simpcomp

A GAP toolbox for simplicial complexes

Version 1.1.9

November 2009

Felix Effenberger
Jonathan Spreer

Felix Effenberger — Email: effenberger@mathematik.uni-stuttgart.de

Jonathan Spreer — Email: spreer@mathematik.uni-stuttgart.de

Address: University of Stuttgart
Department of Mathematics
Institute of Geometry and Topology
D-70550 Stuttgart, Germany

Abstract

simpcomp is a GAP package for working with simplicial complexes. It allows the computation of many properties of simplicial complexes (such as the f-, g- and h-vectors, the face lattice, the automorphism group, (co-)homology with explicit basis computation, intersection form, etc.) and provides the user with functions to compute new complexes from old (simplex links and stars, connected sums, cartesian products, handle additions, bistellar flips, etc.). Furthermore, it comes with an extensive library of known triangulations of manifolds and provides the user with the possibility to create own complex libraries.
simpcomp caches computed properties of a simplicial complex, thus avoiding unnecessary computations, internally handles the vertex labeling of the complexes and insures the consistency of a simplicial complex throughout all operations.
simpcomp relies on the GAP package homology [DHSW04] for its homology computation, but also provides the user with an own (co-)homology algorithm in case the packacke homology is not available. For automorphism group computation the GAP package GRAPE [Soi06] is used, which in turn uses the program `nauty` by Brendan McKay [McK84]. An internal automorphism group calculation algorithm in used as fallback if the GRAPE package is not available.

Copyright

Acknowledgements

A few functions of simpcomp are based on code from other authors. The bistellar flips implementation is based upon work of Frank Lutz [Lut03]. Some functions were carried over from the Homology package by Dumas et al. [DHSW04] – these functions are marked in the documentation and the source code. The internal (co)homology algorithms were implemented by Armin Weiss.

Most of the complexes in the simplicial complex library are taken from the ”Manifold Page” by Frank Lutz [Lut].

The first author acknowledges support by the Deutsche Forschungsgemeinschaft (DFG). This work was carried out as part of the DFG project ”Ku 1203/5-2”.

Contents

Chapter 1

Introduction

simpcomp is a GAP package that provides the user with functions to do calculations and constructions with simplicial complexes (see abstract). It builds on top of the GAP packages homology [DHSW04] by J.-G. Dumas et al. and GRAPE [Soi06] by L. Soicher.

Most parts of this manual can be accessed directly from within GAP using its internal help system.

1.1 Why simpcomp

The origin of simpcomp was a collection of scripts of the two authors that provide basic and often-needed functions and operations for working with simplicial complexes.

simpcomp is written entirely in the GAP scripting language, thus giving the user the possibility to see the behind the scenes and to customize or alter simpcomp functions if needed.

The main benefit when working with simpcomp over implementing the needed functions from scratch is the fact that simpcomp takes care of the error-prone vertex labeling of a complex and the fact that it transparently caches properties already calculated, thus preventing unnecessary double calculations of the same property. It also provides the user with functions to save and load the simplicial complexes to and from files and ships with an extensive library of known triangulations of manifolds and pseudomanifolds. This allows the user to work with many different known triangulations without having to construct them first.

In contrast to a fully fledged software package like polymake [GJ00] providing the most efficient algorithms for each task in form of a heterogeneous package (where algorithms are implemented in various languages), the primary goal when developing simpcomp was not efficieny (this is already limited by the GAP scripting language), but rather ease of use and ease of extensibility by the user in the GAP language with all its mathematical and algebraic capabilities.

Extending simpcomp is possible directly from within GAP, without having to compile anything, see chapter 12.

1.2 How to save time reading this document

The core component in simpcomp is the newly defined object type `SCSimplicialComplex`. When working with this package it is important to understand how objects of this type can be created, accessed and modified. The reader is therefore advised to first skim over chapter 3.

The impatient reader may then directly skip to chapter 11 to see simpcomp in action.

The next advised step is to have a look at the functions for creating object of type SCSimplicialComplex, see the first section of chapter 4.

The rest of chapter 4 contains most of the functions that simpcomp provides, except for the functions related to (co-)homology, bistellar flips and the simplicial complex library that are described in the chapters 5, 6 and 8.

1.3 Organization of this document

This manual accompanying simpcomp is divided into eleven chapters.

- The second chapter provides a short introduction into the theory of simplicial complexes and PL-topology.
- The third chapter is devoted to the description of the GAP object type SCSimplicialComplex that is defined by simpcomp.
- In the fourth chapter functions for working with simplicial complexes that are provided by simpcomp are described.
- Chapter five describes the homology- and cohomology-related functions of simpcomp.
- Chapter six contains a description of the functions related to bistellar flips provided by simpcomp.
- In chapter seven the simplicial complex library and the input output functionality that simpcomp provides is described in detail.
- Chapter eight contains descriptions of functions not fitting in the other chapters, such as the error handling and the email notification system of simpcomp.
- Chapter nine contains the transcript of a demo session with simpcomp showing some of the constructions and calculations with simplicial complexes that can also be used as a first overview of things possible with this package.
- Chapter ten contains a transcript of a demo session with simpcomp, showing some of its functionality.
- Finally, chapter eleven focuses on the description of the internal structure of the simpcomp and deals with aspects of extending the functionality of the package.

1.4 How to assure simpcomp works correctly

As with all software, it is important to test whether simpcomp functions correctly on your system after installing it. GAP has an internal testing mechanism and simpcomp ships with a short testing file that does some sample computations and verifies that the results are correct.

To test the functionality of simpcomp you can run the function SCRunTest (9.3.1) from the GAP console:

Example

```
gap> SCRunTest();
+ test simpcomp package, version 1.1.9
+ GAP4stones: 69988
true
gap>
```

SCRunTest (9.3.1) should return true, otherwise the correct functionality of simpcomp cannot be guaranteed.

1.5 How to cite simpcomp

If you would like to cite simpcomp using BibTeX, you can use the following BibTeX entry for the current simpcomp version:

```
@manual{simpcomp,
 author = "Felix Effenberger and Jonathan Spreer",
 title  = "{\tt simpcomp} - a {\tt GAP} toolkit for simplicial complexes,
           {V}ersion 1.1.9}",
 year   = "2009",
 url    = "\url{http://www.igt.uni-stuttgart.de/LstDiffgeo/simpcomp}",
}
```

If you are not using BibTeX, you can use the following entry inside the bibliography environment of LaTeX.

```
\bibitem{simpcomp}
F.~Effenberger and J.~Spreer,
\emph{{\tt simpcomp} -- a {\tt GAP} toolkit for simplicial complexes},
Version 1.1.9,
2009,
\url{http://www.igt.uni-stuttgart.de/LstDiffgeo/simpcomp}.
```

Chapter 2

Theoretical foundations

The purpose of this chapter is to recall some basic definitions regarding polytopes, triangulations and PL topology. The expert in this field may well skip to the next chapter.

For a more detailed look the authors recommend the books [Hud69], [RS72] on PL-topology and [Zie95], [Grü03] on the theory of polytopes.

An overview of the more recent developments in the field of combinatorial topology can be found in [Lut05] and [Dat07].

2.1 Polytopes and polytopal complexes

A convex *d-polytope* is the convex hull of n points $p_i \in E^d$ in the d-dimensional euclidean space:

$$P = \text{conv}\{v_1, \ldots, v_n\} \subset E^d,$$

where the $v_1, \ldots, v_n$ do not lie in a hyperplane of E^d.

From now on when talking about polytopes in this document always convex polytopes are meant unless explicitly stated otherwise.

For any hyperplane $h \subset E^d$, $P \cap h$ is called a *k-face* of P if $\dim(P \cap h) = k$. The 0-faces are called *vertices*, the 1-faces *edges* and the $(d-1)$-faces are called *facets* of P.

A polytope P is called *regular*, if all its $(d-1)$-faces are congruent regular $(d-1)$-polytopes. A regular 1-polytope is a regular n-gon.

Figure 1 below shows the only five regular convex 3-polytopes (also known as *platonic solids*).

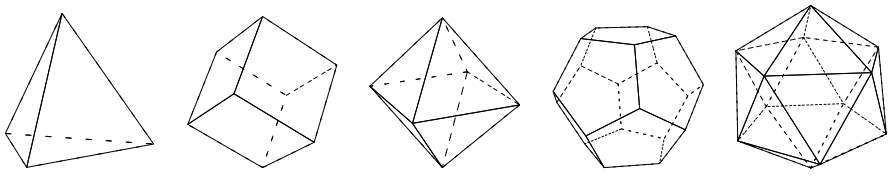

Figure 1. The *platonic solids* as the five regular convex 3-polytopes.

The set of all k-faces of P is called the *k-skeleton* of P, written as $\text{skel}_k(P)$

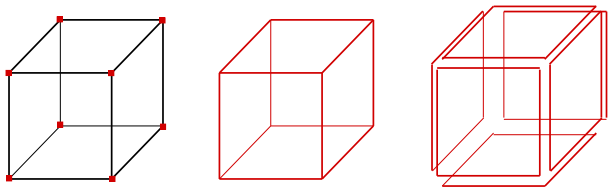

Figure 2. From left to right, drawn in red: the 0-skeleton, the 1-skeleton and the 2-skeleton of the cube.

A *polytopal complex* is a finite collection of polytopes P_i, $1 \leq i \leq n$, for which the intersection of any two polytopes $P_i \cap P_j$ is either empty or a common face of P_i and P_j.

For every d-dimensional polytopal complex the $(d+1)$-tuple, containing its number of i-faces in the i-th entry is called the *f-vector* of the polytopal complex.

Every polytope P gives rise to a polytopal complex consisting of all the proper faces of P. This polytopal complex is called the *boundary complex $C(\partial P)$ of the polytope P*.

Figure 2 below shows the boundary complex of the cube.

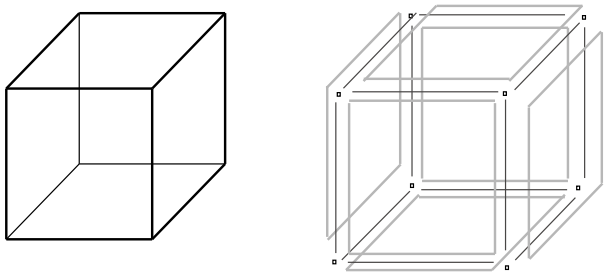

Figure 3. The 3-cube (left) and its boundary complex (right) where the 0-faces shown in black, the 1-faces dark gray and the 2-faces in light gray.

2.2 Simplices and simplicial complexes

A d-dimensional *simplex* or *d-simplex* for short is the convex hull of $d+1$ points in E^d in general position. Thus the d-simplex is the smallest (with respect to the number of vertices) possible d-polytope. Every face of the d-simplex is a m-simplex, $m \leq d$.

A 0-simplex is a point, a 1-simplex is a line segment, a 2-simplex is a triangle, a 3-simplex a tetrahedron, and so on.

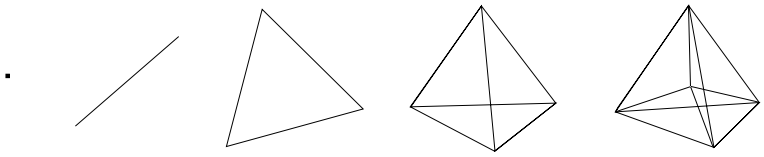

Figure 4. From left to right: a 0-simplex, a 1-simplex, a 2-simplex, a 3-simplex and a Schlegel diagram of a 4-simplex.

A polytopal complex which entirely consists of simplices is called a *simplicial complex* (for this it actually suffices that the facets of a polytopal complex are simplices).

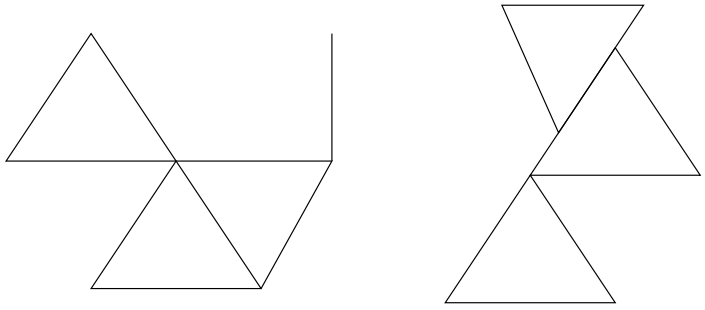

Figure 4. A simplicial complex (left) and a collection of simplices that does not form a simplicial complex (right).

The dimension of a simplicial complex is the maximal dimension of a facet. A simplicial complex is said to be *pure* if all facets are of the same dimension. A pure simplicial complex of dimension d satisfies the *pseudomanifold condition* if every $(d-1)$-face is part of exactly two facets.

Other properties (faces, facets, etc.) are defined in the same way as for polytopes and polytopal complexes.

2.3 From geometry to combinatorics

Every d-simplex has an *underlying set* in E^d, as the set of all points of that simplex. In the same way one can define the *underlying set* $|C|$ of a simplicial complex C. If the underlying set of a simplicial complex C is a topological manifold, then C is called *triangulated manifold* (or *triangulation of* $|C|$).

One can also go the other way and assign an abstract simplicial complex (in form of a *poset*) to a geometrical one by identifying each simplex with its vertex set. This obviously defines a set of sets with a natural partial ordering given by the inclusion (a socalled *poset*).

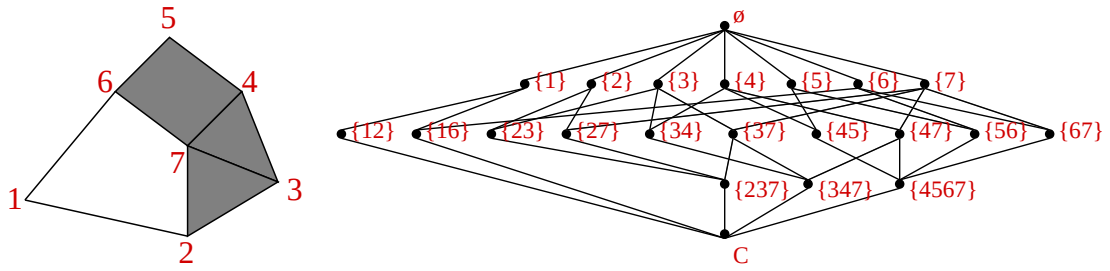

Figure 5. A geometrical simplicial complex (left) and its abstract version in form of a poset (right).

Let v be a vertex of C. The set of all facets that contain v is called *star of v in C* and is denoted by $\mathrm{star}_C(v)$. The subcomplex of $\mathrm{star}_C(v)$ that contains all faces that does not contain v is called *link of v in C*, written as $\mathrm{lk}_C(v)$.

A *combinatorial* 0*-sphere* is a 0-dimensional simplicial complex consisting only of two (different) vertices. Let us now come to the notion of a *combinatorial manifold*:

A *combinatorial d-manifold* is a d-dimensional simplicial complex whose vertex links are all combinatorial $(d-1)$-spheres. A *combinatorial pseudomanifold* is a simplicial complex whose vertex links are all combinatorial $(d-1)$-manifolds.

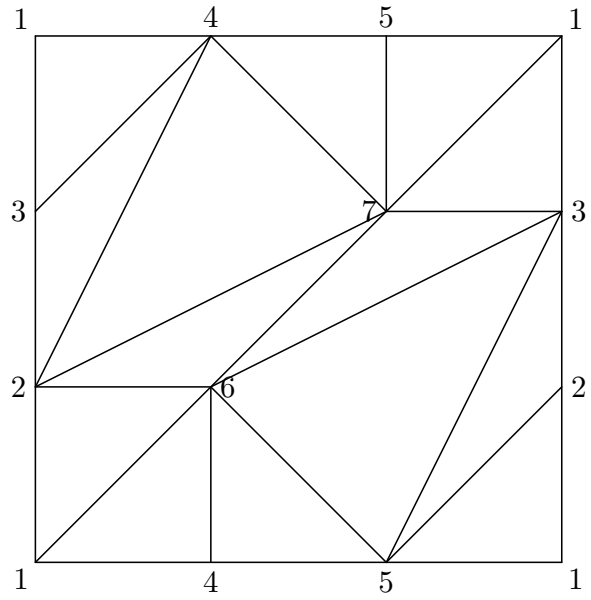

Figure 6. A simplicial complex that is a vertex-minimal combinatorial triangulation of the torus T^2 (so called Möbius' torus) – each vertex link is a hexagon.

Note, that every combinatorial manifold is a triangulated manifold. The opposite is wrong: for example, there exists a triangulation of the 5-sphere that is not combinatorial, the so called *Edward's sphere*, see [BL00].

A combinatorial manifold carries an induced PL-structure and can be understood in terms of an abstract simplicial complex. If the complex has d vertices there exists a natural embedding of C into the $(d-1)$ simplex and, thus, into E^{d-1}. In general, there is no canonical embedding into any lower dimensional space. However, combinatorial methods allow to examine a given simplicial complex independently from an embedding and, in particular, independently from vertex coordinates.

Some fundamental properties of an abstract simplicial complex C are the following:

Dimensionality. The dimension of C.

f, g and h-vector. The f-vector (f_k equals the number of k-faces of a simplicial complex), the g- and h-vector can be obtained from the f-vector via linear transformations.

Euler characteristic The Euler characteristic as the alternating sum over the Betti numbers / the f-vector.

(Co-)Homology. The simplicical (co-)homology groups and Betti numbers.

Connectedness and closeness. Whether C is strongly connected, path connected, has a boundary or not.

Symmetries. The automorphism group, i. e. the group of all permutations on the set of vertex labels that do not change the complex as a whole.

All of those properties and many more can be computed on a strictly combinatorial basis.

2.4 Normal surfaces and combinatorial slicings

The intersection of a tetrahedron Δ with a plane that does not intersect any vertex of Δ is called a *normal subset* of Δ. A closed *PL*-surface, properly embedded into a combinatorial 3-manifold M, which is equal to a finite union of normal subsets of tetrahedra of M is called *normal surface*.

Let M be a closed comb. 3-mfld., $\Delta \in M$ a tetrahedron and $v, w \in V$ two vertices in M. A function $f : M \to \mathbb{R}$ with $f|_\Delta$ is linear for any Δ and $f(v) \neq f(w)$ whenever $v \neq w$ is called *regular simplexwise linear (rsl) function* or *simplicial Morse function*. We call a level set $f^{-1}(\alpha)$, $\alpha \in \mathrm{Im}(f) \subset \mathbb{R}$ of an rsl-function that does not hit any vertex on M a *slicing* of M.

See [Küh95] for an introduction to the theory of polyhedral Morse functions.

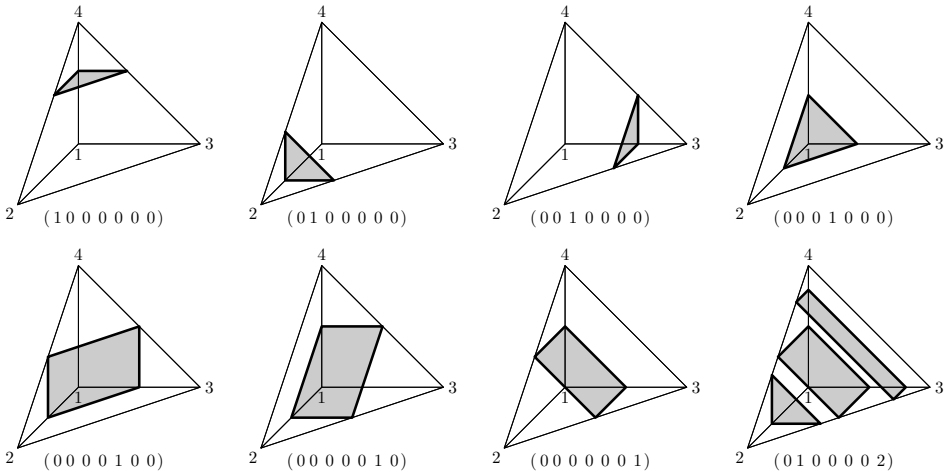

Figure 7. The seven different normal subsets of the tetrahedron.

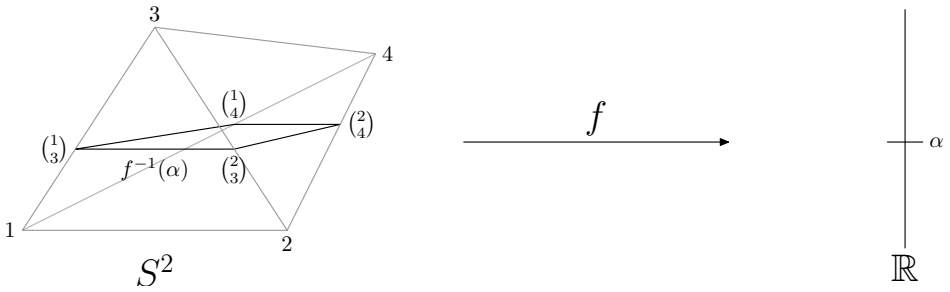

Figure 8. One dimensional slicing of the 2-sphere (represented as the boundary of the 3-simplex) seen as a level set of a regular point of a simplicial Morse function.

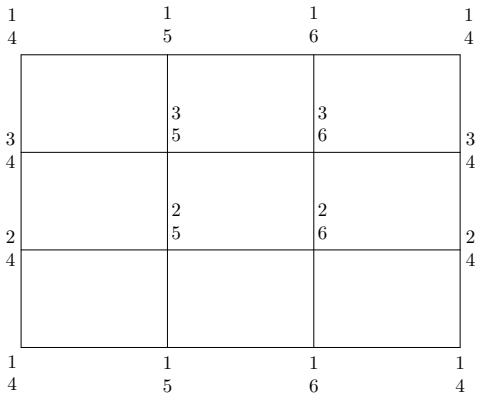

Figure 9. Handlebody decomposition of genus 1 of a 6-vertex 3-sphere - a 3×3-grid torus.

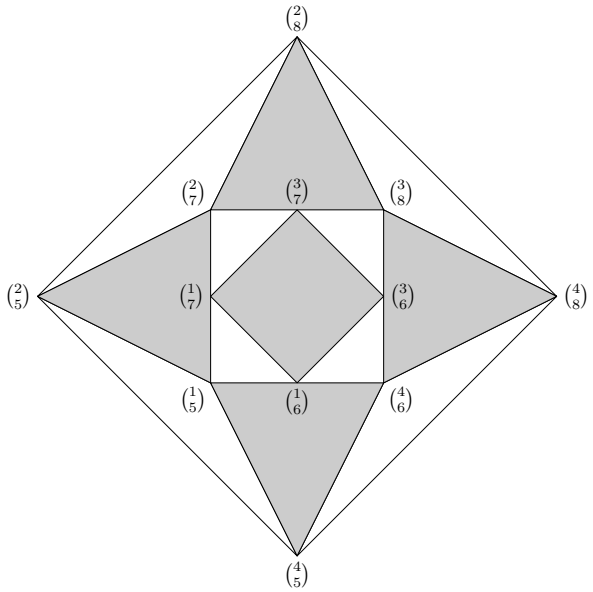

Figure 10. Separating sphere of an 8-vertex cylinder $S^2_4 \times [0,1]$ - A cuboctahedron (drawn as a Schlegel diagram of a quadrilateral face).

