RAMP

The Research Assistant for Maniplexes and Polytopes

0.3

28 July 2020

Gabe Cunningham

Mark Mixer

Gordon Williams

Gabe Cunningham
Email: gabriel.cunningham@umb.edu
Homepage: http://www.gabrielcunningham.com
Address: Gabe Cunningham
Department of Mathematics
University of Massachusetts Boston
100 William T. Morrissey Blvd.
Boston MA 02125

Mark Mixer
Email: mixerm@wit.edu

Gordon Williams
Email: giwilliams@alaska.edu

Copyright

Acknowledgements

We appreciate very much all past and future comments, suggestions and contributions to this package and its documentation provided by GAP users and developers.

Contents

1 Graphs for Maniplexes **5**
1.1 Graphs for maniplexes functions 5

2 Basics **12**
2.1 Constructors 12

3 Combinatorics and Structure **14**
3.1 Faces 14
3.2 Flatness 16
3.3 Schlafli symbol 16
3.4 Basics 16
3.5 Zigzags and holes 18

4 Constructions **19**
4.1 Extensions, amalgamations, and quotients 19
4.2 Duality 20
4.3 Products 20

5 Databases **22**
5.1 Regular polyhedra 22

6 Families of Polytopes **23**
6.1 Classical Polytopes 23
6.2 Uniform Polyhedra 25

7 Groups **28**
7.1 Groups 28

8 Mixing of Maniplexes **30**
8.1 Mixing of Maniplexes functions 30

9 Properties **32**
9.1 Orientability 32

10 Actions **33**
10.1 Faithfulness 33

11 Posets **34**
11.1 Poset constructors 34
11.2 Poset attributes 37
11.3 Working with posets 40
11.4 Elements of posets, also known as faces. 42
11.5 Element Constructors 43
11.6 Element operations 44

12 Products of Posets and Digraphs **45**
12.1 Construction methods 45

13 Comparing maniplexes **48**
13.1 Quotients and covers 48

14 ramp automatic generated documentation **50**
14.1 ramp automatic generated documentation of methods 50

15 Utility functions **52**
15.1 Utility functions 52

Index **54**

Chapter 1

Graphs for Maniplexes

1.1 Graphs for maniplexes functions

1.1.1 DirectedGraphFromListOfEdges (for IsList,IsList)

▷ `DirectedGraphFromListOfEdges(list, list)` (operation)

Returns: `IsGraph`. Note this returns a directed graph.

Given a list of vertices and a list of directed-edges (represented as ordered pairs), this outputs the directed graph with the appropriate vertex and directed-edge set.

Here we have a directed cycle on 3 vertices.

Example
```
gap> g:= DirectedGraphFromListOfEdges([1,2,3],[[1,2],[2,3],[3,1]]);
rec( adjacencies := [ [ 2 ], [ 3 ], [ 1 ] ], group := Group(()),
 isGraph := true, names := [ 1, 2, 3 ], order := 3,
 representatives := [ 1, 2, 3 ], schreierVector := [ -1, -2, -3 ] )
```

1.1.2 GraphFromListOfEdges (for IsList,IsList)

▷ `GraphFromListOfEdges(list, list)` (operation)

Returns: `IsGraph`. Note this returns an undirected graph.

Given a list of vertices and a list of (directed) edges (represented as ordered pairs), this outputs the simple underlying graph with the appropriate vertex and directed-edge set.

Here we have a simple complete graph on 4 vertices.

Example
```
gap> g:= GraphFromListOfEdges([1,2,3,4],[[1,2],[2,3],[3,1], [1,4], [2,4], [3,4]]);
rec(
 adjacencies := [ [ 2, 3, 4 ], [ 1, 3, 4 ], [ 1, 2, 4 ], [ 1, 2, 3 ] ],
 group := Group(()), isGraph := true, isSimple := true,
 names := [ 1, 2, 3, 4 ], order := 4, representatives := [ 1, 2, 3, 4 ]
   , schreierVector := [ -1, -2, -3, -4 ] )
```

1.1.3 UnlabeledFlagGraph (for IsGroup)

▷ `UnlabeledFlagGraph(group)` (operation)

Returns: `IsGraph`. Note this returns an undirected graph.

Given a group (assumed to be the connection group of a maniplex), this outputs the simple underlying flag graph.

Here we build the flag graph for the cube from its connection group.

Example
```
gap> g:= UnlabeledFlagGraph(ConnectionGroup(Cube(3)));
rec(
 adjacencies := [ [ 3, 11, 20 ], [ 7, 13, 18 ], [ 1, 4, 10 ],
     [ 3, 25, 34 ], [ 26, 28, 35 ], [ 7, 13, 41 ], [ 2, 6, 8 ],
     [ 7, 27, 32 ], [ 28, 33, 35 ], [ 3, 20, 45 ], [ 1, 14, 23 ],
     [ 15, 17, 24 ], [ 2, 6, 31 ], [ 11, 25, 44 ], [ 12, 45, 47 ],
     [ 18, 28, 40 ], [ 12, 19, 27 ], [ 2, 16, 21 ], [ 17, 22, 24 ],
     [ 1, 10, 38 ], [ 18, 32, 40 ], [ 19, 41, 48 ], [ 11, 35, 44 ],
     [ 12, 19, 34 ], [ 4, 14, 37 ], [ 5, 38, 42 ], [ 8, 17, 30 ],
     [ 5, 9, 16 ], [ 39, 41, 48 ], [ 27, 32, 47 ], [ 13, 33, 39 ],
     [ 8, 21, 30 ], [ 9, 31, 46 ], [ 4, 24, 37 ], [ 5, 9, 23 ],
     [ 43, 45, 47 ], [ 25, 34, 48 ], [ 20, 26, 43 ], [ 29, 31, 46 ],
     [ 16, 21, 42 ], [ 6, 22, 29 ], [ 26, 40, 43 ], [ 36, 38, 42 ],
     [ 14, 23, 46 ], [ 10, 15, 36 ], [ 33, 39, 44 ], [ 15, 30, 36 ],
     [ 22, 29, 37 ] ], group := Group(()), isGraph := true,
 isSimple := true, names := [ 1 .. 48 ], order := 48,
 representatives := [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
     15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
     31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46,
     47, 48 ],
 schreierVector := [ -1, -2, -3, -4, -5, -6, -7, -8, -9, -10, -11,
     -12, -13, -14, -15, -16, -17, -18, -19, -20, -21, -22, -23, -24,
     -25, -26, -27, -28, -29, -30, -31, -32, -33, -34, -35, -36, -37,
     -38, -39, -40, -41, -42, -43, -44, -45, -46, -47, -48 ] )
```

This also works with a maniplex input. Here we build the flag graph for the cube.

Example
```
gap> g:= UnlabeledFlagGraph(Cube(3));
```

1.1.4 FlagGraphWithLabels (for IsGroup)

▷ `FlagGraphWithLabels(`*group*`)` (operation)

Returns: a triple [`IsGraph`, `IsList`, `IsList`].

Given a group (assumed to be the connection group of a maniplex), this outputs a triple [graph,list,list]. The graph is the unlabeled flag graph of the connection group. The first list gives the undirected edges in the flag graphs. The second list gives the labels for these edges.

Here we again build the flag graph for the cube from its connection group, but this time keep track of labels of the edges.

Example
```
gap> g:= FlagGraphWithLabels(ConnectionGroup(Cube(3)));
[ rec(
     adjacencies := [ [ 3, 11, 20 ], [ 7, 13, 18 ], [ 1, 4, 10 ],
         [ 3, 25, 34 ], [ 26, 28, 35 ], [ 7, 13, 41 ], [ 2, 6, 8 ],
         [ 7, 27, 32 ], [ 28, 33, 35 ], [ 3, 20, 45 ], [ 1, 14, 23 ],
         [ 15, 17, 24 ], [ 2, 6, 31 ], [ 11, 25, 44 ], [ 12, 45, 47 ],
         [ 18, 28, 40 ], [ 12, 19, 27 ], [ 2, 16, 21 ],
         [ 17, 22, 24 ], [ 1, 10, 38 ], [ 18, 32, 40 ],
```

```
        [ 19, 41, 48 ], [ 11, 35, 44 ], [ 12, 19, 34 ],
        [ 4, 14, 37 ], [ 5, 38, 42 ], [ 8, 17, 30 ], [ 5, 9, 16 ],
        [ 39, 41, 48 ], [ 27, 32, 47 ], [ 13, 33, 39 ],
        [ 8, 21, 30 ], [ 9, 31, 46 ], [ 4, 24, 37 ], [ 5, 9, 23 ],
        [ 43, 45, 47 ], [ 25, 34, 48 ], [ 20, 26, 43 ],
        [ 29, 31, 46 ], [ 16, 21, 42 ], [ 6, 22, 29 ],
        [ 26, 40, 43 ], [ 36, 38, 42 ], [ 14, 23, 46 ],
        [ 10, 15, 36 ], [ 33, 39, 44 ], [ 15, 30, 36 ],
        [ 22, 29, 37 ] ], group := Group(()), isGraph := true,
    isSimple := true, names := [ 1 .. 48 ], order := 48,
    representatives := [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
        14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
        29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43,
        44, 45, 46, 47, 48 ],
    schreierVector := [ -1, -2, -3, -4, -5, -6, -7, -8, -9, -10, -11,
        -12, -13, -14, -15, -16, -17, -18, -19, -20, -21, -22, -23,
        -24, -25, -26, -27, -28, -29, -30, -31, -32, -33, -34, -35,
        -36, -37, -38, -39, -40, -41, -42, -43, -44, -45, -46, -47,
        -48 ] ),
  [ [ 1, 3 ], [ 1, 11 ], [ 1, 20 ], [ 2, 7 ], [ 2, 13 ], [ 2, 18 ],
    [ 3, 4 ], [ 3, 10 ], [ 4, 25 ], [ 4, 34 ], [ 5, 26 ], [ 5, 28 ],
    [ 5, 35 ], [ 6, 7 ], [ 6, 13 ], [ 6, 41 ], [ 7, 8 ], [ 8, 27 ],
    [ 8, 32 ], [ 9, 28 ], [ 9, 33 ], [ 9, 35 ], [ 10, 20 ],
    [ 10, 45 ], [ 11, 14 ], [ 11, 23 ], [ 12, 15 ], [ 12, 17 ],
    [ 12, 24 ], [ 13, 31 ], [ 14, 25 ], [ 14, 44 ], [ 15, 45 ],
    [ 15, 47 ], [ 16, 18 ], [ 16, 28 ], [ 16, 40 ], [ 17, 19 ],
    [ 17, 27 ], [ 18, 21 ], [ 19, 22 ], [ 19, 24 ], [ 20, 38 ],
    [ 21, 32 ], [ 21, 40 ], [ 22, 41 ], [ 22, 48 ], [ 23, 35 ],
    [ 23, 44 ], [ 24, 34 ], [ 25, 37 ], [ 26, 38 ], [ 26, 42 ],
    [ 27, 30 ], [ 29, 39 ], [ 29, 41 ], [ 29, 48 ], [ 30, 32 ],
    [ 30, 47 ], [ 31, 33 ], [ 31, 39 ], [ 33, 46 ], [ 34, 37 ],
    [ 36, 43 ], [ 36, 45 ], [ 36, 47 ], [ 37, 48 ], [ 38, 43 ],
    [ 39, 46 ], [ 40, 42 ], [ 42, 43 ], [ 44, 46 ] ],
  [ 3, 2, 1, 3, 1, 2, 2, 1, 3, 1, 2, 3, 1, 1, 3, 2, 2, 1, 3, 1, 2, 3,
    3, 2, 3, 1, 2, 3, 1, 2, 2, 1, 1, 3, 1, 2, 3, 1, 2, 3, 2, 3, 2, 2,
    1, 1, 3, 2, 3, 2, 1, 1, 3, 3, 2, 3, 1, 1, 2, 1, 3, 3, 3, 2, 3, 1,
    2, 3, 1, 2, 1, 2 ] ]
```

This also works with a maniplex input. Here we build the flag graph for the cube.

