity ...> ] ] >
gap> isoG2 := IsomorphismStandardGroupoid( G2, [-44,-43,-42] );
groupoid homomorphism :
[ [ [(1,2), (1,2)] : -6 -> -6], [(2,3), (2,3)] : -6 -> -6],
  [ [ (), <identity> of ... ] : -6 -> -5], [((), <identity ...> ) : -6 ->
    -4] ],
  [ [(1,2) : -44 -> -44], [(2,3) : -44 -> -44], [() : -44 -> -43],
    [() : -44 -> -42] ] ]
```

5.3.5 IsomorphismPermGroupoid

- ▷ `IsomorphismPermGroupoid(gpd)` (attribute)
- ▷ `RegularActionHomomorphismGroupoid(gpd)` (attribute)
- ▷ `IsomorphismPcGroupoid(gpd)` (attribute)

The attribute `IsomorphismPermGroupoid(gpd)` returns an isomorphism from a groupoid `gpd` to a groupoid with the same objects but with an isomorphic permutation group.

The attribute `RegularActionHomomorphismGroupoid` returns an isomorphism from a groupoid `gpd` to a groupoid with the same objects but with an isomorphic regular presentation. In the example below these two operations produce equivalent permutation groupoids. Only the second is printed as the first is liable to change from one run to the next.

Similarly, the attribute `IsomorphismPcGroupoid(gpd)` attempts to return an isomorphism from the group to a pc-group with the same objects.

Example

```
gap> isoGq8 := IsomorphismPermGroupoid( Gq8 );
gap> regGq8 := RegularActionHomomorphismGroupoid( Gq8 );
groupoid homomorphism :
[ [ [x : -19 -> -19], [y : -19 -> -19], [y2 : -19 -> -19],
    [<identity> of ... : -19 -> -18], [<identity> of ... : -19 -> -17] ],
  [ [(1,2,4,6)(3,8,7,5) : -19 -> -19], [(1,3,4,7)(2,5,6,8) : -19 -> -19],
    [(1,4)(2,6)(3,7)(5,8) : -19 -> -19], [() : -19 -> -18],
    [() : -19 -> -17] ] ]
gap> Pq8 := Image( regGq8 ); SetName( Pq8, "Pq8" );
single piece groupoid: < Group( [ (1,2,4,6)(3,8,7,5), (1,3,4,7)(2,5,6,8),
  (1,4)(2,6)(3,7)(5,8) ] ), [ -19, -18, -17 ] >
gap> e7 := Arrow( Gq8, x*y, -18, -17 );
gap> ImageElm( regGq8, e7 );
[(1,5,4,8)(2,7,6,3) : -18 -> -17]
gap> isoGc4 := IsomorphismPcGroupoid( Gc4 );
groupoid homomorphism :
[ [ [(5,6,7,8) : -9 -> -9], [() : -9 -> -8], [() : -9 -> -7] ],
  [ [f1 : -9 -> -9], [<identity> of ... : -9 -> -8],
    [<identity> of ... : -9 -> -7] ] ]
```

5.4 Homomorphisms to a connected groupoid

5.4.1 HomomorphismToSinglePiece (for groupoids)

- ▷ `HomomorphismToSinglePiece(src, rng, piecehoms)` (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 individual homomorphisms *from* the single pieces (in the correct order!). See section 3.1 for the corresponding operation on homomorphisms of magmas with objects.

In the following example the source `V2` of `homV2` has two pieces, and both of the component homomorphisms are isomorphisms.

Example

```

gap> V2 := UnionOfPieces( Gq8, Gc4 );;
gap> genPq8 := GeneratorsOfGroupoid( Pq8 );;
gap> imGc4 := [ genPq8[1], genPq8[4], genPq8[5] ];
[ [(1,2,4,6)(3,8,7,5) : -19 -> -19], [() : -19 -> -18], [() : -19 -> -17] ]
gap> genGc4 := GeneratorsOfGroupoid( Gc4 );;
gap> homGc4 := GroupoidHomomorphism( Gc4, Pq8, genGc4, imGc4 );
groupoid homomorphism : Gc4 -> Pq8
[ [ [(5,6,7,8) : -9 -> -9], [() : -9 -> -8], [() : -9 -> -7] ],
  [ [(1,2,4,6)(3,8,7,5) : -19 -> -19], [() : -19 -> -18], [() : -19 -> -17] ] ]
gap> homV2 := HomomorphismToSinglePiece( V2, Pq8, [ regGq8, homGc4 ] );
groupoid homomorphism :
[ [ [ [x : -19 -> -19], [y : -19 -> -19], [y2 : -19 -> -19],
      [<identity> of ... : -19 -> -18], [<identity> of ... : -19 -> -17] ],
    [ [(1,2,4,6)(3,8,7,5) : -19 -> -19], [(1,3,4,7)(2,5,6,8) : -19 -> -19],
      [(1,4)(2,6)(3,7)(5,8) : -19 -> -19], [() : -19 -> -18],
      [() : -19 -> -17] ] ],
  [ [ [(5,6,7,8) : -9 -> -9], [() : -9 -> -8], [() : -9 -> -7] ],
    [ [(1,2,4,6)(3,8,7,5) : -19 -> -19], [() : -19 -> -18],
      [() : -19 -> -17] ] ] ]
gap> ImageElm( homV2, e7 );
[(1,5,4,8)(2,7,6,3) : -18 -> -17]

```

5.4.2 GroupoidHomomorphismFromHomogeneousDiscrete

▷ GroupoidHomomorphismFromHomogeneousDiscrete(*src*, *rng*, *homs*, *oims*) (operation)

This operation requires the source; the range; a list of homomorphisms from the object groups to the image object groups; and a list of the image objects. The source must be homogeneous and discrete, and the range must be a single piece. The example uses the groupoid HDc6 constructed in subsection HomogeneousGroupoid (4.1.5).

Example

```

gap> c3a := Subgroup( a4, [(1,2,3)] );; c3b := Subgroup( a4, [(1,2,4)] );;
gap> c3c := Subgroup( a4, [(1,3,4)] );; c3d := Subgroup( a4, [(2,3,4)] );;
gap> hc6a := GroupHomomorphismByImages( c6, c3a,
> [(11,12,13)(14,15)], [(1,2,3)] );;
gap> hc6b := GroupHomomorphismByImages( c6, c3b,
> [(11,12,13)(14,15)], [(1,2,4)] );;
gap> hc6c := GroupHomomorphismByImages( c6, c3c,
> [(11,12,13)(14,15)], [(1,3,4)] );;
gap> hc6d := GroupHomomorphismByImages( c6, c3d,
> [(11,12,13)(14,15)], [(2,3,4)] );;
gap> mor6 := GroupoidHomomorphismFromHomogeneousDiscrete( HDc6, Ga4,
> [ hc6a, hc6b, hc6c, hc6d ], [-15,-14,-12,-11] );
groupoid homomorphism : HDc6 -> Ga4
gap> e6 := Arrow( HDc6, (11,12,13), -25, -25 );;
gap> ImageElm( mor6, e6 );

```



```
[(1,3,4) : -12 -> -12]
```

5.5 Homomorphisms to more than one piece

5.5.1 HomomorphismByUnion (for groupoids)

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

As in section 3.3, when the range H has more than one connected component, a homomorphism is a union of homomorphisms, one for each piece in the range.

Example

```
gap> W1 := UnionOfPieces( Ha4, Gd8 );;
gap> W2 := UnionOfPieces( Ka4, Kk4 );;
gap> SetName( W1, "[Ha4,Gd8]" ); SetName( W2, "[Ka4,Kk4]" );
gap> homW := HomomorphismByUnion( W1, W2, [ isoHa4, hom8 ] );;
gap> Display( homW );
groupoid homomorphism: [Ha4,Gd8] -> [Ka4,Kk4] with pieces :
homomorphism to single piece groupoid: Ha4 -> Ka4
root group homomorphism:
(1,2,3) -> (1,2,3)
(2,3,4) -> (2,3,4)
object map: [ -14, -13, -12 ] -> [ -30, -29, -28 ]
ray images: [ (), (), () ]
homomorphism to single piece groupoid: Gd8 -> Kk4
root group homomorphism:
(5,6,7,8) -> (1,4)(2,3)
(5,7) -> ()
object map: [ -9, -8, -7 ] -> [ -14, -13, -12 ]
ray images: [ (), (1,3,4), (1,4)(2,3) ]
```

5.5.2 IsomorphismGroupoids

▷ `IsomorphismGroupoids(A, B)` (operation)

When A, B are two single piece groupoids, they are isomorphic provided they have the same number of objects and the root groups are isomorphic.

When $A = [A_1, \dots, A_n]$, $B = [B_1, \dots, B_n]$ are both unions of connected groupoids, they are isomorphic if there is a permutation π of $[1, \dots, n]$ such that A_i is isomorphic to $B_{\pi(i)}$ for all i .

Example

```
gap> s3b := Group( (4,6,8)(5,7,9), (4,9)(5,8)(6,7) );;
gap> s3c := Group( (4,6,8)(5,7,9), (5,9)(6,8) );;
gap> Gb := SinglePieceGroupoid( s3b, [-6,-5,-4] );;
gap> Gc := SinglePieceGroupoid( s3c, [-16,-15,-14] );;
gap> SetName( Gb, "Gb" ); SetName( Gc, "Gc" );
gap> c6b := Group( (1,2,3,4,5,6) );;
```

```

gap> c6c := Group( (7,8)(9,10,11) );;
gap> Hb := SinglePieceGroupoid( c6b, [-10,-9,-8,-7] );;
gap> Hc := SinglePieceGroupoid( c6c, [-20,-19,-18,-17] );;
gap> SetName( Hb, "Hb" ); SetName( Hc, "Hc" );
gap> IsomorphismGroupoids( Gb, Gc );
groupoid homomorphism : Gb -> Gc
[ [ [(4,6,8)(5,7,9) : -6 -> -6], [(4,9)(5,8)(6,7) : -6 -> -6],
    [() : -6 -> -5], [() : -6 -> -4] ],
  [ [(4,6,8)(5,7,9) : -16 -> -16], [(5,9)(6,8) : -16 -> -16],
    [() : -16 -> -15], [() : -16 -> -14] ] ]
gap> IsomorphismGroupoids( Gb, Hb );
fail
gap> B := UnionOfPieces( [ Gb, Hb ] );;
gap> C := UnionOfPieces( [ Gc, Hc ] );;
gap> isoBC := IsomorphismGroupoids( B, C );;
gap> Print( List( PiecesOfMapping(isoBC), p -> [Source(p),Range(p)] ) );
[ [ Hb, Hc ], [ Gb, Gc ] ]

```

Chapter 6

Automorphisms of Groupoids

In this chapter we consider automorphisms of single piece groupoids; then homogeneous discrete groupoids; and finally homogeneous groupoids. We also consider matrix representations and groupoid actions.

6.1 Automorphisms of single piece groupoids

6.1.1 GroupoidAutomorphismByObjectPerm

- ▷ `GroupoidAutomorphismByObjectPerm(gpd, imobs)` (operation)
- ▷ `GroupoidAutomorphismByGroupAuto(gpd, gpiso)` (operation)
- ▷ `GroupoidAutomorphismByNtuple(gpd, imrays)` (operation)
- ▷ `GroupoidAutomorphismByRayShifts(gpd, imrays)` (operation)

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

- given a permutation π of the n objects, we define

$$\alpha_\pi : G \rightarrow G, (g : o_i \rightarrow o_j) \mapsto (g : o_{\pi i} \rightarrow o_{\pi j});$$

- given an automorphism θ of the root group g , we define

$$\alpha_\theta : G \rightarrow G, (g : o_i \rightarrow o_j) \mapsto (\theta g : o_i \rightarrow o_j);$$

- given $L = [g_1, g_2, g_3, \dots, g_n] \in g^n$ we define

$$\alpha_L : G \rightarrow G, (g : o_i \rightarrow o_j) \mapsto (g_i^{-1} g g_j : o_i \rightarrow o_j).$$

If $g_1 = 1_g$, then for all j the rays $(r_j : o_1 \rightarrow o_j)$ are shifted by g_j : so they map to $(r_j g_j : o_1 \rightarrow o_j)$. So the operation `GroupoidAutomorphismByRayShifts` is the special case of `GroupoidAutomorphismByNtuple` when $g_1 = 1$.

Example

```
gap> perm1 := [-13,-12,-14];;
```

```

gap> aut1 := GroupoidAutomorphismByObjectPerm( Ha4, perm1 );;
gap> Display( aut1 );
homomorphism to single piece groupoid: Ha4 -> Ha4
root group homomorphism:
(1,2,3) -> (1,2,3)
(2,3,4) -> (2,3,4)
object map: [ -14, -13, -12 ] -> [ -13, -12, -14 ]
ray images: [ (), (), () ]
gap> d := Arrow( Ha4, (1,3,4), -12, -13 );
[(1,3,4) : -12 -> -13]
gap> d1 := ImageElm( aut1, d );
[(1,3,4) : -14 -> -12]
gap> gensa4 := GeneratorsOfGroup( a4 );;
gap> alpha2 := GroupHomomorphismByImages( a4, a4, gensa4, [(2,3,4), (1,3,4)] );;
gap> aut2 := GroupoidAutomorphismByGroupAuto( Ha4, alpha2 );;
gap> Display( aut2 );
homomorphism to single piece groupoid: Ha4 -> Ha4
root group homomorphism:
(1,2,3) -> (2,3,4)
(2,3,4) -> (1,3,4)
object map: [ -14, -13, -12 ] -> [ -14, -13, -12 ]
ray images: [ (), (), () ]
gap> d2 := ImageElm( aut2, d1 );
[(1,2,4) : -14 -> -12]
gap> L3 := [(1,2)(3,4), (1,3)(2,4), (1,4)(2,3)];;
gap> aut3 := GroupoidAutomorphismByNtuple( Ha4, L3 );;
gap> Display( aut3 );
homomorphism to single piece groupoid: Ha4 -> Ha4
root group homomorphism:
(1,2,3) -> (1,4,2)
(2,3,4) -> (1,4,3)
object map: [ -14, -13, -12 ] -> [ -14, -13, -12 ]
ray images: [ (), (1,4)(2,3), (1,3)(2,4) ]
gap> d3 := ImageElm( aut3, d2 );
[(2,3,4) : -14 -> -12]
gap> L4 := [(), (1,3,2), (2,4,3)];;
gap> aut4 := GroupoidAutomorphismByRayShifts( Ha4, L4 );;
gap> Display( aut4 );
homomorphism to single piece groupoid: Ha4 -> Ha4
root group homomorphism:
(1,2,3) -> (1,2,3)
(2,3,4) -> (2,3,4)
object map: [ -14, -13, -12 ] -> [ -14, -13, -12 ]
ray images: [ (), (1,3,2), (2,4,3) ]
gap> d4 := ImageElm( aut4, d3 );
[() : -14 -> -12]
gap> h4 := Arrow( Ha4, (2,3,4), -12, -13 );;
gap> aut1234 := aut1*aut2*aut3*aut4;;
gap> Display( aut1234 );
homomorphism to single piece groupoid: Ha4 -> Ha4
root group homomorphism:
(1,2,3) -> (1,4,3)

```