Chapter 3

The GAP object types SCSimplicialComplex and SCNormalSurface

The simpcomp package defines a new GAP object type SCSimplicialComplex, which is used to store simplicial complexes and their properties. As a property should not be calculated multiple times if the complex was not altered, simpcomp caches all calculated properties (the "known properties") of a simplicial complex within the SCSimplicialComplex object such that each property is only calculated once (see SCForceRecalc (3.1.4) for a way of switching off this behaviour). Internally, this behaviour is accomplished by making SCSimplicialComplex a subtype of SCPropertyObject, a GAP object type defined by simpcomp that provides the means to store (i.e. cache) certain properties of objects and also provides means of (de)serializing these objects to and from a XML format (see section 8).

3.1 The object types SCSimplicialComplex and SCPropertyObject

The object type SCPropertyObject provides a GAP object consisting of permanent and temporary attributes. While the permanent attributes are serialized to and from XML (and thus are persistent), this is not the case for temporary attributes. The permanent and temporary properties of a SCPropertyObject can be accessed directly with the functions SCPropertyByName, SCPropertyTmpByName and changed with SCPropertySet, SCPropertyTmpSet. But this direct access to property objects is discouraged when working with simpcomp, as the internal consistency of the objects cannot be guaranteed when the properties of the objects are accessed (and modified) in this way. Instead, the access to the object properties via so called property handlers is strongly recommended (and actually the SCProperty...-functions are only meant for internal use). For each property that an object of type SCPropertyObject can have, a function referred to as property handler ensures the integrity of the property and handles caching of this property. See section 3.2 below for a description of how to access properties of a SCSimplicialComplex object. Note that all properties returned by such property handlers are immutable objects. You can use ShallowCopy to obtain mutable copies. For a list of properties that SCSimplicialComplex handles, see chapter 4. For a few fundamental methods and functions (such as checking the object class, copying objects of this type,

etc.) for SCSimplcicialComplex see below.

3.1.1 SCIsSimplicialComplex

◊ SCIsSimplicialComplex(object) (filter)

Returns: true, if object is of type SCSimplicialComplex, false otherwise.

Filter for the category of a simplicial complex SCSimplicialComplex. The category SCSimplicialComplex is derived from the category SCPropertyObject.

Example

```
gap> c:=SCEmpty();;
gap> SCIsSimplicialComplex(c);
true
```

3.1.2 SCCopy

◊ SCCopy(complex) (operation)

Returns: a copy of complex upon success, fail otherwise.

Makes a "deep copy" of complex – this is a copy such that all properties of the copy can be altered without changing the original complex.

Example

```
gap> c:=SCBdSimplex(4);;
gap> d:=SCCopy(c)-1;;
gap> c.Facets=d.Facets;
false
```

Example

```
gap> c:=SCBdSimplex(4);;
gap> d:=SCCopy(c);;
gap> IsIdenticalObj(c,d);
false
```

3.1.3 ShallowCopy (SCSimplicialComplex)

◊ ShallowCopy (SCSimplicialComplex)(complex) (operation)

Returns: a copy of complex upon success, fail otherwise.

Makes a copy of complex. This is actually a "deep copy" such that all properties of the copy can be altered without changing the original complex. Internally calls SCCopy (3.1.2).

Example

```
gap> c:=SCLib.Load(87);;
gap> d:=ShallowCopy(c)+10;;
gap> c.Facets=d.Facets;
false
```

3.1.4 SCForceRecalc

◊ `SCForceRecalc( complex, flag )` (function)

Returns: `true` upon success, `fail` otherwise.

An object of the type `SCSimplicialComplex` normally caches its previously calculated properties such that each property only has to be calculated once. With this function this behaviour can be changed, if `flag` equals `true`, caching is turned off, for `flag` equals `false`, caching is turned on.

Example

```
gap> c:=SCFromFacets(Combinations([1..16],15));;
gap> c.F;; time;
2000
gap> c.F;; time;
0
gap> SCForceRecalc(c,true);
true
gap> c.F;; time;
2036
```

3.2 Accessing properties of a `SCSimplicialComplex` object

As described in section 3.1 the object type `SCSimplicialComplex` has properties that are handled by so called property handlers. These handlers take care of the internal consistency of objects of type `SCSimplicialComplex` and (re)compute necessary object attributes if necessary. There are two ways of accessing properties of a `SCSimplicialComplex` object. The first is to call a property handler function of the property one wishes to calculate directly. The first argument of such a property handler function is always the simplicial complex for which the property should be calculated, optionally followed by further arguments of the property handler function. An example would be:

Example

```
gap> c:=SCBdSimplex(3);; # create a simplicial complex object
gap> SCFVector(c);
[ 4, 6, 4 ]
gap> SCSkel(c,0);
[ [ 1 ], [ 2 ], [ 3 ], [ 4 ] ]
```

Here the functions `SCFVector` and `SCSkel` are the property handler functions, see chapter 10 for a list of all property handlers of a `SCSimplicialComplex` object. Apart from this (standard) method of calling the property handlers directly with a `SCSimplicialComplex` object, simpcomp provides the user with another more object oriented method of calling property handlers of a `SCSimplicialComplex` object inderectly and more conveniently:

Example

```
gap> c:=SCBdSimplex(3);; # create a simplicial complex object
gap> c.F;
[ 4, 6, 4 ]
gap> c.Skel(0);
[ [ 1 ], [ 2 ], [ 3 ], [ 4 ] ]
```

Note that the code in this example calculates the same properties as in the first example above, but the properties of a `SCSimplicialComplex` object are accessed via the `.` operator (the record access

operator). As GAP lacks object orientation, this behaviour is internally accomplished via a helper function (that calls the property handlers), such that accessing properties in this (indirect) way is a tiny bit slower than with the direct method described above. But this tiny drawback in speed is traded for a great deal of less and easier things that have to be typed (also note that `c.F` is accessed like a list, although internally the property handler function is called). If one is interested in utmost efficiency one must either use the direct calling method of property handlers or use a software package that is optimized for performance, such as polymake [GJ00]. For each property handler of a `SCSimplicialComplex` object the short object oriented form of this property handler is given in the description of the property handler function. In most cases it is just formed by dropping the prefix "`SC`" of the property handler functions.

3.3 Overloaded operators of `SCSimplicialComplex`

simpcomp overloads some standard operations for the object type `SCSimplcicialComplex` if this definition is intuitive and mathematically sound. See a list of overloaded operators below.

3.3.1 Operation + (SCSimplicialComplex, Integer)

◊ `Operation + (SCSimplicialComplex, Integer)( complex, value )` (operation)

Returns: the simplicial complex passed as argument upon success, `fail` otherwise.

Positively shifts the vertex labels of `complex` by the amount specified in `value`.

Example

```
gap> c:=SCBdSimplex(3)+10;;
gap> c.Facets;
[[11, 12, 13], [11, 12, 14], [11, 13, 14], [12, 13, 14]]
```

3.3.2 Operation - (SCSimplicialComplex, Integer)

◊ `Operation - (SCSimplicialComplex, Integer)( complex, value )` (operation)

Returns: the simplicial complex passed as argument unpon success, `fail` otherwise.

Negatively shifts the vertex labels of `complex` by the amount specified in `value`.

Example

```
gap> c:=SCBdSimplex(3)-1;;
gap> c.Facets;
[[0, 1, 2], [0, 1, 3], [0, 2, 3], [1, 2, 3]]
```

3.3.3 Operation mod (SCSimplicialComplex, Integer)

◊ `Operation mod (SCSimplicialComplex, Integer)( complex, value )` (operation)

Returns: the simplicial complex passed as argument unpon success, `fail` otherwise.

Takes all vertex labels of `complex` modulo the value specified in `value`. Warning: this might result in different vertices being assigned the same label, so be careful.

Example

```
gap> c:=(SCBdSimplex(3)*10) mod 7;;
gap> c.Facets;
[[3, 6, 2], [3, 6, 5], [3, 2, 5], [6, 2, 5]]
```

3.3.4 Operation ^ (SCSimplicialComplex, Integer)

◊ Operation ^ (SCSimplicialComplex, Integer)(complex, value) (operation)

Returns: a new simplicial complex upon success, fail otherwise.

Forms the value-th cartesian power of complex, i.e. the value-fold cartesian product of copies of complex. The complex passed as argument is not altered.

Example

```
gap> c:=SCBdSimplex(2)^2; #a torus
[SimplicialComplex

 Properties known: Dim, Facets, Name, TopologicalType, VertexLabels.

 Name="(S^1_3)^2"
 Dim=2
 TopologicalType="(S^1)^2"

/SimplicialComplex]
```

3.3.5 Operation + (SCSimplicialComplex, SCSimplicialComplex)

◊ Operation + (SCSimplicialComplex, SCSimplicialComplex)(complex1, complex2) (operation)

Returns: a new simplicial complex upon success, fail otherwise.

Forms the connected sum of complex1 and complex2. Currently uses the lexicographically first facets of both complex to do the gluing. The complexes passed as arguments are not altered.

Example

```
gap> SCLib.SearchByName("RP^3");
[ [ 45, "RP^3" ], [ 103, "RP^3=L(2,1) (VT)" ], [ 246, "(S^2~S^1)#RP^3" ],
  [ 247, "(S^2xS^1)#RP^3" ], [ 283, "(S^2~S^1)#2#RP^3" ],
  [ 285, "(S^2xS^1)#2#RP^3" ], [ 409, "RP^3#RP^3" ], ...
gap> SCLib.SearchByName("S^2~S^1");
[ [ 14, "S^2~S^1 (VT)" ], [ 29, "S^2~S^1 (VT)" ], [ 34, "S^2~S^1 (VT)" ],
  [ 42, "S^2~S^1 (VT)" ], [ 48, "S^2~S^1 (VT)" ], [ 49, "S^2~S^1 (VT)" ],
  [ 84, "S^2~S^1 (VT)" ], [ 87, "S^2~S^1 (VT)" ], ...
gap> c:=SCLib.Load(45)+SCLib.Load(14); #form RP^3#(S^2~S^1)
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels, Vertices.

 Name="RP^3#+-S^2~S^1 (VT)"
 Dim=3

/SimplicialComplex]
```

3.3.6 Operation - (SCSimplicialComplex, SCSimplicialComplex)

◊ Operation - (SCSimplicialComplex, SCSimplicialComplex)(complex1, complex2) (operation)

Returns: a new simplicial complex upon success, fail otherwise.

Calls SCDifference (4.6.5)(complex1, complex2)

3.3.7 Operation * (SCSimplicialComplex, SCSimplicialComplex)

◊ Operation * (SCSimplicialComplex, SCSimplicialComplex)(complex1, complex2) (operation)

Returns: a new simplicial complex upon success, fail otherwise.

Forms the cartesian product of complex1 and complex2.

Example

```
gap> SCLib.SearchByName("RP^2");
[ [ 3, "RP^2 (VT)" ], [ 284, "RP^2xS^1" ] ]
gap> c:=SCLib.Load(3)*SCBdSimplex(3); #form RP^2 x S^2
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="RP^2 (VT)xS^2_4"
 Dim=4

/SimplicialComplex]
```

3.3.8 Operation = (SCSimplicialComplex, SCSimplicialComplex)

◊ Operation = (SCSimplicialComplex, SCSimplicialComplex)(complex1, complex2) (operation)

Returns: true or false upon success, fail otherwise.

Calculates whether two simplicial complexes are equal, i.e. have equal facet sets in standard labeling. Returns true if the facet sets in standard labeling are equal as sets, false otherwise. Note: this check only compares the facet lists in standard labeling.

Example

```
gap> c:=SCBdSimplex(3);;
gap> c=c+10;
true
gap> c=SCBdCrossPolytope(4);
false
```

3.3.9 Operation Union (SCSimplicialComplex, SCSimplicialComplex)

◊ Operation Union (SCSimplicialComplex, SCSimplicialComplex)(complex1, complex2) (operation)

Returns: a new simplicial complex upon success, fail otherwise.

Computes the union of two simplicial complexes by calling SCUnion (4.6.24).

Example

```
gap> c:=Union(SCBdSimplex(3),SCBdSimplex(3)+3); #a wedge of two 2-spheres
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.
```

```
 Name="S^2_4 cup S^2_4"
 Dim=2

/SimplicialComplex]
```

3.3.10 Operation Difference (SCSimplicialComplex, SCSimplicialComplex)

◊ Operation Difference (SCSimplicialComplex, SCSimplicialComplex)(complex1, complex2) (operation)

Returns: a new simplicial complex upon success, fail otherwise.

Computes the "difference" of two simplicial complexes by calling SCDifference (4.6.5).

Example

```
gap> c:=SCBdSimplex(3);;
gap> d:=SC([[1,2,3]]);;
gap> disc:=Difference(c,d);;
gap> disc.Facets;
[ [ 1, 2, 4 ], [ 1, 3, 4 ], [ 2, 3, 4 ] ]
gap> empty:=Difference(d,c);;
gap> empty.Dim;
-1
gap>
```

3.3.11 Operation Intersection (SCSimplicialComplex, SCSimplicialComplex)

◊ Operation Intersection (SCSimplicialComplex, SCSimplicialComplex)(complex1, complex2) (operation)

Returns: a new simplicial complex upon success, fail otherwise.

Computes the "intersection" of two simplicial complexes by calling SCIntersection (4.6.7).

Example

```
gap> c:=SCBdSimplex(3);;
gap> d:=SCBdSimplex(3);;
gap> d:=SCMove(d,[[1,2,3],[]]);;
gap> d:=d+1;;
gap> s1:=SCIntersection(c,d);
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="S^2_4 cap unnamed complex n"
 Dim=1

/SimplicialComplex]
gap> s1.Facets;
[ [ 2, 3 ], [ 2, 4 ], [ 3, 4 ] ]
gap>
```

3.4 SCSimplicialComplex as a subtype of Set

Apart from being a subtype of SCPropertyObject, an object of type SCSimplicialComplex also behaves like a GAP Set type. The elements of the set are given by the facets of the simplical complex, grouped by their dimensionality, i.e. if complex is an object of type SCSimplicialComplex, c[1] refers to the 0-faces of complex, c[2] to the 1-faces, etc.

3.4.1 Size (SCSimplicialComplex)

◊ Size (SCSimplicialComplex)(complex) (operation)

Returns: an integer upon success, fail otherwise.

Returns the "size" of a simplicial complex. This is $d+1$, where d is the dimension of the complex. $d+1$ is returned instead of d, as all lists in GAP are indexed beginning with 1 – thus this also holds for all the face lattice related properties of the complex.

Example

```
gap> SCLib.SearchByAttribute("F=Â·[12,66,108,54]");
[ [ 116, "S^2xS^1 (VT)" ], [ 117, "S^2xS^1 (VT)" ],
  [ 118, "(S^2xS^1)#(S^2xS^1) (VT)" ], [ 119, "S^2~S^1 (VT)" ],
  [ 120, "(S^2xS^1)#(S^2xS^1) (VT)" ], [ 121, "(S^2xS^1)#(S^2xS^1) (VT)" ],
  [ 122, "S^2xS^1 (VT)" ], [ 123, "(S^2xS^1)#(S^2xS^1) (VT)" ], ...
gap> c:=SCLib.Load(116);;
gap> for i in [1..Size(c)] do Print(c.F[i],"\n"); od;
12
66
108
54
```

3.4.2 Length (SCSimplicialComplex)

◊ Length (SCSimplicialComplex)(complex) (operation)

Returns: an integer upon success, fail otherwise.

Returns the "size" of a simplicial complex by calling Size(complex).

Example

```
gap> SCLib.SearchByAttribute("F=Â¸[12,66,108,54]");
[ [ 116, "SË2xSË1 (VT)" ], [ 117, "SË2xSË1 (VT)" ],
[ 118, "(SË2xSË1)#(SË2xSË1) (VT)" ], [ 119, "SË2ËSË1 (VT)" ],
[ 120, "(SË2xSË1)#(SË2xSË1) (VT)" ], [ 121, "(SË2xSË1)#(SË2xSË1) (VT)" ],
[ 122, "SË2xSË1 (VT)" ], [ 123, "(SË2xSË1)#(SË2xSË1) (VT)" ], ...
gap> c:=SCLib.Load(116);;
gap> for i in [1..Length(c)] do Print(c.F[i],"\n"); od;
12
66
108
54
```

3.4.3 Operation [] (SCSimplicialComplex)

◊ Operation [] (SCSimplicialComplex)(complex, pos) (operation)

Returns: a list of faces upon success, fail otherwise.

Returns the $(pos-1)$-dimensional faces of `complex` as list. If $pos \leq 0$ or $pos \geq d+2$, where d is the dimension of `complex`, the empty set is returned.

Example

```
gap> SCLib.SearchByName("K^2");
[ [ 19, "K^2 (VT)" ], [ 230, "K^2 (VT)" ] ]
gap> c:=SCLib.Load(230);;
gap> c[2];
[ [ 1, 2 ], [ 1, 3 ], [ 1, 7 ], [ 1, 9 ], [ 1, 13 ], [ 1, 14 ], [ 2, 3 ],
  [ 2, 4 ], [ 2, 8 ], [ 2, 10 ], [ 2, 14 ], [ 3, 4 ], [ 3, 5 ], [ 3, 9 ],
  [ 3, 11 ], [ 4, 5 ], [ 4, 6 ], [ 4, 10 ], [ 4, 12 ], [ 5, 6 ], [ 5, 7 ],
  [ 5, 11 ], [ 5, 13 ], [ 6, 7 ], [ 6, 8 ], [ 6, 12 ], [ 6, 14 ], [ 7, 8 ],
  [ 7, 9 ], [ 7, 13 ], [ 8, 9 ], [ 8, 10 ], [ 8, 14 ], [ 9, 10 ], [ 9, 11 ],
  [ 10, 11 ], [ 10, 12 ], [ 11, 12 ], [ 11, 13 ], [ 12, 13 ], [ 12, 14 ],
  [ 13, 14 ] ]
gap> c[4];
[  ]
gap>
```

3.4.4 IsBound (SCSimplicialComplex)

◊ `IsBound (SCSimplicialComplex)( complex, pos )` (operation)

Returns: `true` or `false` upon success, `fail` otherwise.

Returns `true` when `complex` has proper faces of dimension $(pos-1)$, `false` otherwise.

Example

```
gap> c:=SCBdCrossPolytope(4);;
gap> IsBound(c[4]);
true
gap> IsBound(c[5]);
false
```

3.4.5 Iterator (SCSimplicialComplex)

◊ `Iterator (SCSimplicialComplex)( complex )` (operation)

Returns: an iterator on the face lattice of `complex` upon success, `fail` otherwise.

Provides an iterator object for the face lattice of a simplicial complex.

Example

```
gap> c:=SCBdCrossPolytope(4);;
gap> for faces in c do Print(Length(faces),"\n"); od;
8
24
32
16
```

3.5 The object types `SCNormalSurface` as asubtyope of `SCPropertyObject`

The simpcomp package defines a GAP object type SCNormalSurface to describe normal surfaces / slicings (level sets of discrete Morse functions) in combinatorial 3-manifolds. Internally SCNormalSurface is a subtype of SCPropertyObject and, thus, mostly behaves like a SCSimplicialComplex object (see section 3.1). For a very short introduction to normal surfaces see 2.4, for some fundamental methods and functions for SCNormalSurface see below.

3.6 Overloaded operators of SCNormalSurface

As with the object type SCSimplcicialComplex, simpcomp overloads some standard operations for the object type SCNormalSurface. See a list of overloaded operators below.

3.6.1 Operation + (SCNormalSurface, Integer)

◊ Operation + (SCNormalSurface, Integer)(complex, value) (operation)

Returns: the normal surface passed as argument upon success, fail otherwise.
Positively shifts the vertex labels of complex by the amount specified in value.

Example

```
gap> ns:=SCNSSlicing(SCBdSimplex(4),[[1],[2..5]]);;
gap> ns.Facets;
[ [ [ 1, 2 ], [ 1, 3 ], [ 1, 4 ] ], [ [ 1, 2 ], [ 1, 3 ], [ 1, 5 ] ],
  [ [ 1, 2 ], [ 1, 4 ], [ 1, 5 ] ], [ [ 1, 3 ], [ 1, 4 ], [ 1, 5 ] ] ]
gap> ns:=ns + 2;;
gap> ns.Facets;
[ [ [ 3, 4 ], [ 3, 5 ], [ 3, 6 ] ], [ [ 3, 4 ], [ 3, 5 ], [ 3, 7 ] ],
  [ [ 3, 4 ], [ 3, 6 ], [ 3, 7 ] ], [ [ 3, 5 ], [ 3, 6 ], [ 3, 7 ] ] ]
gap>
```

3.6.2 Operation - (SCNormalSurface, Integer)

◊ Operation - (SCNormalSurface, Integer)(complex, value) (operation)

Returns: the normal surface passed as argument unpon success, fail otherwise.
Negatively shifts the vertex labels of complex by the amount specified in value.

Example

```
gap> ns:=SCNSSlicing(SCBdSimplex(4),[[1],[2..5]]);;
gap> ns.Facets;
[ [ [ 1, 2 ], [ 1, 3 ], [ 1, 4 ] ], [ [ 1, 2 ], [ 1, 3 ], [ 1, 5 ] ],
  [ [ 1, 2 ], [ 1, 4 ], [ 1, 5 ] ], [ [ 1, 3 ], [ 1, 4 ], [ 1, 5 ] ] ]
gap> ns:=ns - 2;;
gap> ns.Facets;
[ [ [ -1, 0 ], [ -1, 1 ], [ -1, 2 ] ], [ [ -1, 0 ], [ -1, 1 ], [ -1, 3 ] ],
  [ [ -1, 0 ], [ -1, 2 ], [ -1, 3 ] ], [ [ -1, 1 ], [ -1, 2 ], [ -1, 3 ] ] ]
gap>
```

3.6.3 Operation mod (SCNormalSurface, Integer)

◊ Operation mod (SCNormalSurface, Integer)(complex, value) (operation)

Returns: the normal surface passed as argument unpon success, fail otherwise.

Takes all vertex labels of `complex` modulo the value specified in `value`. Warning: this might result in different vertices being assigned the same label, so be careful.