Example
```
gap> g:= FlagGraphWithLabels(Cube(3));
```

1.1.5 LayerGraph (for IsGroup, IsInt, IsInt)

▷ `LayerGraph([group, int, int])` (operation)

Returns: `IsGraph`. Note this returns an undirected graph.

Given a group (assumed to be the connection group of a maniplex), and two integers, this outputs the simple underlying graph given by incidences of faces of those ranks. Note: There are no warnings yet to make sure that i,j are bounded by the rank.

Here we build the graph given by the 6 faces and 12 edges of a cube from its connection group.

Example
```
gap> g:= LayerGraph(ConnectionGroup(Cube(3)),2,1);
rec(
 adjacencies := [ [ 7, 10, 12, 17 ], [ 8, 10, 15, 18 ],
     [ 7, 9, 13, 14 ], [ 8, 11, 13, 16 ], [ 9, 12, 16, 18 ],
     [ 11, 14, 15, 17 ], [ 1, 3 ], [ 2, 4 ], [ 3, 5 ], [ 1, 2 ],
     [ 4, 6 ], [ 1, 5 ], [ 3, 4 ], [ 3, 6 ], [ 2, 6 ], [ 4, 5 ],
     [ 1, 6 ], [ 2, 5 ] ], group := Group(()), isGraph := true,
 isSimple := true, names := [ 1 .. 18 ], order := 18,
 representatives := [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
     15, 16, 17, 18 ],
 schreierVector := [ -1, -2, -3, -4, -5, -6, -7, -8, -9, -10, -11,
     -12, -13, -14, -15, -16, -17, -18 ] )
```

This also works with a maniplex input. Here we build the graph given by the 6 faces and 12 edges of a cube.

Example
```
gap> g:= LayerGraph(Cube(3),2,1);;
```

1.1.6 Skeleton (for IsManiplex)

▷ `Skeleton(maniplex)` (operation)

Returns: `IsGraph`. Note this returns an undirected graph.

Given a maniplex, this outputs the 0-1 skeleton. The vertices are the 0-faces, and the edges are the 1-faces.

Here we build the skeleton of the dodecahedron.

Example
```
gap> g:= Skeleton(Dodecahedron());;
```

1.1.7 Hasse (for IsManiplex)

▷ `Hasse(group)` (operation)

Returns: `IsGraph`. Note this returns a directed graph.

Given a group, assumed to be the connection group of a maniplex, this outputs the Hasse Diagram as a directed graph. Note: The unique minimal and maximal face are assumed.

Here we build the Hasse Diagram of a 3-simplex from its representation as a maniplex.

Example
```
gap> Hasse(Simplex(3));
rec(
 adjacencies := [ [  ], [ 1 ], [ 1 ], [ 1 ], [ 1 ], [ 2, 4 ],
     [ 2, 3 ], [ 3, 5 ], [ 2, 5 ], [ 4, 5 ], [ 3, 4 ], [ 6, 9, 10 ],
     [ 6, 7, 11 ], [ 8, 10, 11 ], [ 7, 8, 9 ], [ 12, 13, 14, 15 ] ],
 group := Group(()), isGraph := true, names := [ 1 .. 16 ],
 order := 16,
 representatives := [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
     15, 16 ],
 schreierVector := [ -1, -2, -3, -4, -5, -6, -7, -8, -9, -10, -11,
     -12, -13, -14, -15, -16 ] )
```

1.1.8 QuotientByLabel (for IsObject,IsList, IsList, IsList)

▷ `QuotientByLabel(object, list, list, list)` (operation)

Returns: `IsGraph`. Note this returns an undirected graph.

Given a graph, its edges, and its edge labels, and a sublist of labels, this creates the underlying simple graph of the quotient identifying vertices connected by labels not in the sublist.

Here we start with the flag graph of the 3-cube (with edge labels 1,2,3), and identify any vertices not connected by edge by edges of label 1. We can then check that this new graph is bipartite.

Example

```
gap> P:=Cube(3);;
gap> f:=FlagGraphWithLabels(P);;
gap> g:=f[1];;
gap> ed:=f[2];;
gap> lab:=f[3];  #Note This triple is to be replace by a single object.
[ 3, 2, 1, 3, 1, 2, 1, 2, 3, 2, 1, 3, 2, 1, 1, 3, 2, 2, 3, 1, 3, 1, 2, 3, 2, 1, 1, 2, 2, 3, 1, 3,
  3, 1, 2, 1, 3, 2, 2, 1, 2, 2, 3, 1, 1, 3, 1, 3, 3, 2, 1, 2, 1, 3, 3, 1, 3, 2, 2, 2, 2, 3, 3, 1,
gap> Q:=QuotientByLabel(g,ed,lab,[1]);
rec( adjacencies := [ [ 5, 6, 8 ], [ 3, 4, 7 ], [ 2, 6, 8 ], [ 2, 5, 8 ], [ 1, 4, 7 ], [ 1, 3, 7
 isSimple := true, names := [ 1 .. 8 ], order := 8, representatives := [ 1, 2, 3, 4, 5, 6, 7, 8 ]
gap> IsBipartite(Q);
true
```

1.1.9 EdgeLabeledGraphFromEdges (for IsList, IsList,IsList)

▷ `EdgeLabeledGraphFromEdges(list, list, list)` (operation)

Returns: `IsEdgeLabeledGraph`.

Given a list of vertices, a list of edges, and a list of edge labels, this represents the edge labeled (multi)-graph with those parameters. Semi-edges are represented by a singleton in the edge list. Loops are represented by edges [i,i]

Here we have an edge labeled cycle graph with 6 vertices and edges alternating in labels 0,1.

Example

```
V:=[1..6];;
Edges:=[[1,2],[2,3],[3,4],[4,5],[5,6],[6,1]];;
L:=[0,1,0,1,0,1];;
gamma:=EdgeLabeledGraphFromEdges(V,Edges,L);
```

1.1.10 FlagGraph (for IsGroup)

▷ `FlagGraph(group)` (operation)

Returns: `IsEdgeLabeledGraph`.

Given group, assumed to be a connection group, output the labeled flag graph. The input could also be a maniplex, then the connection group is calculated.

Here we have the flag graph of the 3-simplex from its connection group.

Example

```
C:=ConnectionGroup(Simplex(3));;
gamma:=FlagGraph(C);
```

1.1.11 UnlabeledSimpleGraph (for IsEdgeLabeledGraph)

▷ `UnlabeledSimpleGraph(`*edge-labeled-graph*`)` (operation)

Returns: `IsGraph`.

Given an edge labeled (multi) graph, it returns the underlying simple graph, with semi-edges, loops, and muliple-edges removed.

Here we have underlying simple graph for the flag graph of the cube.

Example
```
gamma:=UnlabeledSimpleGraph(FlagGraph(Cube(3)));
```

1.1.12 EdgeLabelPreservingAutomorphismGroup (for IsEdgeLabeledGraph)

▷ `EdgeLabelPreservingAutomorphismGroup(`*edge-labeled-graph*`)` (operation)

Returns: `IsGroup`.

Given an edge labeled (multi) graph, it returns automorphism group (preserving the labels). Note, for now the labels are assumed to be [1..n]. Note This tends to be very slow. I would like to look for a way to go back and forth between flag automorphisms and poset automorphisms, as the latter are much faster to compute.

Here we have the automorphism group of the flag graph of the cube.

Example
```
g:=EdgeLabelPreservingAutomorphismGroup(FlagGraph(Cube(3)));;
Size(g);
```

1.1.13 Simple (for IsEdgeLabeledGraph)

▷ `Simple(`*edge-labeled-graph*`)` (operation)

Returns: `IsEdgeLabeledGraph` .

Given an edge labeled (multi) graph, it returns another edge labeled graph where semi-edges, loops, and multiple edges are removed. Note only the "first" edge label is retained if there are multiple edges.

1.1.14 ConnectedComponents (for IsEdgeLabeledGraph, IsList)

▷ `ConnectedComponents(`*edge-labeled-graph*`)` (operation)

Returns: `IsGraph`.

Given an edge labeled (multi) graph and a list of labels, it returns connected components of the graph not using edges in the list of labels. Note if the second argument is not used, it is assumed to be an empty list, and the connected components of the original graph are returned.

Here we see that each connected component of the flag graph of the cube (which has labels 1,2,3) where edges of label 2 are removed, is a 4 cycle.

Example
```
gamma:=ConnectedComponents(FlagGraph(Cube(3)),[2]);
```

1.1.15 PRGraph (for IsGroup)

▷ `PRGraph(`*group*`)` (operation)

Returns: `IsEdgeLabeledGraph` .

Given a group, it returns the permutation representation graph for that group. When the group is a string C-group this is also called a CPR graph. The labels of the edges are [1...r] where r is the number of generators of the group.

Here we see the CPR graph of the automorphism group of a cube (acting on its 8 vertices).

Example
```
G:=AutomorphismGroup(Cube(3));
H:=Group(G.2,G.3);
phi:=FactorCosetAction(G,H);
G2:=Range(phi);
gamma:=PRGraph(G2);
```

1.1.16 CPRGraphFromGroups (for IsGroup,IsGroup)

▷ `CPRGraphFromGroups(group, subgroup)` (operation)

Returns: `IsEdgeLabeledGraph`.

Given a group and a subgroup. Returns the graph of the action of the first group on cosets of the subgroup.

Chapter 2

Basics

2.1 Constructors

2.1.1 UniversalSggi

▷ `UniversalSggi(n)` (operation)
▷ `UniversalSggi(sym)` (operation)

In the first form, returns the universal Coxeter Group of rank n. In the second form, returns the Coxeter Group with Schlafli symbol sym.

2.1.2 ReflexibleManiplex (for IsGroup)

▷ `ReflexibleManiplex(g)` (operation)

Given a group g (which should be a string C-group), returns the reflexible maniplex with that automorphism group, where the privileged generators are those returned by GeneratorsOfGroup(g).

2.1.3 ReflexibleManiplex (for IsList)

▷ `ReflexibleManiplex(sym)` (operation)

Returns the universal reflexible maniplex (in fact, regular polytope) with Schlafli symbol `sym`.

