Gpd

Groupoids, graphs of groups, and graphs of groupoids

Version 1.03

October 2007

Emma Moore
Chris Wensley

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

Chris Wensley — Email: c.d.wensley@bangor.ac.uk
— Homepage: http://www.informatics.bangor.ac.uk/~cwensley/
— 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 1.03, released on October 8th 2007.

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

Copyright

Contents

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. Some of the utility functions in the XMod package for crossed modules are used.

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 XMod 2.010 for GAP 4.4 - Murat Alp and Chris Wensley
-------------------------------------------------------------
-------------------------------------------------------------
loading Gpd 1.03 for GAP 4.4 - Emma Moore and Chris Wensley
-------------------------------------------------------------
true
gap> ?Groupoid
Help: several entries match this topic - type ?2 to get match [2]

[1] Gpd: Groupoid
[2] loops (not loaded): groupoid
[3] Gpd: Groupoids
[4] Gpd: Groupoids: their elements and attributes
[5] Gpd: GroupoidByUnion
[6] Gpd: GroupoidElement
[7] Gpd: GroupoidMorphismWithCommonRange
[8] Gpd: GroupoidMorphism
[9] Gpd: GroupoidMorphismByComponents
[10] Gpd: GroupoidMorphismByUnion
[11] Gpd: GroupoidsOfGraphOfGroupoids
```

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 all example commands in the Gpd manual
+ GAP4stones: 0
true
```

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

Chapter 2

Groupoids

A *groupoid* is a (mathematical) category in which every element is invertible. It consists of a set of components, each of which is a connected groupoid.

A *connected groupoid* is determined by a set of *objects* `obs` and an *object group* `grp`. Connected groupoids are represented as `IsConnectedGroupoidRep`, a record with components `objects` and `group`. The objects of a connected 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 connected groupoid with one object.

A *groupoid* is a set of one or more connected groupoids, its connected *components*, and is represented as `IsGroupoidRep`, with attribute `ComponentsOfGroupoid`.

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 Groupoids: their elements and attributes

2.1.1 ConnectedGroupoid

◊ `ConnectedGroupoid(`*`obs, grp`*`)` (operation)
◊ `GroupAsGroupoid(`*`obj, grp`*`)` (operation)
◊ `GroupoidByUnion(`*`comps`*`)` (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:

- a list of objects, the object group;
- a single object, a group being converted to a groupoid;
- 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 := ConnectedGroupoid( [-9,-8,-7], d8 );
Perm Connected Groupoid:
< [ -9, -8, -7 ], d8 >
gap> c6 := Group( (5,6,7)(8,9) );;
gap> SetName( c6, "c6" );
gap> Gc6 := GroupAsGroupoid( -6, c6 );
Perm Connected Groupoid:
< [ -6 ], c6 >
gap> Gd8c6 := GroupoidByUnion( [ Gd8, Gc6 ] );;
gap> Display( Gd8c6 );
Perm Groupoid with 2 components:
< 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" );
```

2.1.2 ComponentsOfGroupoid

◊ ComponentsOfGroupoid(*gpd*) (attribute)
◊ ObjectsOfGroupoid(*gpd*) (attribute)

When a groupoid consists of two or more components, we require their object lists to be disjoint. The components 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 components.

Example

```
gap> ComponentsOfGroupoid( Gd8c6 );
[ Gd8, Gc6 ]
gap> ObjectsOfGroupoid( Gd8c6 );
[ -9, -8, -7, -6 ]
```

2.1.3 IsPermGroupoid

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

A groupoid is a permutation groupoid if all its components 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( -22, f2 );;
gap> q8 := SmallGroup( 8, 4 );;
gap> Gq8 := Groupoid( [ -28, -27 ], q8 );;
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 components:
< 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
```

2.1.4 GroupoidElement

◊ GroupoidElement(*gpd, elt, tail, head*) (operation)
◊ IsElementOfGroupoid(*elt*) (property)
◊ GroupElement(*elt*) (attribute)
◊ Tail(*elt*) (attribute)
◊ Head(*elt*) (attribute)
◊ Size(*gpd*) (attribute)

A *groupoid element* e is a triple consisting of a group element, GroupElement(e) or e![1]; the tail (source) object, Tail(e) or e![2]; and the head (target) object, Head(e) or e![3].

