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

Contents

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 or polytope 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 or polytope 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 or polytope 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, or, polytope*)   (operation)

   Returns: IsGraph. Note this returns an undirected graph.

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

1.1.7 Skeleton (for IsPolytope)

▷ Skeleton(*arg*)   (operation)

   Here we build the skeleton of the dodecahedron.

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

1.1.8 Hasse (for IsPolytope)

▷ 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 ] )
```

Chapter 2

Constructions

2.1 Extensions, amalgamations, and quotients

2.1.1 UniversalPolytope (for IsInt)

▷ UniversalPolytope(*n*)   (operation)

Returns the universal polytope of rank *n*.

2.1.2 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.

2.1.3 QuotientPolytope (for IsManiplex, IsList)

▷ QuotientPolytope(*M, rels*)   (operation)

Returns the quotient of *M* by *rels*, which may be given as either a list of Tietze words, such as [[1,2,1,0,1,2,1,0]] or as a string like "(r0 r1 r2 r1)^2, (r0 r1 r2)^4".

2.1.4 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.

2.1.5 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.

2.1.6 TrivialExtension (for IsManiplex)

▷ TrivialExtension(*M*) (operation)

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

2.1.7 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.

2.1.8 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.

2.1.9 Medial (for IsManiplex)

▷ Medial(*M*) (operation)

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

2.2 Duality

2.2.1 Dual (for IsManiplex)

▷ Dual(*M*) (attribute)

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

2.2.2 IsSelfDual (for IsManiplex)

▷ IsSelfDual(*P*) (property)

Returns: Whether this polytope is isomorphic to its dual.

2.2.3 Petrial (for IsManiplex)

▷ Petrial(*P*) (attribute)

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

2.2.4 IsSelfPetrial (for IsManiplex)

▷ IsSelfPetrial(*P*) (property)

Returns: Whether this polytope is isomorphic to its Petrial.

2.3 Products

2.3.1 PyramidOver (for IsManiplex)

▷ PyramidOver(*M*) (operation)

Returns the pyramid over *M*. Currently only works for finite maniplexes.

2.3.2 PrismOver (for IsManiplex)

▷ PrismOver(*M*) (operation)

Returns the prism over *M*. Currently only works for finite maniplexes.

Chapter 3

Databases

3.1 Regular polyhedra

3.1.1 DegeneratePolyhedra (for IsInt)

▷ DegeneratePolyhedra(*maxsize*)   (operation)

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

3.1.2 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.

3.1.3 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 4

Combinatorics and Structure

4.1 Faces

4.1.1 NumberOfIFaces (for IsManiplex, IsInt)

▷ NumberOfIFaces(*M, i*)   (operation)

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

4.1.2 NumberOfVertices (for IsManiplex)

▷ NumberOfVertices(*M*)   (attribute)

    Returns the number of vertices of *M*.

4.1.3 NumberOfEdges (for IsManiplex)

▷ NumberOfEdges(*M*)   (attribute)

    Returns the number of edges of *M*.

4.1.4 NumberOfFacets (for IsManiplex)

▷ NumberOfFacets(*M*)   (attribute)

    Returns the number of facets of *M*.

4.1.5 NumberOfRidges (for IsManiplex)

▷ NumberOfRidges(*M*)   (attribute)

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

4.1.6 Fvector (for IsManiplex)

▷ Fvector(*M*) (attribute)

Returns the f-vector of *M*.

4.1.7 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.

4.1.8 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.

4.1.9 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.

4.1.10 Facets (for IsManiplex)

▷ Facets(*M*) (attribute)

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

4.1.11 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).

4.1.12 Facet (for IsManiplex)

▷ Facet(*M*) (attribute)

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

4.1.13 VertexFigures (for IsManiplex)

▷ VertexFigures(*M*)   (attribute)

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

4.1.14 VertexFigure (for IsManiplex, IsInt)

▷ VertexFigure(*M, k*)   (operation)

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

4.1.15 VertexFigure (for IsManiplex)

▷ VertexFigure(*M*)   (attribute)

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

4.2 Schlafli symbol

4.2.1 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.

4.2.2 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).

4.3 Basics

4.3.1 Size (for IsManiplex)

▷ Size(*M*)   (attribute)

Returns the number of flags of the maniplex *M*.

4.3.2 RankManiplex (for IsManiplex)

▷ RankManiplex(*M*)   (attribute)

Returns the rank of the maniplex *M*.