2.1.4 ReflexibleManiplex (for IsList, IsList)

▷ `ReflexibleManiplex(symbol, relations)` (operation)

Returns the reflexible maniplex with the given Schlafli symbol and with the given relations. The formatting of the relations is quite flexible. All of the following work:

Example
```
q := ReflexibleManiplex([4,3,4], "(r0 r1 r2)^3, (r1 r2 r3)^3");
q := ReflexibleManiplex([4,3,4], "(r0 r1 r2)^3 = (r1 r2 r3)^3 = 1");
p := ReflexibleManiplex([infinity], "r0 r1 r0 = r1 r0 r1");
```

If the option set_schlafli is set, then we set the Schlafli symbol to the one given. This may not be the correct Schlafli symbol, since the relations may cause a collapse, so this should only be used if you know that the Schlafli symbol is correct.

2.1.5 ReflexibleManiplex (for IsString)

▷ `ReflexibleManiplex(name)` (operation)

Returns the regular polytope with the given symbolic name. Examples: ReflexibleManiplex("{3,3,3}"); ReflexibleManiplex("{4,3}_3"); If the option set_schlafli is set, then we set the Schlafli symbol to the one given. This may not be the correct Schlafli symbol, since the relations may cause a collapse, so this should only be used if you know that the Schlafli symbol is correct.

2.1.6 Maniplex (for IsGroup)

▷ `Maniplex(G)` (operation)

Returns a maniplex with connection group `G`, where `G` is assumed to be a permutation group on the flags.

2.1.7 Maniplex (for IsReflexibleManiplex, IsGroup)

▷ `Maniplex(M, G)` (operation)

Given a reflexible maniplex `M` and a subgroup `G` of the flag-stabilizer of the base flag of `M`, returns the maniplex `M/G`.

2.1.8 Maniplex (for IsFunction, IsList)

▷ `Maniplex(F, inputs)` (operation)

Constructs a formal polytope, represented by an operation `F` and a list of arguments `inputs`.

2.1.9 Maniplex (for IsPoset)

▷ `Maniplex(P)` (operation)

Returns a maniplex with poset `P`.

2.1.10 IsPolytopal (for IsManiplex)

▷ `IsPolytopal(M)` (property)

Returns: `true` or `false`

Returns whether the maniplex `M` is a polytope. Currently only implemented for reflexible maniplexes.

Chapter 3

Combinatorics and Structure

3.1 Faces

3.1.1 NumberOfIFaces (for IsManiplex, IsInt)

▷ `NumberOfIFaces(M, i)` (operation)

Returns The number of `i`-faces of `M`.

3.1.2 NumberOfVertices (for IsManiplex)

▷ `NumberOfVertices(M)` (attribute)

Returns the number of vertices of `M`.

3.1.3 NumberOfEdges (for IsManiplex)

▷ `NumberOfEdges(M)` (attribute)

Returns the number of edges of `M`.

3.1.4 NumberOfFacets (for IsManiplex)

▷ `NumberOfFacets(M)` (attribute)

Returns the number of facets of `M`.

3.1.5 NumberOfRidges (for IsManiplex)

▷ `NumberOfRidges(M)` (attribute)

Returns the number of ridges ((n-2)-faces) of `M`.

3.1.6 Fvector (for IsManiplex)

▷ `Fvector(M)` (attribute)

Returns the f-vector of *M*.

3.1.7 Section (for IsManiplex, IsInt, IsInt)

▷ `Section(M, j, i)` (operation)

Returns the section F_j / F_i, where F_j is the j-face of the base flag of *M* and F_i is the i-face of the base flag.

3.1.8 Section (for IsManiplex, IsInt, IsInt, IsInt)

▷ `Section(M, j, i, k)` (operation)

Returns the section F_j / F_i, where F_j is the j-face of flag number *k* of *M* and F_i is the i-face of the same flag.

3.1.9 Sections (for IsManiplex, IsInt, IsInt)

▷ `Sections(M, j, i)` (operation)

Returns all sections of type F_j / F_i, where F_j is a j-face and F_i is an incident i-face.

3.1.10 Facets (for IsManiplex)

▷ `Facets(M)` (attribute)

Returns the facet-types of *M* (i.e. the maniplexes corresponding to the facets).

3.1.11 Facet (for IsManiplex, IsInt)

▷ `Facet(M, k)` (operation)

Returns the facet of *M* that contains the flag number *k* (that is, the maniplex corresponding to the facet).

3.1.12 Facet (for IsManiplex)

▷ `Facet(M)` (attribute)

Returns the facet of *M* that contains flag number 1 (that is, the maniplex corresponding to the facet).

3.1.13 VertexFigures (for IsManiplex)

▷ `VertexFigures(M)` (attribute)

Returns the types of vertex-figures of M (i.e. the maniplexes corresponding to the vertex-figures).

3.1.14 VertexFigure (for IsManiplex, IsInt)

▷ `VertexFigure(M, k)` (operation)

Returns the vertex-figures of M that contains flag number k.

3.1.15 VertexFigure (for IsManiplex)

▷ `VertexFigure(M)` (attribute)

Returns the vertex-figures of M that contains the base flag.

3.2 Flatness

3.2.1 IsFlat

▷ `IsFlat(M)` (property)
▷ `IsFlat(M, i, j)` (operation)

Returns: `true` or `false`

In the first form, returns true if every vertex of the maniplex M is incident to every facet. In the second form, returns true if every i-face of the maniplex M is incident to every j-face.

3.3 Schlafli symbol

3.3.1 SchlafliSymbol (for IsManiplex)

▷ `SchlafliSymbol(M)` (attribute)

Returns the Schlafli symbol of the maniplex M. Each entry is either an integer or a set of integers, where entry number i shows the polygons that we obtain as sections of (i+1)-faces over (i-2)-faces.

3.3.2 IsEquivelar (for IsManiplex)

▷ `IsEquivelar(M)` (property)

Returns: the the maniplex M is equivelar; i.e., whether its Schlafli Symbol consists of integers at each position (no lists).

3.4 Basics

3.4.1 Size (for IsManiplex)

▷ `Size(M)` (attribute)

Returns the number of flags of the maniplex *M*. Synonym: `NumberOfFlags`.

3.4.2 RankManiplex (for IsManiplex)

▷ `RankManiplex(M)` (attribute)

Returns the rank of the maniplex *M*.

3.4.3 IsTight (for IsManiplex and IsPolytopal)

▷ `IsTight(P)` (property)

Returns: `true` or `false`

Returns whether the polytope *P* is tight, meaning that it has a Schlafli symbol {k_1, ..., k_{n-1}} and has 2 k_1 ... k_{n-1} flags, which is the minimum possible. This property doesn't make any sense for non-polytopal maniplexes, which aren't constrained by this lower bound.

3.4.4 IsDegenerate (for IsManiplex)

▷ `IsDegenerate(M)` (property)

Returns: `true` or `false`

Returns whether the maniplex *M* has any sections that are digons. We may eventually want to include maniplexes with even smaller sections.

3.4.5 SymmetryTypeGraph (for IsManiplex)

▷ `SymmetryTypeGraph(M)` (attribute)

Returns the Symmetry Type Graph of the maniplex *M*, encoded as a permutation group on Rank(*M*) generators.

3.4.6 NumberOfFlagOrbits (for IsManiplex)

▷ `NumberOfFlagOrbits(M)` (attribute)

Returns the number of orbits of the automorphism group of *M* on its flags.

3.4.7 FlagOrbitRepresentatives (for IsManiplex)

▷ `FlagOrbitRepresentatives(M)` (attribute)

Returns one flag from each orbit under the action of AutomorphismGroup(*M*).

3.4.8 IsReflexible (for IsManiplex)

▷ `IsReflexible(M)` (property)

Returns: Whether the maniplex *M* is reflexible (has one flag orbit).

3.4.9 IsChiral (for IsManiplex)

▷ `IsChiral(M)` (property)

Returns: Whether the maniplex `M` is chrial.

3.4.10 IsRotary (for IsManiplex)

▷ `IsRotary(M)` (property)

Returns: Whether the maniplex `M` is rotary; i.e., whether it is either reflexible or chiral.

3.5 Zigzags and holes

3.5.1 ZigzagLength (for IsManiplex, IsInt)

▷ `ZigzagLength(M, j)` (operation)

Returns: The lengths of `j`-zigzags of the 3-maniplex `M`. This corresponds to the lengths of orbits under r0 (r1 r2)^j.

3.5.2 ZigzagVector (for IsManiplex)

▷ `ZigzagVector(M)` (attribute)

Returns: The lengths of all zigzags of the 3-maniplex `M`. A rank 3 maniplex of type {p, q} has Floor(q/2) distinct zigzag lengths because the j-zigzags are the same as the (q-j)-zigzags.

3.5.3 PetrieLength (for IsManiplex)

▷ `PetrieLength(M)` (attribute)

Returns: The length of the petrie polygons of the maniplex `M`.

3.5.4 HoleLength (for IsManiplex, IsInt)

▷ `HoleLength(M, j)` (operation)

Returns: The lengths of `j`-holes of the 3-maniplex `M`. This corresponds to the lengths of orbits under r0 (r1 r2)^(j-1) r2.

3.5.5 HoleVector (for IsManiplex)

▷ `HoleVector(M)` (attribute)

Returns: The lengths of all zigzags of the 3-maniplex `M`. A rank 3 maniplex of type {p, q} has Floor(q/2) distinct zigzag lengths because the j-zigzags are the same as the (q-j)-zigzags.

Chapter 4

Constructions

4.1 Extensions, amalgamations, and quotients

4.1.1 UniversalPolytope (for IsInt)

▷ `UniversalPolytope(n)` (operation)

Returns the universal polytope of rank `n`.

4.1.2 FlatRegularPolyhedron (for IsInt, IsInt, IsInt, IsInt)

▷ `FlatRegularPolyhedron(p, q, i, j)` (operation)

Returns the flat regular polyhedron with automorphism group [p, q] / (r2 r1 r0 r1 = (r0 r1)^i (r1 r2)^j). This function does not currently validate the inputs to make sure that the output makes sense.

4.1.3 UniversalExtension (for IsManiplex)

▷ `UniversalExtension(M)` (operation)

Returns the universal extension of `M`, i.e. the maniplex with facets isomorphic to `M` that covers all other maniplexes with facets isomorphic to `M`. Currently only defined for reflexible maniplexes.

4.1.4 UniversalExtension (for IsManiplex, IsInt)

▷ `UniversalExtension(M, k)` (operation)

Returns the universal extension of `M` with last entry of Schlafli symbol `k`. Currently only defined for reflexible maniplexes.

4.1.5 TrivialExtension (for IsManiplex)

▷ `TrivialExtension(M)` (operation)

Returns the trivial extension of `M`, also known as {`M/`, 2}.

4.1.6 FlatExtension (for IsManiplex, IsInt)

▷ `FlatExtension(M, k)` (operation)

Returns the flat extension of *M* with last entry of Schlafli symbol *k*. (As defined in "Flat Extensions of Abstract Polytopes".) Currently only defined for reflexible maniplexes.

4.1.7 Amalgamate (for IsManiplex, IsManiplex)

▷ `Amalgamate(M1, M2)` (operation)

Returns the amalgamation of *M1* and *M2*. Implicitly assumes that *M1* and *M2* are compatible. Currently only defined for reflexible maniplexes.