Example

```
gap> ns:=SCNSSlicing(SCBdSimplex(4),[[1],[2..5]]);;
gap> ns.Facets;
[ [ [ 1, 2 ], [ 1, 3 ], [ 1, 4 ] ], [ [ 1, 2 ], [ 1, 3 ], [ 1, 5 ] ],
  [ [ 1, 2 ], [ 1, 4 ], [ 1, 5 ] ], [ [ 1, 3 ], [ 1, 4 ], [ 1, 5 ] ] ]
gap> ns:=ns mod 2;;
gap> ns.Facets;
[ [ [ 1, 0 ], [ 1, 1 ], [ 1, 0 ] ], [ [ 1, 0 ], [ 1, 1 ], [ 1, 1 ] ],
  [ [ 1, 0 ], [ 1, 0 ], [ 1, 1 ] ], [ [ 1, 1 ], [ 1, 0 ], [ 1, 1 ] ] ]
gap>
```

3.6.4 Operation = (SCNormalSurface, SCNormalSurface)

◊ `Operation = (SCNormalSurface, SCNormalSurface)( complex1, complex2 )` (operation)

Returns: `true` or `false` upon success, `fail` otherwise.

Calculates whether two normal surface are equal, i.e. have equal facet sets in standard labeling. Returns `true` if the facet sets in standard labeling are equal as sets, `false` otherwise. Note: this check only compares the facet lists in standard labeling.

Example

```
gap> ns:=SCNSSlicing(SCBdSimplex(4),[[1],[2..5]]);;
gap> ns1:=SCNSSlicing(SCBdSimplex(4),[[1],[2..5]]);;
gap> ns2:=SCNSSlicing(SCBdSimplex(4),[[2],[1,3,4,5]]);;
gap> ns1=ns2;
true
gap> ns2:=SCNSSlicing(SCBdSimplex(4),[[1,2],[3,4,5]]);;
gap> ns1=ns2;
false
gap>
```

3.6.5 Operation Union (SCNormalSurface, SCNormalSurface)

◊ `Operation Union (SCNormalSurface, SCNormalSurface)( complex1, complex2 )` (operation)

Returns: a new normnal surface upon success, `fail` otherwise.

Computes the union of two normal surfaces by calling `SCUnion` (4.6.24).

Example

```
gap> SCLib.SearchByAttribute("F=[6,15,18,9]");
[ [ 4, "S^3 (VT)" ] ]
gap> c:=SCLib.Load(4);;
gap> ns1:=SCNSSlicing(c,[[1,3,5],[2,4,6]]);;
gap> ns2:=SCNSSlicing(c,[[1,3,5],[2,4,6]])+10;;
gap> SCNSTopologicalType(ns1);
"T^2"
gap> SCNSTopologicalType(ns2);
"T^2"
gap> ns3:=Union(ns1,ns2);;
```

```
gap> SCNSTopologicalType(ns3);
"T^2 U T^2"
gap>
```

3.7 `SCNormalSurface` as a subtype of `Set`

Like objects of type `SCSimplicialComplex`, an object of type `SCNormalSurface` behaves like a GAP `Set` type. The elements of the set are given by the facets of the normal surface, grouped by their dimensionality and type, i.e. if `complex` is an object of type `SCNormalSurface`, `c[1]` refers to the 0-faces of `complex`, `c[2]` to the 1-faces, `c[3]` to the triangles and `c[4]` to the quadrilaterals. See section 3.4 for details.

Chapter 4

Functions for simplicial complexes

4.1 Creating an `SCSimplicialComplex` object from a facet list

This section contains functions to generate or to construct new simplicial complexes. Some of them obtain new complexes from existing ones, some generate new complexes.

4.1.1 SCFromFacets

◊ `SCFromFacets( facets )` (function)

Returns: simplicial complex of type `SCSimplicialComplex` upon success, fail otherwise.

Constructs a simplicial complex object from the given facet list. The facet list `facets` has to be a duplicate free list (or set) which consists of duplicate free entries, which are in turn lists or sets. Any vertex labeling can be used, even sets as vertex labels are allowed. Internally the vertices are mapped to the standard labeling 1..*N*, where *N* is the number of vertices of the complex, but the vertex labels of the original complex are stored in the property "VertexLabels", see `SCLabels` (4.4.3) and the `SCRelabel..` functions like `SCRelabel` (4.4.4) or `SCRelabelStandard` (4.4.5).

Example

```
gap> c:=SCFromFacets([[1,2,5], [1,4,5], [1,4,6], [2,3,5],
                      [3,4,6], [3,5,6]]);
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="unnamed complex n"
 Dim=2

/SimplicialComplex]
gap> c:=SCFromFacets([["a","d","e"], ["a","b","e"], ["b","c","e"],
                      ["c","e","f"], ["c","d","f"], ["a", "d", "f"]]);
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="unnamed complex n"
 Dim=2
```

```
/SimplicialComplex]
```

4.1.2 SC

◊ SC(facets) (function)

Returns: simplicial complex of type SCSimplicialComplex upon success, fail otherwise.

A shorter function to create a simplicial complex from a facet list, just calls SCFromFacets (4.1.1)(facets).

Example

```
gap> c:=SC(Combinations([1..6],5));
[SimplicialComplex

 Properties known: Dim, Facets, VertexLabels.

 Name="unnamed complex n"
 Dim=4

/SimplicialComplex]
```

4.1.3 SCFromDifferenceCycles

◊ SCFromDifferenceCycles(diffcycles) (function)

Returns: a simplicial complex upon success, fail otherwise.

Creates a simplicial complex object from the list of difference cycles provided. If diffcycles is of length 1 the computation is equivalent to the one in SCDifferenceCycleExpand (4.3.8). Otherwise the induced modulus of all cycles has to be equal and the union of all expanded difference cycles is returned.

Example

```
gap> c:=SCFromDifferenceCycles([[1,1,6],[2,3,3]]);;
gap> c.F;
[ 8, 24, 16 ]
gap> c.Homology;
[ [ 0, [  ] ], [ 2, [  ] ], [ 1, [  ] ] ]
gap> c.Chi;
0
gap> c.HasBoundary;
false
gap> SCIsPseudoManifold(c);
true
gap> SCIsManifold(c);
#I  SCIsManifold: link is sphere.
#I  SCIsManifold: transitive automorphism group, checking only one link.
true
gap>
```

4.1.4 SCFromGenerators

◊ `SCFromGenerators( group, generators )` (function)

Returns: simplicial complex of type `SCSimplicialComplex` upon success, fail otherwise.

Constructs a simplicial complex object from the set of `generators` on which the group `group` acts, i.e. a complex which has `group` as automorphism group that consists of the orbits specified by the list of representatives passed in `generators`. Internally calls `SCFromFacets` (4.1.1).

Example

```
gap> #group: (C7 : C3) : C2, order 42
gap> G:=Group([(2,6,5,7,3,4),(1,3,5,7,2,4,6)]);;
gap> c:=SCFromGenerators(G,[[ 1, 2, 4 ]]);
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="unnamed complex n"
 Dim=2

/SimplicialComplex]
gap> SCLib.DetermineTopologicalType(c);
[ [ true, 5 ] ]
gap> # a torus
```

4.2 Generating some standard triangulations

4.2.1 SCBdCrossPolytope

◊ `SCBdCrossPolytope( dim )` (function)

Returns: a new simplicial complex upon success, `fail` otherwise.

Generates the boundary of the *dim*-dimensional cross polytope β^d.

Example

```
gap> SCBdCrossPolytope(3); # the octahedron
[SimplicialComplex

 Properties known: Chi, Dim, F, Facets, HasBoundary, Homology, IsConnected,
                   IsStronglyConnected, Name, TopologicalType, VertexLabels.

 Name="Bd(\beta^3)"
 Dim=2
 Chi=2
 F=[ 6, 12, 8 ]
 Homology=[ [ 0, [ ] ], [ 0, [ ] ], [ 1, [ ] ] ]
 IsConnected=true
IsStronglyConnected=true
 TopologicalType="S^2"

/SimplicialComplex]
```

4.2.2 SCBdSimplex

◊ SCBdSimplex(dim) (function)

Returns: a new simplicial complex upon success, fail otherwise.
Generates the boundary of the d-simplex.

Example

```
gap> SCBdSimplex(5);
[SimplicialComplex

 Properties known: AutomorphismGroup, AutomorphismGroupOrder,
                   AutomorphismGroupStructure, AutomorphismGroupTransitivity,
                   Chi, Dim, F, Facets, Generators, HasBoundary, Homology,
                   IsConnected, IsStronglyConnected, Name, TopologicalType,
                   VertexLabels.

 Name="S^4_6"
 Dim=4
 AutomorphismGroupStructure="S6"
 AutomorphismGroupTransitivity=6
 Chi=2
 F=[ 6, 15, 20, 15, 6 ]
 Homology=[ [ 0, [ ] ], [ 0, [ ] ], [ 0, [ ] ], [ 0, [ ] ], [ 1, [ ] ] ]
 IsConnected=true
 IsStronglyConnected=true
 TopologicalType="S^4"

/SimplicialComplex]
```

4.2.3 SCEmpty

◊ SCEmpty( ) (function)

Returns: simplicial complex with empty facet list upon success, fail otherwise.
Generates an empty complex (of dimension -1).

Example

```
gap> SCEmpty();
[SimplicialComplex

 Properties known: Dim, Faces, Facets, Name, VertexLabels.

 Name="empty complex"
 Dim=-1

/SimplicialComplex]
```

4.2.4 SCSimplex

◊ SCSimplex(dim) (function)

Returns: a new simplicial complex upon success, fail otherwise.
Generates the dim-simplex.

Example

```
gap> SCSimplex(3);
[SimplicialComplex

 Properties known: Chi, Dim, Facets, Name, TopologicalType, VertexLabels.

 Name="B^3_4"
 Dim=3
 Chi=1
 TopologicalType="B^3"

/SimplicialComplex]
```

4.3 Generating new complexes from old

4.3.1 SCCartesianPower

◊ `SCCartesianPower( complex, n )` (function)

Returns: a new simplicial complex upon success, `fail` otherwise.

The new complex is *PL*-homeomorphic to n-times the cartesian product of `complex`, of dimensions $n \cdot \dim$ and has $f^n_{\dim} \cdot n \cdot \frac{2n-1}{2^{n-1}}!$ facets where dim denotes the dimension of `complex`.

Example

```
gap> c:=SCBdSimplex(2);;
gap> 4torus:=SCCartesianPower(c,4);
[SimplicialComplex

 Properties known: Dim, Facets, Name, TopologicalType, VertexLabels.

 Name="(S^1_3)^4"
 Dim=4
 TopologicalType="(S^1)^4"

/SimplicialComplex]
gap> 4torus.Homology;
[ [ 0, [  ] ], [ 4, [  ] ], [ 6, [  ] ], [ 4, [  ] ], [ 1, [  ] ] ]
gap> 4torus.Chi;
0
gap> 4torus.F;
[ 81, 1215, 4050, 4860, 1944 ]
gap>
```

4.3.2 SCCartesianProduct

◊ `SCCartesianProduct( complex1, complex2 )` (function)

Returns: a new simplicial complex upon success, `fail` otherwise.

Computes the simplicial cartesian product of `complex1` and `complex2` where `complex1` and `complex2` are pure, simplicial complexes. The original vertex labeling of `complex1` and `complex2` is

changed into the standard one. The new complex has vertex labels of type $[v_i, v_j]$ where v_i is a vertex of `complex1` and v_j is a vertex of `complex2`.

If n_i, $i = 1,2$ are the number facets and if d_i, $i = 1,2$ are the dimensions of `complexi`, then the new complex has $n_1 \cdot n_2 \cdot \binom{d_1+d_2}{d_1}$ facets. The number of vertices of the new complex equals the product of the numbers of vertices of the arguments.

Example

```
gap> c1:=SCBdSimplex(2);;
gap> c2:=SCBdSimplex(3);;
gap> c3:=SCCartesianProduct(c1,c2);
[SimplicialComplex

 Properties known: Dim, Facets, Name, TopologicalType, VertexLabels.

 Name="S^1_3xS^2_4"
 Dim=3
 TopologicalType="S^1xS^2"

/SimplicialComplex]
gap> c3.Homology;
[ [ 0, [  ] ], [ 1, [  ] ], [ 1, [  ] ], [ 1, [  ] ] ]
gap> c3.F;
[ 12, 48, 72, 36 ]
gap>
```

4.3.3 SCConnectedComponents

◊ `SCConnectedComponents( complex )` (function)

Returns: a list of simplicial complexes upon success, `fail` otherwise.

Computes all connected components of an arbitrary simplicial complex.

Example

```
gap> c:=SC([[1,2,3],[3,4,5],[4,5,6,7,8]]);;
gap> SCRename(c,"connected complex");;
gap> SCConnectedComponents(c);
[ [SimplicialComplex

     Properties known: Dim, Facets, Name, VertexLabels.

     Name="Connected component #1 of connected complex"
     Dim=4

    /SimplicialComplex] ]
gap> c:=SC([[1,2,3],[4,5],[6,7,8]]);;
gap> SCRename(c,"non-connected complex");;
gap> SCConnectedComponents(c);
[ [SimplicialComplex

     Properties known: Dim, Facets, Name, VertexLabels.

     Name="Connected component #1 of non-connected complex"
     Dim=2
```

```
      /SimplicialComplex], [SimplicialComplex

       Properties known: Chi, Dim, Facets, Name, VertexLabels.

       Name="Connected component #2 of non-connected complex"
       Dim=1
       Chi=0

      /SimplicialComplex], [SimplicialComplex

       Properties known: Dim, Facets, Name, VertexLabels.

       Name="Connected component #3 of non-connected complex"
       Dim=2

      /SimplicialComplex] ]
  gap>
```

4.3.4 SCConnectedProduct

◊ `SCConnectedProduct( complex, n )` (function)

Returns: a simplicial complex upon success, `fail` otherwise.

If $n \geq 2$, the function internally calls $1\times$ `SCConnectedSum` (4.3.5) and $(n-2)\times$ `SCConnectedSumMinus` (4.3.6).

Example

```
gap> SCLib.SearchByName("T^2");
[ [ 5, "T^2 (VT)" ], [ 7, "T^2 (VT)" ], [ 11, "T^2 (VT)" ],
  [ 12, "T^2 (VT)" ], [ 20, "T^2 (VT)" ], [ 22, "(T^2)#2" ],
  [ 27, "(T^2)#3" ], [ 41, "T^2 (VT)" ], [ 44, "(T^2)#4" ], ...
gap> torus:=SCLib.Load(5);;
gap> genus10:=SCConnectedProduct(torus,10);
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="T^2 (VT)#+-T^2 (VT)#+-T^2 (VT)#+-T^2 (VT)#+-T^2 (VT)#+-T^2 (VT)#+-T^2 (\
VT)#+-T^2 (VT)#+-T^2 (VT)#+-T^2 (VT)"
 Dim=2

/SimplicialComplex]
gap> genus10.Chi;
-18
gap> genus10.F;
[ 43, 183, 122 ]
gap>
```

4.3.5 SCConnectedSum

◊ `SCConnectedSum( complex1, complex2 )` (function)

Returns: a new simplicial complex upon success, `fail` otherwise.

The function removes the first simplex of `complex1` and `complex2` and glues them together along the boundary. The boundary components of the complexes are glued in with a twist (a transposition on the vertex labels of the boundary of `complex2`). Thus, `SCConnectedSum` is different from `SCConnectedSumMinus` (4.3.6) whenever `complex1` and `complex2` do not allow an orientation reversing homeomorphism.

Example

```
gap> SCLib.SearchByName("T^2");
[ [ 5, "T^2 (VT)" ], [ 7, "T^2 (VT)" ], [ 11, "T^2 (VT)" ],
  [ 12, "T^2 (VT)" ], [ 20, "T^2 (VT)" ], [ 22, "(T^2)#2" ],
  [ 27, "(T^2)#3" ], [ 41, "T^2 (VT)" ], [ 44, "(T^2)#4" ], ...
gap> torus:=SCLib.Load(5);;
gap> genus2:=SCConnectedSum(torus,torus);
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="T^2 (VT)#+-T^2 (VT)"
 Dim=2

/SimplicialComplex]
gap> genus2.Homology;
[ [ 0, [  ] ], [ 4, [  ] ], [ 1, [  ] ] ]
gap> genus2.Chi;
-2
gap>
```

Example

```
gap> SCLib.SearchByName("CP^2");
[ [ 17, "CP^2 (VT)" ], [ 88, "CP^2#CP^2" ], [ 89, "CP^2#-CP^2" ],
  [ 186, "CP^2#(S^2xS^2)" ], [ 499, "(S^3~S^1)#(CP^2)^{#5} (VT)" ] ]
gap> cp2:=SCLib.Load(17);;
# CP^2 # CP^2 (signature of intersection form is 2)
gap> c1:=SCConnectedSum(cp2,cp2);;
# CP^2 # - CP^2 (signature of intersection form is 0)
gap> c2:=SCConnectedSumMinus(cp2,cp2);;
gap> c1.F=c2.F;
true
gap> c1.ASDet=c2.ASDet;
true
gap> SCIsIsomorphic(c1,c2);
false
gap> PrintArray(SCIntersectionForm(c1));
[ [  1,  0 ],
  [  0,  1 ] ]
gap> PrintArray(SCIntersectionForm(c2));
[ [   1,   0 ],
  [   0,  -1 ] ]
```

```
gap>
```

4.3.6 SCConnectedSumMinus

◊ `SCConnectedSumMinus( complex1, complex2 )` (function)

Returns: a new simplicial complex upon success, `fail` otherwise.

The function removes the first simplex of `complex1` and `complex2` and glues them together along the boundary. The boundary components of the complexes are glued in without twist. Thus, `SCConnectedSumMinus` is different from `SCConnectedSum` (4.3.5) whenever `complex1` and `complex2` do not allow an orientation reversing homeomorphism.

Example

```
gap> SCLib.SearchByName("T^2");
[ [ 5, "T^2 (VT)" ], [ 7, "T^2 (VT)" ], [ 11, "T^2 (VT)" ],
  [ 12, "T^2 (VT)" ], [ 20, "T^2 (VT)" ], [ 22, "(T^2)#2" ],
  [ 27, "(T^2)#3" ], [ 41, "T^2 (VT)" ], [ 44, "(T^2)#4" ], ...
gap> torus:=SCLib.Load(5);;
gap> genus2:=SCConnectedSumMinus(torus,torus);
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="T^2 (VT)#+-T^2 (VT)"
 Dim=2

/SimplicialComplex]
gap> genus2.Homology;
[ [ 0, [  ] ], [ 4, [  ] ], [ 1, [  ] ] ]
gap> genus2.Chi;
-2
gap>
```

Example

```
gap> SCLib.SearchByName("CP^2");
[ [ 17, "CP^2 (VT)" ], [ 88, "CP^2#CP^2" ], [ 89, "CP^2#-CP^2" ],
  [ 186, "CP^2#(S^2xS^2)" ], [ 499, "(S^3~S^1)#(CP^2)^{#5} (VT)" ] ]
gap> cp2:=SCLib.Load(17);;
# CP^2 # CP^2 (signature of intersection form is 2)
gap> c1:=SCConnectedSum(cp2,cp2);;
# CP^2 # - CP^2 (signature of intersection form is 0)
gap> c2:=SCConnectedSumMinus(cp2,cp2);;
gap> c1.F=c2.F;
true
gap> c1.ASDet=c2.ASDet;
true
gap> SCIsIsomorphic(c1,c2);
false
gap> PrintArray(SCIntersectionForm(c1));
[ [  1,  0 ],
  [  0,  1 ] ]
gap> PrintArray(SCIntersectionForm(c2));
```

```
  [ [   1,   0 ],
    [   0,  -1 ] ]
  gap>
```

4.3.7 SCDifferenceCycleCompress

◊ `SCDifferenceCycleCompress( simplex, modulus )` (function)

Returns: list with possibly duplicate entries upon success, `fail` otherwise.

A difference cycle is returned, i. e. a list of integer values of length $(d+1)$, if d is the dimension of `complex`, and a sum equal to `modulus`. In some sense this is the inverse operation of `SCDifferenceCycleExpand` (4.3.8).

Example

```
gap> sphere:=SCBdSimplex(4);;
gap> gens:=SCGenerators(sphere);
[ [ [ 1, 2, 3, 4 ], [ [ 5 ] ] ] ]
gap> diffcycle:=SCDifferenceCycleCompress(gens[1][1],5);
[ 1, 1, 1, 2 ]
gap> c:=SCDifferenceCycleExpand([1,1,1,2]);;
gap> c.Facets;
[ [ 1, 2, 3, 4 ], [ 1, 2, 3, 5 ], [ 1, 2, 4, 5 ], [ 1, 3, 4, 5 ],
  [ 2, 3, 4, 5 ] ]
gap>
```

4.3.8 SCDifferenceCycleExpand

◊ `SCDifferenceCycleExpand( diffcycle )` (function)

Returns: a simplicial complex upon success, `fail` otherwise.

`diffcycle` induces a simplex $\Delta = (v_1,\ldots,v_{n+1})$ by v_1 =`diffcycle[1]`, $v_i = v_{i-1}+$ `diffcycle[i]` and a cyclic group action by $\mathbb{Z}_\sigma$ where $\sigma = \sum$ `diffcycle[i]` is the modulus of `diffcycle`. The function returns the $\mathbb{Z}_\sigma$-orbit of Δ.

Note, that modulo operations in GAP are often a little bit cumbersome, since all integer ranges usually start from 1.

Example

```
gap> c:=SCDifferenceCycleExpand([1,1,2]);;
gap> c.Facets;
[ [ 1, 2, 3 ], [ 1, 2, 4 ], [ 1, 3, 4 ], [ 2, 3, 4 ] ]
gap>
```

4.3.9 SCFVectorBdCrossPolytope

◊ `SCFVectorBdCrossPolytope( dim )` (function)

Returns: a list of integers of size `dim + 1` upon success, `fail` otherwise.

Computes the f-vector of the d-dimensional cross polytope without generating the underlying complex.

Example

```
SCFVectorBdCrossPolytope(50);
[100, 4900, 156800, 3684800, 67800320, 1017004800, 12785203200,
  137440934400, 1282782054400, 10518812846080, 76500457062400,
  497252970905600, 2907017368371200, 15365663232819200, 73755183517532160,
  322678927889203200, 1290715711556812800, 4732624275708313600,
  1594147124449116160 0, 4941856085792260096 0, 1411958881654931456 00,
  3722437051635728384 00, 9063324995286990848 00, 2039248123939572940800,
  4241636097794311716864, 8156992495758291763200, 1450131999245918535680 0,
  2382359713046866165760 0, 3614614737036624527360 0, 5060460631851274338304 0,
  6529626621743579791360 0, 7753931613320501002240 0, 8458834487258728366080 0,
  8458834487258728366080 0, 7733791531207980220416 0, 6444826276006650183680 0,
  4877165830491519057920 0, 3337008199809986723840 0, 2053543507575376445440 0,
  1129448929166457044992 0, 5509506971543692902400, 2361217273518725529600,
  878592473867432755200, 279552150776001331200, 74547240206933688320,
  16205921784116019200, 2758454771764428800, 344806846470553600,
  28147497671065600, 1125899906842624]
```

4.3.10 SCFVectorBdSimplex

◊ `SCFVectorBdSimplex( dim )` (function)

Returns: a list of integers of size `dim + 1` upon success, `fail` otherwise.