The Size of a groupoid is the number of its elements which, for a connected 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. Conjugate a^b is defined when a is a loop at the tail of b.

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( [ GroupElement( e2 ), Tail( e2 ), Head( 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 ]
```

2.1.5 IsConnectedGroupoid

◊ IsConnectedGroupoid(*gpd*) (property)
◊ IsDiscreteGroupoid(*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.

2.2 Subgroupoids

2.2.1 SubgroupoidByComponents

◊ SubgroupoidByComponents(*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 SubgroupoidByComponents is the most general. Its two parameters are a groupoid and a list of connected components, where each component is specified as a list [obs,sgp], obs is a subset of the objects in one of the components of gpd, and sgp is a subgroup of the group in that component.

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 connected 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, [ [[-9, k4d]], [[-8,-7], c4d] ] );;
gap> SetName( Ud8, "Ud8" );
gap> Display( Ud8 );
Perm Groupoid with 2 components:
< 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 components:
< [ -7 ], d8 >
< [ -6 ], c6 >
gap> DiscreteSubgroupoid( Gd8c6, [-9,-8], [ c4d, k4d ] );
Perm Groupoid with components:
< [ -9 ], c4d >
< [ -8 ], k4d >
gap> FullIdentitySubgroupoid( Ud8 );
Perm Groupoid with components:
< [ -9 ], id(k4d) >
< [ -8, -7 ], id(c4d) >
```

2.3 Stars, Costars and Homsets

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

Example

```
gap> star9 := ObjectStar( Gd8, -9 );
<star at [ -9 ] with group d8>
```

```
gap> 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]
```

2.3.2 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 ]
```

2.4 Left, right and double cosets

2.4.1 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 $g1$ is related to $g2$ if and only if $g2 = u * g1$ for some element u of U. The right coset containing g is written Ug. These right cosets 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 component 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 component, 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] ]
```

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

Chapter 3

Morphisms of Groupoids

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

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

Note that when a morphism 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 mapping the two-object groupoid with trivial group to the free group $\langle a \rangle$ on one generator, when the image is $[1, a, a^{-1}]$.

3.1 Morphisms to a connected groupoid

3.1.1 MorphismOfConnectedGroupoids

◊ `MorphismOfConnectedGroupoids(`*G, H, imobs, hom*`)` (function)
◊ `Source(`*mor*`)` (attribute)
◊ `Range(`*mor*`)` (attribute)
◊ `ComponentImages(`*mor*`)` (attribute)

As usual, there are a variety of morphism constructors. The basic construction is a morphism $G \to H$ with H connected, which is implemented as `IsMorphismToConnectedGroupoidRep` with attributes `Source`, `Range` and `ComponentImages`. If G is also connected, we may apply `MorphismOfConnectedGroupoids`, requiring:

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

Example

```
gap> d12 := Group( (15,16,17,18,19,20, (15,20)(16,19)(17,18) );;
gap> Gd12 := ConnectedGroupoid( [-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 := SubgroupoidByComponents( 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> md8 := MorphismOfConnectedGroupoids( Gd8, Gs3d, [-34,-35,-36], hd8 );
Groupoid morphism : Gd8 -> Gs3d
gap> IsBijectiveOnObjects( md8 );
true
gap> Display( md8 );
Morphism to connected groupoid:
[ Gd8 ] -> [ Gs3d ]
  object map = [ [ -9, -8, -7 ], [ -34, -35, -36 ] ]
homomorphism = [ [ (1,2,3,4), (1,3) ], [ (), (15,20)(16,19)(17,18) ] ]
```

3.1.2 MorphismToConnectedGroupoid

◊ MorphismToConnectedGroupoid(*G, H, obhoms*) (function)
◊ GroupoidMorphismWithCommonRange(*G, H, mors*) (function)
◊ GroupoidMorphism(*args*) (function)

When *G* is not connected the appropriate operation is `MorphismToConnectedGroupoid( G, H, obhoms )`, where the third parameter is a list `ComponentImages( mor )` of `[imobs,hom]` pairs.

The operation `GroupoidMorphismWithCommonRange` combines into a single morphism two or more morphisms, in the list `mors`, with range *H* and source a subgroupoid of *G*.

The global function `GroupoidMorphism` will normally find the appropriate operation to call.