4.1.8 Medial (for IsManiplex)

▷ `Medial(M)` (operation)

Given a 3-maniplex *M*, returns its medial.

4.2 Duality

4.2.1 Dual (for IsManiplex)

▷ `Dual(M)` (attribute)

Returns: The maniplex that is dual to *M*.

4.2.2 IsSelfDual (for IsManiplex)

▷ `IsSelfDual(P)` (property)

Returns: Whether this polytope is isomorphic to its dual.
Also works for IsPoset objects.

4.2.3 Petrial (for IsManiplex)

▷ `Petrial(P)` (attribute)

Returns: The Petrial (Petrie dual) of *P*. Note that this is not necessarily a polytope.

4.2.4 IsSelfPetrial (for IsManiplex)

▷ `IsSelfPetrial(P)` (property)

Returns: Whether this polytope is isomorphic to its Petrial.

4.3 Products

4.3.1 PyramidOver (for IsManiplex)

▷ `PyramidOver(M)` (operation)

Returns the pyramid over M.

4.3.2 Pyramid (for IsInt)

▷ `Pyramid(k)` (operation)

Returns the pyramid over a k-gon.

4.3.3 PrismOver (for IsManiplex)

▷ `PrismOver(M)` (operation)

Returns the prism over M.

4.3.4 Prism (for IsInt)

▷ `Prism(k)` (operation)

Returns the prism over a k-gon.

Chapter 5

Databases

5.1 Regular polyhedra

5.1.1 DegeneratePolyhedra (for IsInt)

▷ DegeneratePolyhedra(*maxsize*) (operation)

Returns all degenerate polyhedra (of type {2, q} and {p, 2}) with up to *maxsize* flags.

5.1.2 FlatRegularPolyhedra (for IsInt)

▷ FlatRegularPolyhedra(*maxsize*) (operation)

Returns all nondegenerate flat regular polyhedra with up to *maxsize* flags. Currently supports a maxsize of 4000 or less.

5.1.3 SmallRegularPolyhedra (for IsInt)

▷ SmallRegularPolyhedra(*maxsize*) (operation)

Returns all regular polyhedra with up to *maxsize* flags. Currently supports a maxsize of 4000 or less. You can also set options "nondegenerate" and "nonflat".

Example

```
L1 := SmallRegularPolyhedra(500);;
L2 := SmallRegularPolyhedra(1000 : nondegenerate);;
L3 := SmallRegularPolyhedra(2000 : nondegenerate, nonflat);;
```

Chapter 6

Families of Polytopes

6.1 Classical Polytopes

6.1.1 Vertex

▷ `Vertex()` (operation)

Returns the universal 0-polytope.

6.1.2 Edge

▷ `Edge()` (operation)

Returns the universal 1-polytope.

6.1.3 Pgon (for IsInt)

▷ `Pgon(p)` (operation)

Returns the p-gon.

6.1.4 Cube (for IsInt)

▷ `Cube(n)` (operation)

Returns the n-cube.

6.1.5 HemiCube (for IsInt)

▷ `HemiCube(n)` (operation)

Returns the n-hemi-cube.

6.1.6 CrossPolytope (for IsInt)

▷ `CrossPolytope(n)` (operation)

Returns the n-cross-polytope.

6.1.7 HemiCrossPolytope (for IsInt)

▷ `HemiCrossPolytope(n)` (operation)

Returns the n-hemi-cross-polytope.

6.1.8 Simplex (for IsInt)

▷ `Simplex(n)` (operation)

Returns the n-simplex.

6.1.9 CubicTiling (for IsInt)

▷ `CubicTiling(n)` (operation)

Returns the rank n+1 polytope; the tiling of E^n by n-cubes.

6.1.10 Dodecahedron

▷ `Dodecahedron()` (operation)

Returns the dodecahedron, {5, 3}.

6.1.11 HemiDodecahedron

▷ `HemiDodecahedron()` (operation)

Returns the hemi-dodecahedron, {5, 3}_5.

6.1.12 Icosahedron

▷ `Icosahedron()` (operation)

Returns the icosahedron, {3, 5}.

6.1.13 HemiIcosahedron

▷ `HemiIcosahedron()` (operation)

Returns the hemi-icosahedron, {3, 5}_5.

6.1.14 24Cell

▷ `24Cell()` (operation)

Returns the 24-cell, {3, 4, 3}.

6.1.15 Hemi24Cell

▷ `Hemi24Cell()` (operation)

Returns the hemi-24-cell, {3, 4, 3}_6.

6.1.16 120Cell

▷ `120Cell()` (operation)

Returns the 120-cell, {5, 3, 3}.

6.1.17 Hemi120Cell

▷ `Hemi120Cell()` (operation)

Returns the hemi-120-cell, {5, 3, 3}_15.

6.1.18 600Cell

▷ `600Cell()` (operation)

Returns the 600-cell, {3, 3, 5}.

6.1.19 Hemi600Cell

▷ `Hemi600Cell()` (operation)

Returns the hemi-600-cell, {3, 3, 5}_15.

6.2 Uniform Polyhedra

6.2.1 TruncatedOctahedron

▷ `TruncatedOctahedron()` (operation)
Returns: maniplex
Constructs the truncated octahedron.

6.2.2 TruncatedCube

▷ `TruncatedCube()` (operation)
Returns: maniplex
Constructs the truncated octahedron.

6.2.3 Icosadodecahedron

▷ `Icosadodecahedron()` (operation)

Returns: maniplex

Constructs the icosadodecahedron.

6.2.4 TruncatedIcosahedron

▷ `TruncatedIcosahedron()` (operation)

Returns: maniplex

Constructs the truncated icosahedron.

6.2.5 SmallRhombicuboctahedron

▷ `SmallRhombicuboctahedron()` (operation)

Returns: maniplex

Constructs the small rhombicuboctahedron.

6.2.6 Pseudorhombicuboctahedron

▷ `Pseudorhombicuboctahedron()` (operation)

Returns: maniplex

Constructs the pseudorhombicuboctahedron.

6.2.7 SnubCube

▷ `SnubCube()` (operation)

Returns: maniplex

Constructs the snub cube.

6.2.8 SmallRhombicosidodecahedron

▷ `SmallRhombicosidodecahedron()` (operation)

Returns: maniplex

Constructs the small rhombicosidodecahedron.

6.2.9 GreatRhombicosidodecahedron

▷ `GreatRhombicosidodecahedron()` (operation)

Returns: maniplex

Constructs the great rhombicosidodecahedron.

6.2.10 SnubDodecahedron

▷ `SnubDodecahedron()` (operation)

Returns: maniplex

Constructs the small snub dodecahedron.

6.2.11 TruncatedDodecahedron

▷ `TruncatedDodecahedron()` (operation)

Returns: maniplex

Constructs the truncated dodecahedron.

6.2.12 GreatRhombicuboctahedron

▷ `GreatRhombicuboctahedron()` (operation)

Returns: maniplex

Constructs the great rhombicuboctahedron.

Chapter 7

Groups

7.1 Groups

7.1.1 AutomorphismGroup (for IsManiplex)

▷ `AutomorphismGroup(M)` (attribute)

Returns the automorphism group of *M*. This group is not guaranteed to be in any particular form.

7.1.2 AutomorphismGroupFpGroup (for IsManiplex)

▷ `AutomorphismGroupFpGroup(M)` (attribute)

Returns the automorphism group of *M* as a finitely presented group.

7.1.3 AutomorphismGroupPermGroup (for IsManiplex)

▷ `AutomorphismGroupPermGroup(M)` (attribute)

Returns the automorphism group of *M* as a permutation group.

7.1.4 AutomorphismGroupOnFlags (for IsManiplex)

▷ `AutomorphismGroupOnFlags(M)` (attribute)

Returns the automorphism group of *M* as a permutation group action on the flags of *M*.

7.1.5 ConnectionGroup (for IsManiplex)

▷ `ConnectionGroup(M)` (attribute)

Returns the connection group of *M* as a permutation group. We may eventually allow other types of connection groups. Synonym: MonodromyGroup

7.1.6 EvenConnectionGroup (for IsManiplex)

▷ `EvenConnectionGroup(M)` (attribute)

Returns the even-word subgroup of the connection group of `M` as a permutation group.

7.1.7 RotationGroup (for IsManiplex)

▷ `RotationGroup(M)` (attribute)

Returns the rotation group of `M`. This group is not guaranteed to be in any particular form.

7.1.8 ChiralityGroup (for IsRotaryManiplex)

▷ `ChiralityGroup(M)` (attribute)

Returns the chirality group of the rotary maniplex `M`. This is the kernel of the group epimorphism from the rotation group of `M` to the rotation group of its maximal reflexible quotient. In particular, the chirality group is trivial if and only if `M` is reflexible.

7.1.9 ExtraRelators (for IsReflexibleManiplex)

▷ `ExtraRelators(M)` (attribute)

For a reflexible maniplex `M`, returns the relators needed to define its automorphism group as a quotient of the string Coxeter group given by its Schlafli symbol. Not particularly robust at the moment.

7.1.10 ExtraRotRelators (for IsRotaryManiplex)

▷ `ExtraRotRelators(M)` (attribute)

For a reflexible maniplex `M`, returns the relators needed to define its rotation group as a quotient of the rotation group of a string Coxeter group given by its Schlafli symbol. Not particularly robust at the moment.

7.1.11 IsStringC (for IsGroup)

▷ `IsStringC(G)` (operation)

For an sggi `G`, returns whether the group is a string C group. It does not check whether `G` is an sggi.

7.1.12 IsStringCPlus (for IsGroup)

▷ `IsStringCPlus(G)` (operation)

For a "string rotation group" `G`, returns whether the group is a string C+ group. It does not check whether `G` is a string rotation group.

Chapter 8

Mixing of Maniplexes

8.1 Mixing of Maniplexes functions

8.1.1 Mix (for IsPermGroup, IsPermGroup)

▷ `Mix(permgroup, permgroup)` (operation)

Returns: `IsGroup` .

Given two (permutation) groups returns the mix of those groups. Note, also works with FPgroups. Here we build the mix of the connection groups of a 3-cube and an edge.

Example
```
gap> g1:=ConnectionGroup(Cube(3));
<permutation group with 3 generators>
gap> g2:=ConnectionGroup(Edge());
Group([ (1,2) ])
gap> Mix(g1,g2);
<permutation group with 3 generators>
```

8.1.2 Mix (for IsFpGroup, IsFpGroup)

▷ `Mix(fpgroup, fpgroup)` (operation)

Returns the Mix of two Finitely Presented groups gp and gq.

8.1.3 Mix (for IsReflexibleManiplex, IsReflexibleManiplex)

▷ `Mix(maniplex, maniplex)` (operation)

Returns: `IsReflexibleManiplex` .

Given maniplexes returns the IsReflexibleManiplex from the mix of their connection groups

8.1.4 CtoL (for IsInt,IsInt,IsInt,IsInt)

▷ `CtoL(int, int, int, int)` (operation)

Returns: `IsInteger` .

CtoL Returns an integer between 1 and N*M associated with the pair [a,b]. LtoC Returns an ordered pair [a,b] associated with the integer between 1 and N*M. a should range between 1 and N, and b should range between 1 and M N is how many columns (x coordinates), M is how many rows (y coordinates) in a matrix Functions are inverses.

