

Gpd

Groupoids, graphs of groups, and graphs of groupoids

Version 1.05

November 2008

Emma Moore
Chris Wensley

Emma Moore — Email: emmajmoore@yahoo.co.uk

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

Abstract

The Gpd package for GAP4 provides functions for the computation with groupoids (categories with every arrow invertible) and their morphisms; for graphs of groups, and graphs of groupoids.

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

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

The current version is 1.05, released on 21st November 2008, and now includes the more basic structure of *magma with objects*. It had to be released hurriedly, due to the change of website, so some of the function are no longer available. A new version will be released as soon as possible.

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

Copyright

© 2000-2008 Emma Moore and Chris Wensley

Acknowledgements

This gpd package is released under the GNU General Public License (GPL). This file is part of gpd, though as documentation it is released under the GNU Free Documentation License (see <http://www.gnu.org/licenses/licenses.html#FDL>).

gpd 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.

gpd is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with gpd; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

For more details, see <http://www.fsf.org/licenses/gpl.html>.

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

Contents

1	Introduction	5
2	Many-object structures	7
2.1	Magmas with objects	7
2.1.1	MagmaWithObjects	7
2.1.2	MultiplicativeElementWithObjects	8
2.1.3	IsSinglePiece	8
3	Homomorphisms of many-object structures	10
3.1	Homomorphisms of magmas with objects	10
3.1.1	MagmaWithObjectsHomomorphism	10
4	Groupoids	12
4.1	Groupoids: their elements and attributes	12
4.1.1	SinglePieceGroupoid	12
4.1.2	Pieces	13
4.1.3	IsPermGroupoid	13
4.1.4	GroupoidElement	14
4.1.5	IsSinglePiece	15
4.2	Subgroupoids	15
4.2.1	SubgroupoidByPieces	15
4.3	Stars, Costars and Homsets	16
4.3.1	ObjectStar	16
4.3.2	IdentityElement	17
4.4	Left, right and double cosets	17
4.4.1	RightCoset	17
4.5	Conjugation	18
4.5.1	\backslash^{\wedge}	18
5	Homomorphisms of Groupoids	20
5.1	Homomorphisms to a connected groupoid	20
5.1.1	GroupoidHomomorphism	20
6	Graphs of Groups and Groupoids	22
6.1	Digraphs	22
6.1.1	FpWeightedDigraph	22
6.2	Graphs of Groups	23

6.2.1	GraphOfGroups	23
6.2.2	IsGraphOfFpGroups	24
6.2.3	RightTransversalsOfGraphOfGroups	25
6.3	Words in a Graph of Groups and their normal forms	25
6.3.1	GraphOfGroupsWord	25
6.3.2	ReducedGraphOfGroupsWord	26
6.4	Free products with amalgamation and HNN extensions	26
6.4.1	FreeProductWithAmalgamation	26
6.4.2	HnnExtension	28
6.5	GraphsOfGroupoids and their Words	29
6.5.1	GraphOfGroupoids	29
6.5.2	GraphOfGroupoidsWord	30
7	Development History	31
7.1	Versions of the Package	31
7.2	What needs to be done next?	31

Chapter 1

Introduction

Groupoids are mathematical categories in which every arrow is invertible. The Gpd 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 rewrite systems.

Gpd is implemented using GAP 4.4.

The information parameter InfoGpd 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( "gpd" );
-----
loading Gpd 1.05 for GAP 4.4 - Emma Moore and Chris Wensley
-----
true
```

For version 1.05 the package has been completely restructured, starting with *magnas with objects* and their mappings, and building up to groupoids via semigroups with objects and monoids with objects. This development is ongoing, and this manual does not mention all the available functions. A new version will be released as soon as possible.

Once the package is loaded, it is possible to check the correct installation by running the test suite of the package with the following command. (The test file itself is `tst/gpd_manual.tst`.)

Example

```
gap> ReadPackage( "gpd", "tst/testall.g" );
+ Testing examples in Chapter 2 of the Gpd manual
+ GAP4stones: 1250
+ Testing examples in Chapter 3 of the Gpd manual
+ GAP4stones: infinity
+ Testing examples in Chapter 4 of the Gpd manual
+ GAP4stones: 416
+ Testing examples in Chapter 5 of the Gpd manual
+ GAP4stones: 2500
```

- + Testing examples in Chapter 6 of the Gpd manual
- + GAP4stones: 28

You may reference this package by mentioning [\[BMPW02\]](#) and [\[Moo01\]](#).

Chapter 2

Many-object structures

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*.

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].

2.1 Magmas with objects

2.1.1 MagmaWithObjects

◇ MagmaWithObjects (args)	(function)
◇ ObjectList (mwo)	(attribute)
◇ SemigroupWithObjects (args)	(function)
◇ MonoidWithObjects (args)	(function)

The simplest construction for a magma with objects is to take a magma m and form arrows (u, x, v) for every x in m and every pair of objects (u, v) . Multiplication is defined by $(u, x, v) * (v, y, w) = (u, x * y, w)$.

Any finite, ordered set is in principle acceptable as the objects of M , but we will restrict ourselves to sets of negative integers here.

Example