Example

```
gap> hc6 := GroupHomomorphismByImages( c6, s3d, [(5,6,7)(8,9)], [(15,19)(16,18)] );;
gap> mc6 := GroupoidMorphism( Gc6, Gs3d, [ -36 ], hc6 );;
gap> md8c6 := MorphismToConnectedGroupoid( Gd8c6, Gs3d,
              [ [[-34,-35,-36], hd8], [[-36], hc6] ] );
Groupoid morphism : Gd8+Gc6 -> Gs3d
gap> Display( md8c6 );
Morphism to connected groupoid with components:
[ Gd8 ] -> [ Gs3d ]
  object map = [ [ -9, -8, -7 ], [ -34, -35, -36 ] ]
homomorphism = [ [ (1,2,3,4), (1,3) ], [ (), (15,20)(16,19)(17,18) ] ]
[ Gc6 ] -> [ Gs3d ]
  object map = [ [ -6 ], [ -36 ] ]
homomorphism = [ [ (5,6,7)(8,9) ], [ (15,19)(16,18) ] ]
gap>
gap> gens3d := GeneratorsOfGroup( s3d );;
gap> imas3d := [ (15,17,19)(16,18,20), (15,16)(17,20)(18,19) ];;
gap> as3d := GroupHomomorphismByImages( s3d, s3d, gens3d, imas3d );;
gap> aGs3d := MorphismOfConnectedGroupoids( Gs3d, Gs3d, [-35,-34,-36], as3d );;
gap> IsGroupoidAutomorphism( aGs3d );
true
gap> Order( aGs3d );
3
gap> Gs3dd8 := GroupoidByUnion( [ Gd8, Gs3d ] );;
gap> SetName( Gs3dd8, "Gs3d+Gd8" );;
gap> common := GroupoidMorphismWithCommonRange( Gs3dd8, Gs3d, [ aGs3d, md8 ] );;
```

```
gap> Display( common );
Morphism to connected groupoid with components:
[ Gs3d ] -> [ Gs3d ]
  object map = [ [ -36, -35, -34 ], [ -35, -34, -36 ] ]
homomorphism = [ [ (15,17,19)(16,18,20), (15,20)(16,19)(17,18) ],
  [ (15,17,19)(16,18,20), (15,16)(17,20)(18,19) ] ]
[ Gd8 ] -> [ Gs3d ]
  object map = [ [ -9, -8, -7 ], [ -34, -35, -36 ] ]
homomorphism = [ [ (1,2,3,4), (1,3) ], [ (), (15,20)(16,19)(17,18) ] ]
```

3.1.3 IsomorphismNewObjects

◊ IsomorphismNewObjects(*gpd, obs*) (operation)
◊ IdentityMapping(*gpd*) (attribute)

The function IsomorphismNewObjects provides an isomorphism to an isomorphic groupoid with a new set of objects. The function IdentityMapping, already provided for groups, is also implemented for groupoids.

Example

```
gap> idGd8 := IdentityMapping( Gd8 );
Groupoid morphism: [ Gd8 ] -> [ Gd8 ]
gap> copys3d := IsomorphismNewObjects( Gs3d, [-24,-25,-26] );;
gap> Hs3d := Range( copys3d );;
gap> SetName( Hs3, "Hs3" );
gap> Display( copys3d );
Morphism to connected groupoid:
[ Gs3d ] -> [ Hs3d ]
  object map = [ [ -36, -35, -34 ], [ -32, -31, -30 ] ]
homomorphism = [ [ (15,17,19)(16,18,20), (15,20)(16,19)(17,18) ],
  [ (15,17,19)(16,18,20), (15,20)(16,19)(17,18) ] ]
```

3.1.4 InclusionMappingGroupoids

◊ InclusionMappingGroupoids(*G, U*) (operation)
◊ RestrictionMappingGroupoids(*hom, src, rng*) (operation)

We have added functions InclusionMappingGroupoids and RestrictionMappingGroupoids, corresponding to InclusionMappingGroups and RestrictionMappingGroups for groups. At present RestrictionMappingGroups is only implemented for the case where the source and range of the morphism are connected.