```

(2,3,4) -> (1,2,3)
object map: [ -14, -13, -12 ] -> [ -13, -12, -14 ]
ray images: [ (), (2,3,4), (1,3,4) ]
gap> d4 = ImageElm( aut1234, d );
true
gap> inv1234 := InverseGeneralMapping( aut1234 );
gap> Display( inv1234 );
homomorphism to single piece groupoid: Ha4 -> Ha4
root group homomorphism:
(1,2,3) -> (1,4,3)
(2,3,4) -> (2,4,3)
object map: [ -14, -13, -12 ] -> [ -12, -14, -13 ]
ray images: [ (), (1,3,2), (1,3,4) ]

```

6.1.2 GroupoidInnerAutomorphism

- ▷ GroupoidInnerAutomorphism(*gpd*, *arrow*) (operation)
- ▷ GroupoidInnerAutomorphismNormalSubgroupoid(*gpd*, *subgpd*, *arrow*) (operation)

Given an arrow $a = (c : p \rightarrow q) \in G$ with $p \neq q$, the *inner automorphism* α_a of G by a is the mapping $g \mapsto g^a$ where conjugation of arrows is defined in section 4.5. It is easily checked that if $L_a = [1, \dots, 1, c^{-1}, 1, \dots, 1, c, 1, \dots, 1]$, with c^{-1} in position p and c in position q , then

$$\alpha_a = \alpha_{(p,q)} * \alpha_{L_a}.$$

Similarly, when $p = q$, then $\alpha_a = \alpha_{L_a}$ where now $L_a = [1, \dots, 1, c, 1, \dots, 1]$, with c in position p .

— Example —

```

gap> inn1 := GroupoidInnerAutomorphism( Ha4, h4 );
gap> Display( inn1 );
homomorphism to single piece groupoid: Ha4 -> Ha4
root group homomorphism:
(1,2,3) -> (1,2,3)
(2,3,4) -> (2,3,4)
object map: [ -14, -13, -12 ] -> [ -14, -12, -13 ]
ray images: [ (), (2,4,3), (2,3,4) ]
gap> d5 := ImageElm( inn1, d4 );
[(2,3,4) : -14 -> -13]

```

Conjugation may also be applied to certain normal subgroupoids of G . Firstly, let N be the wide subgroupoid of G determined by a normal subgroup n of the root group. Then, provided the group element of a is in n , the inner automorphism by a may be applied to N .

— Example —

```

gap> Nk4 := SubgroupoidBySubgroup( Ha4, k4 );
gap> SetName( Nk4, "Nk4" );
gap> e4 := Arrow( Ha4, (1,2)(3,4), -14, -13 );
gap> inn2 := GroupoidInnerAutomorphismNormalSubgroupoid( Ha4, Nk4, e4 );

```

```

gap> Display( inn2 );
homomorphism to single piece groupoid: Nk4 -> Nk4
root group homomorphism:
(1,2)(3,4) -> (1,2)(3,4)
(1,3)(2,4) -> (1,3)(2,4)
object map: [ -14, -13, -12 ] -> [ -13, -14, -12 ]
ray images: [ (), (), (1,2)(3,4) ]

```

Secondly, if H is a homogeneous, discrete subgroupoid of G and if the group element of a is in the common vertex groups, then the inner automorphism may be applied to H .

Example

```

gap> Ma4 := MaximalDiscreteSubgroupoid( Ha4 );;
gap> SetName( Ma4, "Ma4" );
gap> inn3 := GroupoidInnerAutomorphismNormalSubgroupoid( Ha4, Ma4, e4 );;
gap> Display( inn3 );
homogeneous discrete groupoid mapping: [ Ma4 ] -> [ Ma4 ]
images of objects: [ -13, -14, -12 ]
object homomorphisms:
GroupHomomorphismByImages( a4, a4, [ (1,2,3), (2,3,4) ], [ (1,4,2), (1,4,3) ] )
GroupHomomorphismByImages( a4, a4, [ (1,2,3), (2,3,4) ], [ (1,4,2), (1,4,3) ] )
GroupHomomorphismByImages( a4, a4, [ (1,2,3), (2,3,4) ], [ (1,2,3), (2,3,4) ] )

```

6.1.3 Automorphisms of a groupoid with rays

Let S be a wide subgroupoid with rays of a standard groupoid G .

An automorphism α of the root group H extends to the whole of S with the rays fixed by the automorphism: $(r_i^{-1}hr_j : o_i \rightarrow o_j) \mapsto (r_i^{-1}(\alpha h)r_j : o_i \rightarrow o_j)$.

An automorphism of G obtained by permuting the objects may map S to a different subgroupoid. So we construct an isomorphism ι from S to a standard groupoid T , construct α permuting the objects of T , and return $\iota * \alpha * \iota^{-1}$.

For an automorphism by ray shifts we require that the shifts are elements of the root group of S .

Example

```

gap> ## (1) automorphism by group auto
gap> a6 := GroupHomomorphismByImages( k4, k4,
> [ (1,2)(3,4), (1,3)(2,4) ], [ (1,3)(2,4), (1,4)(2,3) ] );;
gap> aut6 := GroupoidAutomorphismByGroupAuto( Kk4, a6 );
groupoid homomorphism : Kk4 -> Kk4
[ [ (1,2)(3,4) : -14 -> -14], [(1,3)(2,4) : -14 -> -14],
  [(1,3,4) : -14 -> -13], [(1,4)(2,3) : -14 -> -12] ],
  [ [(1,3)(2,4) : -14 -> -14], [(1,4)(2,3) : -14 -> -14],
    [(1,3,4) : -14 -> -13], [(1,4)(2,3) : -14 -> -12] ] ]
gap> a := Arrow( Kk4, (1,3)(2,4), -12, -12 );;
gap> ImageElm( aut6, a );
[(1,4)(2,3) : -12 -> -12]
gap> b := Arrow( Kk4, (1,4,2), -12, -13 );;
gap> ImageElm( aut6, b );

```

```

[(1,2,3) : -12 -> -13]
gap> ## (2) automorphism by object perm
gap> aut7 := GroupoidAutomorphismByObjectPerm( Kk4, [-13,-12,-14] );
groupoid homomorphism : Kk4 -> Kk4
[ [ [(1,2)(3,4) : -14 -> -14], [(1,3)(2,4) : -14 -> -14],
    [(1,3,4) : -14 -> -13], [(1,4)(2,3) : -14 -> -12] ],
  [ [(1,4)(2,3) : -13 -> -13], [(1,2)(3,4) : -13 -> -13],
    [(2,3,4) : -13 -> -12], [(1,4,3) : -13 -> -14] ] ]
gap> ImageElm( aut7, a );
[(1,3)(2,4) : -14 -> -14]
gap> ImageElm( aut7, b );
[(1,3)(2,4) : -14 -> -12]
gap> ## (3) automorphism by ray shifts
gap> aut8 := GroupoidAutomorphismByRayShifts( Kk4,
>      [ (), (1,4)(2,3), (1,3)(2,4) ] );
groupoid homomorphism : Kk4 -> Kk4
[ [ [(1,2)(3,4) : -14 -> -14], [(1,3)(2,4) : -14 -> -14],
    [(1,3,4) : -14 -> -13], [(1,4)(2,3) : -14 -> -12] ],
  [ [(1,2)(3,4) : -14 -> -14], [(1,3)(2,4) : -14 -> -14],
    [(1,2,3) : -14 -> -13], [(1,2)(3,4) : -14 -> -12] ] ]
gap> ImageElm( aut8, a );
[(1,3)(2,4) : -12 -> -12]
gap> ImageElm( aut8, b );
[(1,2,3) : -12 -> -13]
gap> ## (4) combine these three automorphisms
gap> aut678 := aut6 * aut7 * aut8;
groupoid homomorphism : Kk4 -> Kk4
[ [ [(1,2)(3,4) : -14 -> -14], [(1,3)(2,4) : -14 -> -14],
    [(1,3,4) : -14 -> -13], [(1,4)(2,3) : -14 -> -12] ],
  [ [(1,2)(3,4) : -13 -> -13], [(1,3)(2,4) : -13 -> -13],
    [(1,4,3) : -13 -> -12], [(1,3,2) : -13 -> -14] ] ]
gap> ImageElm( aut678, a );
[(1,4)(2,3) : -14 -> -14]
gap> ImageElm( aut678, b );
[(1,4)(2,3) : -14 -> -12]
gap> ## (5) conjugation by an arrow
gap> e8 := Arrow( Kk4, (1,3)(2,4), -14, -12 );
gap> aut9 := GroupoidInnerAutomorphism( Kk4, e8 );
groupoid homomorphism : Kk4 -> Kk4
[ [ [(1,2)(3,4) : -14 -> -14], [(1,3)(2,4) : -14 -> -14],
    [(1,3,4) : -14 -> -13], [(1,4)(2,3) : -14 -> -12] ],
  [ [(1,2)(3,4) : -12 -> -12], [(1,3)(2,4) : -12 -> -12],
    [(1,4,2) : -12 -> -13], [(1,4)(2,3) : -12 -> -14] ] ]

```

6.1.4 AutomorphismGroupOfGroupoid

- ▷ AutomorphismGroupOfGroupoid(*gpd*) (operation)
- ▷ NiceObjectAutoGroupGroupoid(*gpd*, *aut*) (operation)

As above, let G be the direct product of a group g and a complete digraph with n objects. The

AutomorphismGroup $\text{Aut}(G)$ 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].

If H is the union of k groupoids, all isomorphic to G , then $\text{Aut}(H)$ is isomorphic to $S_k \times \text{Aut}(G)$.

The function `NiceObjectAutoGroupGroupoid` takes a groupoid and a subgroup of its automorphism group and returns a *nice monomorphism* from this automorphism group to a pc-group, if one is available. The current implementation is experimental. Note that `ImageElm` at present only works on generating elements.

Example

```
gap> AHa4 := AutomorphismGroupOfGroupoid( Ha4 );
Aut(Ha4)
gap> Agens := GeneratorsOfGroup( AHa4 );
gap> Length( Agens );
8
gap> NHa4 := NiceObject( AHa4 );
gap> MHa4 := NiceMonomorphism( AHa4 );
gap> Size( AHa4 );    ## (3!)x24x(12^2)
20736
gap> SetName( AHa4, "AHa4" );
gap> SetName( NHa4, "NHa4" );
gap> ## either of these names may be returned
gap> names := [ "((A4 x A4 x A4) : C2) : C3) : C2",
> "(C2 x C2 x C2 x C2 x C2 x C2) : (((C3 x C3 x C3) : C3) : (C2 x C2))" ];
gap> StructureDescription( NHa4 ) in names;
true
gap> ## cannot test images of Agens because of random variations
gap> ## Now do some tests!
gap> mgi := MappingGeneratorsImages( MHa4 );
gap> autgen := mgi[1];
gap> pcgen := mgi[2];
gap> ngen := Length( autgen );
gap> ForAll( [1..ngen], i -> Order(autgen[i]) = Order(pcgen[i]) );
true
```

6.1.5 Inner automorphisms

The inner automorphism subgroup $\text{Inn}(G)$ of the automorphism group of G is the group of inner automorphisms $\wedge a : b \mapsto b^a$ for $a \in G$. It is *not* the case that the map $G \rightarrow \text{Inn}(G), a \mapsto \wedge a$ preserves multiplication. Indeed, when $a = (o, g, p), b = (p, h, r) \in G$ with objects p, q, r all distinct, then

$$\wedge(ab) = (\wedge a)(\wedge b)(\wedge a) = (\wedge b)(\wedge a)(\wedge b).$$

(Compare this with the permutation identity $(pq)(qr)(pq) = (pr) = (qr)(pq)(qr)$.) So the map $G \rightarrow \text{Inn}(G)$ is of type `IsMappingWithObjectsByFunction`.

In the example we convert the automorphism group `AGa4` into a single object groupoid, and then define the inner automorphism map.

Example