8.1.5 FlagMix (for IsManiplex, IsManiplex)

▷ `FlagMix(`*permgroup, permgroup*`)` (operation)

Returns: `IsManiplex` .

Given two (permutation) groups gp, gg this returns the maniplex of the "flag" mix of two maniplexes with connection groups gp and gq.

Chapter 9

Properties

9.1 Orientability

9.1.1 IsOrientable (for IsManiplex)

▷ `IsOrientable(p)` (property)

Returns: `true` or `false`

A polytope is orientable if its flag graph is bipartite. Currently only implemented for regular polytopes.

9.1.2 IsIOrientable (for IsManiplex, IsList)

▷ `IsIOrientable(p, I)` (operation)

For a subset I of {0, ..., n-1}, a polytope if I-orientable if every closed path in its flag graph contains an even number of edges with colors in I. Currently only implemented for regular polytopes.

9.1.3 IsVertexBipartite (for IsManiplex)

▷ `IsVertexBipartite(p)` (property)

Returns: `true` or `false`

A polytope is vertex-bipartite if its 1-skeleton is bipartite. This is equivalent to being I-orientable for I = {0}.

9.1.4 IsFacetBipartite (for IsManiplex)

▷ `IsFacetBipartite(p)` (property)

Returns: `true` or `false`

A polytope is facet-bipartite if the 1-skeleton of its dual is bipartite. This is equivalent to being I-orientable for I = {n-1}.

Chapter 10

Actions

10.1 Faithfulness

10.1.1 IsVertexFaithful (for IsReflexibleManiplex)

▷ `IsVertexFaithful(M)` (property)

Returns: `true` or `false`

Returns whether the reflexible maniplex *M* is vertex-faithful; i.e., whether the action of the automorphism group on the vertices is faithful.

10.1.2 IsFacetFaithful (for IsReflexibleManiplex)

▷ `IsFacetFaithful(M)` (property)

Returns: `true` or `false`

Returns whether the reflexible maniplex *M* is facet-faithful; i.e., whether the action of the automorphism group on the facets is faithful.

10.1.3 MaxVertexFaithfulQuotient (for IsReflexibleManiplex)

▷ `MaxVertexFaithfulQuotient(M)` (operation)

Returns the maximal vertex-faithful reflexible maniplex covered by *M*.

Chapter 11

Posets

I'm in the process of reconciling all of this, but there are going to be a number of ways to `define` a poset:

- As an `IsPosetOfFlags`, where the underlying description is an ordered list of length $n+2$. Each of the $n+2$ list elements is a list of faces, and the assumption is that these are the faces of rank $i-2$, where i is the index in the master list (e.g., `l[1][1]` would usually correspond to the unique -1 face of a polytope – and there won't be an `l[1][2]`). Each face is then a list of the flags incident with that face.

- As an `IsPosetOfIndices`, where the underlying description is a binary relation on a set of indices, which correspond to labels for the elements of the poset.

- If the poset is known to be atomic, then by a description of the faces in terms of the atoms... usually we'll just need the list of the elements of maximal rank, from which all other elements may be obtained.

- As an `IsPosetOfElements`, where the elements could be anything, and we have a known function determining the partial order on the elements.

Usually, we assume that the poset will have a natural rank function on it.

11.1 Poset constructors

11.1.1 PosetFromFaceListOfFlags (for IsList)

▷ `PosetFromFaceListOfFlags(list)` (operation)

Returns: `IsPosetOfFlags`. Note that the function is INTENTIONALLY agnostic about whether it is being given full poset or not.

Given a *list* of lists of faces in increasing rank, where each face is described by the incident flags, gives you a IsPosetOfFlags object back. Note that if you don't include faces or ranks, this function doesn't know about about them!

Here we have a poset using the `IsPosetOfFlags` description for the triangle.

Example

```
gap> poset:=PosetFromFaceListOfFlags([[[]],[[1,2],[3,6],[4,5]],[[1,4],[2,3],[5,6]],[[1,2,3,4,5,6]
A poset using the IsPosetOfFlags representation with 8 faces.
gap> FaceListOfPoset(poset);
[ [ [  ] ], [ [ 1, 2 ], [ 3, 6 ], [ 4, 5 ] ], [ [ 1, 4 ], [ 2, 3 ], [ 5, 6 ] ], [ [ 1, 2, 3, 4, 5
```

11.1.2 PosetFromConnectionGroup (for IsPermGroup)

▷ PosetFromConnectionGroup(*g*) (operation)

Returns: IsPosetOfFlags with IsP1=true.

Given a group, returns a poset with an internal representation as a list of faces ordered by rank, where each face is represented as a list of the flags it contains. Note that this function includes the minimal (empty) face and the maximal face of the maniplex. Note that the i-faces correspond to the $i+1$ item in the list because of how GAP indexes lists.

Example
```
gap> g:=Group([(1,4)(2,3)(5,6),(1,2)(3,6)(4,5)]);
Group([ (1,4)(2,3)(5,6), (1,2)(3,6)(4,5) ])
gap> PosetFromConnectionGroup(g);
A poset using the IsPosetOfFlags representation with 8 faces.
```

11.1.3 PosetFromManiplex (for IsManiplex)

▷ PosetFromManiplex(*mani*) (operation)

Returns: IsPosetOfFlags

Given a maniplex, returns a poset of the maniplex with an internal representation as a list of faces ordered by rank, where each face is represented as a list of the flags it contains. Note that this function does include the minimal (empty) face and the maximal face of the maniplex. Note that the i-faces correspond to the $i+1$ item in the list because of how GAP indexes lists.

Example
```
gap> p:=HemiCube(3);
Regular 3-polytope of type [ 4, 3 ] with 24 flags
gap> PosetFromManiplex(p);
A poset using the IsPosetOfFlags representation with 15 faces.
```

11.1.4 PosetFromPartialOrder (for IsBinaryRelation)

▷ PosetFromPartialOrder(*partialOrder*) (operation)

Returns: IsPosetOfIndices

Given a partial order on a finite set of size n, this function will create a partial order on [1..n].

Example
```
gap> l:=List([[1,1],[1,2],[1,3],[1,4],[2,4],[2,2],[3,3],[4,4]],x->Tuple(x));
gap> r:=BinaryRelationByElements(Domain([1..4]), l);
<general mapping: Domain([ 1 .. 4 ]) -> Domain([ 1 .. 4 ]) >
gap> poset:=PosetFromPartialOrder(r);
A poset using the IsPosetOfIndices representation
gap> h:=HasseDiagramBinaryRelation(PartialOrder(poset));
<general mapping: Domain([ 1 .. 4 ]) -> Domain([ 1 .. 4 ]) >
gap> UnderlyingRelation(h);
Domain([ DirectProductElement( [ 1, 2 ] ), DirectProductElement( [ 1, 3 ] ), DirectProductElement
```

Note that what we've accomplished here is the poset containing the elements 1, 2, 3, 4 with partial order determined by whether the first element divides the second. The essential information about the poset can be obtained from the Hasse diagram.

11.1.5 PosetFromElements (for IsList)

▷ `PosetFromElements(list_of_faces, {func})` (operation)

Returns: `IsPosetOfElements`

This is for gathering elements with a known ordering *func* on two variables into a poset. Note... you should expect to get complete garbage if you send it a list of faces of different types. If your list of faces HasFlagList or HasAtomList, you may omit the function. Also note, the expectation is that *func* behaves similarly to IsSubset, i.e., *func* (x,y)=true means y is less than x in the order. Also worth noting, is that the internal representation of this kind of poset can and does keep both the partial order on the indices, and the list of faces corresponding to those indices, and the binary relation *func* (if the *list_of_faces* elements all have HasFlagList or HasAtomList, this will be the operation `PairCompareFlagsList` or `PairCompareAtomsList`).

Example
```
gap> g:=SymmetricGroup(3);
Sym( [ 1 .. 3 ] )
gap> asg:=AllSubgroups(g);
[ Group(()), Group([ (2,3) ]), Group([ (1,2) ]), Group([ (1,3) ]), Group([ (1,2,3) ]), Group([ (1
gap> poset:=PosetFromElements(asg,IsSubgroup);
A poset using the IsPosetOfIndices representation
gap> HasseDiagramBinaryRelation(PartialOrder(poset));
<general mapping: Domain([ 1 .. 6 ]) -> Domain([ 1 .. 6 ]) >
gap> UnderlyingRelation(last);
Domain([ DirectProductElement( [ 1, 2 ] ), DirectProductElement( [ 1, 3 ] ), DirectProductElement
gap> ElementsList(poset){[2,6]};
[ Group([ (2,3) ]), Group([ (1,2,3), (2,3) ]) ]
```

Here we have an example of how we can store a partially ordered set, and recover information about which objects are incident in the partial order. Another interesting example:

Example
```
gap> poset:=PosetFromManiplex(HemiCube(3));
A poset using the IsPosetOfFlags representation with 15 faces.
gap> rfl:=RankedFaceListOfPoset(poset);
[ [ [  ], -1 ], [ [ 1, 6, 2, 9, 3, 13 ], 0 ], [ [ 4, 14, 16, 23, 11, 21 ], 0 ], [ [ 5, 22, 8, 17,
  [ [ 7, 10, 12, 24, 15, 18 ], 0 ], [ [ 1, 5, 2, 8 ], 1 ], [ [ 3, 11, 13, 21 ], 1 ], [ [ 4, 12, 1
  [ [ 6, 15, 9, 18 ], 1 ], [ [ 10, 19, 24, 20 ], 1 ], [ [ 14, 22, 23, 17 ], 1 ],
  [ [ 1, 5, 6, 22, 15, 14, 12, 4 ], 2 ], [ [ 2, 8, 3, 19, 11, 10, 16, 7 ], 2 ],
  [ [ 9, 18, 13, 24, 21, 20, 23, 17 ], 2 ],
  [ [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24 ], 3
gap> Apply(rfl,PosetElementFromListOfFlags);
gap> pos2:=PosetFromElements(rfl);
A poset using the IsPosetOfIndices representation
gap>  List(FacesList(pos2),x->Rank(x));
[ -1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 2, 3 ]
```

11.1.6 Helper functions for special partial orders

▷ `PairCompareFlagsList(list1, list2)` (operation)

▷ `PairCompareAtomsList(list1, list2)` (operation)

Returns: `true` or `false`

The functions PairCompareFlagsList and PairCompareAtomsList are used in poset construction. Function assumes *list1* and *list2* are of the form `[listOfFlags,i]` where `listOfFlags` is a list of flags in the face and `i` is the rank of the face. Allows comparison of HasFlagList elements. Function assumes *list1* and *list2* are of the form `[listOfAtoms,int]` where `listOfAtoms` is a list of flags in the face and `int` is the rank of the face. Allows comparison of HasAtomList elements.

11.1.7 DualPoset (for IsPoset)

▷ `DualPoset(poset)` (operation)

Returns: dual

Given a *poset*, will construct a poset isomorphic to the dual of *poset*.