```
gap> tm := [[1,2,4,3],[1,2,4,3],[3,4,2,1],[3,4,2,1]]; 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" );
gap> m1 := MagmaElement(m,1);;
gap> m2 := MagmaElement(m,2);;
gap> m3 := MagmaElement(m,3);;
gap> m4 := MagmaElement(m,4);;
gap> One(m);
fail
gap> M78 := MagmaWithObjects( [-8,-7], m );
Magma with objects :-
  objects = [ -8, -7 ]
  magma = m
gap> [ IsAssociative(M78), IsCommutative(M78) ];
[ false, false ]

```

2.1.2 MultiplicativeElementWithObjects

◇ `MultiplicativeElementWithObjects(mwo, elt, tail, head)` (operation)

Elements 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.

Example

```

gap> a78 := MultiplicativeElementWithObjects( M78, m4, -7, -8 );
[m2 : -7 -> -8]
gap> b87 := MultiplicativeElementWithObjects( M78, m3, -8, -7 );
[m3 : -8 -> -7]
gap> ba := b87*a78;
[m4 : -8 -> -8]
gap> ab := a78*b87;
[m4 : -7 -> -7]
gap> a78^2;
fail
gap> ba^2;
[m1 : -8 -> -8]

```

2.1.3 IsSinglePiece

◇ `IsSinglePiece(mwo)` (property)

◇ `IsDirectProductWithCompleteGraph(mwo)` (property)

If the partial composition is forgotten, then a digraph is left (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 considerably use in GAP, so (for now?) we use the term `IsSinglePiece` to describe a connected magma with objects.

Example

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

Chapter 3

Homomorphisms of many-object structures

A *homomorphism* f from a magma with objects M to a magma with objects N consists of a map f_O from the objects of M to those of N together with a map f_A from the arrows of M to those of N which is compatible with tail and head and which preserves multiplication:

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

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

3.1 Homomorphisms of magmas with objects

3.1.1 MagmaWithObjectsHomomorphism

◇ MagmaWithObjectsHomomorphism(<i>args</i>)	(function)
◇ MagmaHomomorphismFromSinglePiece(<i>src, rng, hom, imobs</i>)	(operation)
◇ HomomorphismToSinglePiece(<i>src, rng, images</i>)	(operation)
◇ HomomorphismByUnion(<i>src, rng, homs</i>)	(operation)

As usual, there are a variety of homomorphism constructors. The basic construction is a homomorphism $M \rightarrow N$ with both M and N connected, which is implemented as `IsHomomorphismToSinglePieceRep` with attributes `Source`, `Range` and `PieceImages`. We require the following information:

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

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

Example

```
gap> tup1 := [ Tuple([m1,m2]), Tuple([m2,m1]), Tuple([m3,m4]), Tuple([m4,m3]) ];
gap> f1 := GeneralMappingByElements( m, m, tup1 );
f1 = <general mapping: m -> m >
gap> IsMagmaHomomorphism( f1 );
true
```

```
gap> tup2 := [ Tuple([m1,m1]), Tuple([m2,m1]), Tuple([m3,m1]), Tuple([m4,m1]) ];;
gap> f2 := GeneralMappingByElements( m, m, tup2 );;
gap> IsMagmaHomomorphism( f2 );
true
gap> map1 := HomomorphismFromSinglePiece( M78, M78, [-8,-7], f1 );
magma with objects homomorphism : M78 -> M78
gap> Display( map1 );
Mapping to single piece magma:
[ M78 ] -> [ M78 ]
magma mapping: <mapping: m -> m >
  object map: [ -8, -7 ] -> [ -8, -7 ]
Homomorphism to connected magma:
[ M78 ] -> [ M78 ]
  object map = [ [ -8, -7 ], [ -8, -7 ] ]
homomorphism = <homomorphism: m -> m >
gap> idm := f1*f1;;
gap> idmap := HomomorphismFromSinglePiece( M78, M78, idm, [-7,-8] );
gap> map2 := HomomorphismFromSinglePiece( M78, M78, f2, [-7,-8] );
```

Chapter 4

Groupoids

Many of the names of the functions described in this chapter have changed, due to the introduction of magmas with objects, so the chapter is full of errors. A new version will be released as soon as possible.

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

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

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

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

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 elements and attributes

4.1.1 SinglePieceGroupoid

◇ `SinglePieceGroupoid(grp, obs)` (operation)
◇ `Groupoid(args)` (function)

There are a variety of constructors for groupoids, with one or two parameters. The global function `Groupoid` will normally find the appropriate one to call, the options being:

- the object group, a list of objects;
- a group being converted to a groupoid, a single object;
- a list of groupoids which have already been constructed.

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

Example

```

gap> d8 := Group( (1,2,3,4), (1,3) );
gap> SetName( d8, "d8" );
gap> Gd8 := SinglePieceGroupoid( d8, [-9,-8,-7] );
Perm single piece groupoid:
< d8, [ -9, -8, -7 ] >
gap> c6 := Group( (5,6,7)(8,9) );
gap> SetName( c6, "c6" );
gap> Gc6 := DomainWithSingleObject( c6, -6 );
Perm SinglePiece Groupoid:
< c6, [ -6 ] >
gap> Gd8c6 := UnionOfPieces( [ Gd8, Gc6 ] );
gap> Display( Gd8c6 );
Perm Groupoid with 2 pieces:
< objects: [ -9, -8, -7 ]
  group: d8 = <[ (1,2,3,4), (1,3) ]> >
  < objects: [ -6 ]
    group: c6 = <[ (5,6,7)(8,9) ]> >
gap> SetName( Gd8, "Gd8" ); SetName( Gc6, "Gc6" ); SetName( Gd8c6, "Gd8+Gc6" );

```

4.1.2 Pieces

◇ *Pieces(gpd)*

(attribute)

◇ *ObjectList(gpd)*

(attribute)

When a groupoid consists of two or more pieces, we require their object lists to be disjoint. The pieces are sorted by the first object in their object lists, which must be disjoint. The list *ObjectsOfGroupoid* of a groupoid is the sorted concatenation of the objects in the pieces.

Example