Example

```
gap> SetName( ComponentsOfGroupoid(Ud8)[1], "Ud8a" );
gap> SetName( ComponentsOfGroupoid(Ud8)[2], "Ud8b" );
gap> inc := InclusionMappingGroupoids( Gd8, Ud8 );;
gap> Display( inc );
```

```
Morphism to connected groupoid with components:
[ Ud8a ] -> [ Gd8 ]
  object map = [ [ -9 ], [ -9 ] ]
homomorphism = [ [ (1,2)(3,4), (1,3)(2,4) ], [ (1,2)(3,4), (1,3)(2,4) ] ]
[ Ud8b ] -> [ Gd8 ]
  object map = [ [ -8, -7 ], [ -8, -7 ] ]
homomorphism = [ [ (1,2,3,4) ], [ (1,2,3,4) ] ]
gap> res := RestrictionMappingGroupoids( md8, Ud8, Gs3d );
gap> Display( res );
Morphism to connected groupoid with components:
[ Ud8a ] -> [ Gs3d ]
  object map = [ [ -9 ], [ -34 ] ]
homomorphism = [ [ (1,2)(3,4), (1,3)(2,4) ], [ (15,20)(16,19)(17,18), () ] ]
[ Ud8b ] -> [ Gs3d ]
  object map = [ [ -8, -7 ], [ -35, -36 ] ]
homomorphism = [ [ (1,2,3,4) ], [ () ] ]
```

3.2 General Morphisms and composites

3.2.1 GroupoidMorphismByComponents

◊ `GroupoidMorphismByComponents(`*G, H, mors*`)` (operation)
◊ `GroupoidMorphismByUnion(`*mors*`)` (function)

When H is not connected, a groupoid morphism from G to H is a union of a list `mors` of morphisms of connected groupoids.

There are two ways of combining `m1 : G1->H1` and `m2 : G2->H2` into a single morphism. The operation `GroupoidMorphismByUnion( [m1,m2] );` constucts the groupoid unions `G12,H12` of `G1` with `G2` and `H1` with `H2`, and then a morphism `m12 : G12->H12`. If `G12,H12` have already been constructed, the operation `GroupoidMorphismByComponents( G12, H12, [m1,m2] );` may be more appropriate.

Example

```
gap> imad8 := [ (1,2,3,4), (2,4) ];;
gap> ad8 := GroupHomomorphismByImages( d8, d8, [(1,2,3,4),(1,3)], imad8 );;
gap> aGd8 := MorphismOfConnectedGroupoids( Gd8, Gd8, [-8,-7,-9], ad8 );;
gap> mor1 := GroupoidMorphismByUnion( [ aGd8, mc6 ] );;
gap> mor2 := GroupoidMorphismByComponents( Gd8c6, Gs3dd8, [ aGd8, mc6 ] );;
gap> Display( mor2 );
Groupoid Morphism: Gd8+Gc6 -> Gs3d+Gd8 with components :
[ Gc6 ] -> [ Gs3d ]
  object map = [ [ -6 ], [ -36 ] ]
homomorphism = [ [ (5,6,7)(8,9) ], [ (15,19)(16,18) ] ]
[ Gd8 ] -> [ Gd8 ]
  object map = [ [ -9, -8, -7 ], [ -8, -7, -9 ] ]
homomorphism = [ [ (1,2,3,4), (1,3) ], [ (1,2,3,4), (2,4) ] ]
gap> mor1 = mor2;
true
```

3.2.2 IsGroupoidMorphism

◊ IsGroupoidMorphism(*mor*) (property)
◊ IsInjectiveOnObjects(*mor*) (property)
◊ IsSurjectiveOnObjects(*mor*) (property)
◊ IsBijectiveOnObjects(*mor*) (property)
◊ IsGroupoidAutomorphism(*mor*) (property)

Here are some of the basic properties of groupoid morphisms.

Example

```
gap> IsGroupoidMorphism( res );
true
gap> IsInjectiveOnObjects( res );
true
gap> IsInjective( res );
false
gap> IsSurjectiveOnObjects( inc );
true
gap> IsSurjective( inc );
false
gap> IsGroupoidAutomorphism( aGd8 );
true
gap> eGd8 := MorphismOfConnectedGroupoids( Gd8, Gd8, [-7,-7,-9], ad8 );;
gap> IsGroupoidEndomorphism( eGd8 );
true
```

3.2.3 ImagesOfObjects

◊ ImagesOfObjects(*mor*) (attribute)
◊ ImagesSource(*mor*) (attribute)
◊ ImageElm(*mor, elt*) (operation)

Here are some of examples of operations that return images. These are not yet fully implemented: for example, ImageElm and ImagesSource require the range of mor to be connected.