Example
```
gap> p:=PosetFromManiplex(Cube(3));; c:=PosetFromManiplex(CrossPolytope(3));;
gap> IsIsomorphicPoset(DualPoset(DualPoset(p)),p);
true
gap> IsIsomorphicPoset(DualPoset(p),c);
true
gap> IsIsomorphicPoset(DualPoset(p),p);
false
```

11.2 Poset attributes

Posets have many properties we might be interested in. Here's a few.

11.2.1 MaximalChains (for IsPoset)

▷ `MaximalChains(poset)` (attribute)

Gives the list of maximal chains in a poset in terms of the elements of the poset. Synonym function is `FlagsList`. Tends to work faster (sometimes significantly) if the poset `HasPartialOrder`.

Synonym is `FlagsList`.

Example
```
gap> poset:=PosetFromManiplex(HemiCube(3));
A poset using the IsPosetOfFlags representation with 15 faces.
gap> rfl:=RankedFaceListOfPoset(poset);;
gap> Apply(rfl,PosetElementFromListOfFlags);
gap> pos2:=PosetFromElements(rfl);
A poset using the IsPosetOfIndices representation
gap> MaximalChains(pos2)[1];
[ An element of a poset., An element of a poset., An element of a poset., An element of a poset.,
  An element of a poset. ]
gap> List(last,Rank);
[ -1, 0, 1, 2, 3 ]
```

11.2.2 RankPoset (for IsPoset)

▷ `RankPoset(poset)` (attribute)

If the poset IsP1, ranks are assumed to run from -1 to n, and function will return n. If `IsP1(poset)=false`, ranks are assumed to run from 1 to n.

11.2.3 ElementsList (for IsPoset)

▷ `ElementsList(`*`poset`*`)` (attribute)

Will recover the list of faces of the poset, format may depend on *type* of representation of `poset`.

- We also have `FacesList` as a synonym for this command.

11.2.4 OrderingFunction (for IsPoset)

▷ `OrderingFunction(`*`poset`*`)` (attribute)

`OrderingFunction` is an attribute of a poset which stores a function for ordering elements.

11.2.5 IsFlaggable (for IsPoset)

▷ `IsFlaggable(`*`poset`*`)` (property)

Returns: `true` or `false`

Checks or creates the value of the attribute `IsFlaggable` for an `IsPoset`. Point here is to see if the structure of the poset is sufficient to determine the flag graph. For IsPosetOfFlags this is another way of saying that the intersection of the faces (thought of as collections of flags) containing a flag is that selfsame flag. (Might be equivalent to prepolytopal... but Gabe was tired and Gordon hasn't bothered to think about it yet.) Now also works with generic poset element types (not just `IsPosetOfFlags`).

11.2.6 IsAtomic (for IsPoset)

▷ `IsAtomic(`*`poset`*`)` (property)

Returns: `true` or `false`

Checks if *`poset`* is atomic. *Not a computed value.*

11.2.7 PartialOrder (for IsPoset)

▷ `PartialOrder(`*`poset`*`)` (attribute)

Returns: `partial order`

HasPartialOrder Checks if *`poset`* has a declared partial order (binary relation). SetPartialOrder assigns a partial order to the *`poset`*. `Note, currently something that is not computed, just declared.`

11.2.8 ListIsP1Poset (for IsList)

▷ `ListIsP1Poset(`*`list`*`)` (operation)

Returns: `true` or `false`

Given *`list`*, comprised of sublists of faces ordered by rank, each face listing the flags on the face, this function will tell you if the list corresponds to a P1 poset or not.

11.2.9 IsP1 (for IsPoset)

▷ IsP1(*poset*) (property)

Returns: `true` or `false`

Determines whether a poset has property P1 from ARP.

Example
```
gap> p:=PosetFromElements(AllSubgroups(AlternatingGroup(4)),IsSubgroup);
A poset using the IsPosetOfIndices representation
gap> IsP1(p);
true
gap> p2:=PosetFromFaceListOfFlags([[[1],[2]],[[1,2]]]);
A poset using the IsPosetOfFlags representation with 3 faces.
gap> IsP1(p2);
false
```

11.2.10 IsP2 (for IsPoset)

▷ IsP2(*poset*) (property)

Returns: `true` or `false`

Determines whether a poset has property P2 from ARP.

Example
```
gap> poset:=PosetFromManiplex(HemiCube(3));
gap> IsP2(poset);
true
```

Another nice example

Example
```
gap> g:=AlternatingGroup(4);;a:=AllSubgroups(g);;poset:=PosetFromElements(a,IsSubgroup);
A poset using the IsPosetOfIndices representation
gap> IsP2(poset);
false
```

11.2.11 IsP3 (for IsPoset)

▷ IsP3(*poset*) (property)

Returns: `true` or `false`

Determines whether a poset is strongly flag connected (property P3' from ARP). May also be called with command `IsStronglyFlagConnected`. If you are not working with a pre-polytope, expect this to take a LONG time.

Helper for IsP3

11.2.12 IsFlagConnected (for IsPoset)

▷ IsFlagConnected(*poset*) (operation)

Determines whether a poset is flag connected.

11.2.13 IsP4 (for IsPoset)

▷ IsP4(*poset*) (property)

Returns: `true` or `false`

Determines whether a poset satisfies the diamond condition. May also be invoked using `IsDiamondCondition`.

11.2.14 IsPolytope (for IsPoset)

▷ IsPolytope(*poset*) (property)

Returns: `true` or `false`

Determines whether a poset is an abstract polytope.

Example
```
gap> poset:=PosetFromManiplex(Cube(3));
A poset using the IsPosetOfFlags representation with 28 faces.
gap> IsPolytope(poset);
true
gap> KnownPropertiesOfObject(poset);
[ "IsP1", "IsP2", "IsP3", "IsP4", "IsPolytope" ]
gap> poset2:=PosetFromElements(AllSubgroups(AlternatingGroup(4)),IsSubgroup);
A poset using the IsPosetOfIndices representation
gap> IsPolytope(poset2);
false
gap> KnownPropertiesOfObject(poset2);
[ "IsP1", "IsP2", "IsPolytope" ]
```

11.2.15 IsPrePolytope (for IsPoset)

▷ IsPrePolytope(*poset*) (property)

Returns: `true` or `false`

Determines whether a poset is an abstract pre-polytope.

11.3 Working with posets

11.3.1 IsIsomorphicPoset (for IsPoset,IsPoset)

▷ IsIsomorphicPoset(*poset1*, *poset2*) (operation)

Returns: `true` or `false`

Determines whether *poset1* and *poset2* are isomorphic by checking to see if their Hasse diagrams are isomorphic.

Example
```
gap>  IsIsomorphicPoset( PosetFromManiplex( PyramidOver( Cube(3) ) ),  PosetFromManiplex( PrismOv
false
gap>  IsIsomorphicPoset( PosetFromManiplex( PyramidOver( Cube(3) ) ), PosetFromManiplex( PyramidC
true
```

11.3.2 PosetIsomorphism (for IsPoset,IsPoset)

▷ `PosetIsomorphism(poset1, poset2)` (operation)

Returns: map on face indices

When *poset1* and *poset2* are isomorphic, will give you a map from the faces of *poset1* to the faces of *poset2*.

11.3.3 FlagsAsListOfFacesFromPoset (for IsPoset)

▷ `FlagsAsListOfFacesFromPoset(poset)` (operation)

Returns: `IsList`

Given a *poset*, this will give you a version of the list of flags in terms of the faces described in the *poset*. Note that the flag list does not include the empty face or the maximal face. In other words, this gives a list of flags where each face is described in terms of its (enumerated) list of incident flags.

11.3.4 AdjacentFlag (for IsPosetOfFlags,IsList,IsInt)

▷ `AdjacentFlag(poset, flag, i)` (operation)

Returns: `flag(s)`

Given a poset, a flag, and a rank, this function will give you the *i*-adjacent flag. Note that adjacencies are listed from ranks 0 to one less than the dimension. You can replace *flag* with the integer corresponding to that flag. Appending `true` to the arguments will give the position of the flag instead of its description from `FlagsAsListOfFacesFromPoset`.

11.3.5 AdjacentFlags (for IsPoset,IsList,IsInt)

▷ `AdjacentFlags(poset, flagaslistoffaces, adjacencyrank)` (operation)

If your poset isn't P4, there may be multiple adjacent maximal chains at a given rank. This function handles that case. May substitute `IsInt` for `flagaslistoffaces` corresponding to position of `flag` in list of maximal chains.

11.3.6 EqualChains (for IsList,IsList)

▷ `EqualChains(flag1, flag2)` (operation)

Determines whether two chains are equal.

11.3.7 ConnectionGeneratorOfPoset (for IsPoset,IsInt)

▷ `ConnectionGeneratorOfPoset(poset, i)` (operation)

Returns: A permutation on the flags.

Given a *poset* and an integer i, this function will give you the associated permutation for the rank i-connection.

11.3.8 ConnectionGroup (for IsPoset)

▷ `ConnectionGroup(`*`poset`*`)` (attribute)

Returns: `IsPermGroup`

Given a *`poset`* that is `IsPrePolytope`, this function will give you the connection group.

11.3.9 AutomorphismGroup (for IsPoset)

▷ `AutomorphismGroup(`*`poset`*`)` (attribute)

Given a *`poset`*, gives the automorphism group of the poset as an action on the maximal chains.

11.3.10 AutomorphismGroupOnElements (for IsPoset)

▷ `AutomorphismGroupOnElements(`*`poset`*`)` (attribute)

Given a *`poset`*, gives the automorphism group of the poset as an action on the elements.

11.3.11 FaceListOfPoset (for IsPoset)

▷ `FaceListOfPoset(`*`poset`*`)` (operation)

Returns: `list`

Gives a list of faces collected into lists ordered by increasing rank.

11.3.12 FacesByRankOfPoset (for IsPoset)

▷ `FacesByRankOfPoset(`*`poset`*`)` (operation)

Returns: `list`

Gives lists of faces ordered by rank. Also sets the rank for each of the faces.

11.3.13 HasseDiagramOfPoset (for IsPoset)

▷ `HasseDiagramOfPoset(`*`poset`*`)` (operation)

Returns: directed graph

11.4 Elements of posets, also known as faces.

11.4.1 RankPosetElement (for IsPosetElement)

▷ `RankPosetElement(`*`posetelement, {face}`*`)` (attribute)

Returns: `true` or `false`

The rank of a poset element. Alternately `RankFace(`*`IsPosetElement`*`)`.

11.4.2 FlagList (for IsPosetElement)

▷ `FlagList(`*`posetelement, {face}`*`)` (attribute)

Returns: `list`

Description of *`posetelement`* n as a list of incident flags (when present).

11.4.3 FromPoset (for IsPosetElement)

▷ `FromPoset(posetelement, {face})` (attribute)

Returns: `poset`

Gives the poset to which the face belongs (when present).

11.4.4 AtomList (for IsPosetElement)

▷ `AtomList(posetelement, {face})` (attribute)

Returns: `list`

Description of *posetelement* n as a list of atoms (when present).

11.5 Element Constructors

11.5.1 PosetElementWithOrder (for IsObject,IsFunction)

▷ `PosetElementWithOrder(obj, func)` (operation)

Returns: `IsFace`

Creates a `face` with *obj* and ordering function `func`.