```

gap> Pieces( Gd8c6 );
[ Gd8, Gc6 ]
gap> ObjectList( Gd8c6 );
[ -9, -8, -7, -6 ]

```

4.1.3 IsPermGroupoid

◇ *IsPermGroupoid(gpd)*

(property)

◇ *IsPcGroupoid(gpd)*

(property)

◇ *IsFpGroupoid(gpd)*

(property)

A groupoid is a permutation groupoid if all its pieces have permutation groups. Most of the examples in this chapter are permutation groupoids, but in principle any type of group known to GAP may be used.

In the following example *Gf2* is an fp-groupoid, while *Gq8* is a pc-groupoid.

Example

```

gap> f2 := FreeGroup( 2 );;
gap> SetName( f2, "f2" );
gap> Gf2 := Groupoid( f2, -22 );;
gap> q8 := SmallGroup( 8, 4 );;
gap> Gq8 := Groupoid( q8, [ -28, -27 ] );;
gap> SetName( q8, "q8" ); SetName( Gq8, "Gq8" );
gap> Gf2q8 := Groupoid( [ Gf2, Gq8 ] );;
gap> [ IsFpGroupoid( Gf2 ), IsPcGroupoid( Gq8 ), IsPcGroupoid( Gf2q8 ) ];
[ true, true, false ]
gap> G4 := Groupoid( [ Gd8c6, Gf2, Gq8 ] );;
gap> Display( G4 );
Groupoid with 4 pieces:
< objects: [ -28, -27 ]
  group: q8 = <[ f1, f2, f3 ]> >
< objects: [ -22 ]
  group: f2 = <[ f1, f2 ]> >
< objects: [ -9, -8, -7 ]
  group: d8 = <[ (1,2,3,4), (1,3) ]> >
< objects: [ -6 ]
  group: c6 = <[ (5,6,7)(8,9) ]> >
gap> G4 = Groupoid( [ Gq8, Gf2, Gd8c6 ] );
true

```

4.1.4 GroupoidElement

◇ GroupoidElement(<i>gpd</i> , <i>elt</i> , <i>tail</i> , <i>head</i>)	(operation)
◇ IsElementOfGroupoid(<i>elt</i>)	(property)
◇ Arrow(<i>elt</i>)	(attribute)
◇ Arrowtail(<i>elt</i>)	(attribute)
◇ Arrowhead(<i>elt</i>)	(attribute)
◇ Size(<i>gpd</i>)	(attribute)

A *groupoid element* e is a triple consisting of a group element, $\text{Arrow}(e)$ or $e![1]$; the tail (source) object, $\text{Arrowtail}(e)$ or $e![2]$; and the head (target) object, $\text{Arrowhead}(e)$ or $e![3]$.

The Size of a groupoid is the number of its elements which, for a single piece groupoid, is the product of the size of the group with the square of the number of objects.

Groupoid elements have a *partial composition*: two elements may be multiplied when the head of the first coincides with the tail of the second.

Example

```

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

```

```

gap> cycle := prod*e3;
[(1,4,3,2) : -9 -> -9]
gap> cycle^2;
[(1,3)(2,4) : -9 -> -9]
gap> Order( cycle );
4
gap> cycle^e1;
[(1,2,3,4) : -8 -> -8]
gap> [ Size( Gd8 ), Size( Gc6 ), Size( Gd8c6 ), Size( Gf2q8 ) ];
[ 72, 6, 78, infinity ]

```

4.1.5 IsSinglePiece

◇ IsSinglePiece(*gpd*) (property)
 ◇ IsDiscrete(*gpd*) (property)

The forgetful functor, which forgets the composition of elements, maps the category of groupoids and their morphisms to the category of digraphs and their morphisms. Applying this functor to a particular groupoid gives the *underlying digraph* of the groupoid. A groupoid is *connected* if its underlying digraph is connected (and so complete). A groupoid is *discrete* if it is a union of groups, so that all the arcs in its underlying digraph are loops. It is sometimes convenient to call a groupoid element a *loop* when its tail and head are the same object.

4.2 Subgroupoids

4.2.1 SubgroupoidByPieces

◇ SubgroupoidByPieces(*gpd*, *obhoms*) (operation)
 ◇ Subgroupoid(*args*) (function)
 ◇ FullSubgroupoid(*gpd*, *obs*) (operation)
 ◇ MaximalDiscreteSubgroupoid(*gpd*) (attribute)
 ◇ DiscreteSubgroupoid(*gpd*, *obs*, *sgps*) (operation)
 ◇ FullIdentitySubgroupoid(*gpd*) (attribute)
 ◇ DiscreteIdentitySubgroupoid(*gpd*) (attribute)

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

There are a variety of constructors for a subgroupoid of a groupoid. The operation SubgroupoidByPieces is the most general. Its two parameters are a groupoid and a list of pieces, where each piece is specified as a list [*obs*, *sgp*], *obs* is a subset of the objects in one of the pieces of *gpd*, and *sgp* is a subgroup of the group in that piece.

The FullSubgroupoid of a groupoid *gpd* on a subset *obs* of its objects contains all the elements of *gpd* with tail and head in *obs*.

A subgroupoid is *discrete* if it is a union of groups. The MaximalDiscreteSubgroupoid of *gpd* is the union of all the single-object full subgroupoids of *gpd*.

An *identity subgroupoid* has trivial object groups, but need not be discrete. A single piece identity groupoid is sometimes called a *tree groupoid*.

The global function Subgroupoid should call the appropriate operation.

Example