Computes the f-vector of the d-simplex without generating the underlying complex.

Example

```
SCFVectorBdSimplex(100);
[101, 5050, 166650, 4082925, 79208745, 1267339920, 17199613200,
  202095455100, 2088319702700, 19212541264840, 158940114100040,
  1192050855750300, 8160963550905900, 51297485177122800, 297525414027312240,
  1599199100396803290, 7995995501984016450, 37314645675925410100,
  163006083742200475700, 668324943343021950370, 2577824781465941808570,
  9373908296239788394800, 32197337191432316660400, 1046413458721550291463 00,
  3222953452862374897706 04, 9420940862213095854833 04,
  2616928017281415515231400, 6916166902815169575968700,
  1740966151398301307054190 0, 4178318763355923136930056 0,
  9569697812847436862001096 0, 2093371396560376813562739 75,
  4377049283717151519267546 75, 8754098567434303038535093 50,
  1675784582908852295948146470, 3072271735332895875904935195,
  5397234129638871133346507775, 9090078534128625066688855200,
  1468397301666931741542045840 0, 2276015817583744199390171052 0,
  3386267435917277955190254492 0, 4837524908453254221700363560 0,
  6637534176714930211170266280 0, 8749476869306044369269896460 0,
  1108267070112098953440853551 60, 1349194694049511769406256497 60,
  1578844854738790368454129944 00, 1776200461581139164510896187 00,
  1921196417628579096307704039 00, 1998044274333722260160012200 56,
  1998044274333722260160012200 56, 1921196417628579096307704039 00,
  1776200461581139164510896187 00, 1578844854738790368454129944 00,
  1349194694049511769406256497 60, 1108267070112098953440853551 60,
  8749476869306044369269896460 0, 6637534176714930211170266280 0,
  4837524908453254221700363560 0, 3386267435917277955190254492 0,
  2276015817583744199390171052 0, 1468397301666931741542045840 0,
  9090078534128625066688855200, 5397234129638871133346507775,
  3072271735332895875904935195, 1675784582908852295948146470,
```

```
  875409856743430303853509350, 43770492837171515151926754675,
  20933713965603768135627397 5, 95696978128474368620010960,
  41783187633559231369300560, 17409661513983013070541900,
  69161669028151695759687 00, 26169280172814155152314 00,
  94209408622130958548330 4, 32229534528623748977060 4,
  10464134587215502914630 0, 32197337191432316660400, 9373908296239788394800,
  2577824781465941808570, 668324943343021950370, 163006083742200475700,
  37314645675925410100, 7995995501984016450, 1599199100396803290,
  297525414027312240, 51297485177122800, 8160963550905900, 1192050855750300,
  158940114100040, 19212541264840, 2088319702700, 202095455100, 17199613200,
  1267339920, 79208745, 4082925, 166650, 5050, 101]
```

4.3.11 SCStronglyConnectedComponents

◊ `SCStronglyConnectedComponents( complex )` (function)

Returns: a list of simplicial complexes upon success, `fail` otherwise.

Computes all strongly connected components of an arbitrary simplicial complex.

Example

```
gap> c:=SC([[1,2,3],[3,4,5],[4,5,6,7,8,9],[6,7,8,9,10,11]]);;
gap> comps:=SCStronglyConnectedComponents(c);
[ [SimplicialComplex

     Properties known: Dim, Facets, Name, VertexLabels.

     Name="Strongly connected component #1 of unnamed complex n20"
     Dim=2

    /SimplicialComplex], [SimplicialComplex

     Properties known: Dim, Facets, Name, VertexLabels.

     Name="Strongly connected component #2 of unnamed complex n20"
     Dim=5

    /SimplicialComplex], [SimplicialComplex

     Properties known: Dim, Facets, Name, VertexLabels.

     Name="Strongly connected component #3 of unnamed complex n20"
     Dim=5

    /SimplicialComplex] ]
gap> comps[1].Facets;
[ [ 1, 2, 3 ] ]
gap> comps[2].Facets;
[ [ 3, 4, 5 ], [ 4, 5, 6, 7, 8, 9 ] ]
gap> comps[3].Facets;
[ [ 6, 7, 8, 9, 10, 11 ] ]
gap>
```

4.4 Vertex labelings and label operations

Functions operating on the labels of a complex such as the name or the vertex labeling.

Internally, simpcomp uses the standard labeling $[1,\ldots,n]$. It is recommended to use simple vertex labels like integers end, whenever possible, the standard labeling, see also `SCRelabelStandard` (4.4.5).

4.4.1 SCLabelMax

◊ `SCLabelMax( complex )` (function)

Returns: maximal vertex label of `complex` upon success, `fail` otherwise.

The maximum over all vertex labels is determined by the gap function `MaximumList`.

Example

```
gap> c:=SCBdSimplex(3);;
gap> SCRelabel(c,[10,100,100000,3500]);;
gap> SCLabelMax(c);
100000
gap>
```

Example

```
gap> c:=SCBdSimplex(3);;
gap> SCRelabel(c,["a","bbb",5,[1,1]]);
true
gap> SCLabelMax(c);
"bbb"
gap>
```

4.4.2 SCLabelMin

◊ `SCLabelMin( complex )` (function)

Returns: minimal vertex label of `complex` upon success, `fail` otherwise.

The minimum over all vertex labels is determined by the gap function `MinimumList`.

Example

```
gap> c:=SCBdSimplex(3);;
gap> SCRelabel(c,[10,100,100000,3500]);;
gap> SCLabelMin(c);
10
gap>
```

Example

```
gap> c:=SCBdSimplex(3);;
gap> SCRelabel(c,["a","bbb",5,[1,1]]);
true
gap> SCLabelMin(c);
5
gap>
```

4.4.3 SCLabels

◊ SCLabels(complex) (function)

Returns: a list of vertex labels upon success, fail otherwise.

Returns the vertex labels of complex as a list.

Example

```
gap> c:=SCFromFacets(Combinations(["a","b","c","d"],3));;
gap> SCLabels(c);
[ "a", "b", "c", "d" ]
gap> c.Labels;
[ "a", "b", "c", "d" ]
```

4.4.4 SCRelabel

◊ SCRelabel(complex, maptable) (function)

Returns: true upon success, fail otherwise.

maptable has to be a list of length f_0. The function maps the i-th entry of maptable to the i-th entry of the current vertex labels. If complex has the standard vertex labeling $[1,\ldots,n]$ i is mapped to maptable[i].

Internally the property "VertexLabels" of complex is replaced by maptable.

Example

```
gap> SCLib.SearchByAttribute("F[1]=12");
[ [ 56, "S^2 (VT)" ], [ 57, "T^2 (VT)" ], [ 58, "T^2 (VT)" ],
  [ 59, "T^2 (VT)" ], [ 60, "T^2 (VT)" ], [ 61, "(T^2)#2 (VT)" ],
  [ 62, "(P^2)#4 (VT)" ], [ 63, "(T^2)#2 (VT)" ], [ 64, "(T^2)#2 (VT)" ],
gap> c:=SCLib.Load(56);;
gap> SCPropertyByName(c,"VertexLabels");
[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 ]
gap> SCRelabel(c,["a","b","c","d","e","f","g","h","i","j","k","l"]);
true
gap> SCLabels(c);
[ "a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l" ]
gap>
```

4.4.5 SCRelabelStandard

◊ SCRelabelStandard(complex) (function)

Returns: true upon success upon success, fail otherwise.

Maps vertex labels $v_1,\ldots,v_n$ of complex to $[1,\ldots,n]$.

Example

```
gap> SCLib.SearchByAttribute("F[1]=12");
[ [ 56, "S^2 (VT)" ], [ 57, "T^2 (VT)" ], [ 58, "T^2 (VT)" ],
  [ 59, "T^2 (VT)" ], [ 60, "T^2 (VT)" ], [ 61, "(T^2)#2 (VT)" ],
  [ 62, "(P^2)#4 (VT)" ], [ 63, "(T^2)#2 (VT)" ], [ 64, "(T^2)#2 (VT)" ],
gap> c:=SCLib.Load(56);;
gap> SCRelabel(c,[4..15]);
true
gap> SCVertices(c);
[ 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ]
```

```
gap> SCRelabelStandard(c);
true
gap> SCLabels(c);
[ 1 .. 12 ]
gap>
```

4.4.6 SCRelabelTransposition

◊ `SCRelabelTransposition( complex, pair )` (function)

Returns: `true` upon success, `fail` otherwise.

Permutes vertex labels of a single pair of vertices. `pair` has to be a list of length 2 and a sublist of the property "VertexLabels".

The function is equivalent to `SCRelabel` (4.4.4) with `maptable` $= [VertexLabels[1],\ldots,VertexLabels[i],\ldots,VertexLabels[j],\ldots,VertexLabels[n]]$ if `pair` $= [VertexLabels[j],VertexLabels[i]]$, $j \leq i$, $j \neq i$.

Example

```
gap> c:=SCBdSimplex(3);;
gap> SCVertices(c);
[ 1, 2, 3, 4 ]
gap> SCRelabelTransposition(c,[1,2]);;
gap> SCLabels(c);
[ 2, 1, 3, 4 ]
gap>
```

4.4.7 SCRename

◊ `SCRename( complex, name )` (function)

Returns: `true` upon success, `fail` otherwise.

Renames a simplicial complex.

Example

```
gap> c:=SCBdSimplex(5);;
gap> SCName(c);
"S^4_6"
gap> SCRename(c,"mySphere");
true
gap> SCName(c);
"mySphere"
```

4.4.8 SCSortComplex

◊ `SCSortComplex( complex )` (function)

Returns: a new simplicial complex from sorted facet list upon success, `fail` otherwise.

Sorts facet list (with arbitrary vertex labels) and returns a new simplicial complex object from sorted facets. In particular all known properties are lost by this operation.

Example

```
gap> c:=SCBdSimplex(3);
[SimplicialComplex

 Properties known: AutomorphismGroup, AutomorphismGroupOrder,
                   AutomorphismGroupStructure, AutomorphismGroupTransitivity,
                   Chi, Dim, F, Facets, Generators, HasBoundary, Homology,
                   IsConnected, IsStronglyConnected, Name, TopologicalType,
                   VertexLabels.

 Name="S^2_4"
 Dim=2
 AutomorphismGroupStructure="S4"
 AutomorphismGroupTransitivity=4
 Chi=2
 F=[ 4, 6, 4 ]
 Homology=[ [ 0, [ ] ], [ 0, [ ] ], [ 1, [ ] ] ]
 IsConnected=true
 IsStronglyConnected=true
 TopologicalType="S^2"

/SimplicialComplex]
gap> SCRelabel(c,[4,3,2,1]);;
gap> SCFacets(c);
[ [ 4, 3, 2 ], [ 4, 3, 1 ], [ 4, 2, 1 ], [ 3, 2, 1 ] ]
gap> d:=SCSortComplex(c);
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="unnamed complex n"
 Dim=2

/SimplicialComplex]
gap> SCFacets(d);
[ [ 1, 2, 3 ], [ 1, 2, 4 ], [ 1, 3, 4 ], [ 2, 3, 4 ] ]
gap>
```

4.4.9 SCUnlabelFace

◊ `SCUnlabelFace( complex, face )` (function)

Returns: a list upon success, `fail` otherwise.

Computes the standard labeling of `face` in `complex`.

Example

```
gap> c:=SCBdSimplex(3);;
gap> SCRelabel(c,["a","bbb",5,[1,1]]);;
gap> SCUnlabelFace(c,["a","bbb",5]);
[ 1, 2, 3 ]
gap>
```

4.5 Computing properties of simplicial complexes

The following functions compute some basic properties of a simplicial complex. In particular, the facets of the complex are not altered by these functions.

Note: Every simplicial complex is internally stored with the standard vertex labeling from 1 to n and a maptable to restore the original vertex labeling. Thus, we have to relabel some of the complex properties (facets, face lattice, generators, etc...) whenever we want to return them to the user. As a consequence, some of the functions exist twice, one of them with the appendix "Ex". These functions return the standard labeling whereas the other ones relabel the result to the original labeling.

4.5.1 SCAltshulerSteinberg

◊ `SCAltshulerSteinberg( complex )` (function)

Returns: a non-negative integer upon success, `fail` otherwise.

Computes the Altshuler-Steinberg determinant.

Definition: Let v_i, $1 \leq i \leq n$ be the vertices and let F_j, $1 \leq j \leq m$ be the facets of a pure simplicial complex C, then the determinant of $AS \in \mathbb{Z}^{n \times m}$, $AS_{ij} = 1$ if $v_i \in F_j$, $AS_{ij} = 0$ otherwise, is called the Altshuler-Steinberg-determinant.

Example

```
gap> SCLib.SearchByName("T^2");
[ [ 5, "T^2 (VT)" ], [ 7, "T^2 (VT)" ], [ 11, "T^2 (VT)" ],
  [ 12, "T^2 (VT)" ], [ 20, "T^2 (VT)" ], [ 22, "(T^2)#2" ],
  [ 27, "(T^2)#3" ], [ 41, "T^2 (VT)" ], [ 44, "(T^2)#4" ], ...
gap> torus:=SCLib.Load(5);;
gap> SCAltshulerSteinberg(torus);
73728
```

Example

```
gap> c:=SCBdSimplex(3);;
gap> SCAltshulerSteinberg(c);
9
gap> c:=SCBdSimplex(4);;
gap> SCAltshulerSteinberg(c);
16
gap> c:=SCBdSimplex(5);;
gap> SCAltshulerSteinberg(c);
25
```

4.5.2 SCAutomorphismGroup

◊ `SCAutomorphismGroup( complex )` (function)

Returns: a group upon success, `fail` otherwise

Computes the automorphism group of a strongly connected pseudomanifold `complex`. Necessarily the group is a subgroup of the symmetric group S_n where n is the number of vertices of the simplicial complex.

The function uses an efficient algorithm provided by the package GRAPE (see [Soi06], which is based on the program `nauty` by Brendan McKay [McK84]). If the package GRAPE is not available, this function call falls back to `SCAutomorphismGroupInternal` (4.5.3).

Example

```
gap> SCLib.SearchByName("K3");
[ [ 584, "K3 surface" ] ]
gap> k3surf:=SCLib.Load(584);;
gap> SCAutomorphismGroup(k3surf);
((C2 x C2 x C2 x C2) : C5) : C3
gap>
```

4.5.3 SCAutomorphismGroupInternal

◊ SCAutomorphismGroupInternal(complex) (function)

Returns: permutation group in structure description upon success, fail otherwise.

Computes the group of all automorphisms on the set of vertices of a simplicial complex that do not change the complex as a whole. Necessarily the group is a subgroup of S_n where n is the number of vertices of the simplicial complex.

The group is given in structure description, provided by the GAP function StructureDescription(). Note, that this is not always unique, since every non trivial semi-direct product is denoted by ":".

Example

```
gap> c:=SCBdSimplex(5);;
gap> SCAutomorphismGroupInternal(c);
S6
```

Example

```
gap> c:=SCBdSimplex(2);;
gap> g:=SCAutomorphismGroupInternal(c);
S3
gap> List(g);
[(), (1,3), (1,2,3), (2,3), (1,3,2), (1,2)]
```

4.5.4 SCBoundary

◊ SCBoundary(complex) (function)

Returns: a new simplicial complex upon success, fail otherwise.

Computes the boundary of a combinatorial pseudomanifold. The boundary is returned as a simplicial complex and stored as a property of complex.

If the complex is not a pseudomanifold the function returns fail.

Example

```
gap> c:=SC([[1,2,3,4],[1,2,3,5],[1,2,4,5],[1,3,4,5]]);
[SimplicialComplex

Properties known: Dim, Facets, Name, VertexLabels.

Name="unnamed complex n"
Dim=3

/SimplicialComplex]
```

```
gap> SCBoundary(c);
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="Bd(unnamed complex 5)"
 Dim=2

/SimplicialComplex]
gap> c;
[SimplicialComplex

 Properties known: Boundary, Dim, Faces, Facets, HasBoundary, Name,
                   VertexLabels.

 Name="unnamed complex 5"
 Dim=3
 HasBoundary=true

/SimplicialComplex]
gap>
```

4.5.5 SCDim

◊ `SCDim( complex )` (function)

Returns: an integer value ≥ -1 upon success, `fail` otherwise.

Computes the dimension of a simplicial complex. If the complex is not pure, the dimension of the highest dimensional simplex is returned.

Example

```
gap> complex:=SC([[1,2,3], [1,2,4], [1,3,4], [2,3,4]]);;
gap> SCDim(complex);
2
gap>
```

Example

```
gap> c:=SC([[1], [2,4], [3,4], [5,6,7,8]]);;
gap> SCDim(c);
3
gap>
```

4.5.6 SCDualGraph

◊ `SCDualGraph( complex )` (function)

Returns: A 1-dimensional simplicial complex upon success, `fail` otherwise

Computes the dual graph of `complex`.

Example

```
gap> sphere:=SCBdSimplex(5);;
gap> graph:=SCFaces(sphere,1);
```

```
[ [ 1, 2 ], [ 1, 3 ], [ 1, 4 ], [ 1, 5 ], [ 1, 6 ], [ 2, 3 ], [ 2, 4 ],
  [ 2, 5 ], [ 2, 6 ], [ 3, 4 ], [ 3, 5 ], [ 3, 6 ], [ 4, 5 ], [ 4, 6 ],
  [ 5, 6 ] ]
gap> graph:=SC(graph);;
gap> dualGraph:=SCDualGraph(sphere);
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="dual graph of S^4_6"
 Dim=1

/SimplicialComplex]
gap> graph.Facets = dualGraph.Facets;
true
gap>
```

4.5.7 SCEulerCharacteristic

◊ `SCEulerCharacteristic( complex )` (function)

Returns: integer upon success, `fail` otherwise.

Computes the Euler characteristic of a simplicial complex C

$$\chi(C) = \sum_{i=0}^{d} (-1)^i f_i$$

where f_i denotes the i-th component of the f-vector.

Example

```
gap> complex:=SCFromFacets([[1,2,3], [1,2,4], [1,3,4], [2,3,4]]);;
gap> SCEulerCharacteristic(complex);
2
```

Example

```
gap> s2:=SCBdSimplex(3);;
gap> s2.EulerCharacteristic;
2
```

4.5.8 SCFVector

◊ `SCFVector( complex )` (function)

Returns: a list of integers upon success, `fail` otherwise.

Computes the f-vector of a simplicial complex. I. e. the number of i-dimensional faces for $0 \leq i \leq d$, where d is the dimension of the complex.

Example

```
gap> complex:=SC([[1,2,3], [1,2,4], [1,3,4], [2,3,4]]);;
gap> SCFVector(complex);
[4, 6, 4]
gap>
```

4.5.9 SCFaceLattice

◊ SCFaceLattice(complex) (function)

Returns: a list of face lists upon success, fail otherwise.

Computes the entire face lattice of a d-dimensional simplicial complex, i. e. all of its i-skeletons for $0 \leq i \leq d$. The faces are returned in the original labeling.

Example

```
gap> c:=SC([["a","b","c"],["a","b","d"], ["a","c","d"], ["b","c","d"]]);;
gap> SCFaceLattice(c);
[ [ ["a"], ["b"], ["c"], ["d"] ],
  [ ["a", "b"], ["a", "c"], ["a", "d"],
          ["b", "c"], ["b", "d"], ["c", "d"] ],
  [ ["a", "b", "c"], ["a", "b", "d"],
    ["a", "c", "d"], ["b", "c", "d"] ] ]
gap>
```

4.5.10 SCFaceLatticeEx

◊ SCFaceLatticeEx(complex) (function)

Returns: a list of face lists upon success, fail otherwise.

Computes the entire face lattice of a d-dimensional simplicial complex, i. e. all of its i-skeletons for $0 \leq i \leq d$. The faces are returned in the standard labeling.

Example

```
gap> c:=SC([["a","b","c"],["a","b","d"], ["a","c","d"], ["b","c","d"]]);;
gap> SCFaceLatticeEx(c);
[ [ [1], [2], [3], [4] ],
  [ [1, 2], [1, 3], [1, 4], [2, 3], [2, 4], [3, 4] ],
  [ [1, 2, 3], [1, 2, 4], [1, 3, 4], [2, 3, 4] ] ]
gap>
```

4.5.11 SCFaces

◊ SCFaces(complex, dim) (function)

Returns: a face list upon success, fail otherwise.

This is a synonym of the function SCSkel (4.5.44).

4.5.12 SCFacesEx

◊ SCFacesEx(complex, dim) (function)

Returns: a face list upon success, fail otherwise.

This is a synonym of the function SCSkelEx (4.5.45).

4.5.13 SCFacets

◊ SCFacets(complex) (function)

Returns: a facet list upon success, fail otherwise.

Returns the facets of a simplicial complex in the original vertex labeling.

Example

```
gap> c:=SC([[2,3],[3,4],[4,2]]);;
gap> SCFacets(c);
[ [ 2, 3 ], [ 3, 4 ], [ 2, 4 ] ]
gap>
```

4.5.14 SCFacetsEx

◊ SCFacetsEx(complex) (function)

Returns: a facet list upon success, fail otherwise.

Returns the facets of a simplicial complex as they are stored, i. e. with standard vertex labeling from 1 to n.

Example

```
gap> c:=SC([[2,3],[3,4],[4,2]]);;
gap> SCFacetsEx(c);
[ [ 1, 2 ], [ 2, 3 ], [ 1, 3 ] ]
gap>
```

4.5.15 SCFpBettiNumbers

◊ SCFpBettiNumbers(complex, p) (function)

Returns: a list of non-negative integers upon success, fail otherwise.

Computes the $\mathbb{F}_p$ Betti numbers of a simplicial complex for any prime number p.

Example

```
gap> SCLib.SearchByName("K^2");
[ [ 19, "K^2 (VT)" ], [ 230, "K^2 (VT)" ] ]
gap> kleinBottle:=SCLib.Load(19);;
gap> SCHomology(kleinBottle);
[ [ 0, [  ] ], [ 1, [ 2 ] ], [ 0, [  ] ] ]
gap> SCFpBettiNumbers(kleinBottle,2);
[ 1, 2, 1 ]
gap> SCFpBettiNumbers(kleinBottle,3);
[ 1, 1, 0 ]
gap>
```

4.5.16 SCFundamentalGroup

◊ SCFundamentalGroup(complex) (function)

Returns: an fp group upon success, fail otherwise.