```
gap> AHa40 := Groupoid( AHa4, [0] );
```



```

single piece groupoid: < Aut(Ha4), [ 0 ] >
gap> conj := function(a)
>   return ArrowNC( Ha4, 1, GroupoidInnerAutomorphism(Ha4,a), 0, 0 );
> end;;
gap> inner := MappingWithObjectsByFunction( Ha4, AHa40, conj, [0,0,0] );;
gap> a1 := Arrow( Ha4, (1,2,3), -14, -13 );;
gap> inner1 := ImageElm( inner, a1 );;
gap> a2 := Arrow( Ha4, (2,3,4), -13, -12 );;
gap> inner2 := ImageElm( inner, a2 );;
gap> a3 := a1*a2;
[(1,3)(2,4) : -14 -> -12]
gap> inner3 := ImageElm( inner, a3 );
[groupoid homomorphism : Ha4 -> Ha4
[ [ [(1,2,3) : -14 -> -14], [(2,3,4) : -14 -> -14], [() : -14 -> -13],
    [() : -14 -> -12] ],
  [ [(1,3,4) : -12 -> -12], [(1,2,4) : -12 -> -12], [(1,3)(2,4) : -12 -> -13],
    [() : -12 -> -14] ] ] : 0 -> 0]
gap> (inner3 = inner1*inner2*inner1) and (inner3 = inner2*inner1*inner2);
true

```

6.1.6 GroupoidAutomorphismByGroupAutos

▷ GroupoidAutomorphismByGroupAutos(*gpd*, *auts*)

(operation)

Homogeneous, discrete groupoids are the second type of groupoid for which a method is provided for AutomorphismGroupOfGroupoid. 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. So, if the object groups g have automorphism group $\text{Aut}(g)$ and there are n objects, the automorphism group of the groupoid has size $n!|\text{Aut}(g)|^n$.

Example

```

gap> Dd8 := HomogeneousDiscreteGroupoid( d8, [ -13..-10 ] );
homogeneous, discrete groupoid: < d8, [ -13 .. -10 ] >
gap> aut10 := GroupoidAutomorphismByObjectPerm( Dd8, [-12,-10,-11,-13] );
groupoid homomorphism : morphism from a homogeneous discrete groupoid:
[ -13, -12, -11, -10 ] -> [ -12, -10, -11, -13 ]
object homomorphisms:
IdentityMapping( d8 )
IdentityMapping( d8 )
IdentityMapping( d8 )
IdentityMapping( d8 )
gap> gend8 := GeneratorsOfGroup( d8 );;
gap> g1 := gend8[1];;
gap> g2 := gend8[2];;
gap> b1 := IdentityMapping( d8 );;
gap> b2 := GroupHomomorphismByImages( d8, d8, gend8, [g1, g2*g1] );;
gap> b3 := GroupHomomorphismByImages( d8, d8, gend8, [g1^g2, g2] );;
gap> b4 := GroupHomomorphismByImages( d8, d8, gend8, [g1^g2, g2^(g1*g2)] );;

```

```

gap> aut11 := GroupoidAutomorphismByGroupAutos( Dd8, [b1,b2,b3,b4] );
groupoid homomorphism : morphism from a homogeneous discrete groupoid:
[ -13, -12, -11, -10 ] -> [ -13, -12, -11, -10 ]
object homomorphisms:
IdentityMapping( d8 )
GroupHomomorphismByImages( d8, d8, [ (5,6,7,8), (5,7) ],
[ (5,6,7,8), (5,8)(6,7) ] )
GroupHomomorphismByImages( d8, d8, [ (5,6,7,8), (5,7) ], [ (5,8,7,6), (5,7) ] )
GroupHomomorphismByImages( d8, d8, [ (5,6,7,8), (5,7) ], [ (5,8,7,6), (6,8) ] )
gap> ADd8 := AutomorphismGroupOfGroupoid( Dd8 );
<group with 4 generators>
gap> Size( ADd8 );      ## 4!*8^4
98304
gap> genADd8 := GeneratorsOfGroup( ADd8 );
gap> Length( genADd8 );
4
gap> w := GroupoidAutomorphismByGroupAutos( Dd8, [b2,b1,b1,b1] );
gap> x := GroupoidAutomorphismByGroupAutos( Dd8, [b3,b1,b1,b1] );
gap> y := GroupoidAutomorphismByObjectPerm( Dd8, [ -12, -11, -10, -13 ] );
gap> z := GroupoidAutomorphismByObjectPerm( Dd8, [ -12, -13, -11, -10 ] );
gap> ok := ForAll( genADd8, a -> a in[ w, x, y, z ] );
true
gap> NADd8 := NiceObject( ADd8 );
gap> MADd8 := NiceMonomorphism( ADd8 );
gap> w1 := ImageElm( MADd8, w );
gap> x1 := ImageElm( MADd8, x );
gap> y1 := ImageElm( MADd8, y );
gap> z1 := ImageElm( MADd8, z );
gap> u := z*w*y*x*z;
groupoid homomorphism : morphism from a homogeneous discrete groupoid:
[ -13, -12, -11, -10 ] -> [ -11, -13, -10, -12 ]
object homomorphisms:
IdentityMapping( d8 )
GroupHomomorphismByImages( d8, d8, [ (5,6,7,8), (5,7) ],
[ (5,6,7,8), (5,8)(6,7) ] )
IdentityMapping( d8 )
GroupHomomorphismByImages( d8, d8, [ (5,6,7,8), (5,7) ], [ (5,8,7,6), (5,7) ] )
gap> u1 := z1*w1*y1*x1*z1;
(1,2,4,3)(5,17,23,11,6,18,24,16)(7,19,25,15,9,21,27,13)(8,20,26,14,10,22,28,12)
gap> imu := ImageElm( MADd8, u );
gap> u1 = imu;
true

```

6.1.7 AutomorphismGroupoidOfGroupoid

▷ AutomorphismGroupoidOfGroupoid(gpd)

(attribute)

If G is a single piece groupoid with automorphism group $\text{Aut}(G)$, and if H is the union of k pieces, all isomorphic to G , then the automorphism group of H is the wreath product $S_k \ltimes \text{Aut}(G)$. However, we find it more convenient to construct the *automorphism groupoid* of H . This is a single

piece groupoid $\text{AUT}(H)$ with k objects - the object lists of the pieces of H - and root group $\text{Aut}(G)$. Isomorphisms between the root groups of the k pieces may be applied to the generators of $\text{Aut}(G)$ to construct automorphism groups of these pieces, and then isomorphisms between these automorphism groups. We then construct $\text{AUT}(H)$ using `GroupoidByIsomorphisms`.

In the special case that H is homogeneous, there is no need to construct a collection of automorphism groups. Rather, the rays of $\text{AUT}(H)$ are given by `IsomorphismNewObjects`. For the example we use `HGd8` constructed in subsection `HomogeneousGroupoid (4.1.5)`.

Example

```
gap> HGd8 := HomogeneousGroupoid( Gd8,
> [ [-39,-38,-37], [-36,-35,-34], [-33,-32,-31] ] );
gap> SetName( HGd8, "HGd8" );
gap> AHGd8 := AutomorphismGroupoidOfGroupoid( HGd8 );
Aut(HGd8)
gap> ObjectList( AHGd8 );
[ [ -39, -38, -37 ], [ -36, -35, -34 ], [ -33, -32, -31 ] ]
gap> RaysOfGroupoid( AHGd8 ){[2..3]};
[ groupoid homomorphism :
  [ [ [(5,6,7,8) : -39 -> -39], [(5,7) : -39 -> -39], [( ) : -39 -> -38],
    [( ) : -39 -> -37] ],
    [ [(5,6,7,8) : -36 -> -36], [(5,7) : -36 -> -36], [( ) : -36 -> -35],
    [( ) : -36 -> -34] ] ], groupoid homomorphism :
  [ [ [(5,6,7,8) : -39 -> -39], [(5,7) : -39 -> -39], [( ) : -39 -> -38],
    [( ) : -39 -> -37] ],
    [ [(5,6,7,8) : -33 -> -33], [(5,7) : -33 -> -33], [( ) : -33 -> -32],
    [( ) : -33 -> -31] ] ] ]
gap> obgp := ObjectGroup( AHGd8, [ -36, -35, -34 ] );
gap> Size( obgp );    ## 3!*8^3
3072
```

6.2 Matrix representations of groupoids

Suppose that gpd is the direct product of a group G and a complete digraph, and that $\rho : G \rightarrow M$ is an isomorphism to a matrix group M . Then, if rep is the isomorphic groupoid with the same objects and root group M , there is an isomorphism μ from gpd to rep mapping $(g : i \rightarrow j)$ to $(\rho g : i \rightarrow j)$.

When gpd is a groupoid with rays, a representation can be obtained by restricting a representation of its parent.

Example

```
gap> reps := IrreducibleRepresentations( a4 );
gap> rep4 := reps[4];
Pcgs([ (2,4,3), (1,3)(2,4), (1,2)(3,4) ]) ->
[ [ [ 0, 0, 1 ], [ 1, 0, 0 ], [ 0, 1, 0 ] ],
  [ [ -1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, -1 ] ],
  [ [ 1, 0, 0 ], [ 0, -1, 0 ], [ 0, 0, -1 ] ] ]
gap> Ra4 := Groupoid( Image( rep4 ), Ga4!.objects );
gap> ObjectList( Ra4 ) = [ -15 .. -11 ];
true
gap> gens := GeneratorsOfGroupoid( Ga4 );
```

```

[ [(1,2,3) : -15 -> -15], [(2,3,4) : -15 -> -15], [() : -15 -> -14],
  [() : -15 -> -13], [() : -15 -> -12], [() : -15 -> -11] ]
gap> images := List( gens,
>   g -> Arrow( Ra4, ImageElm(rep4,g![2]), g![3], g![4] ) );
[ [[ [ 0, 0, -1 ], [ 1, 0, 0 ], [ 0, -1, 0 ] ] : -15 -> -15],
  [[ [ 0, 1, 0 ], [ 0, 0, 1 ], [ 1, 0, 0 ] ] : -15 -> -15],
  [[ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ] : -15 -> -14],
  [[ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ] : -15 -> -13],
  [[ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ] : -15 -> -12],
  [[ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ] : -15 -> -11] ]
gap> mor := GroupoidHomomorphismFromSinglePiece( Ga4, Ra4, gens, images );
groupoid homomorphism :
[ [ [(1,2,3) : -15 -> -15], [(2,3,4) : -15 -> -15], [() : -15 -> -14],
  [() : -15 -> -13], [() : -15 -> -12], [() : -15 -> -11] ],
  [ [[ [ 0, 0, -1 ], [ 1, 0, 0 ], [ 0, -1, 0 ] ] : -15 -> -15],
    [[ [ 0, 1, 0 ], [ 0, 0, 1 ], [ 1, 0, 0 ] ] : -15 -> -15],
    [[ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ] : -15 -> -14],
    [[ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ] : -15 -> -13],
    [[ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ] : -15 -> -12],
    [[ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ] : -15 -> -11] ] ]
gap> IsMatrixGroupoid( Ra4 );
true
gap> a := Arrow( Ha4, (1,4,2), -12, -13 );
[(1,4,2) : -12 -> -13]
gap> ImageElm( mor, a );
[[ [ 0, 0, 1 ], [ -1, 0, 0 ], [ 0, -1, 0 ] ] : -12 -> -13]
gap> rmor := RestrictedMappingGroupoids( mor, Ha4 );
groupoid homomorphism :
[ [ [(1,2,3) : -14 -> -14], [(2,3,4) : -14 -> -14], [() : -14 -> -13],
  [() : -14 -> -12] ],
  [ [[ [ 0, 0, -1 ], [ 1, 0, 0 ], [ 0, -1, 0 ] ] : -14 -> -14],
    [[ [ 0, 1, 0 ], [ 0, 0, 1 ], [ 1, 0, 0 ] ] : -14 -> -14],
    [[ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ] : -14 -> -13],
    [[ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ] : -14 -> -12] ] ]
gap> ParentMappingGroupoids( rmor ) = mor;
true

```

6.3 Groupoid actions

Recall from sections 4.5 and GroupoidInnerAutomorphism (6.1.2) the notion of *conjugation* in a groupoid, and the associated inner automorphisms.

It was mentioned there that the map $\wedge : G \rightarrow \text{Aut}(G)$, $a \mapsto \wedge a$, is *not* a groupoid homomorphism. It is in fact a *groupoid action* which we now define. Let $\{p, q, r, u, v\}$ be distinct objects in G and let:

$$a_1 = (c_1 : p \rightarrow q), \quad a_2 = (c_2 : q \rightarrow r), \quad a_3 = (c_3 : q \rightarrow p), \quad a_4 = (c_4 : u \rightarrow v),$$

$$b_1 = (d_1 : p \rightarrow p), \quad b_2 = (d_2 : p \rightarrow p), \quad b_3 = (d_3 : q \rightarrow q), \quad b_4 = (c_3 c_1 : q \rightarrow q)$$

be arrows in G . Then the following *conjugation identities* must be satisfied:

- $\wedge(a_1 a_2) = (\wedge a_1) * (\wedge a_2) * (\wedge a_1) = (\wedge a_2) * (\wedge a_1) * (\wedge a_2),$
- $\wedge(b_1 a_1) = (\wedge b_1) * (\wedge a_1) * (\wedge b_1)^{-1},$

- $\wedge(a_1 b_3) = (\wedge b_3)^{-1} * (\wedge a_1) * (\wedge b_3)$,
- $\wedge(b_1 b_2) = (\wedge b_1) * (\wedge b_2)$,
- $\wedge(a_1 a_3) = (\wedge a_1) * (\wedge a_3) * (\wedge b_4)$,
- $(\wedge a_1) * (\wedge a_4) = (\wedge a_4) * (\wedge a_1)$.

In the following example we check the first of these identities in one particular case.

Example