```
gap> c4d := Subgroup( d8, [ (1,2,3,4) ] );;
gap> k4d := Subgroup( d8, [ (1,2)(3,4), (1,3)(2,4) ] );;
gap> SetName( c4d, "c4d" ); SetName( k4d, "k4d" );
gap> Ud8 := Subgroupoid( Gd8, [ [ k4d, [-9] ], [ c4d, [-8,-7] ] ] );;
gap> SetName( Ud8, "Ud8" );
gap> Display( Ud8 );
Perm Groupoid with 2 pieces:
< objects: [ -9 ]
  group: k4d = <[ (1,2)(3,4), (1,3)(2,4) ]> >
< objects: [ -8, -7 ]
  group: c4d = <[ (1,2,3,4) ]> >
gap> FullSubgroupoid( Gd8c6, [-7,-6] );
Perm Groupoid with pieces:
< [ -7 ], d8 >
< [ -6 ], c6 >
gap> DiscreteSubgroupoid( Gd8c6, [-9,-8], [ c4d, k4d ] );
Perm Groupoid with pieces:
< [ -9 ], c4d >
< [ -8 ], k4d >
gap> FullIdentitySubgroupoid( Ud8 );
Perm Groupoid with pieces:
< [ -9 ], id(k4d) >
< [ -8, -7 ], id(c4d) >
```

4.3 Stars, Costars and Homsets

4.3.1 ObjectStar

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

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

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

Example

```
gap> star9 := ObjectStar( Gd8, -9 );
<star at [ -9 ] with group d8>
gap> for e in star9 do
>   if ( Order( e![1] ) = 4 ) then Print( e, "\n" ); fi;
```



```

> od;
[(1,4,3,2) : -9 -> -9]
[(1,4,3,2) : -9 -> -8]
[(1,4,3,2) : -9 -> -7]
[(1,2,3,4) : -9 -> -9]
[(1,2,3,4) : -9 -> -8]
[(1,2,3,4) : -9 -> -7]
gap> costar6 := ObjectCostar( Gc6, -6 );
<costar at [ -6 ] with group c6>
gap> hset78 := Homset( Ud8, -7, -8 );
<homset -7 -> -8 with group c4d>
gap> for e in hset78 do Print( e, "\n" ); od;
[() : -7 -> -8]
[(1,3)(2,4) : -7 -> -8]
[(1,4,3,2) : -7 -> -8]
[(1,2,3,4) : -7 -> -8]

```

4.3.2 IdentityElement

◇ IdentityElement(*gpd*, *obj*)

(operation)

The identity groupoid element $1_{\{o\}}$ of *gpd* at object *o* is $[e, o, o]$ where *e* is the identity group element in the object group. It is a left identity for the star and a right identity for the costar at that object.

Example

```

gap> i7 := IdentityElement( Gd8, -7 );;
gap> i8 := IdentityElement( Gd8, -8 );;
gap> L := [ i7, i8 ];;
gap> for e in hset78 do Add( L, i7*e*i8 = e ); od;
gap> L;
[ [() : -7 -> -7], [() : -8 -> -8], true, true, true, true ]

```

4.4 Left, right and double cosets

4.4.1 RightCoset

◇ RightCoset(*G*, *U*, *elt*)

(operation)

◇ RightCosetRepresentatives(*G*, *U*)

(operation)

◇ RightCosetsNC(*G*, *U*)

(operation)

◇ LeftCoset(*G*, *U*, *elt*)

(operation)

◇ LeftCosetRepresentatives(*G*, *U*)

(operation)

◇ LeftCosetRepresentativesFromObject(*G*, *U*, *obj*)

(operation)

◇ LeftCosetsNC(*G*, *U*)

(operation)

◇ DoubleCoset(*G*, *U*, *elt*, *V*)

(operation)

◇ DoubleCosetRepresentatives(*G*, *U*, *V*)

(operation)

◇ `DoubleCosetsNC(G, U, V)`

(operation)

If U is a wide subgroupoid of G , the *right cosets* of U in G are the equivalence classes of the relation on the elements of G where g_1 is related to g_2 if and only if $g_2 = u * g_1$ for some element u of U . The right coset containing g is written Ug . These right cosets refine the costars of G and, in particular, $U1_o$ is the costar of U at o , so that (unlike groups) U is itself a coset only when G has a single object.

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

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

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

Example