Computes the first fundamental group of complex which must be a connected simplicial complex and returns it in form of a finitely presented group. The generators of the group are given as 2-tuples that correspond to the edges of complex in standard labeling. You can use GAP's SimplifiedFpGroup to simplify the group presenation.

Example

```
# an RP^2
gap> c:=SC([[ 1, 2, 3 ], [ 1, 2, 6 ], [ 1, 3, 5 ], [ 1, 4, 5 ], [ 1, 4, 6 ],
          [ 2, 3, 4 ], [ 2, 4, 5 ], [ 2, 5, 6 ], [ 3, 4, 6 ], [ 3, 5, 6 ]]);;
```

```
gap> g:=SCFundamentalGroup(c);
<fp group on the generators [ [2,3], [2,4], [2,5], [2,6], [3,4], [3,5],
  [3,6], [4,5], [4,6], [5,6] ]>
gap> Order(g);
2
gap> Elements(g);
[ <identity ...>, [2,4] ]
```

4.5.17 SCGVector

◊ `SCGVector( complex )` (function)

Returns: a list of non-negative integers upon success, `fail` otherwise.

Computes the g-vector of a simplicial complex. The g-vector is defined as follows:

Let h be the h-vector of a d-dimensional simplicial complex C, then

$h_{-1} := 1; \quad g_i := h_i - h_{i-1}; \quad \frac{d-1}{2} \geq i \geq 0$

is called the g-vector of C. For the definition of the h-vector see `SCHVector` (4.5.20). The information contained in g suffices to determine the f-vector of C.

Example

```
gap> SCLib.SearchByName("RP^2");
[ [ 3, "RP^2 (VT)" ], [ 284, "RP^2xS^1" ] ]
gap> rp2_6:=SCLib.Load(3);;
gap> SCFVector(rp2_6);
[ 6, 15, 10 ]
gap> SCHVector(rp2_6);
[ 3, 6, 0 ]
gap> SCGVector(rp2_6);
[ 2 ]
gap>
```

4.5.18 SCGenerators

◊ `SCGenerators( complex )` (function)

Returns: a list of pairs of the form `[ list, integer ]` upon success, `fail` otherwise.

Computes the generators of a simplicial complex in the original vertex labeling.

The generating set of a simplicial complex is a list of simplices that will generate the complex by uniting their G-orbits if G is the group of automorphisms of `complex`.

The function will return the simplices together with the length of the their orbits.

Example

```
gap> SCLib.SearchByName("T^2");
[ [ 5, "T^2 (VT)" ], [ 7, "T^2 (VT)" ], [ 11, "T^2 (VT)" ],
  [ 12, "T^2 (VT)" ], [ 20, "T^2 (VT)" ], [ 22, "(T^2)#2" ],
  [ 27, "(T^2)#3" ], [ 41, "T^2 (VT)" ], [ 44, "(T^2)#4" ], ...
gap> torus:=SCLib.Load(5);;
gap> SCGenerators(torus);
[ [ [ 1, 2, 4 ], 14 ] ]
gap>
```

Example

```
gap> SCLib.SearchByName("K3");
[ [ 584, "K3 surface" ] ]
gap> SCLib.Load(584);
[SimplicialComplex

 Properties known: AltshulerSteinberg, AutomorphismGroup,
                   AutomorphismGroupSize, AutomorphismGroupStructure,
                   AutomorphismGroupTransitivity, Boundary, Chi,
                   ConnectedComponents, Dim, DualGraph, F, Faces, Facets, G,
                   Generators, H, HasBoundary, HasInterior, Homology,
                   Interior, IsCentrallySymmetric, IsConnected,
                   IsEulerianManifold, IsOrientable, IsPM, IsPure,
                   MinimalNonFaces, Name, Neighborliness, Orientation,
                   StronglyConnected, VertexLabels, Vertices.

 Name="K3 surface"
 Dim=4
 AutomorphismGroupSize=240
 AutomorphismGroupStructure="((C2 x C2 x C2 x C2) : C5) : C3"
 AutomorphismGroupTransitivity=2
 Chi=24
 F=[ 16, 120, 560, 720, 288 ]
 G=[ 10, 55 ]
 H=[ 11, 66, 286, -99, 23 ]
 HasBoundary=false
 HasInterior=true
 Homology=[ [ 0, [ ] ], [ 0, [ ] ], [ 22, [ ] ], [ 0, [ ] ], [ 1, [ ] ] ]
 IsCentrallySymmetric=false
 IsConnected=true
 IsEulerianManifold=true
 IsOrientable=true
 IsPM=true
 IsPure=true
 Neighborliness=3

/SimplicialComplex]
gap> SCGenerators(last);
[ [ [ 1, 2, 3, 8, 12 ], 240 ], [ [ 1, 2, 5, 8, 14 ], 48 ] ]
gap>
```

4.5.19 SCGeneratorsEx

◊ `SCGeneratorsEx( complex )` (function)

Returns: a list of pairs of the form `[ list, integer ]` upon success, `fail` otherwise.

Computes the generators of a simplicial complex in the standard vertex labeling.

The generating set of a simplicial complex is a list of simplices that will generate the complex by uniting their G-orbits if G is the group of automorphisms of `complex`.

The function will return the simplices together with the length of the their orbits.

Example

```
gap> SCLib.SearchByName("T^2");
```

```
[ [ 5, "T^2 (VT)" ], [ 7, "T^2 (VT)" ], [ 11, "T^2 (VT)" ],
  [ 12, "T^2 (VT)" ], [ 20, "T^2 (VT)" ], [ 22, "(T^2)#2" ],
  [ 27, "(T^2)#3" ], [ 41, "T^2 (VT)" ], [ 44, "(T^2)#4" ], ...
gap> torus:=SCLib.Load(5);;
gap> SCGeneratorsEx(torus);
[ [ [ 1, 2, 4 ], 14 ] ]
gap>
```

Example

```
gap> SCLib.SearchByName("K3");
[ [ 584, "K3 surface" ] ]
gap> SCLib.Load(584);
[SimplicialComplex

 Properties known: AltshulerSteinberg, AutomorphismGroup,
                   AutomorphismGroupSize, AutomorphismGroupStructure,
                   AutomorphismGroupTransitivity, Boundary, Chi,
                   ConnectedComponents, Dim, DualGraph, F, Faces, Facets, G,
                   Generators, H, HasBoundary, HasInterior, Homology,
                   Interior, IsCentrallySymmetric, IsConnected,
                   IsEulerianManifold, IsOrientable, IsPM, IsPure,
                   MinimalNonFaces, Name, Neighborliness, Orientation,
                   StronglyConnected, VertexLabels, Vertices.

 Name="K3 surface"
 Dim=4
 AutomorphismGroupSize=240
 AutomorphismGroupStructure="((C2 x C2 x C2 x C2) : C5) : C3"
 AutomorphismGroupTransitivity=2
 Chi=24
 F=[ 16, 120, 560, 720, 288 ]
 G=[ 10, 55 ]
 H=[ 11, 66, 286, -99, 23 ]
 HasBoundary=false
 HasInterior=true
 Homology=[ [ 0, [ ] ], [ 0, [ ] ], [ 22, [ ] ], [ 0, [ ] ], [ 1, [ ] ] ]
 IsCentrallySymmetric=false
 IsConnected=true
 IsEulerianManifold=true
 IsOrientable=true
 IsPM=true
 IsPure=true
 Neighborliness=3

/SimplicialComplex]
gap> SCGeneratorsEx(last);
[ [ [ 1, 2, 3, 8, 12 ], 240 ], [ [ 1, 2, 5, 8, 14 ], 48 ] ]
gap>
```

4.5.20 SCHVector

◊ `SCHVector( complex )` (function)

Returns: a list of integers upon success, `fail` otherwise.

Computes the h-vector of a simplicial complex. The h-vector is defined as follows:

Let $f \in \mathbb{N}^{d+1}$ be the f-vector of a d-dimensional simplicial complex, i. e. the number of its i-faces for $0 \leq i \leq d$ C, and let $p(C,x)$ be the polynomial

$p(C,x) = x^{d+1} + f_0 x^d + f_1 + x^{d-1} + \ldots + f_{d-1}x + f_d$

of degree $d+1$. Then $h = (h_0, \ldots, h_d) \in \mathbb{Z}^{d+1}$ is given by

$q(C,x) := p(C,x-1) = x^{d+1} + h_0 x^d + h_1 + x^{d-1} + \ldots + h_{d-1}x + h_d$.

This results into the following formula

$h_k := (-1)^{k+1}\binom{d+1}{k+1} + \sum_{i=0}^{k} (-1)^{k-i}\binom{d-i}{d-k} f_i$.

Example

```
gap> SCLib.SearchByName("RP^2");
[ [ 3, "RP^2 (VT)" ], [ 284, "RP^2xS^1" ] ]
gap> rp2_6:=SCLib.Load(3);;
gap> SCFVector(rp2_6);
[ 6, 15, 10 ]
gap> SCHVector(rp2_6);
[ 3, 6, 0 ]
gap>
```

4.5.21 SCHasBoundary

◊ `SCHasBoundary( complex )` (function)

Returns: `true` / `false` upon success, `fail` otherwise.

Checks, if a combinatorial pseudomanifold `complex` has a boundary. If the complex is closed `false` is returned, if the complex is not a pseudomanifold, `fail` is returned.

Example

```
gap> SCLib.SearchByName("K^2");
[ [ 19, "K^2 (VT)" ], [ 230, "K^2 (VT)" ] ]
gap> kleinBottle:=SCLib.Load(19);;
gap> SCHasBoundary(kleinBottle);
false
gap>
```

Example

```
gap> c:=SC([[1,2,3,4],[1,2,3,5],[1,2,4,5],[1,3,4,5]]);;
gap> SCHasBoundary(c);
true
```

4.5.22 SCHasInterior

◊ `SCHasInterior( )` (function)

Returns: `true` / `false` upon success, `fail` otherwise.

Returns `true` if there exist at least one $(d-1)$-face of `complex` that is not in its boundary.

Example
```
gap> c:=SC([[1,2,3,4],[1,2,3,5],[1,2,4,5],[1,3,4,5]]);;
gap> SCHasInterior(c)
true
gap> c:=SC([[1,2,3,4]]);;
gap> SCHasInterior(c);
false
```

4.5.23 SCHomology

◊ SCHomology(complex) (function)

Returns: a list of pairs of the form [integer, list].

Computes the integral simplicial homology groups for a simplicial complex (internally calls SimplicialHomology(complex.FacetsEx) from the homology package, version 1.4.2., see [DHSW04].

If the homology package is not available, this function call falls back to SCHomologyInternal (5.1.5). The output is a list of homology groups of the form $[H_0,....,H_d]$, where d is the dimension of complex. The format of the homology groups H_i is given in terms of their maximal cyclic subgroups, i.e. a homology group $H_i \cong \mathbb{Z}^f + \mathbb{Z}/t_1\mathbb{Z} \times \cdots \times \mathbb{Z}/t_n\mathbb{Z}$ is returned in form of a list $[f,[t_1,...,t_n]]$, where f is the (integer) free part of H_i and t_i denotes the torsion parts of H_i ordered in weakly incresing size.

Example
```
gap> SCLib.SearchByName("K^2");
[ [ 19, "K^2 (VT)" ], [ 230, "K^2 (VT)" ] ]
gap> kleinBottle:=SCLib.Load(19);;
gap> kleinBottle.Homology;
[ [ 0, [  ] ], [ 1, [ 2 ] ], [ 0, [  ] ] ]
gap>
```

Example
```
gap> SCLib.SearchByName("L_");
[ [ 139, "L_3_1" ], [ 282, "L_4_1" ], [ 300, "L_5_2" ],
  [ 411, "(S^2xS^1)#L_3_1" ], [ 412, "(S^2~S^1)#L_3_1" ], [ 444, "L_5_1" ],
  [ 445, "(S^2xS^1)#2#L_3_1" ], [ 446, "(S^2~S^1)#2#L_3_1" ], ...
gap> c:=SCConnectedSum(SCConnectedProduct(SCLib.Load(446),3),
 SCConnectedProduct(SCLib.Load(445),2));
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels, Vertices.

 Name="(S^2~S^1)#2#L_3_1#+-(S^2~S^1)#2#L_3_1#+-(S^2~S^1)#2#L_3_1#+-(S^2xS^1)#
  2#L_3_1#+-(S^2xS^1)#2#L_3_1"
 Dim=3

/SimplicialComplex]
gap> SCHomology(c);
[ [ 0, [  ] ], [ 10, [ 3, 3, 3, 3, 3 ] ], [ 9, [ 2 ] ], [ 0, [  ] ] ]
gap> SCFpBettiNumbers(c,2);
[ 1, 10, 10, 1 ]
gap> SCFpBettiNumbers(c,3);
[ 1, 15, 14, 0 ]
```

```
gap>
```

4.5.24 SCIncidences

◊ `SCIncidences( complex, dim )` (function)

Returns: a list of face lists upon success, `fail` otherwise.

Returns an array of all `dim`-faces of `complex`. The array is sorted by the incidence of the face in the (`dim`+1)-skeleton of the complex, i. e. the *i*-th entry of the array contains all `dim`-faces with incidence *i*. The faces are returned in the original labeling.

Example

```
gap> c:=SC([[1,2,3],[2,3,4],[3,4,5],[4,5,6],[1,5,6],[1,4,6],[2,3,6]]);;
gap> SCIncidences(c,1);
[ [ [1, 2], [1, 3], [1, 4], [1, 5],
    [2, 4], [2, 6], [3, 5], [3, 6] ],
  [ [1, 6], [3, 4], [4, 5], [4, 6], [5, 6] ],
  [ [2, 3] ] ]
```

4.5.25 SCIncidencesEx

◊ `SCIncidencesEx( complex, dim )` (function)

Returns: a list of face lists upon success, `fail` otherwise.

Returns an array of all `dim`-faces of `complex`. The array is sorted by the incidence of the face in the (`dim`+1)-skeleton of the complex, i. e. the *i*-th entry of the array contains all `dim`-faces with incidence *i*. The faces are returned in the standard labeling.

Example

```
gap> c:=SC([[1,2,3],[2,3,4],[3,4,5],[4,5,6],[1,5,6],[1,4,6],[2,3,6]]);;
gap> SCIncidences(c,1);
[ [ [1, 2], [1, 3], [1, 4], [1, 5],
    [2, 4], [2, 6], [3, 5], [3, 6] ],
  [ [1, 6], [3, 4], [4, 5], [4, 6], [5, 6] ],
  [ [2, 3] ] ]
```

4.5.26 SCInterior

◊ `SCInterior( complex )` (function)

Returns: a new simplicial complex upon success, `fail` otherwise.

Computes all $(d-1)$-faces of a combinatorial d-pseudomanifold C that occur in exactly two facets, i. e. returns the part of the $(d-1)$-skeleton of C that is not part of the boundary.

Example

```
gap> c:=SC([[1,2,3,4],[1,2,3,5],[1,2,4,5],[1,3,4,5]]);;
gap> SCInterior(c).Facets;
[[1, 2, 3], [1, 2, 4], [1, 2, 5], [1, 3, 4], [1, 3, 5],
  [1, 4, 5]]
gap> c:=SC([[1,2,3,4]]);;
gap> SCInterior(c).Facets;
[]
```

4.5.27 SCIsCentrallySymmetric

◊ `SCIsCentrallySymmetric( complex )` (function)

Returns: `true` / `false` upon success, `fail` otherwise.

Checks, if a simplicial complex is centrally symmetric, i. e. if its automorphism group contains a fixed point free involution.

Example

```
gap> c:=SCBdCrossPolytope(4);;
gap> SCIsCentrallySymmetric(c);
true
```

Example

```
gap> c:=SCBdSimplex(4);;
gap> SCIsCentrallySymmetric(c);
false
```

4.5.28 SCIsConnected

◊ `SCIsConnected( complex )` (function)

Returns: `true` / `false` upon success, `fail` otherwise.

Checks, if a simplicial complex is connected.

Example

```
gap> c:=SCBdSimplex(1);;
gap> SCIsConnected(c);
false
gap> c:=SCBdSimplex(2);;
gap> SCIsConnected(c);
true
```

4.5.29 SCIsEmpty

◊ `SCIsEmpty( complex )` (function)

Returns: `true` / `false` upon success, `fail` otherwise.

Checks if `complex` is empty

Example

```
gap> c:=SC([[1]]);;
gap> SCIsEmpty(c);
false
```

Example

```
gap> c:=SC([]);;
gap> SCIsEmpty(c);
true
gap> c:=SC([[]]);;
gap> SCIsEmpty(c);
true
gap>
```

4.5.30 SCIsEulerianManifold

◊ `SCIsEulerianManifold( complex )` (function)

Returns: `true` / `false` upon success, `fail` otherwise.

Checks whether a given simplicial complex is a Eulerian manifold or not, i. e. checks, if all vertex links of a combinatorial pseudomanifold have the Euler characteristic of a sphere. In particular the function returns `false` in case of a bounded manifold.

Example

```
gap> c:=SCBdSimplex(4);;
gap> SCIsEulerianManifold(c);
true
gap> moebius:=SCLib.Load(6); # a moebius strip
gap> SCIsEulerianManifold(moebius);
false
gap>
```

4.5.31 SCIsHomologySphere

◊ `SCIsHomologySphere( complex )` (function)

Returns: `true` or `false` upon success, `fail` otherwise.

Checks, whether the `complex` given is a homology sphere, or not.

Example

```
gap> c:=SC([[2,3],[3,4],[4,2]]);;
gap> SCIsHomologySphere(c);
true
```

4.5.32 SCIsInKd

◊ `SCIsInKd( complex, k )` (function)

Returns: an integer upon success, `fail` otherwise.

Checks, whether the complex is in the class $K^k(d)$ of simplicial complexes that only have k-stacked spheres as vertex links. Returns the minimal determined k for which the membership was tested or 0 if the membership test could not be decided for the given complex. Internally calls `SCIsKStackedSphere` (6.2.5) for all links. Please note that this is a radomized algorithm that may give an indefinite answer to the membership problem.

Example

```
gap> SCLib.SearchByName("S^2~S^1");
[ [ 14, "S^2~S^1 (VT)" ], [ 29, "S^2~S^1 (VT)" ], [ 34, "S^2~S^1 (VT)" ],
  [ 42, "S^2~S^1 (VT)" ], [ 48, "S^2~S^1 (VT)" ], [ 49, "S^2~S^1 (VT)" ],
  [ 84, "S^2~S^1 (VT)" ], [ 87, "S^2~S^1 (VT)" ], [ 90, "(S^2~S^1)#2" ], ...
gap> c:=SCLib.Load(14);;
gap> c.AutomorphismGroup;
D18
gap> SCIsInKd(c,1);
#I  SCIsInKd: checking link 1/9
#I  SCIsKStackedSphere: try 1/50
round 0: [ 7, 15, 10 ]
round 1: [ 6, 12, 8 ]
```

```
round 2: [ 5, 9, 6 ]
round 3: [ 4, 6, 4 ]
Computed locally minimal complex after 4 rounds.
#I  SCIsInKd: complex has transitive automorphism group, all links are
1-stacked.
1
```

4.5.33 SCIsKNeighborly

◊ `SCIsKNeighborly( complex, k )` (function)

Returns: `true` / `false` upon success, `fail` otherwise.

Checks, if a simplicial complex is k-neighborly, i. e. if it contains all possible $\binom{f_0}{k}$ $(k-1)$-simplices, where f_0 denotes the number of simplices.

Example

```
gap> SCLib.SearchByName("RP^2");
[ [ 3, "RP^2 (VT)" ], [ 284, "RP^2xS^1" ] ]
gap> rp2_6:=SCLib.Load(3);;
gap> SCFVector(rp2_6);
[6, 15, 10]
gap> SCIsKNeighborly(rp2_6,2);
true
gap> SCIsKNeighborly(rp2_6,3);
false
```

4.5.34 SCIsOrientable

◊ `SCIsOrientable( complex )` (function)

Returns: `true` / `false` upon success, `fail` otherwise.

Checks, if a combinatorial pseudomanifold is orientable.

Example

```
gap> c:=SCBdCrossPolytope(4);;
gap> SCIsOrientable(c);
true
```

4.5.35 SCIsPseudoManifold

◊ `SCIsPseudoManifold( complex )` (function)

Returns: `true` / `false` upon success, `fail` otherwise.

Checks, if every $(d-1)$-face of `complex` is contained in exactly 2 facets.

Example

```
# Two 2-spheres glued together at [1]
gap> c:=SC([[1,2,3],[1,2,4],[1,3,4],[2,3,4],[1,5,6],[1,5,7],[1,6,7],[5,6,7]]);;
gap> SCIsPseudoManifold(c);
true
# Two circles glued together a 1
gap> c:=SC([[1,2],[2,3],[3,1],[1,4],[4,5],[5,1]]);;
gap> SCIsPseudoManifold(c);
```

```
false
```

4.5.36 SCIsPure

◊ `SCIsPure( complex )` (function)

Returns: a pair of the form `[ integer, boolean ]` upon success, `fail` otherwise.

Checks, if a simplicial complex is pure.

Example

```
gap> c:=SC([[1,2], [1,4], [2,4], [2,3,4]]);;
gap> SCIsPure(c);
false
gap>
```

Example

```
gap> c:=SC([[1,2], [1,4], [2,4]]);;
gap> SCIsPure(c);
true
```

4.5.37 SCIsShellable

◊ `SCIsShellable( complex )` (function)

Returns: `true` / `false` upon success, `fail` otherwise.

Checks, if a bounded simplical complex is shellable. See [Zie95], [Pac87] to learn more about shellings.

Example

```
gap> c:=SCBdCrossPolytope(4);;
gap> c:=Difference(c,SC([[1,3,5,7]]));; # bounded version
gap> SCIsShellable(c);
true
gap>
```

4.5.38 SCIsStronglyConnected

◊ `SCIsStronglyConnected( complex )` (function)

Returns: `true` / `false` upon success, `fail` otherwise.

Checks, if a simplicial complex is strongly connected, i. e. if for any pair of facets $(\hat{\Delta},\tilde{\Delta})$ there exists a sequence of facets $(\Delta_1,\ldots,\Delta_n)$ with $\Delta_1=\hat{\Delta}$ and $\Delta_n=\tilde{\Delta}$ and $\dim(\Delta_i,\Delta_{i+1})=d-1$ for all $1\leq i\leq n-1$.