```
gap> c1 := Arrow( Ha4, (1,2)(3,4), -14, -13);;
gap> innc1 := GroupoidInnerAutomorphism( Ha4, c1 );
groupoid homomorphism : Ha4 -> Ha4
[ [ [(1,2,3) : -14 -> -14], [(2,3,4) : -14 -> -14], [() : -14 -> -13],
  [() : -14 -> -12] ],
  [ [(1,4,2) : -13 -> -13], [(1,4,3) : -13 -> -13], [() : -13 -> -14],
  [(1,2)(3,4) : -13 -> -12] ] ]
gap> c2 := Arrow( Ha4, (1,4,2), -13, -12);;
gap> innc2 := GroupoidInnerAutomorphism( Ha4, c2 );
groupoid homomorphism : Ha4 -> Ha4
[ [ [(1,2,3) : -14 -> -14], [(2,3,4) : -14 -> -14], [() : -14 -> -13],
  [() : -14 -> -12] ],
  [ [(1,2,3) : -14 -> -14], [(2,3,4) : -14 -> -14], [(1,4,2) : -14 -> -12],
  [(1,2,4) : -14 -> -13] ] ]
gap> c12 := c1 * c2;
[(2,4,3) : -14 -> -12]
gap> innc12 := GroupoidInnerAutomorphism( Ha4, c12 );
groupoid homomorphism : Ha4 -> Ha4
[ [ [(1,2,3) : -14 -> -14], [(2,3,4) : -14 -> -14], [() : -14 -> -13],
  [() : -14 -> -12] ],
  [ [(1,4,2) : -12 -> -12], [(2,3,4) : -12 -> -12], [(2,3,4) : -12 -> -13],
  [(2,4,3) : -12 -> -14] ] ]
gap> [ innc1 * innc2 * innc1 = innc12, innc2 * innc1 * innc2 = innc12 ];
[ true, true ]
```

6.3.1 GroupoidActionByConjugation

- | | |
|---|-------------|
| ▷ GroupoidActionByConjugation(<i>gpd</i>) | (operation) |
| ▷ IsGroupoidAction(<i>map</i>) | (Category) |
| ▷ ActionMap(<i>act</i>) | (attribute) |

The operation GroupoidInnerAutomorphism, which produces the conjugation action of G on itself, does satisfy the conjugation identities and so provides a standard example of an action.

An action is a record with fields Source, Range and ActionMap.

The examples repeat those in section GroupoidInnerAutomorphism (6.1.2): firstly with a groupoid acting on itself.

Example

```
gap> act1 := GroupoidActionByConjugation( Ha4 );
```

```

<general mapping: Ha4 -> Aut(Ha4) >
gap> IsGroupoidAction( act1 );
true
gap> amap1 := ActionMap( act1 );;
gap> amap1( h4 ) = inn1;
true

```

Secondly with an action on a single piece, normal subgroupoid.

Example

```

gap> act2 := GroupoidActionByConjugation( Ha4, Nk4 );
<general mapping: Ha4 -> Aut(Nk4) >
gap> IsGroupoidAction( act2 );
true
gap> amap2 := ActionMap( act2 );;
gap> amap2( e4 ) = inn2;
true

```

Thirly with an action on a homogeneous, discrete subgroupoid.

Example

```

gap> act3 := GroupoidActionByConjugation( Ha4, Ma4 );
<general mapping: Ha4 -> Aut(Ma4) >
gap> IsGroupoidAction( act3 );
true
gap> amap3 := ActionMap( act3 );;
gap> amap3( e4 ) = inn3;
true

```

Chapter 7

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.

7.1 Digraphs

7.1.1 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 := [7,8];;
gap> fg1 := FreeGroup("y");;
gap> y := fg1.1;;
gap> A1 := [ [y,7,8], [y^-1,8,7] ];;
gap> D1 := FpWeightedDigraph( fg1, V1, A1 );
weighted digraph with vertices: [ 7, 8 ]
and arcs: [ [ y, 7, 8 ], [ y^-1, 8, 7 ] ]
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> ob3 := [11,12,13];;
gap> A3 := [[z1,11,12],[z2,12,13],[z3,13,11],[z1^-1,12,11],[z2^-1,13,12],[z3^-1,11,13]];;
gap> D3 := FpWeightedDigraph( fg3, ob3, A3 );
weighted digraph with vertices: [ 11, 12, 13 ]
and arcs: [ [ z1, 11, 12 ], [ z2, 12, 13 ], [ z3, 13, 11 ], [ z1^-1, 12, 11 ],
[ z2^-1, 13, 12 ], [ z3^-1, 11, 13 ] ]
gap> inob3 := InvolutoryArcs( D3 );
[ 4, 5, 6, 1, 2, 3 ]
```

7.2 Graphs of Groups

7.2.1 GraphOfGroups

- | | |
|--|-------------|
| ▷ <code>GraphOfGroups(dig, gps, isos)</code> | (operation) |
| ▷ <code>DigraphOfGraphOfGroups(gg)</code> | (attribute) |
| ▷ <code>GroupsOfGraphOfGroups(gg)</code> | (attribute) |
| ▷ <code>IsomorphismsOfGraphOfGroups(gg)</code> | (attribute) |
| ▷ <code>IsGraphOfGroups(dig)</code> | (Category) |

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 in Section 7.4. 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: [ 7, 8 ]
    arcs: [ [ y, 7, 8 ], [ y^-1, 8, 7 ] ]
    groups: [ fa, fb ]
isomorphisms: [ [ [ a^3 ], [ b^2 ] ], [ [ b^2 ], [ a^3 ] ] ]
gap> IsGraphOfGroups( G1 );
true
```

7.2.2 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 );
[ [ a^3 ] -> [ b^2 ], [ b^2 ] -> [ a^3 ] ]
```

7.2.3 RightTransversalsOfGraphOfGroups

- ▷ `RightTransversalsOfGraphOfGroups(gg)` (attribute)
- ▷ `LeftTransversalsOfGraphOfGroups(gg)` (attribute)

Computation with graph of groups words will require, for each arc subgroup h_a , a set of representatives for the left cosets of h_a 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 ] ]
```

7.3 Words in a Graph of Groups and their normal forms

7.3.1 GraphOfGroupsWord

- ▷ `GraphOfGroupsWord(gg, tv, list)` (operation)
- ▷ `IsGraphOfGroupsWord(w)` (property)
- ▷ `GraphOfGroupsOfWord(w)` (attribute)
- ▷ `WordOfGraphOfGroupsWord(w)` (attribute)
- ▷ `TailOfGraphOfGroupsWord(w)` (attribute)
- ▷ `HeadOfGraphOfGroupsWord(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 $y^{-1} = \bar{y}$ 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 $w = [g_1, y_1, g_2, y_2, \dots, g_n, y_n, g_{n+1}]$. 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 .

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, 7, L1 );
(7)a^7.y.b^-6.y^-1.a^-11.y.b^9.y^-1.a^7(7)
gap> IsGraphOfGroupsWord( gw1 );
true
gap> [ TailOfGraphOfGroupsWord( gw1 ), HeadOfGraphOfGroupsWord( gw1 ) ];
[ 7, 7 ]
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 ]
```

7.3.2 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 $[y_i, \text{identity}, y_i\text{bar}]$ 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 $[t_1, y_1, t_2, y_2, \dots, t_n, y_n, k]$ 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:

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

Example

```
gap> nw1 := ReducedGraphOfGroupsWord( gw1 );
(7)a^-1.y.b^-1.y^-1.a^10(7)
```

7.4 Free products with amalgamation and HNN extensions

7.4.1 FreeProductWithAmalgamation

- ▷ `FreeProductWithAmalgamation(gp1, gp2, iso, verts)` (operation)
- ▷ `FreeProductWithAmalgamationInfo(fpa)` (attribute)
- ▷ `IsFreeProductWithAmalgamation(fpa)` (property)

- ▷ `GraphOfGroupsRewritingSystem(fpa)` (attribute)
- ▷ `NormalFormGGRWS(fpa, word)` (attribute)

As we have seen with the trefoil group example in Section 7.2, 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

```
## 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]);;
## 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> inv := InverseGeneralMapping(iso);;
gap> fpa := FreeProductWithAmalgamation( s3, a4, iso, [7,8] );
<fp group on the generators [ f1, f2, f3, f4 ]>
gap> RelatorsOfFpGroup( fpa );
[ f1^2, f2^3, (f2*f1)^2, f3^3, f4^3, (f4*f3)^2, f2*f3^-1 ]
gap> gg1 := GraphOfGroupsRewritingSystem( fpa );;
gap> Display( gg1 );
Graph of Groups with :-
  vertices: [ 7, 8 ]
    arcs: [ [ y, 7, 8 ], [ y^-1, 8, 7 ] ]
    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> gfpa := GeneratorsOfGroup( fpa );;
gap> w2 := (gfpa[1]*gfpa[2]*gfpa[3]^gfpa[4])^3;
(f1*f2*f4^-1*f3*f4)^3
gap> n2 := NormalFormGGRWS( fpa, w2 );
f2*f3*(f4^-1*f2)^2*f4^-1*f3
```

7.4.2 ReducedImageElm

- ▷ ReducedImageElm(hom, eml) (operation)
- ▷ IsMappingToGroupWithGGRWS(map) (property)
- ▷ Embedding(fpa, num) (method)

All fpa-groups are provided with a record attribute, FreeProductWithAmalgamationInfo(fpa) which is a record storing the groups, subgroups and isomorphism involved in their construction. This information record also contains the embeddings of the two groups into the product. The operation ReducedImageElm, applied to a homomorphism h of type IsMappingToGroupWithGGRWS and an element x of the source, finds the usual ImageElm(h, x) and then reduces this to its normal form using the graph of groups rewriting system.

Example

```
gap> fpainfo := FreeProductWithAmalgamationInfo( fpa );
rec( embeddings := [ [ a2, a1 ] -> [ f1, f2 ], [ b1, b2 ] -> [ f3, f4 ] ],
      groups := [ s3, a4 ], isomorphism := [ a1 ] -> [ b1 ],
      positions := [ [ 1, 2 ], [ 3, 4 ] ],
      subgroups := [ Group([ a1 ]), Group([ b1 ]) ], vertices := [ 7, 8 ] )
gap> emb2 := Embedding( fpa, 2 );
[ b1, b2 ] -> [ f3, f4 ]
gap> ImageElm( emb2, b1^b2 );
f4^-1*f3*f4
gap> ReducedImageElm( emb2, b1^b2 );
f4*f3^-1
```

7.4.3 HnnExtension

- ▷ HnnExtension(gp, iso, verts) (operation)
- ▷ HnnExtensionInfo(gp, iso) (attribute)
- ▷ IsHnnExtension(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 H_1, H_2 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, [9] );
<fp group of size infinity 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

```

As with fpa-groups, hnn-groups are provided with a record attribute, `HnnExtensionInfo(hnn)`, storing the group, subgroups and isomorphism involved in their construction.

Example

```

gap> hnninfo := HnnExtensionInfo( hnn );
rec( embeddings := [ [ b1, b2 ] -> [ fe1, fe2 ] ], group := a4,
      isomorphism := [ b1 ] -> [ b2 ],
      subgroups := [ Group([ b1 ]), Group([ b2 ]) ], vertices := [ 9 ] )
gap> emb := Embedding( hnn, 1 );
[ b1, b2 ] -> [ fe1, fe2 ]
gap> ImageElm( emb, b1^b2 );
fe2^-1*fe1*fe2
gap> ReducedImageElm( emb, b1^b2 );
fe2*fe1^-1

```

7.5 GraphsOfGroupoids and their Words

7.5.1 GraphOfGroupoids

- | | |
|---|-------------|
| ▷ <code>GraphOfGroupoids(dig, gpds, subgpds, isos)</code> | (operation) |
| ▷ <code>IsGraphOfPermGroupoids(gg)</code> | (property) |
| ▷ <code>IsGraphOfFpGroupoids(gg)</code> | (property) |
| ▷ <code>GroupoidsOfGraphOfGroupoids(gg)</code> | (attribute) |

- ▷ DigraphOfGraphOfGroupoids(*gg*) (attribute)
- ▷ SubgroupoidsOfGraphOfGroupoids(*gg*) (attribute)
- ▷ IsomorphismsOfGraphOfGroupoids(*gg*) (attribute)
- ▷ RightTransversalsOfGraphOfGroupoids(*gg*) (attribute)
- ▷ LeftTransversalsOfGraphOfGroupoids(*gg*) (attribute)
- ▷ IsGraphOfGroupoids(*dig*) (Category)

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, [-4,-3] );;
gap> ofa := One( fa );;
gap> SetName( Gfa, "Gfa" );
gap> Uhy := Subgroupoid( Gfa, [ [ hy, [-4,-3] ] ] );;
gap> SetName( Uhy, "Uhy" );
gap> Gfb := SinglePieceGroupoid( fb, [-6,-5] );;
gap> ofb := One( fb );;
gap> SetName( Gfb, "Gfb" );
gap> Uhybar := Subgroupoid( Gfb, [ [ hybar, [-6,-5] ] ] );;
gap> SetName( Uhybar, "Uhybar" );
gap> gens := GeneratorsOfGroupoid( Uhy );;
gap> gensbar := GeneratorsOfGroupoid( Uhybar );;
gap> mory := GroupoidHomomorphismFromSinglePiece(
>      Uhy, Uhybar, gens, gensbar );
groupoid homomorphism : Uhy -> Uhybar
[ [ [a^3 : -4 -> -4], [<identity ...> : -4 -> -3] ],
  [ [b^2 : -6 -> -6], [<identity ...> : -6 -> -5] ] ]
gap> morybar := InverseGeneralMapping( mory );
groupoid homomorphism : Uhybar -> Uhy
[ [ [b^2 : -6 -> -6], [<identity ...> : -6 -> -5] ],
  [ [a^3 : -4 -> -4], [<identity ...> : -4 -> -3] ] ]
gap> gg3 := GraphOfGroupoids( D1, [Gfa,Gfb], [Uhy,Uhybar], [mory,morybar] );;
gap> Display( gg3 );
Graph of Groupoids with :-
  vertices: [ 7, 8 ]
    arcs: [ [ y, 7, 8 ], [ y^-1, 8, 7 ] ]
  groupoids:
fp single piece groupoid: Gfa
  objects: [ -4, -3 ]
  group: fa = <[ a ]>
fp single piece groupoid: Gfb
  objects: [ -6, -5 ]
  group: fb = <[ b ]>
subgroupoids: single piece groupoid: Uhy
```

```

objects: [ -4, -3 ]
group: hy = <[ a^3 ]>
single piece groupoid: Uhybar
objects: [ -6, -5 ]
group: hybar = <[ b^2 ]>
isomorphisms: [ groupoid homomorphism : Uhy -> Uhybar
  [ [ [a^3 : -4 -> -4], [<identity ...> : -4 -> -3] ],
    [ [b^2 : -6 -> -6], [<identity ...> : -6 -> -5] ] ],
  groupoid homomorphism : Uhybar -> Uhy
  [ [ [b^2 : -6 -> -6], [<identity ...> : -6 -> -5] ],
    [ [a^3 : -4 -> -4], [<identity ...> : -4 -> -3] ] ] ]
gap> IsGraphOfGroupoids( gg3 );
true

```

7.5.2 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. Analogous to the word $a^7b^{-6}a^{-11}b^9a^7$ in subsection ReducedGraphOfGroupsWord (7.3.2) we shall consider

$$(a^7 : -1 \rightarrow -2) (b^{-6} : -4 \rightarrow -4) (a^{-11} : -2 \rightarrow -1) (b^9 : -3 \rightarrow -4) (a^7 : -2 \rightarrow -1).$$

Compare the normal form nw3 below with the normal form nw1 above.