4.3.3 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.

4.3.4 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.

4.3.5 SymmetryTypeGraph (for IsManiplex)

▷ SymmetryTypeGraph(*M*) (attribute)

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

4.3.6 NumberOfFlagOrbits (for IsManiplex)

▷ NumberOfFlagOrbits(*M*) (attribute)

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

4.3.7 IsReflexible (for IsManiplex)

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

4.3.8 IsChiral (for IsManiplex)

▷ IsChiral(*M*) (property)
    Returns: Whether the maniplex *M* is chrial.

4.3.9 IsRotary (for IsManiplex)

▷ IsRotary(*M*) (property)
    Returns: Whether the maniplex *M* is rotary; i.e., whether it is either reflexible or chiral.

4.3.10 Description (for IsManiplex)

▷ Description(*M*) (attribute)

    Returns a short name for the maniplex *M*, if one is available. For example, Description(Simplex(3)) = "3-simplex".

4.4   Zigzags and holes

4.4.1   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.

4.4.2   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.

4.4.3   PetrieLength (for IsManiplex)

▷ PetrieLength(*M*)   (attribute)

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

4.4.4   HoleLength (for IsReflexibleManiplex)

▷ HoleLength(*M*)   (attribute)

Chapter 5

Families of Polytopes

5.1 Classical Polytopes

5.1.1 Vertex

▷ Vertex()   (operation)

5.1.2 Edge

▷ Edge()   (operation)

5.1.3 Pgon (for IsInt)

▷ Pgon(*p*)   (operation)

5.1.4 Cube (for IsInt)

▷ Cube(*n*)   (operation)

5.1.5 HemiCube (for IsInt)

▷ HemiCube(*n*)   (operation)

5.1.6 CrossPolytope (for IsInt)

▷ CrossPolytope(*n*)   (operation)

5.1.7 HemiCrossPolytope (for IsInt)

▷ HemiCrossPolytope(*n*) (operation)

5.1.8 Simplex (for IsInt)

▷ Simplex(*n*) (operation)

5.1.9 CubicTiling (for IsInt)

▷ CubicTiling(*n*) (operation)

5.1.10 Dodecahedron

▷ Dodecahedron() (operation)

5.1.11 HemiDodecahedron

▷ HemiDodecahedron() (operation)

5.1.12 Icosahedron

▷ Icosahedron() (operation)

5.1.13 HemiIcosahedron

▷ HemiIcosahedron() (operation)

5.1.14 24Cell

▷ 24Cell() (operation)

5.1.15 Hemi24Cell

▷ Hemi24Cell() (operation)

5.1.16 120Cell

▷ 120Cell() (operation)

5.1.17 Hemi120Cell

▷ Hemi120Cell() (operation)

5.1.18 600Cell

▷ 600Cell() (operation)

5.1.19 Hemi600Cell

▷ Hemi600Cell() (operation)

Chapter 6

Groups

6.1 Groups

6.1.1 AutomorphismGroup (for IsManiplex)

▷ AutomorphismGroup(*M*)   (attribute)

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

6.1.2 AutomorphismGroupFpGroup (for IsManiplex)

▷ AutomorphismGroupFpGroup(*M*)   (attribute)

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

6.1.3 AutomorphismGroupPermGroup (for IsManiplex)

▷ AutomorphismGroupPermGroup(*M*)   (attribute)

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

6.1.4 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.

6.1.5 EvenConnectionGroup (for IsManiplex)

▷ EvenConnectionGroup(*M*)   (attribute)

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

6.1.6 RotationGroup (for IsManiplex)

▷ RotationGroup(*M*) (attribute)

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

6.1.7 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.

6.1.8 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.

6.1.9 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.

6.1.10 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.

Chapter 7

Properties

7.1 Orientability

7.1.1 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.

7.1.2 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.

7.1.3 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}.

7.1.4 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 8

Basics

8.1 Constructors

8.1.1 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.

8.1.2 ReflexibleManiplex (for IsGroup)

▷ ReflexibleManiplex(*g*)   (operation)

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

8.1.3 ReflexibleManiplex (for IsList, IsList)

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

Returns an abstract regular polytope 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.

8.1.4 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.

8.1.5 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.

8.1.6 IsPolytopal (for IsManiplex)

▷ IsPolytopal(*M*) (property)

Returns: true or false

Returns whether the maniplex *M* is a polytope.

Chapter 9

Actions

9.1 Faithfulness

9.1.1 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.

9.1.2 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.

9.1.3 MaxVertexFaithfulQuotient (for IsReflexibleManiplex)

▷ MaxVertexFaithfulQuotient(*M*)   (operation)

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

Chapter 10

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.

10.1 Poset constructors

10.1.1 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
```