Example

```
gap> ImagesOfObjects( md8 );
[ [ -34, -35, -36, -36 ] ]
gap> ImagesSource(md8);
Perm Connected Groupoid:
< [ -36, -35, -34 ], Group( [ (), (15,20)(16,19)(17,18) ] ) >
gap> e1;
[(1,2)(3,4) : -9 -> -8]
gap> ImageElm( md8c6, e1 );
[(15,20)(16,19)(17,18) : -34 -> -35]
```

3.2.4 Product

◊ Product(*mors*) (operation)

Morphisms m_1, m_2 can be composed if the image of m_1 is contained in the domain of m_2. Currently composition is only defined for morphisms of connected groupoids.

Example

```
gap> Hk4d := SubgroupoidByComponents( Gd8, [ [[-9,-8,-7], k4d] ] );;
gap> genq8 := GeneratorsOfGroup(q8);;
gap> hq8 := GroupHomomorphismByImages( q8,k4d,genq8,[(1,2)(3,4),(1,3)(2,4),()]);;
gap> mq8 := GroupoidMorphism( Gq8, Hk4d, [-7,-9], hq8 );;
gap> iHk4d := InclusionMappingGroupoids( Gd8, Hk4d );;
gap> mor3 := mq8*iHk4d;;
gap> Display( mor3 );
Morphism to connected groupoid:
[ Gq8 ] -> [ Gd8 ]
  object map = [ [ -28, -27 ], [ -7, -9 ] ]
homomorphism = [ Pcgs([ f1, f2, f3 ]), [ (1,2)(3,4), (1,3)(2,4), () ] ]
gap> mor4 := mor1*aGd8;;
gap> Display( mor4 );
Morphism to connected groupoid:
[ Gq8 ] -> [ Gd8 ]
  object map = [ [ -28, -27 ], [ -9, -8 ] ]
homomorphism = [ Pcgs([ f1, f2, f3 ]), [ (1,4)(2,3), (1,3)(2,4), () ] ]
```

3.2.5 InverseMorphism

◊ InverseMorphism(*mor*) (attribute)
◊ Order(*mor*) (attribute)

The *inverse* of a morphism m is defined when m is bijective on objects and on elements. The operation `InverseMorphism` is a synonym for `InverseGeneralMapping`. The *order* of an automorphism is the smallest power which returns the identity morphism of G.

Example

```
gap> genq8 := GeneratorsOfGroup( q8 );;
gap> imq8 := [ q8.2, q8.1*q8.2, q8.3 ];
[ f2, f1*f2, f3 ]
gap> autq8 := GroupHomomorphismByImages( q8, q8, genq8, imq8 );;
gap> autGq8 := GroupoidMorphism( Gq8, Gq8, [-27,-28], autq8 );;
gap> Display( InverseMorphism( autGq8 ) );
Morphism to connected groupoid:
[ Gq8 ] -> [ Gq8 ]
  object map = [ [ -28, -27 ], [ -27, -28 ] ]
homomorphism = [ [ f2, f1*f2, f3 ], [ f1, f2, f3 ] ]
gap> Order( autGq8 );
6
```

Chapter 4

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.

4.1 Digraphs

4.1.1 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 ]
```

4.2 Graphs of Groups

4.2.1 GraphOfGroups

◊ `GraphOfGroups(`*dig, gps, sgps, isos*`)` (operation)
◊ `DigraphOfGraphOfGroups(`*gg*`)` (attribute)
◊ `GroupsOfGraphOfGroups(`*gg*`)` (attribute)
◊ `SubgroupsOfGraphOfGroups(`*gg*`)` (attribute)
◊ `IsomorphismsOfGraphOfGroups(`*gg*`)` (attribute)

A graph of groups is traditionally defined as consisting of:

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

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

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

These two viewpoints are clearly equivalent.

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

The 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 ] ] ]
```