```
gap> re2 := RightCoset( Gd8, Ud8, e2 );
RightCoset(c4d, [(1,3) : -8 -> -7])
gap> for x in re2 do Print( x, "\n" ); od;
[(1,3) : -8 -> -8]
[(1,3) : -7 -> -8]
[(2,4) : -8 -> -8]
[(2,4) : -7 -> -8]
[(1,4)(2,3) : -8 -> -8]
[(1,4)(2,3) : -7 -> -8]
[(1,2)(3,4) : -8 -> -8]
[(1,2)(3,4) : -7 -> -8]
gap> rcrd8 := RightCosetRepresentatives( Gd8, Ud8 );
[ [ () : -9 -> -9], [ () : -9 -> -8], [ () : -9 -> -7], [(2,4) : -9 -> -9],
  [(2,4) : -9 -> -8], [(2,4) : -9 -> -7], [ () : -8 -> -9], [ () : -8 -> -8],
  [ () : -8 -> -7], [(2,4) : -8 -> -9], [(2,4) : -8 -> -8], [(2,4) : -8 -> -7]
]
gap> lcr7 := LeftCosetRepresentativesFromObject( Gd8, Ud8, -7 );
[ [ () : -7 -> -9], [(2,4) : -7 -> -9], [ () : -7 -> -8], [(2,4) : -7 -> -8] ]
```

4.5 Conjugation

4.5.1 \backslash^{\wedge}

◇ $\backslash^{\wedge}(e1, e2)$

(operation)

When $e2 = c : p \rightarrow q$ and $e1$ has group element b , the conjugate $e1^e2$ has a complicated definition, but may be remembered as follows. All objects are fixed except p, q , which are interchanged. For p, q as source, multiply b on the left by c^{-1}, c respectively; and for p, q as target, multiply b on the right by c, c^{-1} . This product gives the group element of the conjugate.

Example

```

gap> x := GroupoidElement( Gd8, (2,4), -9, -9 );;
gap> y := GroupoidElement( Gd8, (1,2,3,4), -8, -9 );;
gap> z := GroupoidElement( Gd8, (1,3)(2,4), -7, -8 );;
gap> Print( "\nConjugation with elements x, y, and z in Gd8:\n" );
gap> Print( "x = ", x, ", y = ", y, ", z = ", z, "\n" );
x = [(2,4) : -9 -> -9], y = [(1,2,3,4) : -8 -> -9], z = [(1,3) : -8 -> -8]
gap> Print( "x^x = ", x^x, ", x^y = ", x^y, ", x^z = ", x^z, "\n" );
x^x = [(2,4) : -9 -> -9], x^y = [(1,3) : -8 -> -8], x^z = [(2,4) : -9 -> -9]
gap> Print( "y^x = ", y^x, ", y^y = ", y^y, ", y^z = ", y^z, "\n" );
y^x = [() : -8 -> -9], y^y = [(1,4,3,2) : -9 -> -8], y^z = [(1,4)(2,3) : -8 -> -9]
gap> Print( "z^x = ", z^x, ", z^y = ", z^y, ", z^z = ", z^z, "\n" );
z^x = [(1,3) : -8 -> -8], z^y = [(2,4) : -9 -> -9], z^z = [(1,3) : -8 -> -8]

```

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

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. The simplest example of this is given by homomorphism the two-object groupoid with trivial group to the free group $\langle a \rangle$ on one generator, when the image is $[1, a, a^{-1}]$.

5.1 Homomorphisms to a connected groupoid

5.1.1 GroupoidHomomorphism

◇ GroupoidHomomorphism(<i>args</i>)	(function)
◇ GroupoidHomomorphismFromSinglePiece(<i>src, rng, hom, imobs</i>)	(operation)
◇ Source(<i>hom</i>)	(attribute)
◇ Range(<i>hom</i>)	(attribute)

As usual, there are a variety of homomorphism constructors. The basic construction is a homomorphism $G \rightarrow H$ with H connected, which is implemented as `IsHomomorphismToSinglePieceGroupoidRep` with attributes `Source`, `Range` and `PieceImages`. If G is also connected, we may apply `HomomorphismOfSinglePieceGroupoids`, requiring:

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

Example

```
gap> d12 := Group( (15,16,17,18,19,20, (15,20)(16,19)(17,18)) );
gap> Gd12 := SinglePieceGroupoid( [-37,-36,-35,-34], d12 );
gap> SetName( d12, "d12" ); SetName( Gd12, "Gd12" );
gap> s3d := Subgroup( d12, [ (15,17,19)(16,18,20), (15,20)(16,19)(17,18) ] );
gap> Gs3d := SubgroupoidByPieces( Gd12, [ [-36,-35,-34], s3d ] );
gap> SetName( s3d, "s3d" ); SetName( Gs3d, "Gs3d" );
gap> gend8 := GeneratorsOfGroup( d8 );
```

```
gap> imhd8 := [ ( ), (15,20)(16,19)(17,18) ];;
gap> hd8 := GroupHomomorphismByImages( d8, s3d, gend8, imhd8 );
gap> homd8 := GroupoidHomomorphism( Gd8, Gs3d, hd8, [-34,-35,-36] );
groupoid homomorphism : Gd8 -> Gs3d
gap> IsBijectiveOnObjects( homd8 );
true
gap> Display( homd8 );
groupoid mapping: [ Gd8 ] -> [ Gs3d ]
root homomorphism: [ [ (1,2,3,4), (1,3) ], [ ( ), (15,20)(16,19)(17,18) ] ]
images of objects: [ -34, -35, -36 ]
  images of rays: [ ( ), ( ), ( ) ]
```

Chapter 6

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.

6.1 Digraphs

6.1.1 FpWeightedDigraph

◇ `FpWeightedDigraph(verts, arcs)` (attribute)
◇ `IsFpWeightedDigraph(dig)` (attribute)
◇ `InvolutoryArcs(dig)` (attribute)

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

Example