Example

```

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

```

The reduction proceeds as follows.

$$\bullet [a^7 : -3 \rightarrow -4] = [a^{-2} : -3 \rightarrow -4] * [a^9 : -4 \rightarrow -4] \xrightarrow{y} [a^{-2} : -3 \rightarrow -4] * [b^6 : -6 \rightarrow -6]$$

- $[b^6 : -6 \rightarrow -6] * [b^{-6} : -6 \rightarrow -6] = [\text{id} : -6 \rightarrow -6] \xrightarrow{\bar{y}} [\text{id} : -4 \rightarrow -4]$
- $[a^{-2} : -3 \rightarrow -4] * [\text{id} : -4 \rightarrow -4] * [a^{-11} : -4 \rightarrow -3] = [a^{-13} : -3 \rightarrow -3]$
- $[a^{-13} : -3 \rightarrow -3] = [a^{-1} : -3 \rightarrow -3] * [a^{-12} : -3 \rightarrow -3] \xrightarrow{y} [a^{-1} : -3 \rightarrow -3] * [b^{-8} : -5 \rightarrow -5]$
- $[b^{-8} : -5 \rightarrow -5] * [b^9 : -5 \rightarrow -6] = [b^{-1} : -5 \rightarrow -6] * [b^2 : -6 \rightarrow -6]$
- $[b^2 : -6 \rightarrow -6] \xrightarrow{\bar{y}} [a^3 : -4 \rightarrow -4]$
- $[a^3 : -4 \rightarrow -4] * [a^7 : -4 \rightarrow -3] = [a^{10} : -4 \rightarrow -3]$

So the resulting word is $[a^{-1} : -3, -3][b^{-1} : -5, -6][a^{10} : -4, -3]$.

Chapter 8

Double Groupoids

A *double groupoid* is a *double category* in which all the category structures are groupoids. For the most general type of double groupoid there is also an associated pre-crossed module. In this package we consider only *basic double groupoids*, which do not involve pre-crossed modules. The more general case will be discussed in the *XMod* package.

In a double groupoid, as well as objects and arrows, we need a set of *squares*. A square is bounded by four arrows, two horizontal and two vertical, and there is a *horizontal* groupoid structure and a *vertical* groupoid structure on these squares.

Double groupoids can be considered where the vertical arrows come from one groupoid, and the horizontal arrows from another. The double groupoids constructed here are special in that all four arrows come from the same groupoid. We call these *edge-symmetric* double groupoids.

This addition to the package is very experimental, and will be extended as time permits.

8.1 Single piece double groupoids

8.1.1 SinglePieceBasicDoubleGroupoid

- ▷ `SinglePieceBasicDoubleGroupoid(gpd)` (operation)
- ▷ `DoubleGroupoid(args)` (function)
- ▷ `IsDoubleGroupoid(mwo)` (Category)
- ▷ `IsBasicDoubleGroupoid(dgpd)` (Category)

Let G be a connected groupoid with object set Ω . The double groupoid $\square(G)$ on G is constructed by the operation `SinglePieceBasicDoubleGroupoid(G)`.

The global function `DoubleGroupoid` may be used instead of this operation, and will work with various other input parameters.

Example

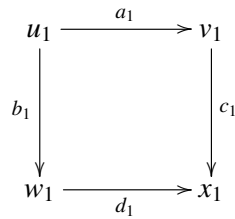
```
gap> DGd8 := SinglePieceBasicDoubleGroupoid( Gd8 );;
gap> DGd8!.groupoid;
Gd8
gap> DGd8!.objects;
[ -9, -8, -7 ]
gap> SetName( DGd8, "DGd8" );
gap> [ IsDoubleGroupoid( DGd8 ), IsBasicDoubleGroupoid( DGd8 ) ];
```

```
[ true, true ]
```

8.1.2 SquareOfArrows

- ▷ SquareOfArrows(*gpd*, *up*, *lt*, *rt*, *dn*) (operation)
- ▷ UpArrow(*sq*) (attribute)
- ▷ LeftArrow(*sq*) (attribute)
- ▷ RightArrow(*sq*) (attribute)
- ▷ DownArrow(*sq*) (attribute)
- ▷ BoundaryOfSquare(*sq*) (operation)
- ▷ DoubleGroupoidOfSquare(*sq*) (operation)
- ▷ IsDoubleGroupoidElement(*arrow*) (Category)

Let $\square(G)$ be the set of *squares* with objects from Ω at each corner; plus two vertical arrows and two horizontal arrows from $\text{Arr}(G)$. The following picture illustrates a square s_1 :



We name these four arrows $\text{UpArrow}(s_1)$, $\text{LeftArrow}(s_1)$, $\text{RightArrow}(s_1)$ and $\text{DownArrow}(s_1)$.

We think of the square s_1 being *based* at the bottom, right-hand corner, x_1 .

The *boundary* of the square is the loop $(x_1, d_1^{-1} b_1^{-1} a_1 c_1, x_1) = (x_1, \delta(s_1), x_1)$. The number of squares in a double groupoid is the product of the number of objects with the size of the group all raised to the fourth power. When viewing or printing a square, the boundary element is shown in the centre.

Example

```
gap> [ Size(DGd8), (3*8)^4 ];
[ 331776, 331776 ]
gap> a1 := Arrow( Gd8, (5,7), -7, -8 );;
gap> b1 := Arrow( Gd8, (6,8), -7, -7 );;
gap> c1 := Arrow( Gd8, (5,6)(7,8), -8, -9 );;
gap> d1 := Arrow( Gd8, (5,6,7,8), -7, -9 );;
gap> bdy1 := d1^-1 * b1^-1 * a1 * c1;
[(6,8) : -9 -> -9]
gap> sq1 := SquareOfArrows( DGd8, a1, b1, c1, d1 );
[-7] ----- (5,7) -----> [-8]
|
(6,8)          (6,8)          (5,6)(7,8)
  v              v
[-7] ----- (5,6,7,8) -----> [-9]
gap> sq1 in DGd8;
true
gap> UpArrow( sq1 );
```

```

[(5,7) : -7 -> -8]
gap> LeftArrow( sq1 );
[(6,8) : -7 -> -7]
gap> RightArrow( sq1 );
[(5,6)(7,8) : -8 -> -9]
gap> DownArrow( sq1 );
[(5,6,7,8) : -7 -> -9]
gap> BoundaryOfSquare( sq1 );
[(6,8) : -9 -> -9]
gap> DoubleGroupoidOfSquare( sq1 );
DGd8
gap> IsDoubleGroupoidElement( sq1 );
true

```

8.1.3 IsCommutingSquare

▷ IsCommutingSquare(sq) (property)

The square s_1 is *commuting* if $a_1 * c_1 = b_1 * d_1$, so that its boundary is the identity. The set of commutative squares in $\square(G)$ forms the *commutative sub-double groupoid* of $\square(G)$.

Example

```

gap> a2 := Arrow( Gd8, (6,8), -8, -9 );;
gap> c2 := Arrow( Gd8, (5,7)(6,8), -9, -8 );;
gap> d2 := Arrow( Gd8, (5,6,7,8), -9, -8 );;
gap> sq2 := SquareOfArrows( DGd8, a2, c1, c2, d2 );
[-8] ----- (6,8) -----> [-9]
    |                               |
    |                               |
(5,6)(7,8)      ( )      (5,7)(6,8)
    v                               v
[-9] ----- (5,6,7,8) -----> [-8]
gap> bdy2 := BoundaryOfSquare( sq2 );
[( ) : -8 -> -8]
gap> [ IsCommutingSquare(sq1), IsCommutingSquare(sq2) ];
[ false, true ]

```

8.1.4 TransposedSquare

▷ TransposedSquare(sq) (operation)
 ▷ IsClosedUnderTransposition(sq) (property)

The transpose of the square s_1 , as with matrix transposition, is obtained by interchanging a_1 with b_1 and c_1 with d_1 . Its boundary is the inverse of the boundary of s_1 .

Example

```

gap> tsq1 := TransposedSquare( sq1 );
[-7] ----- (6,8) -----> [-7]
    |                               |
    |                               |

```

```

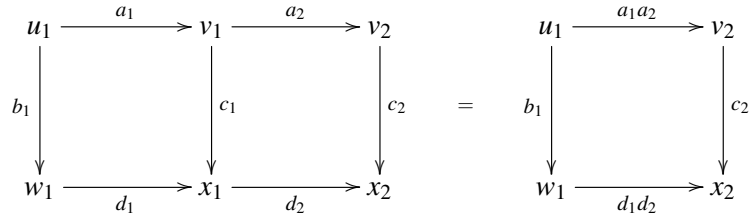
(5,7)      (6,8)      (5,6,7,8)
  v          v
[-8] ----- (5,6)(7,8) ----> [-9]
gap> IsClosedUnderTransposition( sq1 );
false

```

8.1.5 HorizontalProduct

▷ HorizontalProduct(sq1, sq2) (operation)

When $\text{RightArrow}(s_1) = \text{LeftArrow}(s_2)$ we may compose s_1 and s_2 *horizontally* to form the square $s_1(\rightarrow)s_2 = \text{HorizontalProduct}(s_1, s_2)$ as illustrated here:



Notice that the boundary of the composite satisfies the identity:

$$\delta(s_1(\rightarrow)s_2) = (d_1 d_2)^{-1} b_1^{-1} (a_1 a_2) c_2 = d_2^{-1} (d_1^{-1} b_1^{-1} a_1 c_1) d_2 (d_2^{-1} c_1^{-1} a_2 c_2) = (\delta s_1)^{d_2} (\delta s_2).$$

(This operation was called LeftRightProduct in versions up to 1.76.)

Example

```

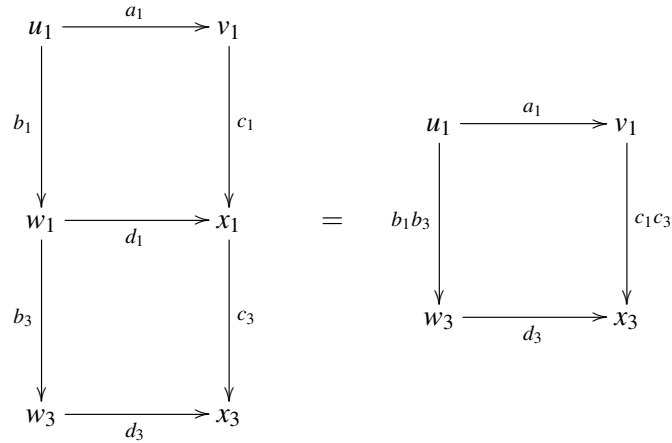
gap> LeftArrow( sq2 ) = RightArrow( sq1 );
true
gap> sq12 := HorizontalProduct( sq1, sq2 );
[-7] ----- (5,7)(6,8) ----> [-9]
      |               |
(6,8)      (5,7)      (5,7)(6,8)
      v               v
[-7] ----- (5,7)(6,8) ----> [-8]
gap> bdy12 := BoundaryOfSquare( sq12 );
[(5,7) : -8 -> -8]
gap> (bdy1^d2) * bdy2 = bdy12;
true

```

8.1.6 VerticalProduct

▷ VerticalProduct(sq1, sq3) (operation)

When $\text{DownArrow}(s_1) = \text{UpArrow}(s_3)$ we may compose s_1 and s_3 *vertically* to form $s_1(\downarrow)s_3 = \text{VerticalProduct}(s_1, s_3)$ illustrated by:



This time the boundary condition satisfies the identity:

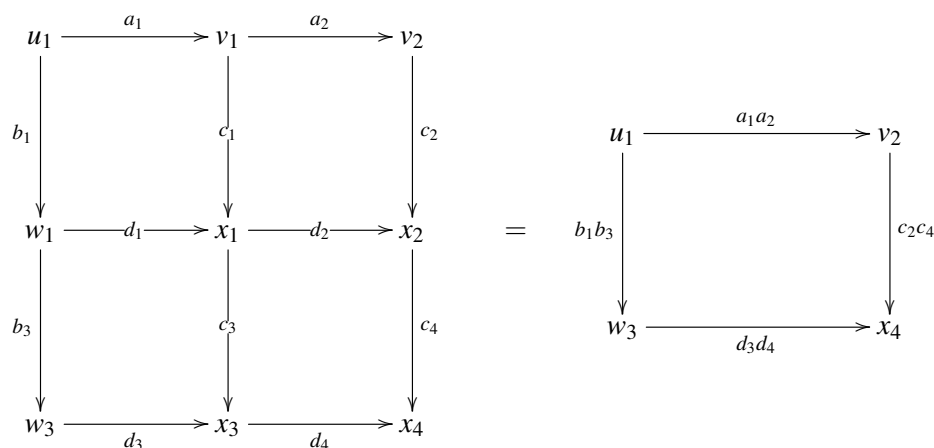
$$\delta(s_1(\downarrow)s_3) = d_3^{-1}(b_1b_3)^{-1}a_1(c_1c_3) = (d_3^{-1}b_3^{-1}d_1c_3)c_3^{-1}(d_1^{-1}b_1^{-1}a_1c_1)c_3 = (\delta s_3)(\delta s_1)^{c_3}.$$

(This operation was called `UpDownProduct` in versions up to 1.76.)