Example

```
# Two 2-spheres, glued along [1]
gap> c:=SC([[1,2,3],[1,2,4],[1,3,4],[2,3,4],[1,5,6],[1,5,7],[1,6,7],[5,6,7]]);
gap> SCIsConnected(c);
true
gap> SCIsStronglyConnected(c);
false
gap>
```

4.5.39 SCMinimalNonFaces

◊ SCMinimalNonFaces(complex) (function)

Returns: a list of face lists upon success, fail otherwise.

Computes all missing proper faces of complex by calling SCMinimalNonFacesEx (4.5.40). The simplices are returned in the original labeling of complex.

Example

```
gap> c:=SCFromFacets(["abc","abd"]);;
gap> SCMinimalNonFaces(c);
[ [  ], [ "cd" ] ]
gap>
```

4.5.40 SCMinimalNonFacesEx

◊ SCMinimalNonFacesEx(complex) (function)

Returns: a list of face lists upon success, fail otherwise.

Computes all missing proper faces of complex, i.e. the missing $(i+1)$-tuples in the i-dim. skeleton of a complex in minimal dimension. A missing $(i+1)$-tuple is not listed if it only consists of missing i-tuples. Note, that whenever complex is k-neighborly the first $k+1$ entries are empty. Note that the simplices returned are in standard labeling $1,\ldots,n$, where n is the number of vertices of complex.

Example

```
gap> SCLib.SearchByName("T^2");
[ [ 5, "T^2 (VT)" ], [ 7, "T^2 (VT)" ], [ 11, "T^2 (VT)" ],
  [ 12, "T^2 (VT)" ], [ 20, "T^2 (VT)" ], [ 22, "(T^2)#2" ],
  [ 27, "(T^2)#3" ], [ 41, "T^2 (VT)" ], [ 44, "(T^2)#4" ], ...
gap> torus:=SCLib.Load(5);;
gap> SCFVector(torus);
[7, 21, 14]
gap> SCMinimalNonFacesEx(torus);
[ [  ], [  ] ]
gap> SCMinimalNonFacesEx(SCBdCrossPolytope(4));
[ [  ], [  ],
  [ [ 1 .. 3 ], [ 1, 2, 4 ], [ 1, 2, 5 ], [ 1, 2, 6 ], [ 1, 2, 7 ],
    [ 1, 2, 8 ], [ 1, 3, 4 ], [ 1, 5, 6 ], [ 1, 7, 8 ], [ 2, 3, 4 ],
    [ 2, 5, 6 ], [ 2, 7, 8 ], [ 3, 4, 5 ], [ 3, 4, 6 ], [ 3, 4, 7 ],
    [ 3, 4, 8 ], [ 3, 5, 6 ], [ 3, 7, 8 ], [ 4, 5, 6 ], [ 4, 7, 8 ],
    [ 5, 6, 7 ], [ 5, 6, 8 ], [ 5, 7, 8 ], [ 6, 7, 8 ] ] ]
gap>
```

4.5.41 SCName

◊ SCName(complex) (function)

Returns: a string upon success, fail otherwise.

Returns the name of complex.

Example

```
gap> c:=SCBdSimplex(5);;
gap> SCName(c);
"S^4_6"
```

Example
```
gap> c:=SC([[1,2],[2,3],[3,1]]);;
gap> SCName(c);
"unnamed complex n"
gap>
```

4.5.42 SCNeighborliness

◊ `SCNeighborliness( complex )` (function)

Returns: a positive integer upon success, `fail` otherwise.

Returns k if a simplicial complex is k-neighborly but not $(k+1)$-neighborly. See also `SCIsKNeighborly` (4.5.33)
Note, that every complex is at least 1-neighborly.

Example
```
gap> c:=SCBdSimplex(4);;
gap> SCNeighborliness(c);
4
gap> c:=SCBdCrossPolytope(4);;
gap> SCNeighborliness(c);
1
gap> SCLib.SearchByAttribute("F[3]=Binomial(F[1],3)");
[ [ 10, "S^5 (VT)" ], [ 17, "CP^2 (VT)" ], [ 18, "S^5 (VT)" ],
  [ 38, "S^5 (VT)" ], [ 39, "S^6 (VT)" ], [ 40, "S^7 (VT)" ],
  [ 53, "S^5 (VT)" ], [ 54, "S^5 (VT)" ], [ 55, "S^7 (VT)" ], ...
gap> cp2:=SCLib.Load(17);;
gap> SCNeighborliness(cp2);
3
```

4.5.43 SCOrientation

◊ `SCOrientation( complex )` (function)

Returns: a list of the type $\{\pm 1\}^{f_d}$ / `[ ]` upon success, `fail` otherwise.

This function tries to compute an orientation of a pure simplicial complex that fulfills the pseudomanifold property. If `complex` is orientable, an orientation in form of a list of orientations for the facets of `complex` is returned, otherwise an empty set.

Example
```
gap> c:=SCBdCrossPolytope(4);;
gap> SCOrientation(c);
[ 1, -1, -1, 1, -1, 1, 1, -1, -1, 1, 1, -1, 1, -1, -1, 1 ]
```

4.5.44 SCSkel

◊ `SCSkel( complex, dim )` (function)

Returns: a face list upon success, `fail` otherwise.

Computes the `dim`-skeleton of a simplcial complex, i. e. all `dim`-faces of `complex`, if `dim` is an integer value. If `dim` is a list, a list of all `dim[i]`-faces of `complex` for each entry `dim[i]` (which has to be an integer) is returned. The faces are returned in the original labeling.

Example

```
gap> SCLib.SearchByName("RP^2");
[ [ 3, "RP^2 (VT)" ], [ 284, "RP^2xS^1" ] ]
gap> rp2_6:=SCLib.Load(3);;
gap> rp2_6:=SC(rp2_6.Facets+10);;
gap> SCSkelEx(rp2_6,1);
[ [ 1, 2 ], [ 1, 3 ], [ 1, 4 ], [ 1, 5 ], [ 1, 6 ], [ 2, 3 ], [ 2, 4 ],
  [ 2, 5 ], [ 2, 6 ], [ 3, 4 ], [ 3, 5 ], [ 3, 6 ], [ 4, 5 ], [ 4, 6 ],
  [ 5, 6 ] ]
gap> SCSkel(rp2_6,1);
[ [ 11, 12 ], [ 11, 13 ], [ 11, 14 ], [ 11, 15 ], [ 11, 16 ], [ 12, 13 ],
  [ 12, 14 ], [ 12, 15 ], [ 12, 16 ], [ 13, 14 ], [ 13, 15 ], [ 13, 16 ],
  [ 14, 15 ], [ 14, 16 ], [ 15, 16 ] ]
gap>
```

4.5.45 SCSkelEx

◊ `SCSkelEx( complex, dim )` (function)

Returns: a face list upon success, `fail` otherwise.

Computes the `dim`-skeleton of a simplcial complex, i. e. all `dim`-faces of `complex`, if `dim` is an integer value. If `dim` is a list, a list of all `dim[i]`-faces of `complex` for each entry `dim[i]` (which has to be an integer) is returned. The faces are returned in the standard labeling.

Example

```
gap> SCLib.SearchByName("RP^2");
[ [ 3, "RP^2 (VT)" ], [ 284, "RP^2xS^1" ] ]
gap> rp2_6:=SCLib.Load(3);;
gap> rp2_6:=SC(rp2_6.Facets+10);;
gap> SCSkelEx(rp2_6,1);
[ [ 1, 2 ], [ 1, 3 ], [ 1, 4 ], [ 1, 5 ], [ 1, 6 ], [ 2, 3 ], [ 2, 4 ],
  [ 2, 5 ], [ 2, 6 ], [ 3, 4 ], [ 3, 5 ], [ 3, 6 ], [ 4, 5 ], [ 4, 6 ],
  [ 5, 6 ] ]
gap> SCSkel(rp2_6,1);
[ [ 11, 12 ], [ 11, 13 ], [ 11, 14 ], [ 11, 15 ], [ 11, 16 ], [ 12, 13 ],
  [ 12, 14 ], [ 12, 15 ], [ 12, 16 ], [ 13, 14 ], [ 13, 15 ], [ 13, 16 ],
  [ 14, 15 ], [ 14, 16 ], [ 15, 16 ] ]
gap>
```

4.5.46 SCSpanningTree

◊ `SCSpanningTree( complex )` (function)

Returns: a spanning tree of `complex` upon success, `fail` otherwise.

Computes a spanning tree of `complex` which must be a connected simplicial complex with a greedy algorithm and returns it in form of a `SCSimplicialComplex` object.

Example

```
gap> c:=SC([["a","b","c"],["a","b","d"], ["a","c","d"], ["b","c","d"]]);;
gap> s:=SCSpanningTree(c);
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.
```

```
 Name="spanning tree of unnamed complex n"
 Dim=1

/SimplicialComplex]
gap> s.Facets;
[ [ "a", "b" ], [ "a", "c" ], [ "a", "d" ] ]
```

4.5.47 SCVertices

◊ `SCVertices( complex )` (function)

Returns: a list of vertex labels upon success, `fail` otherwise.

Returns vertex labels.

Example

```
gap> sphere:=SC([["x",45,[1,1]],["x",45,["b",3]],["x",[1,1],
  ["b",3]],[45,[1,1],["b",3]]]);;
gap> SCVerticesEx(sphere);
[1, 2, 3, 4]
gap> SCVertices(sphere);
[ 45, [ 1, 1 ], "x", [ "b", 3 ] ]
```

4.5.48 SCVerticesEx

◊ `SCVerticesEx( complex )` (function)

Returns: $[1,\ldots,n]$ upon success, `fail` otherwise.

Returns $[1,\ldots,n]$, where n is the number of vertices of `complex`.

Example

```
gap> c:=SC([[1,4,5],[4,9,8],[12,13,14,15,16,17]]);;
gap> SCVerticesEx(c);
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

4.6 Operations on simplicial complexes

The following functions perform operations on simplicial complexes. Most of them return simplicial complexes. Thus, this section is closely related to section ”Generate new complexes from old” earlier in this chapter. However, the data generated here is rather seen as an intrinsic attribute of the original complex and not as an independent complex.

4.6.1 SCAlexanderDual

◊ `SCAlexanderDual( complex )` (function)

Returns: a new simplicial complex upon success, `fail` otherwise.

The Alexander dual of `complex` is the simplicial complex whose faces are the complements of the nonfaces of `complex`.

Example

```
# a square
gap> c:=SC([[1,2],[2,3],[3,4],[4,1]]);;
gap> dual:=SCAlexanderDual(c);;
gap> dual.F;
[4, 2]
gap> dual.IsConnected;
false
gap> dual.Facets;
[[1, 3], [2, 4]]
```

4.6.2 SCCollapseGreedy

◊ SCCollapseGreedy(complex) (function)

Returns: a new simplicial complex upon success, fail otherwise.

Employs a greedy collapsing algorithm in order to collapse complex. The source code of this function is taken from [Lut].

Example

```
gap> SCLib.SearchByName("T^2");
[ [ 5, "T^2 (VT)" ], [ 7, "T^2 (VT)" ], [ 11, "T^2 (VT)" ],
  [ 12, "T^2 (VT)" ], [ 20, "T^2 (VT)" ], [ 22, "(T^2)#2" ],
  [ 27, "(T^2)#3" ], [ 41, "T^2 (VT)" ], [ 44, "(T^2)#4" ], ...
gap> torus:=SCLib.Load(5);;
gap> torus:=SCLib.Load(5);;
gap> bdtorus:=SCDifference(torus,SC([torus.Facets[1]]));;
gap> coll:=SCCollapseGreedy(bdtorus);
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="collapsed version of T^2 (VT) \ unnamed complex n"
 Dim=1

/SimplicialComplex]
gap> coll.Facets;
[ [ 3, 6 ], [ 3, 7 ], [ 5, 6 ], [ 5, 7 ], [ 6, 7 ] ]
gap> sphere:=SCBdSimplex(4);;
gap> bdsphere:=SCDifference(sphere,SC([sphere.Facets[1]]));;
gap> coll:=SCCollapseGreedy(bdsphere);
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="collapsed version of S^3_5 \ unnamed complex n"
 Dim=0

/SimplicialComplex]
gap> coll.Facets;
[ [ 5 ] ]
gap>
```

4.6.3 SCCone

◊ SCCone( ) (function)

Returns: a new simplicial complex upon success, fail otherwise.

Takes a new vertex v and unites every facet with it. The function returns the new complex in the standard vertex labeling from 1 to $(n+1)$. The apex of the cone is $(n+1)$.

Example

```
gap> SCLib.SearchByName("RP^3");
[ [ 45, "RP^3" ], [ 103, "RP^3=L(2,1) (VT)" ], [ 246, "(S^2~S^1)#RP^3" ],
  [ 247, "(S^2xS^1)#RP^3" ], [ 283, "(S^2~S^1)#2#RP^3" ],
  [ 285, "(S^2xS^1)#2#RP^3" ], [ 409, "RP^3#RP^3" ], ...
gap> rp3:=SCLib.Load(45);;
gap> rp3.F;
[11, 51, 80, 40]
gap> cone:=SCCone(rp3);;
gap> cone.F;
[12, 62, 131, 120, 40]
gap>
```

4.6.4 SCDeletedJoin

◊ SCDeletedJoin(complex1, complex2) (function)

Returns: a new simplicial complex upon success, fail otherwise.

Calculates the simplicial deleted join of the simplicial complexes complex1 and complex2. Internally falls back to the homology package [DHSW04] if called with a facet list and does not interfere with the homology package.

Example

```
gap> deljoin:=SCDeletedJoin(SCBdSimplex(3),SCBdSimplex(3));
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="S^2_4 deljoin S^2_4"
 Dim=3

/SimplicialComplex]
gap> bddeljoin:=SCBoundary(deljoin);;
gap> bddeljoin.Homology;
[ [ 1, [  ] ], [ 0, [  ] ], [ 2, [  ] ] ]
gap> deljoin.Facets;
[ [ [ 1, 1 ], [ 2, 1 ], [ 3, 1 ], [ 4, 2 ] ],
  [ [ 1, 1 ], [ 2, 1 ], [ 3, 2 ], [ 4, 1 ] ],
  [ [ 1, 1 ], [ 2, 1 ], [ 3, 2 ], [ 4, 2 ] ],
  [ [ 1, 1 ], [ 2, 2 ], [ 3, 1 ], [ 4, 1 ] ],
  [ [ 1, 1 ], [ 2, 2 ], [ 3, 1 ], [ 4, 2 ] ],
  [ [ 1, 1 ], [ 2, 2 ], [ 3, 2 ], [ 4, 1 ] ],
  [ [ 1, 1 ], [ 2, 2 ], [ 3, 2 ], [ 4, 2 ] ],
```

```
    [ [ 1, 2 ], [ 2, 1 ], [ 3, 1 ], [ 4, 1 ] ],
    [ [ 1, 2 ], [ 2, 1 ], [ 3, 1 ], [ 4, 2 ] ],
    [ [ 1, 2 ], [ 2, 1 ], [ 3, 2 ], [ 4, 1 ] ],
    [ [ 1, 2 ], [ 2, 1 ], [ 3, 2 ], [ 4, 2 ] ],
    [ [ 1, 2 ], [ 2, 2 ], [ 3, 1 ], [ 4, 1 ] ],
    [ [ 1, 2 ], [ 2, 2 ], [ 3, 1 ], [ 4, 2 ] ],
    [ [ 1, 2 ], [ 2, 2 ], [ 3, 2 ], [ 4, 1 ] ] ]
gap>
```

4.6.5 SCDifference

◊ `SCDifference( complex1, complex2 )` (function)

Returns: a new simplicial complex upon success, `fail` otherwise.

Forms the "difference" of two simplicial complexes `complex1` and `complex2` as the simplicial complex formed by the difference of the face lattices of `complex1` minus `complex2`. The two arguments are not altered. Note: for the difference process the vertex labelings of the complexes are taken into account, see also `Operation Difference (SCSimplicialComplex, SCSimplicialComplex)` (3.3.10).

Example

```
gap> c:=SCBdSimplex(3);;
gap> d:=SC([[1,2,3]]);;
gap> disc:=SCDifference(c,d);;
gap> disc.Facets;
[ [ 1, 2, 4 ], [ 1, 3, 4 ], [ 2, 3, 4 ] ]
gap> empty:=SCDifference(d,c);;
gap> empty.Dim;
-1
gap>
```

4.6.6 SCHandleAddition

◊ `SCHandleAddition( complex, f1, f2 )` (function)

Returns: simplicial complex obtained by identifying the vertices of facet `f1` with the ones of facet `f2` in `complex`. `fail` upon an error.

A combinatorial handle addition is possible, whenever we have $\mathrm{d}(v,w) \geq 3$ for any two vertices $v \in$`f1` and $w \in$`f2`, where $\mathrm{d}(\cdot,\cdot)$ is the length of the shortest path from v to w. This condition is not checked by this algorithm. See. [BD08] for further information.

Example

```
gap> c:=SC([[1,2,4],[2,4,5],[2,3,5],[3,5,6],[1,3,6],[1,4,6]]);;
gap> c:=SCUnion(c,SCUnion(SCCopy(c)+3,SCCopy(c)+6));;
gap> c:=SCUnion(c,SC([[1,2,3],[10,11,12]]));;
gap> c.Facets;
[[1, 2, 3], [1, 2, 4], [1, 3, 6], [1, 4, 6], [2, 3, 5],
  [2, 4, 5], [3, 5, 6], [4, 5, 7], [4, 6, 9], [4, 7, 9],
  [5, 6, 8], [5, 7, 8], [6, 8, 9], [7, 8, 10], [7, 9, 12],
  [7, 10, 12], [8, 9, 11], [8, 10, 11], [9, 11, 12], [10, 11, 12]]
gap> c.Homology;
[[0, []], [0, []], [1, []]]
```

```
gap> torus:=SCHandleAddition(c,[1,2,3],[10,11,12]);;
gap> torus.Homology;
[[0, []], [2, []], [1, []]]
gap> ism:=SCIsManifold(torus);;
gap> ism;
true
gap>
```

4.6.7 SCIntersection

◊ `SCIntersection( complex1, complex2 )` (function)

Returns: a new simplicial complex upon success, `fail` otherwise.

Forms the "intersection" of two simplicial complexes `complex1` and `complex2` as the simplicial complex formed by the intersection of the face lattices of `complex1` and `complex2`. The two arguments are not altered. Note: for the intersection process the vertex labelings of the complexes are taken into account.

Example

```
gap> c:=SCBdSimplex(3);;
gap> d:=SCBdSimplex(3)+1;;
gap> d.Facets;
[ [ 2, 3, 4 ], [ 2, 3, 5 ], [ 2, 4, 5 ], [ 3, 4, 5 ] ]
gap> c:=SCBdSimplex(3);;
gap> d:=SCBdSimplex(3);;
gap> d:=SCMove(d,[[1,2,3],[]])+1;;
gap> s1:=SCIntersection(c,d);;
gap> s1.Facets;
[ [ 2, 3 ], [ 2, 4 ], [ 3, 4 ] ]
gap>
```

4.6.8 SCIsIsomorphic

◊ `SCIsIsomorphic( complex1, complex2 )` (function)

Returns: `true` / `false` upon success, `fail` otherwise.

`true` is returned, if `complex1` and `complex2` are combinatorially isomorphic, `false` if not.

Example

```
gap> c1:=SC([[11,12,13],[11,12,14],[11,13,14],[12,13,14]]);;
gap> c2:=SCBdSimplex(3);;
gap> SCIsIsomorphic(c1,c2);
true
gap> c3:=SCBdCrossPolytope(3);;
gap> SCIsIsomorphic(c1,c3);
false
```

4.6.9 SCIsSubcomplex

◊ `SCIsSubcomplex( sc1, sc2 )` (function)

Returns: `true` if `complex1` and `complex2` are combinatorially isomorphic, `false` otherwise.

Returns `true` if `sc2` is a subcomplex of `sc1`, `false` otherwise. If dim(`sc2`) $\leq$ dim(`sc1`) the facets of `sc2` are compared with the dim(`sc2`)-skeleton of `sc1`. Only works for pure simplicial complexes.

Example

```
gap> SCLib.SearchByAttribute("F[1]=10");
[ [ 19, "K^2 (VT)" ], [ 20, "T^2 (VT)" ], [ 21, "S^3 (VT)" ],
  [ 22, "(T^2)#2" ], [ 23, "S^3 (VT)" ], [ 24, "S^2xS^1 (VT)" ],
  [ 25, "S^3 (VT)" ], [ 26, "S^4 (VT)" ], [ 27, "(T^2)#3" ], ...
gap> t2:=SCLib.Load(20);;
gap> c:=SCBdSimplex(9);;
gap> t2.F;
[10, 30, 20]
gap> c.F;
[10, 45, 120, 210, 252, 210, 120, 45, 10]
gap> SCIsSubcomplex(c,t2);
true
gap> SCIsSubcomplex(t2,c);
false
```

4.6.10 SCIsomorphism

◊ `SCIsomorphism( complex1, complex2 )` (function)

Returns: a list of pairs of vertex labels upon success, `fail` otherwise.

Returns an isomorphism of `complex1` to `complex2` in the original labeling if they are combinatorially isomorphic, `false` otherwise. Internally calls `SCIsomorphismEx` (4.6.11)`(complex1,complex2);`

Example

```
gap> c1:=SC([[11,12,13],[11,12,14],[11,13,14],[12,13,14]]);;
gap> c2:=SCBdSimplex(3);;
gap> SCIsomorphism(c1,c2);
[ [ 11, 1 ], [ 12, 2 ], [ 13, 3 ], [ 14, 4 ] ]
gap> SCIsomorphismEx(c1,c2);
[ [ [ 1, 1 ], [ 2, 2 ], [ 3, 3 ], [ 4, 4 ] ] ]
gap>
```

4.6.11 SCIsomorphismEx

◊ `SCIsomorphismEx( complex1, complex2 )` (function)

Returns: a list of pairs of vertex labels upon success, `fail` otherwise.

Returns an isomorphism of `complex1` to `complex2` in the standard labeling if they are combinatorially isomorphic, `false` otherwise. If f-vector and Altshuler-Steinberg determinant are equal, the internal function `SCIntFunc.SCComputeIsomorphismsEx"(complex1,complex2,true)` is called.

Example

```
gap> c1:=SC([[11,12,13],[11,12,14],[11,13,14],[12,13,14]]);;
gap> c2:=SCBdSimplex(3);;
gap> SCIsomorphism(c1,c2);
[ [ 11, 1 ], [ 12, 2 ], [ 13, 3 ], [ 14, 4 ] ]
gap> SCIsomorphismEx(c1,c2);
[ [ [ 1, 1 ], [ 2, 2 ], [ 3, 3 ], [ 4, 4 ] ] ]
```

```
gap>
```

4.6.12 SCJoin

◊ SCJoin(complex1, complex2) (function)

Returns: a new simplicial complex upon success, fail otherwise.

Calculates the simplicial join of the simplicial complexes complex1 and complex2. Internally falls back to the homology package [DHSW04] and does not interfere with the homology package. Note that the vertex labelings of the complexes passed as arguments are not propagated to the new complex.

Example

```
gap> sphere:=SCJoin(SCBdSimplex(2),SCBdSimplex(2));
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="S^1_3 join S^1_3"
 Dim=3

/SimplicialComplex]
gap> SCHasBoundary(sphere);
false
gap> sphere.Facets;
[ [ [ 1, 1 ], [ 1, 2 ], [ 2, 1 ], [ 2, 2 ] ],
  [ [ 1, 1 ], [ 1, 2 ], [ 2, 1 ], [ 2, 3 ] ],
  [ [ 1, 1 ], [ 1, 2 ], [ 2, 2 ], [ 2, 3 ] ],
  [ [ 1, 1 ], [ 1, 3 ], [ 2, 1 ], [ 2, 2 ] ],
  [ [ 1, 1 ], [ 1, 3 ], [ 2, 1 ], [ 2, 3 ] ],
  [ [ 1, 1 ], [ 1, 3 ], [ 2, 2 ], [ 2, 3 ] ],
  [ [ 1, 2 ], [ 1, 3 ], [ 2, 1 ], [ 2, 2 ] ],
  [ [ 1, 2 ], [ 1, 3 ], [ 2, 1 ], [ 2, 3 ] ],
  [ [ 1, 2 ], [ 1, 3 ], [ 2, 2 ], [ 2, 3 ] ] ]
gap> sphere.Homology;
[ [ 0, [  ] ], [ 0, [  ] ], [ 0, [  ] ], [ 1, [  ] ] ]
gap>
```

Example

```
gap> ball:=SCJoin(SC([[1]]),SCBdSimplex(2));
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="unnamed complex n join S^1_3"
 Dim=2

/SimplicialComplex]
gap> ball.Homology;
[ [ 0, [  ] ], [ 0, [  ] ], [ 0, [  ] ] ]
gap> ball.Facets;
[ [ [ 1, 1 ], [ 2, 1 ], [ 2, 2 ] ], [ [ 1, 1 ], [ 2, 1 ], [ 2, 3 ] ],
```

```
    [ [ 1, 1 ], [ 2, 2 ], [ 2, 3 ] ] ]
  gap>
```

4.6.13 SCLink

◊ `SCLink( complex, face )` (function)

Returns: a new simplicial complex upon success, `fail` otherwise.

Computes the link of `face` in `complex`, i. e. all facets containing `face`, reduced by `face`. if `complex` is pure, the resulting complex is of dimension dim(`complex`) - dim(`face`) -1.

Example

```
gap> SCLib.SearchByName("RP^2");
[ [ 3, "RP^2 (VT)" ], [ 284, "RP^2xS^1" ] ]
gap> rp2:=SCLib.Load(3);;
gap> SCVertices(rp2);
[1, 2, 3, 4, 5, 6]
gap> SCLink(rp2,[1]);
[SimplicialComplex

Properties known: Dim, Facets, Name, VertexLabels.