```
gap> V1 := [ 5, 6 ];;
gap> f1 := FreeGroup( "y" );
gap> y := f1.1;;
gap> A1 := [ [ y, 5, 6 ], [ y^-1, 6, 5 ] ];
gap> D1 := FpWeightedDigraph( V1, A1 );
weighted digraph with vertices: [ 5, 6 ]
and arcs: [ [ y, 5, 6 ], [ y^-1, 6, 5 ] ]
gap> inv1 := InvolutoryArcs( D1 );
[ 2, 1 ]
```

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

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

Example

```
gap> f3 := FreeGroup( 3, "z" );;
gap> z1 := f3.1;; z2 := f3.2;; z3 := f3.3;;
gap> V3 := [ 7, 8, 9 ];;
gap> A3 := [[z1,7,8],[z2,8,9],[z3,9,7],[z1^-1,8,7],[z2^-1,9,8],[z3^-1,7,9]];;
gap> D3 := FpWeightedDigraph( V3, A3 );
weighted digraph with vertices: [ 7, 8, 9 ]
and arcs: [ [ z1, 7, 8 ], [ z2, 8, 9 ], [ z3, 9, 7 ], [ z1^-1, 8, 7 ],
            [ z2^-1, 9, 8 ], [ z3^-1, 7, 9 ] ]
[ gap> inv3 := InvolutoryArcs( D3 );
  [ 4, 5, 6, 1, 2, 3 ]
```

6.2 Graphs of Groups

6.2.1 GraphOfGroups

◇ <code>GraphOfGroups(dig, gps, sgps, isos)</code>	(operation)
◇ <code>DigraphOfGraphOfGroups(gg)</code>	(attribute)
◇ <code>GroupsOfGraphOfGroups(gg)</code>	(attribute)
◇ <code>SubgroupsOfGraphOfGroups(gg)</code>	(attribute)
◇ <code>IsomorphismsOfGraphOfGroups(gg)</code>	(attribute)

A graph of groups is traditionally defined as consisting of:

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

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

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

These two viewpoints are clearly equivalent.

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

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

We shall be representing free products with amalgamation of groups and HNN extensions of groups, so we take as our first example the trefoil group with generators a, b and relation $a^3 = b^2$.

For this we take digraph D1 above with an infinite cyclic group at each vertex, generated by a and b respectively. The two subgroups will be generated by a^3 and b^2 with the obvious isomorphisms.

Example

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

6.2.2 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 ] ]
```


6.2.3 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 ] ]
```

6.3 Words in a Graph of Groups and their normal forms

6.3.1 GraphOfGroupsWord

◇ GraphOfGroupsWord (gg, tv, list) (operation)

◇ IsGraphOfGroupsWord (w) (property)

◇ GraphOfGroupsOfWord (w) (attribute)

◇ WordOfGraphOfGroupsWord (w) (attribute)

◇ GGTail (w) (attribute)

◇ GGHead (w) (attribute)

If G is a graph of groups with underlying digraph D , the following groupoids may be considered. First there is the free groupoid or path groupoid on D . Since we want each involutory pair of arcs to represent inverse elements in the groupoid, we quotient out by the relations $y^{-1} = \bar{y}$ 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 .

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

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

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

Example

```
gap> L1 := [ a^7, 1, b^-6, 2, a^-11, 1, b^9, 2, a^7 ];;
gap> gw1 := GraphOfGroupsWord( G1, 5, L1 );
(5)a^7.y.b^-6.y^-1.a^-11.y.b^9.y^-1.a^7(5)
gap> IsGraphOfGroupsWord( gw1 );
true
gap> [ GGTail(gw1), GGHead(gw1) ];
[ 5, 5 ]
gap> GraphOfGroupsOfWord(gw1);
Graph of Groups: 2 vertices; 2 arcs; groups [ fa, fb ]
gap> WordOfGraphOfGroupsWord( gw1 );
[ a^7, 1, b^-6, 2, a^-11, 1, b^9, 2, a^7 ]
```

6.3.2 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 );
(5)a^-1.y.b^-1.y^-1.a^10(5)
```

6.4 Free products with amalgamation and HNN extensions

6.4.1 FreeProductWithAmalgamation

- ◇ FreeProductWithAmalgamation(*gp1*, *gp2*, *iso*) (operation)
- ◇ IsFpaGroup(*fpa*) (property)
- ◇ GraphOfGroupsRewritingSystem(*fpa*) (attribute)
- ◇ NormalFormGGRWS(*fpa*, *word*) (attribute)