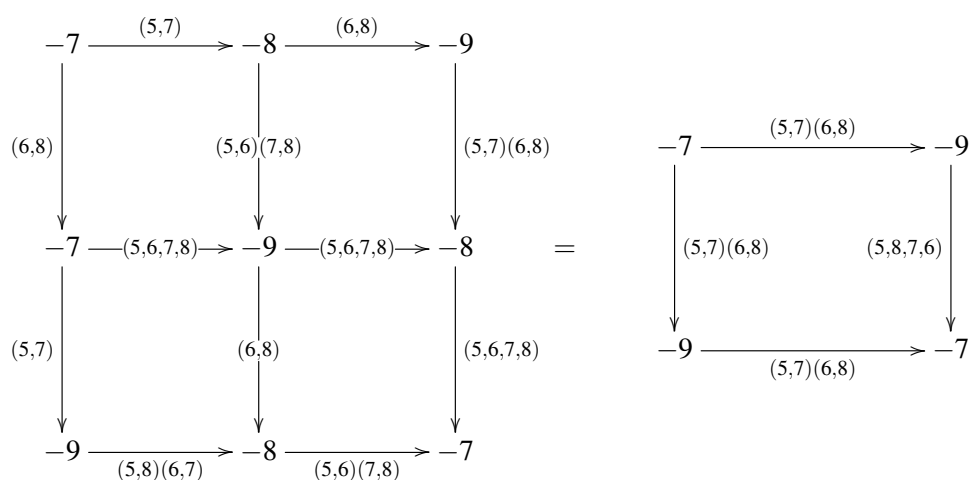
Example

```
gap> b3 := Arrow( Gd8, (5,7), -7, -9 );;
gap> c3 := Arrow( Gd8, (6,8), -9, -8 );;
gap> d3 := Arrow( Gd8, (5,8)(6,7), -9, -8 );;
gap> sq3 := SquareOfArrows( DGd8, d1, b3, c3, d3 );
[-7] ---- (5,6,7,8) ----> [-9]
|
(5,7)      (6,8)      (6,8)
v          v          v
[-9] ---- (5,8)(6,7) ----> [-8]
gap> bdy3 := BoundaryOfSquare( sq3 );
[(6,8) : -8 -> -8]
gap> UpArrow( sq3 ) = DownArrow( sq1 );
true
gap> sq13 := VerticalProduct( sq1, sq3 );
[-7] ----- (5,7) -----> [-8]
|
(5,7)(6,8)      ( )      (5,8,7,6)
v              v              v
[-9] ----- (5,8)(6,7) -----> [-8]
```

Vertical and horizontal compositions commute, so we may construct products such as:



In our example, after adding c_4 and d_4 , it is routine to check that the two ways of computing the product of four squares give the same answer.



Example

```
gap> c4 := Arrow( Gd8, (5,6,7,8), -8, -7 );;
gap> d4 := Arrow( Gd8, (5,6)(7,8), -8, -7 );;
gap> sq4 := SquareOfArrows( DGd8, d2, c3, c4, d4 );
[-9] ----- (5,6,7,8) -----> [-8]
|
(6,8)      (5,6,7,8)      (5,6,7,8)
v
[-8] ----- (5,6)(7,8) -----> [-7]
gap> UpArrow( sq4 ) = DownArrow( sq2 );
true
gap> LeftArrow( sq4 ) = RightArrow( sq3 );
true

gap> sq34 := HorizontalProduct( sq3, sq4 );
[-7] ----- (5,7)(6,8) -----> [-8]
|
|
```

```

(5,7)      (5,8)(6,7)      (5,6,7,8)
  V              V
[-9] ----- (5,7)(6,8) -----> [-7]

gap> sq1234 := VerticalProduct( sq12, sq34 );
[-7] ----- (5,7)(6,8) -----> [-9]
  |
(5,7)(6,8)      (5,6,7,8)      (5,8,7,6)
  V              V
[-9] ----- (5,7)(6,8) -----> [-7]

gap> sq24 := VerticalProduct( sq2, sq4 );
[-8] ----- (6,8) -----> [-9]
  |
(5,8,7,6)      (5,6,7,8)      (5,8,7,6)
  V              V
[-8] ----- (5,6)(7,8) -----> [-7]

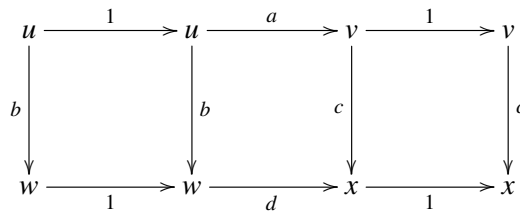
gap> sq1324 := HorizontalProduct( sq13, sq24 );
gap> sq1324 = sq1234;
true

```

8.1.7 HorizontalIdentities

- ▷ HorizontalIdentities(sq) (operation)
- ▷ VerticalIdentities(sq) (operation)
- ▷ HorizontalInverses(sq) (operation)
- ▷ VerticalInverses(sq) (operation)

There is no single identity for the operations `HorizontalProduct` and `VerticalProduct` but there are, for each square, a *left identity*, a *right identity*, an *up identity* and a *down identity*. The composite of the three squares shown below is equal to the central square s , and the other two squares are the left identity $1_L(s)$ and the right identity $1_R(s)$ for s .



Example

```

gap> hid := HorizontalIdentities( sq24 );
gap> hid[1]; Print("\n"); hid[2];
[-8] ----- () -----> [-8]
  |
(5,8,7,6)      ()      (5,8,7,6)
  V              V
[-8] ----- () -----> [-8]

```

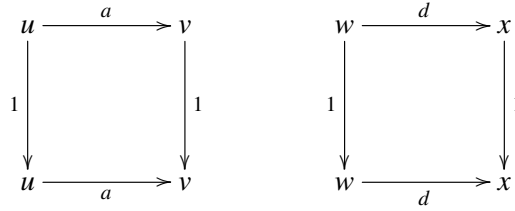


```

[-9] ----- () -----> [-9]
  |               |
(5,8,7,6)      ()      (5,8,7,6)
  v               v
[-7] ----- () -----> [-7]
gap> HorizontalProduct( hid[1], sq24 ) = sq24;
true
gap> HorizontalProduct( sq24, hid[2] ) = sq24;
true

```

Similarly, here are the up identity $1_U(s)$ and the down identity $1_D(s)$ of s :



Example

```

gap> vid := VerticalIdentities( sq24 );;
gap> vid[1]; Print("\n"); vid[2];
[-8] ---- (6,8) ----> [-9]
  |               |
()          ()      ()
  v               v
[-8] ---- (6,8) ----> [-9]

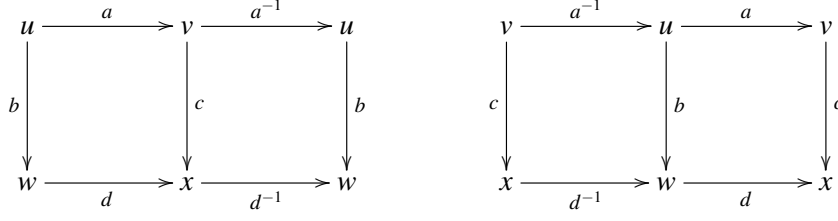
[-8] ---- (5,6)(7,8) ----> [-7]
  |               |
()          ()      ()
  v               v
[-8] ---- (5,6)(7,8) ----> [-7]
gap> VerticalProduct( vid[1], sq24 ) = sq24;
true
gap> VerticalProduct( sq24, vid[2] ) = sq24;
true

```

Confusingly, s has a *horizontal inverse* s_H^{-1} whose product with s is the left identity or right identity:

$$s(\rightarrow)s_H^{-1} = 1_L(s), \quad s_H^{-1}(\rightarrow)s = 1_R(s).$$

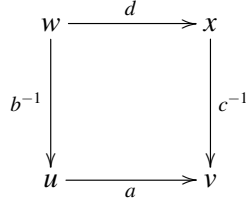
The boundary of s_H^{-1} is $dc^{-1}a^{-1}b = (\delta(s)^{-1})^{d^{-1}}$. Here are the two products:



Example

```
gap> hinv := HorizontalInverse( sq24 );
[-9] ----- (6,8) -----> [-8]
|
(5,8,7,6)      (5,6,7,8)      (5,8,7,6)
  v              v
[-7] ----- (5,6)(7,8) -----> [-8]
gap> HorizontalProduct( hinv, sq24 ) = hid[2];
true
gap> HorizontalProduct( sq24, hinv ) = hid[1];
true
```

Similarly, s has a *vertical inverse* s_V^{-1} whose product with s is an up or down identity: $s(\downarrow)s_V^{-1} = 1_U(s)$ and $s_V^{-1}(\downarrow)s = 1_D(s)$. The boundary is $a^{-1}bdc^{-1} = (\delta(s)^{-1})^{c^{-1}}$.



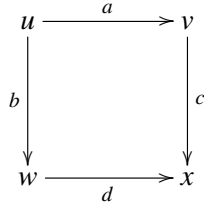
Example

```
gap> vinv := VerticalInverse( sq24 );
[-8] ----- (5,6)(7,8) -----> [-7]
|
(5,6,7,8)      (5,8,7,6)      (5,6,7,8)
  v              v
[-8] ----- (6,8) -----> [-9]
gap> VerticalProduct( vinv, sq24 ) = vid[2];
true
gap> VerticalProduct( sq24, vinv ) = vid[1];
true
```

8.1.8 Horizontal and vertical groupoids in $\square(G)$

Now $\square(G)$ is the maximal double groupoid determined by G , but in general many substructures may be formed. The *horizontal groupoid* structure $\square_H(G)$ on $\square(G)$ has the vertical arrows as objects, and

considers the usual square s



as an arrow from b to c . So the arrows in $\square_H(G)$ are effectively pairs of horizontal arrows $[a, d]$. The vertex groups are isomorphic to $G \times G$; the identity arrow at b is $1_L(s)$; and the inverse arrow of s is s_H^{-1} .

Similarly the *vertical groupoid* structure $\square_V(G)$ on $\square(G)$ has the horizontal arrows as objects and pairs of vertical arrows as arrows. The identity arrow at a is $1_U(s)$, and the inverse arrow of s is s_V^{-1} .

These groupoid structures have not been implemented in this package.

8.2 Double groupoids with more than one piece

As with groupoids, double groupoids may comprise a union of single piece double groupoids with disjoint object sets.

8.2.1 UnionOfPieces (for double groupoids)

- ▷ `UnionOfPieces(pieces)` (operation)
- ▷ `Pieces(dgpd)` (attribute)

The operation `UnionOfPieces` and the attribute `Pieces`, introduced in section 2.5, are also used for double 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 example shows that, as well as taking the union of two double groupoids, the same object may be constructed directly from the underlying groupoids.

Example

```

gap> DGc6 := SinglePieceBasicDoubleGroupoid( Gc6 );;
gap> DGa4 := SinglePieceBasicDoubleGroupoid( Ga4 );;
gap> DGc6s4 := DoubleGroupoid( [ DGc6, DGa4 ] );
double groupoid having 2 pieces :-
1: single piece double groupoid with:
  groupoid = Ga4
  group = a4
  objects = [ -15 .. -11 ]
2: single piece double groupoid with:
  groupoid = Gc6
  group = c6
  objects = [ -10 ]

gap> DGa4c6 := DoubleGroupoid( [ Ga4, Gc6 ] );;
gap> Pieces( DGa4c6 );
[ single piece double groupoid with:
  groupoid = Ga4

```

```

group = a4
objects = [ -15 .. -11 ], single piece double groupoid with:
groupoid = Gc6
group = c6
objects = [ -10 ] ]

```

8.3 Generators of a double groupoid

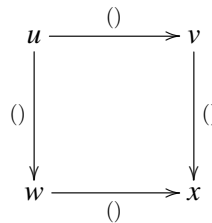
Before considering the general case we investigate two special cases:

- a basic double groupoid with identity group;
- a basic double groupoid with a single object.

8.3.1 DoubleGroupoidWithTrivialGroup

▷ `DoubleGroupoidWithTrivialGroup(obs)` (operation)

When $|\Omega| = n$ the double groupoid with trivial permutation group on these n objects contains n^4 squares of the form:



Example

```

gap> DGtriv := DoubleGroupoidWithTrivialGroup( [-19..-17] );
single piece double groupoid with:
groupoid = single piece groupoid: < Group( [ ( ) ] ), [ -19 .. -17 ] >
group = Group( [ ( ) ] )
objects = [ -19 .. -17 ]

gap> Size(DGtriv);
81

```

8.3.2 DoubleGroupoidWithSingleObject

▷ `DoubleGroupoidWithSingleObject(gp, obj)` (operation)

Given a group G we can form the corresponding groupoid with a single object, and from that a double groupoid on that object. The number of squares is $|G|^4$.

Example

```

gap> DGc4 := DoubleGroupoidWithSingleObject( Group((1,2,3,4)), 0 );

```

```

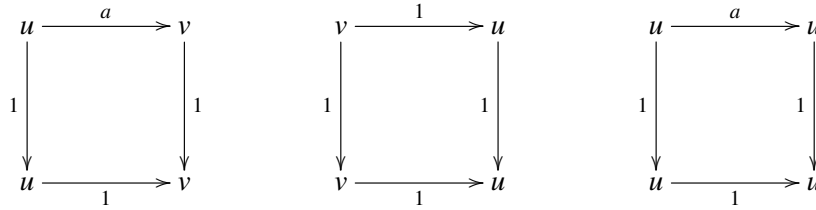
single piece double groupoid with:
  groupoid = single piece groupoid: < Group( [ (1,2,3,4) ] ), [ 0 ] >
    group = Group( [ (1,2,3,4) ] )
    objects = [ 0 ]

gap> Size( DGc4 );
256

```

8.3.3 What is the double groupoid generated by a set of squares?

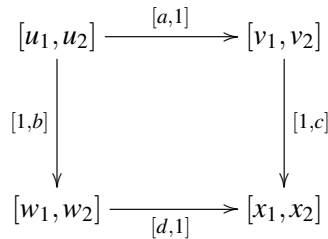
This is a very experimental section. Let us consider the following list of three squares $[s_U(a, u, v), 1_U(v, u), s_U(a, u, u)]$. What is generated by the single square $s_U(a, u, v)$?