Name="unnamed complex n"
Dim=1

/SimplicialComplex]
gap> last.Facets;
[[2, 3], [2, 6], [3, 5], [4, 5], [4, 6]]
```

4.6.14 SCLinks

◊ `SCLinks( complex, dim )` (function)

Returns: a list of new simplicial complexes upon success, `fail` otherwise.

Computes the link of all `dim`-faces in `complex` and stores them in a list of simplicial complexes. Internally calls `SCLink` (4.6.13) for every face.

Example

```
gap> c:=SCBdSimplex(4);;
# all vertex links -> 2 spheres
gap> SCLinks(c,0);
[ [SimplicialComplex

     Properties known: Dim, Facets, Name, VertexLabels.

     Name="lk([ 1 ]) in S^3_5"
     Dim=2

    /SimplicialComplex], # ... etc
]
 # all edge links -> circles
 gap> SCLinks(c,1);
 [ [SimplicialComplex
```

```
     Properties known: Dim, Facets, Name, VertexLabels.

     Name="lk([ 1, 2 ]) in S^3_5"
     Dim=1

    /SimplicialComplex], # ... etc
]
gap>
```

4.6.15 SCNeighbors

◊ `SCNeighbors( complex, face )` (function)

Returns: a list of faces upon success, `fail` otherwise.

Computes the neighboring faces of a given `face` in `complex`. A neighboring face is a face distinct from `face` that must have the same dimension as `face` and intersect `face` in a $(d-1)$-face, where `face` is of dimensions d. The face is returned in original labeling.

Example

```
gap> c:=SCFromFacets(Combinations(["a","b","c"],2));
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="unnamed complex n"
 Dim=1

/SimplicialComplex]
gap> SCNeighbors(c,["a","d"]);
[ [ "a", "b" ], [ "a", "c" ] ]
gap>
```

4.6.16 SCNeighborsEx

◊ `SCNeighborsEx( complex, face )` (function)

Returns: a list of faces upon success, `fail` otherwise.

Computes the neighboring faces of a given `face` in standard labeling in `complex`. A neighboring face is a face distinct from `face` that must have the same dimension as `face` and intersect `face` in a $(d-1)$-face, where `face` is of dimensions d.

Example

```
gap> c:=SCFromFacets(Combinations(["a","b","c"],2));
[SimplicialComplex

 Properties known: Chi, Dim, Facets, Name, VertexLabels.

 Name="unnamed complex n"
 Dim=1

/SimplicialComplex]
```

```
gap> SCLabels(c);
[ "a", "b", "c" ]
gap> SCNeighborsEx(c,[1,2]);
[ [ 1, 3 ], [ 2, 3 ] ]
```

4.6.17 SCShelling

◊ `SCShelling( complex )` (function)

Returns: a permuted list of the facets of `complex` upon success, `fail` otherwise.

`complex` must be a pure, strongly connected pseudomanifold with boundary.

Internally calls `SCShellingExt` (4.6.18)`(complex,false,[]);`. To learn more about shellings see [Zie95], [Pac87].

Example

```
gap> c:=SC([[1,2,3],[1,2,4],[1,3,4]]);;
gap> SCShelling(c);
[[1, 2, 3], [1, 2, 4], [1, 3, 4]]
```

4.6.18 SCShellingExt

◊ `SCShellingExt( sc, all, checkvector )` (function)

Returns: a list of permuted lists of the facets of `sc` of size 1 (`all` = `false`), n (`all` = `true`) if `checkvector` is empty, `true` / `false` if `checkvector` is not empty, `fail` upon an error.

`sc` must be a pure, bounded, strongly connected combinatorial pseudomanifold.

A shelling is an ordering on the facet list of `sc` starting with a facet at the boundary. The facets are removed with respect to that ordering. In every step the intersection of the removed facet and the already removed facets has to be a shelling too.

Every shelling is represented as a permuted version of the facet list of `sc`. A checkvector encodes a shelling in a shorter form. It only contains the indices of the facets. If an order of indices is assigned to `checkvector` the function tests whether it is a valid shelling or not.

See [Zie95], [Pac87] to learn more about shellings.

Example

```
gap> SCLib.SearchByName("RP^2");
[ [ 3, "RP^2 (VT)" ], [ 284, "RP^2xS^1" ] ]
gap> rp2:=SCLib.Load(3);;
gap> rp2:=SCDifference(rp2,SC([rp2.Facets[1]]));; # bounded version
gap> all:=SCShellingExt(rp2,true,[]);;
gap> Size(all);
1488
gap> all[1];
[ [ 1, 2, 6 ], [ 1, 4, 6 ], [ 1, 4, 5 ], [ 1, 3, 5 ], [ 2, 4, 5 ],
  [ 2, 3, 4 ], [ 2, 5, 6 ], [ 3, 4, 6 ], [ 3, 5, 6 ] ]
gap> all:=SCShellingExt(rp2,false,[]);
[ [ [1, 2, 6], [1, 4, 6], [1, 4, 5], [1, 3, 5],
    [2, 4, 5], [2, 3, 4], [2, 5, 6], [3, 4, 6], [3, 5, 6]         ]
]
# valid shelling
gap> all:=SCShellingExt(rp2,false,[4, 9, 3, 2, 5, 6, 7, 1, 8]);
true
```

```
gap> all:=SCShellingExt(rp2,true,[4, 9, 3, 2, 5, 6, 7, 1, 8]);
true
# invalid shelling
gap> all:=SCShellingExt(rp2,true,[1 .. 9]);
#I  SCShellingExt: 2 is not a valid shelling facet.
false
```

4.6.19 SCShellings

◊ `SCShellings( complex )` (function)

Returns: a list of permuted versions of the facet list of `complex`.

Internally calls `SCShellingExt` (4.6.18)`(complex,true,[]);`. To learn more about shellings see [Zie95], [Pac87].

Example

```
gap> c:=SC([[1,2,3],[1,2,4],[1,3,4]]);;
gap> SCShellings(c);
[[[1, 2, 3], [1, 2, 4], [1, 3, 4]],
  [[1, 2, 3], [1, 3, 4], [1, 2, 4]],
  [[1, 2, 4], [1, 2, 3], [1, 3, 4]],
  [[1, 3, 4], [1, 2, 3], [1, 2, 4]],
  [[1, 2, 4], [1, 3, 4], [1, 2, 3]],
  [[1, 3, 4], [1, 2, 4], [1, 2, 3]]
]
```

4.6.20 SCSpan

◊ `SCSpan( complex, subset )` (function)

Returns: a new simplicial complex upon success, `fail` otherwise.

Computes the reduced face lattice of all faces that are a subset of `subset`.

Example

```
gap> c:=SCBdCrossPolytope(4);;
gap> SCVertices(c);
[1, 2, 3, 4, 5, 6, 7, 8]
gap> span:=SCSpan(c,[1,2,3,4]);
[SimplicialComplex

Properties known: Dim, Facets, Name, VertexLabels.

Name="span([ 1, 2, 3, 4 ]) in Bd(\beta^4)"
Dim=1

/SimplicialComplex]
gap> span.Facets;
[[1, 3], [1, 4], [2, 3], [2, 4]]
```

Example

```
gap> c:=SC([[1,2],[1,4,5],[2,3,4]]);;
gap> span:=SCSpan(c,[2,3,5]);
```

```
[SimplicialComplex

Properties known: Dim, Facets, Name, VertexLabels.

Name="span([ 2, 3, 5 ]) in unnamed complex n"
Dim=1

/SimplicialComplex]
gap> SCFacets(span);
[[2, 3], [5]]
```

4.6.21 SCStar

◊ `SCStar( complex, face )` (function)

Returns: a new simplicial complex upon success, `fail` otherwise .

Computes the star of `face` in `complex`, i. e. the set of facets of `complex` that contain `face`.

Example

```
gap> SCLib.SearchByName("RP^2");
[ [ 3, "RP^2 (VT)" ], [ 284, "RP^2xS^1" ] ]
gap> rp2:=SCLib.Load(3);;
gap> SCVertices(rp2);
[1, 2, 3, 4, 5, 6]
gap> SCStar(rp2,[1]);
[SimplicialComplex

 Properties known: Dim, Facets, VertexLabels.

 Name="star([ 1 ]) in RP^2 (VT)"
 Dim=2

/SimplicialComplex]
gap> last.Facets;
[[1, 2, 3], [1, 2, 6], [1, 3, 5], [1, 4, 5], [1, 4, 6]]
```

4.6.22 SCStars

◊ `SCStars( complex, dim )` (function)

Returns: a list of new simplicial complexes upon success, `fail` otherwise.

Computes for every `dim`-face f the set of facets that contain f.

Example

```
gap> SCLib.SearchByName("T^2");
[ [ 5, "T^2 (VT)" ], [ 7, "T^2 (VT)" ], [ 11, "T^2 (VT)" ],
  [ 12, "T^2 (VT)" ], [ 20, "T^2 (VT)" ], [ 22, "(T^2)#2" ],
  [ 27, "(T^2)#3" ], [ 41, "T^2 (VT)" ], [ 44, "(T^2)#4" ], ...
# the minimal 7-vertex torus
gap> torus:=SCLib.Load(5);;
# 7 2-discs as vertex stars
gap> SCStars(torus,0);
[ [SimplicialComplex
```

```
     Properties known: Dim, Facets, Name, VertexLabels.

     Name="star([ 1 ]) in T^2 (VT)"
     Dim=2

    /SimplicialComplex], # ... etc
]
gap>
```

4.6.23 SCSuspension

◊ `SCSuspension( complex )` (function)

Returns: a new simplicial complex upon success, `fail` otherwise.

Calculates the simplicial suspension of the simplicial complex `complex`. Internally falls back to the homology package [DHSW04] and does not interfere with the homology package. Note that the vertex labelings of the complexes passed as arguments are not propagated to the new complex.

Example

```
gap> SCLib.SearchByName("Poincare");
[ [ 563, "Poincare_sphere" ] ]
gap> phs:=SCLib.Load(587);
[SimplicialComplex

 Properties known: AltshulerSteinberg, AutomorphismGroup,
                   AutomorphismGroupSize, AutomorphismGroupStructure,
                   AutomorphismGroupTransitivity, Boundary, Chi,
                   ConnectedComponents, Dim, DualGraph, F, Faces, Facets, G,
                   Generators, H, HasBoundary, HasInterior, Homology,
                   Interior, IsCentrallySymmetric, IsConnected,
                   IsEulerianManifold, IsOrientable, IsPM, IsPure,
                   MinimalNonFaces, Name, Neighborliness, Orientation,
                   StronglyConnected, VertexLabels, Vertices.

 Name="B4"
 Dim=3
 AutomorphismGroupSize=1
 AutomorphismGroupStructure="1"
 AutomorphismGroupTransitivity=0
 Chi=0
 F=[ 17, 117, 200, 100 ]
 G=[ 12, 59 ]
 H=[ 13, 72, 13, 1 ]
 HasBoundary=false
 HasInterior=true
 Homology=[ [ 0, [ ] ], [ 1, [ 4 ] ], [ 0, [ 2 ] ], [ 0, [ ] ] ]
 IsCentrallySymmetric=false
 IsConnected=true
 IsEulerianManifold=true
 IsOrientable=false
 IsPM=true
 IsPure=true
```

```
 Neighborliness=1

/SimplicialComplex]
gap> phs:=SCLib.Load(587);;
gap> susp:=SCSuspension(phs);;
gap> edwardsSphere:=SCSuspension(susp);;
gap> SCIsManifold(edwardsSphere);
...
```

4.6.24 SCUnion

◊ `SCUnion( complex1, complex2 )` (function)

Returns: a new simplicial complex upon success, `fail` otherwise.

Forms the union of two simplicial complexes `complex1` and `complex2` as the simplicial complex formed by the union of their facets sets. The two arguments are not altered. Note: for the union process the vertex labelings of the complexes are taken into account, see also `Operation Union (SCSimplicialComplex, SCSimplicialComplex)` (3.3.9).

Example

```
gap> c:=SCUnion(SCBdSimplex(3),SCBdSimplex(3)+3); #a wedge of two 2-spheres
[SimplicialComplex

 Properties known: Dim, Facets, VertexLabels.

 Name="S^2_4 cup S^2_4"
 Dim=2

/SimplicialComplex]
```

4.6.25 SCVertexIdentification

◊ `SCVertexIdentification( complex, v1, v2 )` (function)

Returns: simplicial complex with pinched point `v1=v2` upon success, `fail` otherwise.

A vertex identification of `v1` and `v2` is possible whenever d(`v1`,`v2`) ≥ 3. This is not checked by this algorithm.

Example

```
gap> c:=SC([[1,2],[2,3],[3,4]]);;
gap> circle:=SCVertexIdentification(c,[1],[4]);;
gap> circle.Facets;
[[1, 2], [1, 3], [2, 3]]
gap> circle.Homology;
[[0, []], [1, []]]
```

4.6.26 SCWedge

◊ `SCWedge( complex1, complex2, ... )` (function)

Returns: a new simplicial complex upon success, `fail` otherwise.

Calculates the wedge product of the complexes supplied as arguments. Note, that the vertex labelings of the complexes passed as arguments are not propagated to the new complex.

Example

```
gap> wedge:=SCWedge(SCBdSimplex(2),SCBdSimplex(2));
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="S^1_3 wedge S^1_3"
 Dim=1

/SimplicialComplex]
gap> wedge.Facets;
[ [ [ 1, 2 ], [ 1, 3 ] ], [ [ 2, 2 ], [ 2, 3 ] ], [ [ 1, 2 ], "a" ],
  [ [ 1, 3 ], "a" ], [ [ 2, 2 ], "a" ], [ [ 2, 3 ], "a" ] ]
gap>
```

Chapter 5

(Co-)Homology of simplicial complexes

Although simpcomp relies on the GAP package "homology" [DHSW04] for its homology calculations due to efficiency reasons, the package contains some additional homology related functionality, which will be explained in this chapter.

5.1 Homology computation

Apart from calculating boundaries of simplices, boundary matrices or the simplicial homology of a given complex, simpcomp is also able to compute a basis of the homology groups.

5.1.1 SCBoundaryOperatorMatrix

◊ `SCBoundaryOperatorMatrix( complex, dim )` (function)

Returns: a rectangular matrix upon success, `fail` otherwise.

Calculates the matrix of the boundary operator $\partial_{\texttt{dim+1}}$. Notice, that each column contains the boundaries of a (`dim`+1)-simplex as a list of oriented `dim`-simplices and that the matrix is stored (GAP-typical) as a list of row vectors (as usual in GAP).

Example

```
gap> c:=SCFromFacets([[1,2,3],[1,2,6],[1,3,5],[1,4,5],[1,4,6],
                      [2,3,4],[2,4,5],[2,5,6],[3,4,6],[3,5,6]]);;
gap> mat:=SCBoundaryOperatorMatrix(c,1);
[ [ 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
  [ -1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0 ],
  [ 0, -1, 0, 0, 0, -1, 0, 0, 0, 1, 1, 1, 0, 0, 0 ],
  [ 0, 0, -1, 0, 0, 0, -1, 0, 0, -1, 0, 0, 1, 1, 0 ],
  [ 0, 0, 0, -1, 0, 0, 0, -1, 0, 0, -1, 0, -1, 0, 1 ],
  [ 0, 0, 0, 0, -1, 0, 0, 0, -1, 0, 0, -1, 0, -1, -1 ] ]
gap>
```

5.1.2 SCBoundarySimplex

◊ `SCBoundarySimplex( simplex, orientation )` (function)

Returns: a list upon success, `fail` otherwise.

Calculates the boundary of a given `simplex`. If the flag `orientation` is set to true, the function returns the boundary as a list of oriented simplices of the form [ORIENTATION, SIMPLEX], where

ORIENTATION is either +1 or -1 and a value of +1 means that SIMPLEX is positively oriented and a value of -1 that SIMPLEX is negatively oriented. If the ORIENTATION flag is false, an unoriented list of simplices is returned, see the example.

Example

```
gap> SCBoundarySimplex([1..5],true);
[ [ -1, [ 2, 3, 4, 5 ] ], [ 1, [ 1, 3, 4, 5 ] ], [ -1, [ 1, 2, 4, 5 ] ],
  [ 1, [ 1, 2, 3, 5 ] ], [ -1, [ 1, 2, 3, 4 ] ] ]
gap> SCBoundarySimplex([1..5],false);
[ [ 2, 3, 4, 5 ], [ 1, 3, 4, 5 ], [ 1, 2, 4, 5 ], [ 1, 2, 3, 5 ],
  [ 1, 2, 3, 4 ] ]
```

5.1.3 SCHomologyBasis

◊ `SCHomologyBasis( complex, dim )` (function)

Returns: a list of pairs of the form `[ integer, list ]` upon success, `fail` otherwise.

Calculates a set of basis vectors for the `dim`-dimensional homology group (with integer coefficients) of `complex`. The entries of the returned list are of the form [MODULUS, [BASEELM1, BASEELM2, ...]], where the value MODULUS is 1 for the basis elements of the free part of the `dim`-th homology group and $k \geq 2$ for the basis elements of the k-torsion part. The basis elements are stored as lists of coefficient-index pairs referring to the simplices of the complex, i.e. a basis element of the form $[[\lambda_1, i], [\lambda_2, j], \ldots]$ encodes the linear combination of simplices of the form $\lambda_1 * \Delta_1 + \lambda_2 * \Delta_2$ with Δ_1=`SCSkel(complex,dim)[i]`, Δ_2=`SCSkel(complex,dim)[j]` and so on.

Example

```
gap> SCLib.SearchByName("(S^2xS^1)#RP^3");
[ [ 247, "(S^2xS^1)#RP^3" ] ]
gap> c:=SCLib.Load(247);;
gap> SCHomologyBasis(c,1);
[ [ 1, [ [ 1, 12 ], [ -1, 7 ], [ 1, 1 ] ] ],
  [ 2, [ [ 1, 68 ], [ -1, 69 ], [ -1, 71 ], [ 2, 72 ], [ -2, 73 ] ] ] ]
```

5.1.4 SCHomologyBasisAsSimplices

◊ `SCHomologyBasisAsSimplices( complex, dim )` (function)

Returns: a list of pairs of the form `[ integer, linear combination of simplices ]` upon success, `fail` otherwise.

In contrast to the function `SCHomologyBasis` (5.1.3) this functSion returns the basis elements in the same format as `SCHomologyBasis` (5.1.3), but every basis element is returned as a linear combination of simplices of `complex`. Each term of the linear combination is of the form [FACTOR, SIMPLEX], where FACTOR is a non-zero integer value.

Example

```
gap> SCLib.SearchByName("(S^2xS^1)#RP^3");
[ [ 247, "(S^2xS^1)#RP^3" ] ]
gap> c:=SCLib.Load(247);;
gap> SCHomologyBasisAsSimplices(c,1);
[ [ 1, [ [ 1, [ 2, 8 ] ], [ -1, [ 1, 8 ] ], [ 1, [ 1, 2 ] ] ] ],
  [ 2, [ [ 1, [ 11, 12 ] ], [ -1, [ 11, 13 ] ], [ -1, [ 12, 13 ] ],
         [ 2, [ 12, 14 ] ], [ -2, [ 13, 14 ] ] ] ] ]
```

5.1.5 SCHomologyInternal

◊ `SCHomologyInternal( complex )` (function)

Returns: a list of pairs of the form `[ integer, list ]` upon success, `fail` otherwise.

This function computes the reduced simplicial homology with integer coefficients of a given simplicial complex `complex` with integer coefficients. It uses the algorithm described in [DKT08].

The output is a list of homology groups of the form $[H_0,....,H_d]$, where d is the dimension of `complex`. The format of the homology groups H_i is given in terms of their maximal cyclic subgroups, i.e. a homology group $H_i \cong \mathbb{Z}^f + \mathbb{Z}/t_1\mathbb{Z} \times \cdots \times \mathbb{Z}/t_n\mathbb{Z}$ is returned in form of a list $[f,[t_1,...,t_n]]$, where f is the (integer) free part of H_i and t_i denotes the torsion parts of H_i ordered in weakly incresing size.

Example