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

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

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

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

Example

```
gap> ## set up the first group s3 and a subgroup c3=<a1>
gap> f1 := FreeGroup( 2, "a" );;
gap> rel1 := [ f1.1^3, f1.2^2, (f1.1*f1.2)^2 ];;
gap> s3 := f1/rel1;;
gap> gs3 := GeneratorsOfGroup(s3);;
gap> SetName( s3, "s3" );
gap> a1 := gs3[1];; a2 := gs3[2];;
gap> H1 := Subgroup(s3,[a1]);;
gap> ## then the second group a4 and subgroup c3=<b1>
gap> f2 := FreeGroup( 2, "b" );;
gap> rel2 := [ f2.1^3, f2.2^3, (f2.1*f2.2)^2 ];;
gap> a4 := f2/rel2;;
gap> ga4 := GeneratorsOfGroup(a4);;
gap> SetName( a4, "a4" );
gap> b1 := ga4[1]; b2 := ga4[2];;
gap> H2 := Subgroup(a4,[b1]);;
gap> ## form the isomorphism and the fpa group
gap> iso := GroupHomomorphismByImages(H1,H2,[a1],[b1]);;
gap> fpa := FreeProductWithAmalgamation( s3, a4, iso );
<fp group on the generators [ fa1, fa2, fa3, fa4 ]>
gap> RelatorsOfFpGroup( fpa );
[ fa1^3, fa2^2, fa1*fa2*fa1*fa2, fa3^3, fa4^3, fa3*fa4*fa3*fa4, fa1*fa3^-1 ]
gap> ggl := GraphOfGroupsRewritingSystem( fpa );;
gap> Display( ggl );
Graph of Groups with :-
  vertices: [ 5, 6 ]
    arcs: [ [ y, 5, 6 ], [ y^-1, 6, 5 ] ]
    groups: [ s3, a4 ]
  subgroups: [ Group( [ a1 ] ), Group( [ b1 ] ) ]
isomorphisms: [ [ [ a1 ], [ b1 ] ], [ [ b1 ], [ a1 ] ] ]
gap> LeftTransversalsOfGraphOfGroups( ggl );
[ [ <identity ..>, a2^-1 ], [ <identity ..>, b2^-1, b1^-1*b2^-1, b1*b2^-1 ] ]
gap> ## choose a word in fpa and find its normal form
gap> gfpa := GeneratorsOfGraphOfGroups( fpa );;
gap> w2 := (gfpa[1]*gfpa[2]*gfpa[3]^gfpa[4])^3;
fa1*fa2*fa4^-1*fa3*fa4*fa1*fa2*fa4^-1*fa3*fa4*fa1*fa2*fa4^-1*fa3*fa4
```

```
gap> n2 := NormalFormGGRWS( fpa, w2 );
fa2*fa3*fa4^-1*fa2*fa4^-1*fa2*fa3^-1*fa4*fa3^-1
```

6.4.2 HnnExtension

◇ `HnnExtension(gp, iso)`

(operation)

◇ `IsHnnGroup(hnn)`

(property)

For *HNN extensions*, the appropriate graph of groups has underlying digraph with just one vertex and one pair of loops, weighted with `FpGroup` generators z, z^{-1} . There is one vertex group G , two isomorphic subgroups 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 $G=a_4$ and the two subgroups are cyclic groups of order 3.

Example

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

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

Example

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

```
gap> hnninfo := HnnInfo( hnn );
rec( group := a4, isomorphism := [ b1 ] -> [ b2 ] )
```

6.5 GraphsOfGroupoids and their Words

6.5.1 GraphOfGroupoids

◇ GraphOfGroupoids(<i>dig</i> , <i>gpds</i> , <i>subgpds</i> , <i>isos</i>)	(operation)
◇ IsGraphOfPermGroupoids(<i>gg</i>)	(property)
◇ IsGraphOfFpGroupoids(<i>gg</i>)	(property)
◇ GroupoidsOfGraphOfGroupoids(<i>gg</i>)	(attribute)
◇ DigraphOfGraphOfGroupoids(<i>gg</i>)	(attribute)
◇ SubgroupoidsOfGraphOfGroupoids(<i>gg</i>)	(attribute)
◇ IsomorphismsOfGraphOfGroupoids(<i>gg</i>)	(attribute)
◇ RightTransversalsOfGraphOfGroupoids(<i>gg</i>)	(attribute)
◇ LeftTransversalsOfGraphOfGroupoids(<i>gg</i>)	(attribute)

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

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

Example

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

```

objects: [ -2, -1 ]
group: fa = <[ a ]>
Fp single piece groupoid: Gfb
objects: [ -4, -3 ]
group: fb = <[ b ]>
subgroupoids: single piece groupoid: Uhy
objects: [ -2, -1 ]
group: hy = <[ a^3 ]>
single piece groupoid: Uhybar
objects: [ -4, -3 ]
group: hybar = <[ b^2 ]>
isomorphisms: [ groupoid mapping : Uhy -> Uhybar,
groupoid mapping : Uhybar -> Uhy ]

```

6.5.2 GraphOfGroupoidsWord

◇ GraphOfGroupoidsWord(<i>gg</i> , <i>tv</i> , <i>list</i>)	(operation)
◇ IsGraphOfGroupoidsWord(<i>w</i>)	(property)
◇ GraphOfGroupoidsOfWord(<i>w</i>)	(attribute)
◇ WordOfGraphOfGroupoidsWord(<i>w</i>)	(attribute)
◇ ReducedGraphOfGroupoidsWord(<i>w</i>)	(operation)
◇ IsReducedGraphOfGroupoidsWord(<i>w</i>)	(property)

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

Example

```

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

```

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

Chapter 7

Development History

7.1 Versions of the Package

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

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

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

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

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

A further extensive rewrite is in progress, introducing magmas with objects and their mappings, but many of the functions fail to work at present, and this manual is far from correct.