The first square does not compose with itself, so cannot generate anything. When constructing a group from generators there is never any need to include an identity - that is always assumed to be included. Perhaps, when constructing a double groupoid, it should be assumed that the `DoubleGroupoidWithTrivialGroup` on the given objects should be automatically included? In that case the square $1_U(v, u)$ is available and can compose on the right to give $s_U(a, u, u)$. This then composes with itself to produce squares $s_U(a^i, u, u)$. Then, composing with identities, we obtain `SinglePieceBasicDoubleGroupoid(G)` where G is the groupoid with group $\langle a \rangle$ and objects $[u, v]$. More work on this area is required!

8.4 Starting with two groupoids

In the literature on double groupoids the construction often starts with two groupoids G_1, G_2 , and squares have horizontal arrows chosen from G_1 and vertical arrows chosen from G_2 . When that is the case, the boundary of a square is not defined, since arrows from G_1 do not compose with those from G_2 . This situation may be modelled here by constructing the direct product groupoid $G = G_1 \times G_2$ and forming a double groupoid on G in which squares have the form:



Example

```

gap> Gd8c6 := DirectProduct( Gd8, Gc6 );

```

```

single piece groupoid: < Group( [ (1,2,3,4), (1,3), (5,6,7)(8,9) ] ),
[ [ -9, -10 ], [ -8, -10 ], [ -7, -10 ] ] >
gap> SetName( Gd8c6, "Gd8c6" );
gap> DGd8c6 := SinglePieceBasicDoubleGroupoid( Gd8c6 );
single piece double groupoid with:
  groupoid = Gd8c6
    group = Group( [ (1,2,3,4), (1,3), (5,6,7)(8,9) ] )
    objects = [ [ -9, -10 ], [ -8, -10 ], [ -7, -10 ] ]

gap> emb1 := Embedding( Gd8c6, 1 );;
gap> emb2 := Embedding( Gd8c6, 2 );;
gap> a5 := Arrow( Gd8, (5,7), -9, -7 );;
gap> a6 := ImageElm( emb1, a5 );
[(1,3) : [ -9, -10 ] -> [ -7, -10 ]]
gap> d5 := Arrow( Gd8, (6,8), -9, -8 );;
gap> d6 := ImageElm( emb1, d5 );
[(2,4) : [ -9, -10 ] -> [ -8, -10 ]]
gap> b5 := Arrow( Gc6, (11,12,13), -10, -10 );;
gap> b6 := ImageElm( emb2, b5 );
[(5,6,7) : [ -9, -10 ] -> [ -9, -10 ]]
gap> c6 := Arrow( Gd8c6, (8,9), [-7,-10], [-8,-10] );;
gap> sq := SquareOfArrows( DGd8c6, a6, b6, c6, d6 );
[[ -9, -10 ]] ----- (1,3) ----> [[ -7, -10 ]]
      |
(5,6,7)      (1,3)(2,4)(5,7,6)(8,9)      (8,9)
      v
[[ -9, -10 ]] ----- (2,4) ----> [[ -8, -10 ]]

```

8.5 Double groupoid homomorphisms

8.5.1 DoubleGroupoidHomomorphism

- ▷ DoubleGroupoidHomomorphism(*src*, *rng*, *hom*) (operation)
- ▷ IsDoubleGroupoidHomomorphism(*mwohom*) (Category)

A homomorphism of double groupoids is determined by a homomorphism *mor* between the underlying groupoids since *mor* determines the images of the four arrows in every square.

In the example we take the endomorphism *md8* of *Gd8*, constructed in section 5.2.1, to produce an endomorphism of *DGd8*.

Example

```

gap> ad8 := GroupHomomorphismByImages( d8, d8,
>      [ (5,6,7,8), (5,7) ], [ (5,8,7,6), (6,8) ] );;
gap> md8 := GroupoidHomomorphism( Gd8, Gd8, ad8,
>      [-7,-9,-8], [(),(5,7),(6,8)] );;
gap> endDGd8 := DoubleGroupoidHomomorphism( DGd8, DGd8, md8 );;
gap> Display( endDGd8 );
double groupoid homomorphism: [ DGd8 ] -> [ DGd8 ]
with underlying groupoid homomorphism:

```

```

homomorphism to single piece groupoid: Gd8 -> Gd8
root group homomorphism:
(5,6,7,8) -> (5,8,7,6)
(5,7) -> (6,8)
object map: [ -9, -8, -7 ] -> [ -7, -9, -8 ]
ray images: [ (), (5,7), (6,8) ]
gap> IsDoubleGroupoidHomomorphism( endDGd8 );
true
gap> sq1;
[-7] ----- (5,7) -----> [-8]
    |
    |
(6,8)      (6,8)      (5,6)(7,8)
    v              v
[-7] ----- (5,6,7,8) -----> [-9]
gap> ImageElm( endDGd8, sq1 );
[-8] ----- (5,7) -----> [-9]
    |
    |
(5,7)      (5,7)      (5,8,7,6)
    v              v
[-8] ----- (5,8)(6,7) -----> [-7]

```

Chapter 9

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.

9.1 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("IsMultiplicativeElementWithObjectsAndInverses",
IsMultiplicativeElementWithObjectsAndOnes);`
- `DeclareCategory("IsGroupoidElement",
IsMultiplicativeElementWithObjectsAndInverses);`

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

- `DeclareCategory("IsDomainWithObjects", IsDomain);`
- `DeclareCategory("IsMagmaWithObjects", IsDomainWithObjects and
IsMultiplicativeElementWithObjectsCollection);`
- `DeclareCategory("IsSemigroupWithObjects", IsMagmaWithObjects and
IsAssociative);`
- `DeclareCategory("IsMonoidWithObjects", IsSemigroupWithObjects and
IsMultiplicativeElementWithObjectsAndOnesCollection);`
- `DeclareCategory("IsGroupoid", IsMonoidWithObjects and
IsGroupoidElementCollection);`

Among 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",
  "IsDomainWithObjects",
  "CategoryCollections(IsMultiplicativeElementWithObjects)",
  "CategoryCollections(IsMultiplicativeElementWithObjectsAndOnes)",
  "CategoryCollections(IsMultiplicativeElementWithObjectsAndInverses)",
  "CategoryCollections(IsGroupoidElement)", "IsMagmaWithObjects",
  "IsSemigroupWithObjects", "IsMonoidWithObjects", "IsGroupoid" ]
gap> FamilyObj( Gd8 );
<Family: "CollectionsFamily(...)">
gap> Display(last);
name:
  CollectionsFamily(...)
required filters:
  IsCollection
implied filters:
  IsListOrCollection
  IsCollection
  IsExtLElement
  CategoryCollections(IsExtLElement)
  IsExtRElement
  CategoryCollections(IsExtRElement)
  CategoryCollections(IsMultiplicativeElement)
  IsOddAdditiveNestingDepthObject
  CategoryCollections(IsMultiplicativeElementWithObjects)
  CategoryCollections(IsMultiplicativeElementWithObjectsAndOnes)
  CategoryCollections(IsMultiplicativeElementWithObjectsAndInverses)
  CategoryCollections(IsGroupoidElement)
gap> KnownAttributesOfObject( Gd8 );
[ "Name", "ObjectList", "GeneratorsOfMagmaWithObjects",
  "GeneratorsOfGroupoid" ]
gap> KnownTruePropertiesOfObject( Gd8 );
[ "IsDuplicateFree", "IsAssociative", "IsSinglePieceDomain",
  "IsDirectProductWithCompleteDigraphDomain" ]
gap> RepresentationsOfObject( Gd8 );
[ "IsComponentObjectRep", "IsAttributeStoringRep", "IsMWOSinglePieceRep" ]
gap> RepresentationsOfObject( U5 );
[ "IsComponentObjectRep", "IsAttributeStoringRep", "IsPiecesRep" ]
```

Similarly, for arrows, we have:

Example

```
gap> e1;
[(5,6,7,8) : -9 -> -8]
gap> CategoriesOfObject( e1 );
```

```
[ "IsExtLElement", "IsExtRElement", "IsMultiplicativeElement",
  "IsMultiplicativeElementWithObjects",
  "IsMultiplicativeElementWithObjectsAndOnes",
  "IsMultiplicativeElementWithObjectsAndInverses", "IsGroupoidElement" ]
gap> FamilyObj( e1 );
<Family: "IsGroupoidElementFamily">
```

9.2 Many object homomorphisms

Homomorphisms of structures with many objects have a similar heirarchy. A few examples:

- `DeclareCategory("IsGeneralMappingWithObjects", IsGeneralMapping);`
- `DeclareSynonymAttr("IsMagmaWithObjectsGeneralMapping",
IsGeneralMappingWithObjects and RespectsMultiplication);`
- `DeclareSynonymAttr("IsMagmaWithObjectsHomomorphism",
IsMagmaWithObjectsGeneralMapping and IsSPGeneralMappingWithObjects);`
- `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
IsComponentObjectRep and IsGeneralMapping, ["Source", "Range",
"MappingToSinglePieceData"]);`
- `DeclareRepresentation("IsMappingWithPiecesRep", IsMagmaWithObjectsHomomorphism
and IsAttributeStoringRep and IsComponentObjectRep and IsGeneralMapping,
["Source", "Range", "PiecesOfMapping"]);`

In previous chapters, `hom9` was a homomorphism from `Gd8` to `Kk4`; and `aut1` was an automorphism of `Ha4`.

Example

```
gap> hom9;
groupoid homomorphism : Gd8 -> Kk4
[ [ [(5,6,7,8) : -9 -> -9], [(5,7) : -9 -> -9], [() : -9 -> -8],
  [() : -9 -> -7] ],
  [ [(1,4)(2,3) : -14 -> -14], [() : -14 -> -14], [(1,3,4) : -14 -> -13],
  [(1,4)(2,3) : -14 -> -12] ] ]
gap> CategoriesOfObject( hom9 );
[ "IsExtLElement", "IsExtRElement", "IsMultiplicativeElement",
  "IsMultiplicativeElementWithOne", "IsMultiplicativeElementWithInverse",
  "IsAssociativeElement", "IsGeneralMapping", "IsSPGeneralMapping",
  "IsGeneralMappingWithObjects", "IsSPGeneralMappingWithObjects",
  "IsGroupoidHomomorphism" ]
gap> FamilyObj( hom9 );
```