11.5.2 PosetElementFromListOfFlags (for IsList,IsInt)

▷ `PosetElementFromListOfFlags(list, n)` (operation)

Returns: `IsPosetElement`

This is used to create a face of rank *n* from a *list* of flags of poset. If an IsPoset object is appended to the input will tell the element what poset it belongs to.

11.5.3 PosetElementFromAtomList (for IsList,IsInt)

▷ `PosetElementFromAtomList(list, n)` (operation)

Returns: `IsFace`

Creates a `face` with *list* of atoms at rank *n*. If an IsPoset object is appended to the input will tell the element what poset it belongs to.

11.5.4 PosetElementFromIndex (for IsObject,IsInt)

▷ `PosetElementFromIndex(obj, n)` (operation)

Returns: `IsFace`

Creates a `face` with index *obj* at rank *n*. If an IsPoset object is appended to will tell the element what poset it belongs to.

11.5.5 PosetElementWithPartialOrder (for IsObject, IsBinaryRelation)

▷ `PosetElementWithPartialOrder(obj, order)` (operation)

Returns: `IsFace`

Creates a `face` with index *obj* and BinaryRelation *order* on *obj*.

11.6 Element operations

11.6.1 RankedFaceListOfPoset (for IsPoset)

▷ `RankedFaceListOfPoset(poset)` (operation)

Returns: `list`

Gives a list of [*face*,*rank*] pairs for all the faces of *poset*.

11.6.2 IsSubface (for IsFace,IsFace)

▷ `IsSubface([face1, face1])` (operation)

Returns: `true` or `false`

face1 and *face2* are IsFace or IsPosetElement. Subface will check to make sure *face2* is a subface of *face1*.

11.6.3 AreIncidentElements (for IsObject,IsObject)

▷ `AreIncidentElements(object1, object2)` (operation)

Returns: `true` or `false`

Given two poset elements, will tell you if they are incident.

- Synonym function: `AreIncidentFaces`.

Chapter 12

Products of Posets and Digraphs

This uses the work of Gleason and Hubard.

12.1 Construction methods

Anyone know how to link stuff?

12.1.1 JoinProduct (for IsPoset,IsPoset)

▷ `JoinProduct(poset1, poset2)` (operation)

Returns: poset

Given two posets, this forms the join product. If given two partial orders, returns the join product of the partial orders.

Example
```
gap> p:=PosetFromManiplex(Cube(2));
A poset
gap> rel:=BinaryRelationOnPoints([[1,2],[2]]);
Binary Relation on 2 points
gap> p1:=PosetFromPartialOrder(rel);
A poset using the IsPosetOfIndices representation
gap> j:=JoinProduct(p,p1);
A poset using the IsPosetOfIndices representation
gap> IsIsomorphicPoset(j,PosetFromManiplex(PyramidOver(Cube(2))));
true
```

12.1.2 CartesianProduct (for IsPoset,IsPoset)

▷ `CartesianProduct(polytope1, polytope2)` (operation)

Returns: polytope

Given two polytopes, forms the cartesian product of the polytopes. Should also work if you give it any two posets.

Example
```
gap> p1:=PosetFromManiplex(Edge());
A poset
gap> p2:=PosetFromManiplex(Simplex(2));
A poset
```

```
gap> c:=CartesianProduct(p1,p2);
A poset using the IsPosetOfIndices representation
gap> IsIsomorphicPoset(c,PosetFromManiplex(PrismOver(Simplex(2))));
true
```

12.1.3 DirectSumOfPosets (for IsPoset,IsPoset)

▷ `DirectSumOfPosets(polytope1, polytope2)` (operation)

Returns: polytope

Given two polytopes, forms the direct sum of the polytopes.

Example

```
gap> p1:=PosetFromManiplex(Cube(2));;p2:=PosetFromManiplex(Edge());;
gap> ds:=DirectSumOfPosets(p1,p2);
A poset using the IsPosetOfIndices representation.
gap> IsIsomorphicPoset(ds,PosetFromManiplex(CrossPolytope(3)));
true
```

12.1.4 TopologicalProduct (for IsPoset,IsPoset)

▷ `TopologicalProduct(polytope1, polytope2)` (operation)

Returns: polytope

Given two polytopes, forms the topological product of the polytopes.

Here we demonstrate that the topological product (as expected) when taking the product of a triangle with itself gives us the torus $\{4,4\}_{(3,0)}$ with 72 flags.

Example

```
gap> p:=PosetFromManiplex(Pgon(3));
A poset using the IsPosetOfFlags representation.
gap> tp:=TopologicalProduct(p,p);
A poset using the IsPosetOfIndices representation.
gap> s0 := (5,6);;
gap> s1 := (1,2)(3,5)(4,6);;
gap> s2 := (2,3);;
gap> poly := Group([s0,s1,s2]);;
gap> torus:=PosetFromManiplex(ReflexibleManiplex(poly));
A poset using the IsPosetOfFlags representation.
gap> IsIsomorphicPoset(p,tp);
false
gap> IsIsomorphicPoset(torus,tp);
true
```

12.1.5 Antiprism (for IsPoset)

▷ `Antiprism(polytope)` (operation)

Returns: poset

Given a *polytope* (actually, should work for any poset), will return the antiprism of the *polytope* (poset).

Example

```
gap> p:=PosetFromManiplex(Pgon(3));;
gap> a:=Antiprism(p);;
```

```
gap> IsIsomorphicPoset(a,PosetFromManiplex(CrossPolytope(3)));
true
gap> p:=PosetFromManiplex(Pgon(4));;a:=Antiprism(p);;
gap> d:=DualPoset(p);;ad:=Antiprism(d);;
gap> IsIsomorphicPoset(a,ad);
true
```

Chapter 13

Comparing maniplexes

13.1 Quotients and covers

13.1.1 IsQuotient (for IsManiplex, IsManiplex)

▷ `IsQuotient(M1, M2)` (operation)

Returns whether *M1* is a quotient of *M2*.

13.1.2 IsCover (for IsManiplex, IsManiplex)

▷ `IsCover(M1, M2)` (operation)

Returns whether *M1* is a cover of *M2*.

13.1.3 IsIsomorphicManiplex (for IsManiplex, IsManiplex)

▷ `IsIsomorphicManiplex(M1, M2)` (operation)

Returns whether *M1* is isomorphic to *M2*.

13.1.4 SmallestRegularCover (for IsManiplex)

▷ `SmallestRegularCover(M)` (attribute)

Returns the smallest regular cover of *M*, which is the maniplex whose automorphism group is the connection group of *M*.

13.1.5 QuotientManiplex (for IsReflexibleManiplex, IsString)

▷ `QuotientManiplex(M, relStr)` (operation)

Given a reflexible maniplex *M*, generates the subgroup S of AutomorphismGroup(*M*) given by relStr, and returns the quotient maniplex M / S. For example, QuotientManiplex(CubicTiling(2), "(r0 r1 r2 r1)^5, (r1 r0 r1 r2)^2") returns the toroidal map `{4,4}_{(5,0),(0,2)}`. You can also input this as CubicTiling(2) / "(r0 r1 r2 r1)^5, (r1 r0 r1 r2)^2".

13.1.6 ReflexibleQuotientManiplex (for IsManiplex, IsList)

▷ `ReflexibleQuotientManiplex(M, rels)` (operation)

Given a reflexible maniplex *M*, generates the normal closure N of the subgroup S of AutomorphismGroup(*M*) given by relStr, and returns the quotient maniplex M / N, which will be reflexible. For example, QuotientManiplex(CubicTiling(2), "(r0 r1 r2 r1)^5, (r1 r0 r1 r2)^2") returns the toroidal map {4,4}_{(1,0)}, because the normal closure of the group generated by (r0 r1 r2 r1)^5 and (r1 r0 r1 r2)^2 is the group generated by r0 r1 r2 r1 and r1 r0 r1 r2.

Chapter 14

ramp automatic generated documentation

14.1 ramp automatic generated documentation of methods

14.1.1 UniversalRotationGroup (for IsInt)

▷ `UniversalRotationGroup(n)` (operation)

Returns the rotation subgroup of the universal Coxeter Group of rank n.

14.1.2 UniversalRotationGroup (for IsList)

▷ `UniversalRotationGroup(sym)` (operation)

Returns the rotation subgroup of the Coxeter Group with Schlafli symbol sym.

14.1.3 RotaryManiplex (for IsGroup)

▷ `RotaryManiplex(g)` (operation)

Given a group g (which should be a string rotation group), returns the rotary maniplex with that rotation group, where the privileged generators are those returned by GeneratorsOfGroup(g).

14.1.4 RotaryManiplex (for IsList)

▷ `RotaryManiplex(sym)` (operation)

Returns the universal rotary maniplex (in fact, regular polytope) with Schlafli symbol *sym*.

14.1.5 RotaryManiplex (for IsList, IsList)

▷ `RotaryManiplex(symbol, relations)` (operation)

Returns the rotary maniplex with the given Schlafli symbol and with the given relations. The relations are given by a string that refers to the generators s1, s2, etc. For example:

Example
```
gap> M := RotaryManiplex([4,4], "(s2^-1 s1)^6");;
```

If the option set_schlafli is set, then we set the Schlafli symbol to the one given. This may not be the correct Schlafli symbol, since the relations may cause a collapse, so this should only be used if you know that the Schlafli symbol is correct.

14.1.6 EnantiomorphicForm (for IsRotaryManiplex)

▷ `EnantiomorphicForm(M)` (operation)

The *enantiomorphic form* of a rotary maniplex is the same maniplex, but where we choose the new base flag to be one of the flags that is adjacent to the original base flag. If M is reflexible, then this choice has no effect. Otherwise, if M is chiral, then the enantiomorphic form gives us a different presentation for the rotation group.

14.1.7 Cuboctahedron

▷ `Cuboctahedron()` (operation)

Returns: maniplex

Constructs the cuboctahedron.

14.1.8 TruncatedTetrahedron

▷ `TruncatedTetrahedron()` (operation)

Returns: maniplex

Constructs the truncated tetrahedron.

Chapter 15

Utility functions

15.1 Utility functions

15.1.1 AbstractPolytope

▷ `AbstractPolytope(`*`args`*`)` (function)

Calls `Maniplex(args)` and marks the output as polytopal.

15.1.2 AbstractRegularPolytope

▷ `AbstractRegularPolytope(`*`args`*`)` (function)

Calls `ReflexibleManiplex(args)` and marks the output as polytopal. Also available as `ARP(args)`.

15.1.3 AbstractRotaryPolytope

▷ `AbstractRotaryPolytope(`*`args`*`)` (function)

Calls `RotaryManiplex(args)` and marks the output as polytopal.

15.1.4 TranslatePerm

▷ `TranslatePerm(`*`perm, k`*`)` (function)

Returns a new permutation obtained from *`perm`* by adding k to each moved point.

15.1.5 MultPerm

▷ `MultPerm(`*`perm, multiplier, offset`*`)` (function)

Multiplies together perm, TranslatePerm(perm, offset), TranslatePerm(perm, offset*2), ..., with *`multiplier`* terms, and returns the result.

15.1.6 ParseStringCRels

▷ `ParseStringCRels(rels, g)` (function)

This helper function is used in several maniplex constructors. Given a string *rels* that represents relations in an sggi, and an sggi g, returns a list of elements of g represented by *rels*.