7.2 What needs to be done next?

Computationally, there are three types of connected groupoid:

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

GraphGpd attempted to implement the second case, while `Gpd` 1.01 and 1.03 considered only the first case, and `Gpd` 1.05 extended 1.03 to the second case. Here are some other immediate requirements:

- Automorphism group of a groupoid.
- normal subgroupoids and quotient groupoids;
- more methods for morphisms of groupoids, particularly when the range is not connected;

- ImageElm and ImagesSource for the cases of groupoid morphisms not yet covered;
- Enumerator for IsHomsetCosetsRep;
- free groupoid on a graph;
- methods for FreeProductWithAmalgamation and HnnEntension for pc-groups;
- convert GraphOfGroupsRewritingSystem to the category IsRewritingSystem;
- in XMod, implement crossed modules over groupoids.

References

- [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. [6](#)
- [Bro88] Ronald Brown. *Topology: a geometric account of general topology, homotopy types, and the fundamental groupoid*. Ellis Horwood, Chichester, 1988. [7](#), [12](#)
- [Bro06] Ronald Brown. *Topology and groupoids*. www.groupoids.org, Deganwy, 2006. [7](#), [12](#)
- [Hig76] Philip Higgins. The fundamental groupoid of a graph of groups. *J. London Math. Soc.*, 13:145–149, 1976. [22](#)
- [Moo01] Emma Moore. *Graphs of groups: word computations and free crossed resolutions*. Ph.D. thesis, University of Wales, Bangor, 2001. [5](#), [6](#), [22](#), [25](#), [26](#), [31](#)
- [Ser80] J. Serre. *Trees*. Springer-Verlag, Berlin, 1980. [22](#)

Index

- * for groupoid elements, 14
- $\backslash^{\{ \}$ for groupoid elements, 18
- \backslash^{\cdot} , 18
- Arrow, 14
- Arrowhead, 14
- Arrowtail, 14
- DigraphOfGraphOfGroupoids, 29
- DigraphOfGraphOfGroups, 23
- DiscreteIdentitySubgroupoid, 15
- DiscreteSubgroupoid, 15
- DoubleCoset, 17
- DoubleCosetRepresentatives, 17
- DoubleCosetsNC, 18
- FpWeightedDigraph, 22
- FreeProductWithAmalgamation, 26
- FullIdentitySubgroupoid, 15
- FullSubgroupoid, 15
- GGHead, 25
- GGTail, 25
- GraphOfGroupoids, 29
- GraphOfGroupoidsOfWord, 30
- GraphOfGroupoidsWord, 30
- GraphOfGroups, 23
- GraphOfGroupsOfWord, 25
- GraphOfGroupsRewritingSystem, 26
- GraphOfGroupsWord, 25
- Groupoid, 12
- GroupoidElement, 14
- GroupoidHomomorphism, 20
- GroupoidHomomorphismFromSinglePiece, 20
- GroupoidsOfGraphOfGroupoids, 29
- GroupsOfGraphOfGroups, 23
- HnnExtension, 28
- HomomorphismByUnion, 10
- HomomorphismToSinglePiece, 10
- Homset, 16
- IdentityElement, 17
- InvolutoryArcs, 22
- IsDirectProductWithCompleteGraph, 8
- IsDiscrete, 15
- IsElementOfGroupoid, 14
- IsFpaGroup, 26
- IsFpGroupoid, 13
- IsFpWeightedDigraph, 22
- IsGraphOfFpGroupoids, 29
- IsGraphOfFpGroups, 24
- IsGraphOfGroupoidsWord, 30
- IsGraphOfGroupsWord, 25
- IsGraphOfPcGroups, 24
- IsGraphOfPermGroupoids, 29
- IsGraphOfPermGroups, 24
- IsHnnGroup, 28
- IsomorphismsOfGraphOfGroupoids, 29
- IsomorphismsOfGraphOfGroups, 23
- IsPcGroupoid, 13
- IsPermGroupoid, 13
- IsReducedGraphOfGroupoidsWord, 30
- IsReducedGraphOfGroupsWord, 26
- IsSinglePiece, 8, 15
- LeftCoset, 17
- LeftCosetRepresentatives, 17
- LeftCosetRepresentativesFromObject, 17
- LeftCosetsNC, 17
- LeftTransversalsOfGraphOfGroups, 25
- LefvtTransversalsOfGraphOfGroupoids, 29
- MagmaHomomorphismFromSinglePiece, 10
- MagmaWithObjects, 7
- MagmaWithObjectsHomomorphism, 10
- MaximalDiscreteSubgroupoid, 15
- MonoidWithObjects, 7
- MultiplicativeElementWithObjects, 8

NormalFormGGRWS, [26](#)

ObjectCostar, [16](#)

ObjectList, [7](#), [13](#)

ObjectStar, [16](#)

Pieces, [13](#)

Range, [20](#)

ReducedGraphOfGroupoidsWord, [30](#)

ReducedGraphOfGroupsWord, [26](#)

RightCoset, [17](#)

RightCosetRepresentatives, [17](#)

RightCosetsNC, [17](#)

RightTransversalsOfGraphOfGroupoids, [29](#)

RightTransversalsOfGraphOfGroups, [25](#)

SemigroupithObjects, [7](#)

SinglePieceGroupoid, [12](#)

Size, [14](#)

Source, [20](#)

Subgroupoid, [15](#)

SubgroupoidByPieces, [15](#)

SubgroupoidsOfGraphOfGroupoids, [29](#)

SubgroupsOfGraphOfGroups, [23](#)

WordOfGraphOfGroupoidsWord, [30](#)

WordOfGraphOfGroupsWord, [25](#)