```

<Family: "GroupoidHomomorphismFamily">
gap> Display( last );
name:
  GroupoidHomomorphismFamily
required filters:
  IsGroupoidHomomorphism
implied filters:
  CanEasilyCompareElements
  HasCanEasilyCompareElements
  CanEasilySortElements
  HasCanEasilySortElements
  IsExtLElement
  IsExtRElement
  IsMultiplicativeElement
  IsMultiplicativeElementWithOne
  IsMultiplicativeElementWithInverse
  IsAssociativeElement
  IsGeneralMapping
  IsSPGeneralMapping
  RespectsMultiplication
  HasRespectsMultiplication
  IsGeneralMappingWithObjects
  IsSPGeneralMappingWithObjects
  IsGroupoidHomomorphism
gap> FamilyObj( hom9 ) = FamilyObj( aut1 );
true
gap> KnownTruePropertiesOfObject( hom9 );
[ "CanEasilyCompareElements", "CanEasilySortElements",
  "RespectsMultiplication", "IsGroupWithObjectsHomomorphism",
  "IsGeneralMappingToSinglePiece", "IsGeneralMappingFromSinglePiece",
  "IsInjectiveOnObjects", "IsSurjectiveOnObjects" ]
gap> KnownTruePropertiesOfObject( aut1 );
[ "CanEasilyCompareElements", "CanEasilySortElements", "IsEndoGeneralMapping",
  "RespectsMultiplication", "IsGroupWithObjectsHomomorphism",
  "IsGeneralMappingToSinglePiece", "IsGeneralMappingFromSinglePiece",
  "IsInjectiveOnObjects", "IsSurjectiveOnObjects",
  "IsEndomorphismWithObjects", "IsAutomorphismWithObjects",
  "IsGroupoidAutomorphismByObjectPerm" ]
gap> KnownAttributesOfObject( hom9 );
[ "Range", "Source", "MappingGeneratorsImages", "MappingToSinglePieceData",
  "ImagesOfObjects", "ImageElementsOfRays", "RootGroupHomomorphism" ]
gap> KnownAttributesOfObject( aut1 );
[ "Order", "Range", "Source", "MappingGeneratorsImages", "AutomorphismDomain",
  "MappingToSinglePieceData", "ImagesOfObjects", "ImageElementsOfRays",
  "RootGroupHomomorphism" ]

```

Chapter 10

Development History

10.1 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.

`Gpd` became an accepted `GAP` package in May 2015.

In April 2017 the package was renamed again, as `groupoids`.

In August 2017 the implementation of groupoid homomorphisms was completely revised with the emphasis now on a mapping from a set of generating arrows to their images.

In September 2017 various functions were revised so that, at last, the operation `DiscreteNormalPreXModWithObjects` in `XMod` works again. This constructs a crossed module of groupoids with a connected range and a homogeneous, discrete source.

In recent versions there have been a number of changes of function name, such as `IsDigraph` becoming `IsGroupoidDigraph`. This is in order to avoid conflicts with the `Digraphs` package.

In version 1.62 of October 2018 there were significant changes to the operations constructing free products with amalgamation and HNN extensions. There was a plan to move this material to a new package `Rewriting`, but that has not happened.

Version 1.71 of August 2022 contains a complete revision of right, left and double cosets of groupoids. (The initial declaration of `LeftCoset` was moved to the `Utils` package.)

Version 1.73 of February 2023 contained a first attempt at an implementation of double groupoids, as described in Chapter 8. This was then extensively revised in version 1.77 of July 2025. This experimental material is liable to be changed and extended. A more general version of double groupoids has been introduced in the `XMod` package.

10.2 What needs to be done next?

- 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 `ImageSource` for the cases of groupoid morphisms not yet covered;
- `Enumerator` for `IsHomsetCosetsRep`;
- free groupoid on a graph;
- convert `GraphOfGroupsRewritingSystem` to the category `IsRewritingSystem`;
- in `XMod`, continue to work on crossed modules over groupoids;

References

- [AW10] M. Alp and C. D. Wensley. Automorphisms and homotopies of groupoids and crossed modules. *Applied Categorical Structures*, 18:473–495, 2010. 2, 5, 35, 56, 92
- [BMPW02] R. Brown, E. J. Moore, T. Porter, and C. D. Wensley. Crossed complexes, and free crossed resolutions for amalgamated sums and hnn-extensions of groups. *Georgian Math. J.*, 9:623–644, 2002. 5
- [Bro88] R. Brown. *Topology: a geometric account of general topology, homotopy types, and the fundamental groupoid*. Ellis Horwood, Chichester, 1988. 6, 19
- [Bro06] R. Brown. *Topology and groupoids*. Booksurge LLC, S.Carolina, 2006. 6, 19
- [GH17] S. Gutsche and M. Horn. *AutoDoc - Generate documentation from GAP source code (Version 2017.09.15)*, 2017. GAP package, <https://github.com/gap-packages/AutoDoc>. 2
- [Hig76] P. Higgins. The fundamental groupoid of a graph of groups. *J. London Math. Soc.*, 13:145–149, 1976. 63
- [Hig05] P. Higgins. *Categories and groupoids*. Reprints in Theory and Applications of Categories, 2005. <http://www.tac.mta.ca/tac/reprints/articles/7/tr7abs.html>. 6
- [Hor17] M. Horn. *GitHubPagesForGAP - Template for easily using GitHub Pages within GAP packages (Version 0.2)*, 2017. GAP package, <https://gap-system.github.io/GitHubPagesForGAP/>. 2
- [LN17] F. Lübeck and M. Neunhöffer. *GAPDoc (version 1.6)*. RWTH Aachen, 2017. GAP package, <http://www.math.rwth-aachen.de/~Frank.Luebeck/GAPDoc/index.html>. 2
- [Moo01] E. J. Moore. *Graphs of Groups: Word Computations and Free Crossed Resolutions*. PhD thesis, University of Wales, Bangor, 2001. <https://research.bangor.ac.uk/en/studentTheses/graphs-of-groups-word-computations-and-free-crossed-resolutions/>. 5, 63, 66, 67, 92
- [Ser80] J. Serre. *Trees*. Springer-Verlag, Berlin, 1980. 63

Index

- * for groupoid elements, [25](#)
- \wedge , [35](#)
- \wedge
 - for arrows, [35](#)
 - for groupoids, [36](#)
- ActionMap, [61](#)
- Arrow, [8](#)
- Arrow
 - for groupoid elements, [25](#)
- AutomorphismGroup, [56](#)
- AutomorphismGroupOfGroupoid, [55](#)
- AutomorphismGroupoidOfGroupoid, [58](#)
- BoundaryOfSquare, [75](#)
- ConjugateGroupoid, [36](#)
- conjugation identities, [60](#)
- Cosets (left,right,double), [33](#)
- costar, [26](#)
- DigraphOfGraphOfGroupoids, [70](#)
- DigraphOfGraphOfGroups, [64](#)
- DirectProductOp, [24](#)
- DiscreteSubgroupoid, [31](#)
- DiscreteTrivialSubgroupoid, [31](#)
- double coset, [33](#)
- DoubleCoset, [33](#)
- DoubleCosetRepresentatives, [33](#)
- DoubleCosets, [33](#)
- DoubleGroupoid, [74](#)
- DoubleGroupoidHomomorphism, [86](#)
- DoubleGroupoidOfSquare, [75](#)
- DoubleGroupoidWithSingleObject, [84](#)
- DoubleGroupoidWithTrivialGroup, [84](#)
- DownArrow, [75](#)
- ElementOfArrow
 - for groupoids, [25](#)
 - for magmas with objects, [8](#)
- Embedding
 - for fpa-groups, [69](#)
 - for groupoids, [24](#)
- FpWeightedDigraph, [63](#)
- FreeProductWithAmalgamation, [67](#)
- FreeProductWithAmalgamationInfo, [67](#)
- FullTrivialSubgroupoid, [31](#)
- GeneratorsOfMagmaWithObjects, [11](#)
- GeneratorsOfMonoidWithObjects, [11](#)
- GeneratorsOfSemigroupWithObjects, [11](#)
- GraphOfGroupoids, [70](#)
- GraphOfGroupoidsOfWord, [72](#)
- GraphOfGroupoidsWord, [72](#)
- GraphOfGroups, [64](#)
- GraphOfGroupsOfWord, [66](#)
- GraphOfGroupsRewritingSystem, [67](#)
- GraphOfGroupsWord, [66](#)
- Groupoid, [19](#)
- groupoid action, [60](#)
- GroupoidActionByConjugation, [61](#)
- GroupoidAutomorphismByGroupAuto, [51](#)
- GroupoidAutomorphismByGroupAutos, [57](#)
- GroupoidAutomorphismByNtuple, [51](#)
- GroupoidAutomorphismByObjectPerm, [51](#)
- GroupoidAutomorphismByRayShifts, [51](#)
- GroupoidByIsomorphisms, [36](#)
- GroupoidElement, [25](#)
- GroupoidHomomorphism, [40](#)
- GroupoidHomomorphismFromHomogeneous-Discrete, [48](#)
- GroupoidHomomorphismFromSinglePiece, [40](#)
- GroupoidInnerAutomorphism, [53](#)
- GroupoidInnerAutomorphismNormal-Subgroupoid, [53](#)
- GroupoidsOfGraphOfGroupoids, [70](#)
- GroupoidWithMonoidObjects, [38](#)
- GroupsOfGraphOfGroups, [64](#)

- HeadOfArrow
 - for groupoids, 25
 - for magmas with objects, 8
- HeadOfGraphOfGroupsWord, 66
- HnnExtension, 69
- HnnExtensionInfo, 69
- HomogeneousDiscreteGroupoid, 23
- HomogeneousDiscreteSubgroupoid, 31
- HomogeneousGroupoid, 23
- HomomorphismByUnion
 - for groupoids, 49
 - for magmas with objects, 17
- HomomorphismFromSinglePiece, 14
- HomomorphismToSinglePiece
 - for groupoids, 47
 - for magmas with objects, 14
- Homset, 26
- horizontal groupoid, 82
- HorizontalIdentities, 80
- HorizontalInverses, 80
- HorizontalProduct, 77
- identity subgroupoid, 31
- IdentityArrow, 25
- ImageElementsOfRays, 43
- ImagesOfObjects, 43
- InclusionMappingGroupoids, 44
- inner automorphism, 53
- inner automorphism group, 56
- inverse arrow, 26
- InvolutoryArcs, 63
- IsAutomorphismWithObjects
 - for groupoid homomorphisms, 42
- IsAutomorphismWithObjects, 17
- IsBasicDoubleGroupoid, 74
- IsBijectiveOnObjects
 - for groupoid homomorphisms, 42
- IsBijectiveOnObjects, 17
- IsClosedUnderTransposition, 76
- IsCommutingSquare, 76
- IsDirectProductWithCompleteDigraph, 8
- IsDiscreteMagmaWithObjects, 8
- IsDomainWithObjects, 7
- IsDoubleGroupoid, 74
- IsDoubleGroupoidElement, 75
- IsDoubleGroupoidHomomorphism, 86
- IsEndomorphismWithObjects
 - for groupoid homomorphisms, 42
- IsEndomorphismWithObjects, 17
- IsFpGroupoid, 21
- IsFpWeightedDigraph, 63
- IsFreeGroupoid, 21
- IsFreeProductWithAmalgamation, 67
- IsFullSubgroupoid, 28
- IsGraphOfFpGroupoids, 70
- IsGraphOfFpGroups, 65
- IsGraphOfGroupoids, 71
- IsGraphOfGroupoidsWord, 72
- IsGraphOfGroups, 64
- IsGraphOfGroupsWord, 66
- IsGraphOfPcGroups, 65
- IsGraphOfPermGroupoids, 70
- IsGraphOfPermGroups, 65
- IsGroupoid, 19
- IsGroupoidAction, 61
- IsGroupoidByIsomorphisms, 36
- IsGroupoidElement, 25
- IsGroupoidHomomorphism, 40
- IsHnnExtension, 69
- IsHomogeneousDomainWithObjects, 23
- IsHomogeneousDiscreteGroupoidRep, 23
- IsInjective
 - for groupoid homomorphisms, 42
- IsInjectiveOnObjects
 - for groupoid homomorphisms, 42
- IsInjectiveOnObjects, 17
- IsMagmaWithObjects, 7
- IsMappingToGroupWithGGRWS, 69
- IsMappingToSinglePieceRep, 14
- IsMappingWithObjectsByFunction, 18
- IsMatrixGroupoid, 21
- IsMonoidWithObjects, 10
- IsMultiplicativeElementWithObjects, 8
- IsomorphismGroupoids, 49
- IsomorphismNewObjects
 - for groupoids, 45
 - for magmas with objects, 14
- IsomorphismPcGroupoid, 47
- IsomorphismPermGroupoid, 47
- IsomorphismsOfGraphOfGroupoids, 70
- IsomorphismsOfGraphOfGroups, 64
- IsomorphismStandardGroupoid, 46
- IsPcGroupoid, 21

- IsPermGroupoid, 21
- IsReducedGraphOfGroupoidsWord, 72
- IsReducedGraphOfGroupsWord, 67
- IsSemigroupWithObjects, 9
- IsSinglePiece, 8
- IsSinglePieceDomain, 8
- IsSubgroupoid, 27
- IsSurjective
 - for groupoid homomorphisms, 42
- IsSurjectiveOnObjects
 - for groupoid homomorphisms, 42
- IsSurjectiveOnObjects, 17
- IsWideSubgroupoid, 27
- LeftArrow, 75
- LeftCoset, 33
- LeftCosetRepresentatives, 33
- LeftCosetRepresentativesFromObject, 33
- LeftCosets, 33
- LeftTransversalsOfGraphOfGroupoids, 71
- LeftTransversalsOfGraphOfGroups, 66
- loop, 26
- MagmaWithObjects, 6
- MagmaWithObjectsHomomorphism, 14
- MagmaWithSingleObject
 - for groups, 19
 - for semigroups, 9
- MappingToSinglePieceData
 - for groupoids, 43
 - for magmas with objects, 14
- MappingWithObjectsByFunction, 18
- matrix representation, 59
- MaximalDiscreteSubgroupoid, 31
- MonoidWithObjects, 10
- MWOofArrow
 - for groupoids, 25
 - for magmas with objects, 8
- NiceObjectAutoGroupGroupoid, 55
- NormalFormGGRWS, 68
- ObjectCostar, 26
- ObjectGroup, 20
- ObjectGroupHomomorphism, 44
- ObjectList
 - for groupoids, 20
 - for magmas with objects, 6
- ObjectList
 - for groupoids, 22
- ObjectStar, 26
- Order, 26
- ParentList, 28
- ParentMappingGroupoids, 45
- PieceIsomorphisms, 23
- PieceOfObject, 12
- PiecePositions, 30
- Pieces
 - for double groupoids, 83
 - for groupoids, 22
 - for magmas with objects, 12
- PiecesOfMapping, 14
- Projection
 - for groupoids, 24
- Range, 14
- rays, 19
- ReducedGraphOfGroupoidsWord, 72
- ReducedGraphOfGroupsWord, 67
- ReducedImageElm, 69
- RegularActionHomomorphismGroupoid, 47
- ReplaceOnePieceInUnion, 22
- representation by matrices, 59
- RestrictedMappingGroupoids, 45
- RightArrow, 75
- RightCoset, 33
- RightCosetRepresentatives, 33
- RightCosets, 33
- RightTransversalsOfGraphOfGroupoids, 71
- RightTransversalsOfGraphOfGroups, 66
- RootGroup, 20
- RootGroupHomomorphism, 43
- RootObject
 - for groupoids, 20
 - for magmas with objects, 6
- SemigroupWithObjects, 9
- SinglePieceBasicDoubleGroupoid, 74
- SinglePieceGroupoid, 19
- SinglePieceGroupoidWithRays, 32
- SinglePieceMagmaWithObjects, 6
- SinglePieceMonoidWithObjects, 10
- SinglePieceSemigroupWithObjects, 9
- SinglePieceSubgroupoidByGenerators, 32

- Size, [22](#)
- Source, [14](#)
- SquareOfArrows, [75](#)
- standard groupoid, [20](#)
- star, [26](#)
- Subgroupoid, [27](#)
- SubgroupoidByObjects, [28](#)
- SubgroupoidByPieces, [29](#)
- SubgroupoidBySubgroup, [28](#)
- SubgroupoidsOfGraphOfGroupoids, [70](#)
- SubgroupoidWithRays, [29](#)
- TailOfArrow
 - for groupoids, [25](#)
 - for magmas with objects, [8](#)
- TailOfGraphOfGroupsWord, [66](#)
- TransposedSquare, [76](#)
- tree groupoid, [31](#)
- trivial subgroupoid, [31](#)
- UnderlyingFunction, [18](#)
- UnionOfPieces
 - for double groupoids, [83](#)
 - for groupoids, [22](#)
 - for magmas with objects, [12](#)
- UpArrow, [75](#)
- vertical groupoid, [83](#)
- VerticalIdentities, [80](#)
- VerticalInverses, [80](#)
- VerticalProduct, [77](#)
- WordOfGraphOfGroupoidsWord, [72](#)
- WordOfGraphOfGroupsWord, [66](#)