Example
```
g := AutomorphismGroup(CubicTiling(2));;
rels := "(r0 r1 r2 r1)^6";;
newrels := ParseStringCRels(rels, g);
[ (r0*r1*r2*r1)^6 ]
```

For convenience, you may use z1, z2, etc and h1, h2, etc in relations, where zj means r0 (r1 r2)^j and hj means r0 (r1 r2)^j-1 r1. (That is, the former is the word corresponding to j-zigzags of a polyhedron, and the latter corresponds to j-holes.)

15.1.7 ParseRotGpRels

▷ `ParseRotGpRels(rels, g)` (function)

This helper function is used in several maniplex constructors. It is analogous to ParseStringCRels, but for rotation groups instead.

15.1.8 AddOrAppend

▷ `AddOrAppend(L, x)` (function)

Given a list *L* and an object *x*, this calls Append(L, x) if x is a list; otherwise it calls Add(L, x). Note that since strings are internally represented as lists, AddOrAppend(L, "foo") will append the characters 'f', 'o', 'o'.

Index

120Cell, 25
24Cell, 25
600Cell, 25

AbstractPolytope, 52
AbstractRegularPolytope, 52
AbstractRotaryPolytope, 52
AddOrAppend, 53
AdjacentFlag
for IsPosetOfFlags,IsList,IsInt, 41
AdjacentFlags
for IsPoset,IsList,IsInt, 41
Amalgamate
for IsManiplex, IsManiplex, 20
Antiprism
for IsPoset, 46
AreIncidentElements
for IsObject,IsObject, 44
AtomList
for IsPosetElement, 43
AutomorphismGroup
for IsManiplex, 28
for IsPoset, 42
AutomorphismGroupFpGroup
for IsManiplex, 28
AutomorphismGroupOnElements
for IsPoset, 42
AutomorphismGroupOnFlags
for IsManiplex, 28
AutomorphismGroupPermGroup
for IsManiplex, 28

CartesianProduct
for IsPoset,IsPoset, 45
ChiralityGroup
for IsRotaryManiplex, 29
ConnectedComponents
for IsEdgeLabeledGraph, IsList, 10
ConnectionGeneratorOfPoset
for IsPoset,IsInt, 41
ConnectionGroup
for IsManiplex, 28
for IsPoset, 42
CPRGraphFromGroups
for IsGroup,IsGroup, 11
CrossPolytope
for IsInt, 24
CtoL
for IsInt,IsInt,IsInt,IsInt, 30
Cube
for IsInt, 23
CubicTiling
for IsInt, 24
Cuboctahedron, 51

DegeneratePolyhedra
for IsInt, 22
DirectedGraphFromListOfEdges
for IsList,IsList, 5
DirectSumOfPosets
for IsPoset,IsPoset, 46
Dodecahedron, 24
Dual
for IsManiplex, 20
DualPoset
for IsPoset, 37

Edge, 23
EdgeLabeledGraphFromEdges
for IsList, IsList,IsList, 9
EdgeLabelPreservingAutomorphismGroup
for IsEdgeLabeledGraph, 10
ElementsList
for IsPoset, 38
EnantiomorphicForm
for IsRotaryManiplex, 51
EqualChains
for IsList,IsList, 41

EvenConnectionGroup
for IsManiplex, 29
ExtraRelators
for IsReflexibleManiplex, 29
ExtraRotRelators
for IsRotaryManiplex, 29

FaceListOfPoset
for IsPoset, 42
FacesByRankOfPoset
for IsPoset, 42
Facet
for IsManiplex, 15
for IsManiplex, IsInt, 15
Facets
for IsManiplex, 15
FlagGraph
for IsGroup, 9
FlagGraphWithLabels
for IsGroup, 6
FlagList
for IsPosetElement, 42
FlagMix
for IsManiplex, IsManiplex, 31
FlagOrbitRepresentatives
for IsManiplex, 17
FlagsAsListOfFacesFromPoset
for IsPoset, 41
FlatExtension
for IsManiplex, IsInt, 20
FlatRegularPolyhedra
for IsInt, 22
FlatRegularPolyhedron
for IsInt, IsInt, IsInt, IsInt, 19
FromPoset
for IsPosetElement, 43
Fvector
for IsManiplex, 15

GraphFromListOfEdges
for IsList,IsList, 5
GreatRhombicosidodecahedron, 26
GreatRhombicuboctahedron, 27

Hasse
for IsManiplex, 8
HasseDiagramOfPoset
for IsPoset, 42
Hemi120Cell, 25
Hemi24Cell, 25
Hemi600Cell, 25
HemiCrossPolytope
for IsInt, 24
HemiCube
for IsInt, 23
HemiDodecahedron, 24
HemiIcosahedron, 24
HoleLength
for IsManiplex, IsInt, 18
HoleVector
for IsManiplex, 18

Icosadodecahedron, 26
Icosahedron, 24
IsAtomic
for IsPoset, 38
IsChiral
for IsManiplex, 18
IsCover
for IsManiplex, IsManiplex, 48
IsDegenerate
for IsManiplex, 17
IsEquivelar
for IsManiplex, 16
IsFacetBipartite
for IsManiplex, 32
IsFacetFaithful
for IsReflexibleManiplex, 33
IsFlagConnected
for IsPoset, 39
IsFlaggable
for IsPoset, 38
IsFlat
for IsManiplex, 16
for IsManiplex, IsInt, IsInt, 16
IsIOrientable
for IsManiplex, IsList, 32
IsIsomorphicManiplex
for IsManiplex, IsManiplex, 48
IsIsomorphicPoset
for IsPoset,IsPoset, 40
IsOrientable
for IsManiplex, 32
IsP1

for IsPoset, 39
IsP2
for IsPoset, 39
IsP3
for IsPoset, 39
IsP4
for IsPoset, 40
IsPolytopal
for IsManiplex, 13
IsPolytope
for IsPoset, 40
IsPrePolytope
for IsPoset, 40
IsQuotient
for IsManiplex, IsManiplex, 48
IsReflexible
for IsManiplex, 17
IsRotary
for IsManiplex, 18
IsSelfDual
for IsManiplex, 20
IsSelfPetrial
for IsManiplex, 20
IsStringC
for IsGroup, 29
IsStringCPlus
for IsGroup, 29
IsSubface
for IsFace,IsFace, 44
IsTight
for IsManiplex and IsPolytopal, 17
IsVertexBipartite
for IsManiplex, 32
IsVertexFaithful
for IsReflexibleManiplex, 33

JoinProduct
for IsPoset,IsPoset, 45

LayerGraph
for IsGroup, IsInt, IsInt, 7
License, 2
ListIsP1Poset
for IsList, 38

Maniplex
for IsFunction, IsList, 13
for IsGroup, 13
for IsPoset, 13
for IsReflexibleManiplex, IsGroup, 13
MaximalChains
for IsPoset, 37
MaxVertexFaithfulQuotient
for IsReflexibleManiplex, 33
Medial
for IsManiplex, 20
Mix
for IsFpGroup, IsFpGroup, 30
for IsPermGroup, IsPermGroup, 30
for IsReflexibleManiplex, IsReflexibleManiplex, 30
MultPerm, 52

NumberOfEdges
for IsManiplex, 14
NumberOfFacets
for IsManiplex, 14
NumberOfFlagOrbits
for IsManiplex, 17
NumberOfIFaces
for IsManiplex, IsInt, 14
NumberOfRidges
for IsManiplex, 14
NumberOfVertices
for IsManiplex, 14

OrderingFunction
for IsPoset, 38

PairCompareAtomsList
for IsList,IsList, 36
PairCompareFlagsList
for IsList,IsList, 36
ParseRotGpRels, 53
ParseStringCRels, 53
PartialOrder
for IsPoset, 38
Petrial
for IsManiplex, 20
PetrieLength
for IsManiplex, 18
Pgon
for IsInt, 23
PosetElementFromAtomList

for IsList,IsInt, 43
PosetElementFromIndex
for IsObject,IsInt, 43
PosetElementFromListOfFlags
for IsList,IsInt, 43
PosetElementWithOrder
for IsObject,IsFunction, 43
PosetElementWithPartialOrder
for IsObject, IsBinaryRelation, 43
PosetFromConnectionGroup
for IsPermGroup, 35
PosetFromElements
for IsList, 36
PosetFromFaceListOfFlags
for IsList, 34
PosetFromManiplex
for IsManiplex, 35
PosetFromPartialOrder
for IsBinaryRelation, 35
PosetIsomorphism
for IsPoset,IsPoset, 41
PRGraph
for IsGroup, 10
Prism
for IsInt, 21
PrismOver
for IsManiplex, 21
Pseudorhombicuboctahedron, 26
Pyramid
for IsInt, 21
PyramidOver
for IsManiplex, 21

QuotientByLabel
for IsObject,IsList, IsList, IsList, 9
QuotientManiplex
for IsReflexibleManiplex, IsString, 48

RankedFaceListOfPoset
for IsPoset, 44
RankManiplex
for IsManiplex, 17
RankPoset
for IsPoset, 37
RankPosetElement
for IsPosetElement, 42
ReflexibleManiplex
for IsGroup, 12
for IsList, 12
for IsList, IsList, 12
for IsString, 13
ReflexibleQuotientManiplex
for IsManiplex, IsList, 49
RotaryManiplex
for IsGroup, 50
for IsList, 50
for IsList, IsList, 50
RotationGroup
for IsManiplex, 29

SchlafliSymbol
for IsManiplex, 16
Section
for IsManiplex, IsInt, IsInt, 15
for IsManiplex, IsInt, IsInt, IsInt, 15
Sections
for IsManiplex, IsInt, IsInt, 15
Simple
for IsEdgeLabeledGraph, 10
Simplex
for IsInt, 24
Size
for IsManiplex, 17
Skeleton
for IsManiplex, 8
SmallestRegularCover
for IsManiplex, 48
SmallRegularPolyhedra
for IsInt, 22
SmallRhombicosidodecahedron, 26
SmallRhombicuboctahedron, 26
SnubCube, 26
SnubDodecahedron, 26
SymmetryTypeGraph
for IsManiplex, 17

TopologicalProduct
for IsPoset,IsPoset, 46
TranslatePerm, 52
TrivialExtension
for IsManiplex, 19
TruncatedCube, 25
TruncatedDodecahedron, 27
TruncatedIcosahedron, 26

TruncatedOctahedron, 25
TruncatedTetrahedron, 51

UniversalExtension
  for IsManiplex, 19
  for IsManiplex, IsInt, 19
UniversalPolytope
  for IsInt, 19
UniversalRotationGroup
  for IsInt, 50
  for IsList, 50
UniversalSggi
  for IsInt, 12
  for IsList, 12
UnlabeledFlagGraph
  for IsGroup, 5
UnlabeledSimpleGraph
  for IsEdgeLabeledGraph, 10

Vertex, 23
VertexFigure
  for IsManiplex, 16
  for IsManiplex, IsInt, 16
VertexFigures
  for IsManiplex, 16

ZigzagLength
  for IsManiplex, IsInt, 18
ZigzagVector
  for IsManiplex, 18