4.2.2 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 ] ]
```

4.2.3 RightTransversalsOfGraphOfGroups

◊ RightTransversalsOfGraphOfGroups(*gg*) (attribute)
◊ LeftTransversalsOfGraphOfGroups(*gg*) (attribute)

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

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

Example

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

4.3 Words in a Graph of Groups and their normal forms

4.3.1 GraphOfGroupsWord

◊ GraphOfGroupsWord(*gg, tv, list*) (operation)
◊ IsGraphOfGroupsWord(*w*) (property)
◊ GraphOfGroupsOfWord(*w*) (attribute)
◊ WordOfGraphOfGroupsWord(*w*) (attribute)

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

An element of A(G) is a graph of groups word which may be represented by a list of the form $w = [g_1, y_1, g_2, y_2, \ldots, g_n, y_n, g_{n+1}]$. Here each y_i is an arc of D; the head of y_{i-1} is a vertex v_i which is also the tail of y_i; and g_i is an element of the vertex group at v_i. 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> [ Tail(gw1), Head(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 ]
```

4.3.2 ReducedGraphOfGroupsWord

◊ ReducedGraphOfGroupsWord(*w*) (operation)
◊ IsReducedGraphOfGroupsWord(*w*) (property)

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

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

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

Example

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

4.4 Free products with amalgamation and HNN extensions

4.4.1 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> gg1 := GraphOfGroupsRewritingSystem( fpa );;
gap> Display( gg1 );
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( gg1 );
[ [ <identity ..>, a2^-1 ], [ <identity ..>, b2^-1, b1^-1*b2^-1, b1*b2^-1 ] ]
gap> ## choose a word in fpa and find its normal form
gap> gfpa := GeneratorsOfGraphOfGroups( fpa );;
gap> w2 := (gfpa[1]*gfpa[2]*gfpa[3]^gfpa[4])^3;
fa1*fa2*fa4^-1*fa3*fa4*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
```

4.4.2 HnnExtension

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

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

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

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

Example

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

4.5 GraphsOfGroupoids and their Words

4.5.1 GraphOfGroupoids

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

Graphs of groups generalise naturally to graphs of groupoids, forming the class IsGraphOfGroupoids. There is now a groupoid at each vertex and the isomorphism on an arc identifies wide subgroupoids at the tail and at the head. Since all subgroupoids are wide, every groupoid in a connected component 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 := ConnectedGroupoid( [-2,-1], fa );;
gap> Uhy := Subgroupoid( Gfa, [ [[-2,-1], hy ] ] );;
gap> SetName( Gfa, "Gfa" );  SetName( Uhy, "Uhy" );
gap> Gfb := ConnectedGroupoid( [-4,-3], fb );;
gap> Uhybar := Subgroupoid( Gfb, [ [[-4,-3], hybar ] ] );;
gap> SetName( Gfb, "Gfb" );  SetName( Uhybar, "Uhybar" );
gap> mory := MorphismOfConnectedGroupoids( Uhy, Uhybar, [-4,-3], homy );;
gap> morybar := MorphismOfConnectedGroupoids( Uhybar, Uhy, [-2,-1], homybar );;
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 Connected Groupoid: Gfa
  objects: [ -2, -1 ]
    group: fa = <[ a ]>
Fp Connected Groupoid: Gfb
  objects: [ -4, -3 ]
    group: fb = <[ b ]>
subgroupoids: Connected Groupoid: Uhy
  objects: [ -2, -1 ]
    group: hy = <[ a^3 ]>
Connected Groupoid: Uhybar
  objects: [ -4, -3 ]
    group: hybar = <[ b^2 ]>
isomorphisms: [ Groupoid morphism : Uhy -> Uhybar
```

```
    , Groupoid morphism : Uhybar -> Uhy
     ]
```

4.5.2 GraphOfGroupoidsWord

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

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

Example

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

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

Chapter 5

Development History

5.1 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`.

5.2 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 1.04 will extend 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

[Bro06] Ronald Brown. *Topology and groupoids*. www.groupoids.org, Deganwy, 2006. 7

[Hig76] Philip Higgins. The fundamental groupoid of a graph of groups. *J. London Math. Soc.*, 13:145–149, 1976. 20

[Moo01] Emma Moore. *Graphs of groups: word computations and free crossed resolutions*. Ph.D. thesis, University of Wales, Bangor, 2001. 5, 6, 20, 23, 24, 29

[Ser80] J. Serre. *Trees*. Springer-Verlag, Berlin, 1980. 20

Index