10.1.2 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.
```

10.1.3 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.
```

10.1.4 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.

10.1.5   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 ]
```

10.1.6   PairCompareFlagsList (for IsList,IsList)

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

  Returns: true or false

  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.

10.1.7 PairCompareAtomsList (for IsList,IsList)

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

Returns: `true` or `false`

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.

10.2 Poset attributes

10.2.1 ElementsList (for IsPoset)

▷ ElementsList(*poset*) (attribute)

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

10.2.2 OrderingFunction (for IsPoset)

▷ OrderingFunction(*poset*) (attribute)

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

10.2.3 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.

10.2.4 IsFlaggable (for IsPoset)

▷ IsFlaggable(*poset*) (property)

Returns: `list` of flags as lists of poset elements.

Gives the list of maximal chains in a poset in terms of the elements of the poset. Synonym function 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 ]
```

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`).

10.2.5   IsAtomic (for IsPoset)

▷ IsAtomic(*poset*) (property)

Returns: `true` or `false`

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

10.2.6   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.`

10.2.7   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.

10.2.8   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
```

10.2.9   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
```

10.2.10   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. In fact, this is currently pretty slow in general. Being strongly flag connected (or in the case of posets that are P1 and P2, strongly connected) is a combinatorial explosion scenario.

Helper for IsP3

10.2.11   IsFlagConnected (for IsPoset)

▷ IsFlagConnected(*arg*)   (operation)

10.2.12   IsP4 (for IsPoset)

▷ IsP4(*poset*)   (property)

Returns: true or false

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

10.2.13   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" ]
```

10.3   Working with posets

10.3.1   IsPrePolytope (for IsPoset)

▷ IsPrePolytope(*poset*)   (property)
   Returns: `true` or `false`
   Determines whether a poset is an abstract pre-polytope.

10.3.2   AreIncidentElements (for IsObject,IsObject)

▷ AreIncidentElements(*object1, object2*)   (operation)
   Returns: `true` or `false`
   Given two poset elements, will tell you if they are incident.

10.3.3   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.

10.3.4   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 adja-
cencies 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`.

10.3.5   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.

10.3.6   EqualChains (for IsList,IsList)

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

   Determines whether two chains are equal.

10.3.7 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.

10.3.8 ConnectionGroup (for IsPoset)

▷ ConnectionGroup(*poset*) (attribute)

Returns: IsPermGroup

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

10.3.9 AutomorphismGroup (for IsPoset)

▷ AutomorphismGroup(*poset*) (attribute)

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

10.3.10 FaceListOfPoset (for IsPoset)

▷ FaceListOfPoset(*poset*) (operation)

Returns: list

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

10.3.11 FacesByRankOfPoset (for IsPoset)

▷ FacesByRankOfPoset(*poset*) (operation)

Returns: list

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

10.4 Elements of posets, also known as faces.

10.4.1 RankPosetElement (for IsPosetElement)

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

Returns: true or false

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

10.4.2 FlagList (for IsPosetElement)

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

Returns: list

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

10.4.3   FromPoset (for IsPosetElement)

▷ FromPoset(*posetelement, {face}*)   (attribute)
  Returns: poset
  Gives the poset to which the face belongs (when present).

10.4.4   AtomList (for IsPosetElement)

▷ AtomList(*posetelement, {face}*)   (attribute)
  Returns: list
  Description of *posetelement* n as a list of atoms (when present).

10.4.5   PosetElementWithOrder (for IsObject,IsFunction)

▷ PosetElementWithOrder(*obj, func*)   (operation)
  Returns: IsFace
  Creates a face with *obj* and ordering function func.

10.4.6   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.

10.4.7   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.

10.4.8   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.

10.4.9   RankedFaceListOfPoset (for IsPoset)

▷ RankedFaceListOfPoset(*poset*)   (operation)
  Returns: list
  Gives a list of [*face*,*rank*] pairs for all the faces of *poset*.

10.4.10 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*.

Chapter 11

Comparing maniplexes

11.1 Quotients and covers

11.1.1 IsQuotientOf (for IsManiplex, IsManiplex)

▷ IsQuotientOf(*M1, M2*)   (operation)

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

11.1.2 IsCoverOf (for IsManiplex, IsManiplex)

▷ IsCoverOf(*M1, M2*)   (operation)

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

11.1.3 IsIsomorphicTo (for IsManiplex, IsManiplex)

▷ IsIsomorphicTo(*M1, M2*)   (operation)

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

11.1.4 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*.

Chapter 12

ramp automatic generated documentation

12.1 ramp automatic generated documentation of methods

12.1.1 UniversalRotationGroup (for IsInt)

▷ UniversalRotationGroup(*n*) (operation)

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

12.1.2 UniversalRotationGroup (for IsList)

▷ UniversalRotationGroup(*sym*) (operation)

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

12.1.3 RotaryManiplex (for IsGroup)

▷ RotaryManiplex(*arg*) (operation)

12.1.4 RotaryManiplex (for IsList)

▷ RotaryManiplex(*arg*) (operation)

12.1.5 RotaryManiplex (for IsList, IsList)

▷ RotaryManiplex(*arg1, arg2*) (operation)

12.1.6 EnantiomorphicForm (for IsRotaryManiplex)

▷ EnantiomorphicForm(*arg*) (operation)

Index