```
gap> c:=SCFromFacets([[1,2,3],[1,2,6],[1,3,5],[1,4,5],[1,4,6],
                      [2,3,4],[2,4,5],[2,5,6],[3,4,6],[3,5,6]]);;
gap> SCHomologyInternal(c);
[ [ 0, [  ] ], [ 0, [ 2 ] ], [ 0, [  ] ] ]
```

5.2 Cohomology computation

simpcomp can also compute the cohomology groups of simplicial complexes, bases of these cohomology groups, the cup product of two cocycles and the intersection form of (orientable) 4-manifolds. Due to the enormous amount of resources needed for the computation of exact eigenvalues, the signature of the intersection form cannot be computed within `GAP`. However, in some cases the `GAP` function `Eigenvalues` can be used.

5.2.1 SCCoboundaryOperatorMatrix

◊ `SCCoboundaryOperatorMatrix( complex, dim )` (function)

Returns: a rectangular matrix upon success, `fail` otherwise.

Calculates the matrix of the coboundary operator $d^{\mathtt{dim+1}}$ as a list of row vectors.

Example

```
gap> c:=SCFromFacets([[1,2,3],[1,2,6],[1,3,5],[1,4,5],[1,4,6],
                      [2,3,4],[2,4,5],[2,5,6],[3,4,6],[3,5,6]]);
gap> mat:=SCCoboundaryOperatorMatrix(c,1);;
[ [ -1, 1, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
  [ -1, 0, 0, 0, 1, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0 ],
  [ 0, -1, 0, 1, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0 ],
  [ 0, 0, -1, 1, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0 ],
  [ 0, 0, -1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0 ],
  [ 0, 0, 0, 0, 0, -1, 1, 0, 0, -1, 0, 0, 0, 0, 0 ],
  [ 0, 0, 0, 0, 0, 0, -1, 1, 0, 0, 0, 0, -1, 0, 0 ],
  [ 0, 0, 0, 0, 0, 0, 0, -1, 1, 0, 0, 0, 0, 0, -1 ],
  [ 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 1, 0, -1, 0 ],
  [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 1, 0, 0, -1 ] ]
```

5.2.2 SCCohomology

◊ `SCCohomology( complex )` (function)

Returns: a list of pairs of the form `[ integer, list ]` upon success, `fail` otherwise.

This function computes the simplicial cohomology of a given simplicial complex `complex` with integer coefficients. It uses the algorithm described in [DKT08].

The output is a list of cohomology groups of the form $[H^0,....,H^d]$, where d is the dimension of `complex`. The format of the cohomology groups H^i is given in terms of their maximal cyclic subgroups, i.e. a cohomology group $H^i \cong \mathbb{Z}^f + \mathbb{Z}/t_1\mathbb{Z} \times \cdots \times \mathbb{Z}/t_n\mathbb{Z}$ is returned in form of a list $[f,[t_1,...,t_n]]$, where f is the (integer) free part of H^i and t_i denotes the torsion parts of H^i ordered in weakly incresing size.

Example

```
gap> c:=SCFromFacets([[1,2,3],[1,2,6],[1,3,5],[1,4,5],[1,4,6],
                      [2,3,4],[2,4,5],[2,5,6],[3,4,6],[3,5,6]]);
gap> SCCohomology(c);
[ [ 1, [  ] ], [ 0, [  ] ], [ 0, [ 2 ] ] ]
gap>
```

5.2.3 SCCohomologyBasis

◊ `SCCohomologyBasis( complex, dim )` (function)

Returns: a list of pairs of the form `[ integer, list ]` upon success, `fail` otherwise.

Calculates a set of basis vectors for the `dim`-dimensional cohomology group (with integer coefficients) of `complex`. The entries of the returned list are of the form [MODULUS, [BASEELM1, BASEELM2, ...]], where the value MODULUS is 1 for the basis elements of the free part of the `dim`-th cohomology group and $k \geq 2$ for the basis elements of the k-torsion part. The basis elements are stored as lists of coefficient-index pairs referring to the simplices of the complex, i.e. a basis element of the form $[[\lambda_1,i],[\lambda_2,j],\dots]$ encodes the linear combination of simplices of the form $\lambda_1 * \Delta_1 + \lambda_2 * \Delta_2$ with Δ_1=`SCSkel(complex,dim)[i]`, Δ_2=`SCSkel(complex,dim)[j]` and so on.

Example

```
gap> SCLib.SearchByName("SU(3)/SO(3)");
[ [ 222, "SU(3)/SO(3) (VT)" ], [ 520, "SU(3)/SO(3) (VT)" ],
  [ 526, "SU(3)/SO(3) (VT)" ], [ 528, "SU(3)/SO(3) (VT)" ] ]
gap> c:=SCLib.Load(222);;
gap> SCCohomologyBasis(c,3);
[ [ 2, [ [ -9, 259 ], [ 9, 262 ], [ 9, 263 ], [ -9, 270 ], [ 9, 271 ],
          [ -9, 273 ], [ -9, 274 ], [ -18, 275 ], [ -9, 276 ], [ 9, 278 ],
          [ -9, 279 ], [ -9, 280 ], [ 3, 283 ], [ -3, 285 ], [ 3, 289 ],
          [ -3, 294 ], [ 3, 310 ], [ -3, 313 ], [ 3, 316 ], [ -1, 317 ],
          [ -6, 318 ], [ 3, 319 ], [ -6, 320 ], [ 6, 321 ], [ 1, 322 ],
          [ 3, 325 ], [ -1, 328 ], [ 6, 330 ], [ -2, 331 ], [ 12, 332 ],
          [ 7, 333 ], [ -5, 334 ], [ 1, 345 ], [ 3, 355 ], [ -9, 357 ],
          [ 9, 358 ], [ 1, 363 ], [ 12, 365 ], [ -9, 366 ], [ -3, 370 ],
          [ -1, 371 ], [ -3, 372 ], [ 8, 373 ], [ -1, 374 ], [ 6, 375 ],
          [ 9, 376 ], [ 3, 377 ], [ 1, 380 ], [ 3, 383 ], [ -8, 385 ],
          [ -9, 386 ], [ -9, 388 ], [ -18, 404 ], [ 9, 410 ], [ -9, 425 ],
          [ -18, 426 ], [ -9, 427 ], [ 9, 428 ], [ -9, 429 ], [ 3, 433 ],
          [ -3, 435 ], [ -9, 437 ], [ 10, 442 ], [ 12, 445 ], [ 1, 447 ],
          [ -19, 448 ], [ 2, 449 ], [ -1, 450 ], [ -9, 451 ], [ 3, 453 ],
          [ 1, 455 ], [ 1, 457 ], [ -11, 458 ], [ -9, 459 ], [ 9, 461 ],
          [ 9, 462 ], [ -9, 468 ], [ 9, 469 ], [ -18, 471 ], [ -9, 472 ],
          [ 9, 474 ], [ -9, 475 ], [ 9, 488 ], [ 9, 495 ], [ -9, 500 ],
          [ -3, 504 ], [ 9, 505 ], [ 9, 512 ], [ 9, 515 ], [ 6, 519 ],
          [ 18, 521 ], [ -15, 523 ], [ 9, 524 ], [ -3, 525 ], [ 18, 527 ],
```

```
            [ -18, 528 ], [ 6, 529 ], [ 6, 531 ], [ 12, 532 ] ] ] ]
```

5.2.4 SCCohomologyBasisAsSimplices

◊ `SCCohomologyBasisAsSimplices( complex, dim )` (function)

Returns: a list of pars of the form `[ integer, linear combination of simplices ]` upon success, `fail` otherwise.

In contrast to the function `SCCohomologyBasis` (5.2.3) this function returns the basis elements in the same format as `SCCohomologyBasis` (5.2.3), but every basis element is returned as a linear combination of simplices of `complex`. Each term of the linear combination is of the form [FACTOR, SIMPLEX], where FACTOR is a non-zero integer value.

Example

```
gap> SCLib.SearchByName("SU(3)/SO(3)");
[ [ 222, "SU(3)/SO(3) (VT)" ], [ 520, "SU(3)/SO(3) (VT)" ],
  [ 526, "SU(3)/SO(3) (VT)" ], [ 528, "SU(3)/SO(3) (VT)" ] ]
gap> c:=SCLib.Load(222);;
gap> SCCohomologyBasisAsSimplices(c,3);
[ [ 2, [ [ -9, [ 2, 7, 8, 9 ] ], [ 9, [ 2, 7, 8, 12 ] ], [ 9, [ 2, 7, 8, 13 ] ],
          [ -9, [ 2, 7, 11, 12 ] ], [ 9, [ 2, 7, 11, 13 ] ], [ -9, [ 2, 8, 9, 10 ] ],
          [ -9, [ 2, 8, 9, 11 ] ], [ -18, [ 2, 8, 9, 12 ] ], [ -9, [ 2, 8, 9, 13 ] ],
          [ 9, [ 2, 8, 10, 12 ] ], [ -9, [ 2, 8, 10, 13 ] ],
          [ -9, [ 2, 8, 11, 12 ] ], [ 3, [ 2, 9, 10, 12 ] ],
          [ -3, [ 2, 9, 11, 12 ] ], [ 3, [ 3, 4, 5, 7 ] ], [ -3, [ 3, 4, 5, 12 ] ],
          [ 3, [ 3, 4, 10, 12 ] ], [ -3, [ 3, 5, 6, 7 ] ], [ 3, [ 3, 5, 6, 11 ] ],
          [ -1, [ 3, 5, 6, 13 ] ], [ -6, [ 3, 5, 7, 8 ] ], [ 3, [ 3, 5, 7, 10 ] ],
          [ -6, [ 3, 5, 7, 11 ] ], [ 6, [ 3, 5, 7, 12 ] ], [ 1, [ 3, 5, 7, 13 ] ],
          [ 3, [ 3, 5, 8, 12 ] ], [ -1, [ 3, 5, 9, 13 ] ], [ 6, [ 3, 5, 10, 12 ] ],
          [ -2, [ 3, 5, 10, 13 ] ], [ 12, [ 3, 5, 11, 12 ] ],
          [ 7, [ 3, 5, 11, 13 ] ], [ -5, [ 3, 5, 12, 13 ] ], [ 1, [ 3, 6, 9, 13 ] ],
          [ 3, [ 3, 7, 10, 12 ] ], [ -9, [ 3, 7, 11, 12 ] ], [ 9, [ 3, 7, 11, 13 ] ],
          [ 1, [ 3, 8, 9, 13 ] ], [ 12, [ 3, 8, 10, 12 ] ], [ -9, [ 3, 8, 10, 13 ] ],
          [ -3, [ 3, 9, 10, 12 ] ], [ -1, [ 3, 9, 10, 13 ] ],
          [ -3, [ 3, 9, 11, 12 ] ], [ 8, [ 3, 9, 11, 13 ] ],
          [ -1, [ 3, 9, 12, 13 ] ], [ 6, [ 3, 10, 11, 12 ] ],
          [ 9, [ 3, 10, 11, 13 ] ], [ 3, [ 3, 10, 12, 13 ] ], [ 1, [ 4, 5, 6, 8 ] ],
          [ 3, [ 4, 5, 6, 11 ] ], [ -8, [ 4, 5, 6, 13 ] ], [ -9, [ 4, 5, 7, 8 ] ],
          [ -9, [ 4, 5, 7, 11 ] ], [ -18, [ 4, 6, 8, 9 ] ], [ 9, [ 4, 6, 9, 13 ] ],
          [ -9, [ 4, 8, 9, 10 ] ], [ -18, [ 4, 8, 9, 12 ] ], [ -9, [ 4, 8, 9, 13 ] ],
          [ 9, [ 4, 8, 10, 12 ] ], [ -9, [ 4, 8, 10, 13 ] ], [ 3, [ 4, 9, 10, 12 ] ],
          [ -3, [ 4, 9, 11, 12 ] ], [ -9, [ 4, 9, 12, 13 ] ], [ 10, [ 5, 6, 7, 8 ] ],
          [ 12, [ 5, 6, 7, 11 ] ], [ 1, [ 5, 6, 7, 13 ] ], [ -19, [ 5, 6, 8, 9 ] ],
          [ 2, [ 5, 6, 8, 11 ] ], [ -1, [ 5, 6, 8, 12 ] ], [ -9, [ 5, 6, 8, 13 ] ],
          [ 3, [ 5, 6, 9, 11 ] ], [ 1, [ 5, 6, 9, 13 ] ], [ 1, [ 5, 6, 10, 13 ] ],
          [ -11, [ 5, 6, 11, 13 ] ], [ -9, [ 5, 7, 8, 9 ] ], [ 9, [ 5, 7, 8, 12 ] ],
          [ 9, [ 5, 7, 8, 13 ] ], [ -9, [ 5, 7, 11, 12 ] ], [ 9, [ 5, 7, 11, 13 ] ],
          [ -18, [ 5, 8, 9, 12 ] ], [ -9, [ 5, 8, 9, 13 ] ], [ 9, [ 5, 8, 10, 12 ] ],
          [ -9, [ 5, 8, 11, 12 ] ], [ 9, [ 6, 7, 8, 13 ] ], [ 9, [ 6, 7, 11, 13 ] ],
          [ -9, [ 6, 8, 10, 13 ] ], [ -3, [ 6, 9, 11, 12 ] ],
          [ 9, [ 6, 9, 11, 13 ] ], [ 9, [ 7, 8, 9, 13 ] ], [ 9, [ 7, 8, 11, 12 ] ],
          [ 6, [ 7, 9, 11, 12 ] ], [ 18, [ 7, 11, 12, 13 ] ],
```

```
            [ -15, [ 8, 9, 10, 12 ] ], [ 9, [ 8, 9, 10, 13 ] ],
            [ -3, [ 8, 9, 11, 12 ] ], [ 18, [ 8, 10, 11, 12 ] ],
            [ -18, [ 8, 10, 12, 13 ] ], [ 6, [ 9, 10, 11, 12 ] ],
            [ 6, [ 9, 10, 12, 13 ] ], [ 12, [ 9, 11, 12, 13 ] ] ] ] ]
```

5.2.5 SCCupProduct

◊ `SCCupProduct( complex, cocycle1, cocycle2 )` (function)

Returns: a list of oriented simplices upon success, `fail` otherwise.

The cup product is a method of adjoining two cocycles of degree p and q to form a composite cocycle of degree $p+q$.

The construction of the cup product starts with a product of cochains: if `cocycle1` is a p-cochain and `cocylce2` is a q-cochain (given as list of oriented p- (q-)simplices, then

$$\texttt{cocycle1} \smile \texttt{cocycle2}(\sigma) = \texttt{cocycle1}(\sigma \circ \iota_{0,1,\ldots p}) \cdot \texttt{cocycle2}(\sigma \circ \iota_{p,p+1,\ldots,p+q})$$

where σ is a $(p+q)$-simplex and $\iota_S, S \subset \{0,1,...,p+q\}$ is the canonical embedding of the simplex spanned by S into the $(p+q)$-standard simplex.

$\sigma \circ \iota_{0,1,\ldots,p}$ is called the p-th front face and $\sigma \circ \iota_{p,p+1,\ldots,p+q}$ is the q-th back face of σ, respectively.

Example

```
gap> SCLib.SearchByName("K3");
[ [ 584, "K3 surface" ] ]
gap> c:=SCLib.Load(584);;
gap> basis:=SCCohomologyBasisAsSimplices(c,2);;
gap> SCCupProduct(c,basis[1][2],basis[1][2]);
[ [ 1, [ 1, 2, 4, 7, 11 ] ], [ 1, [ 2, 3, 4, 5, 9 ] ] ]
gap> SCCupProduct(c,basis[1][2],basis[2][2]);
[ [ -1, [ 1, 2, 4, 7, 11 ] ], [ -1, [ 1, 2, 4, 7, 15 ] ],
  [ -1, [ 2, 3, 4, 5, 9 ] ] ]
gap>
```

5.2.6 SCIntersectionForm

◊ `SCIntersectionForm( complex )` (function)

Returns: a square matrix of integer values upon success, `fail` otherwise.

For $2d$-dimensional orientable pseudomanifolds M the cup product (see `SCCupProduct` (5.2.5)) defines a bilinear function

$$\mathrm{H}^d(M) \times \mathrm{H}^d(M) \to \mathrm{H}^{2d}(M), (a,b) \mapsto a \cup b$$

called the intersection form. This function returns the intersection form in integral cohomology in form of a matrix `mat` with respect to the basis of $H^d(M)$ computed by `SCCohomologyBasisAsSimplices` (5.2.4). The matrix entry `M[i][j]` equals the intersection number of the `i`-th element with the `j`-th element of the basis.

Example

```
gap> SCLib.SearchByName("CP^2");
[ [ 17, "CP^2 (VT)" ], [ 88, "CP^2#CP^2" ], [ 89, "CP^2#-CP^2" ],
  [ 186, "CP^2#(S^2xS^2)" ], [ 499, "(S^3~S^1)#(CP^2)^{#5} (VT)" ] ]
gap> c:=SCLib.Load(17);;
gap> c1:=SCConnectedSum(c,c);;
gap> c2:=SCConnectedSumMinus(c,c);;
gap> q1:=SCIntersectionForm(c1);;
gap> q2:=SCIntersectionForm(c2);;
gap> PrintArray(q1);
[ [  1,  0 ],
  [  0,  1 ] ]
gap> PrintArray(q2);
[ [   1,   0 ],
  [   0,  -1 ] ]
```

5.2.7 SCIntersectionFormParity

◊ `SCIntersectionFormParity( complex )` (function)

Returns: `0` or `1` upon success, `fail` otherwise.

Computes the parity of the intersection form of `complex` (see `SCIntersectionForm` (5.2.6)). If the intersection for is even (i. e. all diagonal entries are even numbers) `0`, if it is odd (at least one diagonal entry is odd), `1` is returned.

Example

```
gap> SCLib.SearchByName("S^2xS^2");
[ [ 51, "S^2xS^2" ], [ 110, "S^2xS^2 (VT)" ], [ 111, "S^2xS^2 (VT)" ],
  [ 112, "S^2xS^2 (VT)" ], [ 114, "(S^2xS^2)#(S^2xS^2)" ],
  [ 144, "(S^2xS^2)#(S^2xS^2) (VT)" ], [ 145, "(S^2xS^2)#(S^2xS^2) (VT)" ],
  [ 186, "CP^2#(S^2xS^2)" ] ]
gap> c:=SCLib.Load(51);;
gap> SCIntersectionFormParity(c);
0
gap> SCLib.SearchByName("CP^2");
[ [ 17, "CP^2 (VT)" ], [ 88, "CP^2#CP^2" ], [ 89, "CP^2#-CP^2" ],
  [ 186, "CP^2#(S^2xS^2)" ], [ 499, "(S^3~S^1)#(CP^2)^{#5} (VT)" ] ]
gap> c:=SCLib.Load(17);;
gap> SCIntersectionFormParity(c);
1
gap>
```

5.2.8 SCIntersectionFormDimensionality

◊ `SCIntersectionFormDimensionality( complex )` (function)

Returns: an integer upon success, `fail` otherwise.

Returns the dimensionality of the intersection form of `complex`, i. e. the length of a minimal generating set of $H^d(M)$ (where $2d$ is the dimension of `complex`). See `SCIntersectionForm` (5.2.6) for further details.

Example

```
gap> SCLib.SearchByName("CP^2");
[ [ 17, "CP^2 (VT)" ], [ 88, "CP^2#CP^2" ], [ 89, "CP^2#-CP^2" ],
  [ 186, "CP^2#(S^2xS^2)" ], [ 499, "(S^3~S^1)#(CP^2)^{#5} (VT)" ] ]
gap> c:=SCLib.Load(17);; ## gap> SCIntersectionFormParity(c);
1
gap> SCCohomology(c);
[ [ 1, [  ] ], [ 0, [  ] ], [ 1, [  ] ], [ 0, [  ] ], [ 1, [  ] ] ]
gap> SCIntersectionFormDimensionality(c);
1
gap> d:=SCConnectedProduct(c,10);;
gap> SCIntersectionFormDimensionality(d);
10
```

5.2.9 SCIntersectionFormSignature

◊ `SCIntersectionFormSignature( complex )` (function)

Returns: a triple of integers upon success, `fail` otherwise.

Returns the dimensionality and the signature of the intersection form of `complex` as a 3-tuple that contains the dimensionality in the first entry and the number of positive / negative eigenvalues in the second and third entry. See `SCIntersectionForm` (5.2.6) for further details.

Example

```
gap> SCLib.SearchByName("CP^2");
[ [ 17, "CP^2 (VT)" ], [ 88, "CP^2#CP^2" ], [ 89, "CP^2#-CP^2" ],
  [ 186, "CP^2#(S^2xS^2)" ], [ 499, "(S^3~S^1)#(CP^2)^{#5} (VT)" ] ]
gap> c:=SCLib.Load(17);; ## gap> SCIntersectionFormParity(c);
1
gap> SCCohomology(c);
[ [ 1, [  ] ], [ 0, [  ] ], [ 1, [  ] ], [ 0, [  ] ], [ 1, [  ] ] ]
gap> SCIntersectionFormSignature(c);
[ 1, 0, 1 ]
gap> d:=SCConnectedSum(c,c);
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels, Vertices.

 Name="CP^2 (VT)#+-CP^2 (VT)"
 Dim=4

/SimplicialComplex]
gap> SCIntersectionFormSignature(d);
[ 2, 1, 1 ]
gap> d:=SCConnectedSumMinus(c,c);;
gap> SCIntersectionFormSignature(d);
[ 2, 0, 2 ]
```

Chapter 6

Bistellar flips

6.1 Theory

Bistellar flips, first introduced by Pachner in [Pac87] and sometimes also refered to as Pachner moves, can prove useful for the following tasks:

1. Try to decide whether two closed simplicial pseudomanifolds are PL-homeomorphic.
2. Reduce a given triangulation without changing its PL-type.
3. Verify whether a closed simplicial pseudomanifold is a combinatorial manifold.

Bistellar moves are a convenient means to prove the PL equivalence of triangulations, see [Lut03] for an example.

Since bistellar flips do not respect the combinatorial properties of a complex, no attention to the original vertex labels is payed, i. e. the flipped complex will be relabeled whenever its vertex labels become different from the standard labeling (for example after every reverse 0-move).

6.2 Functions for bistellar flips

6.2.1 SCBistellarOptions

◊ `SCBistellarOptions( )` (property)

Record of global variables to adjust output an behavior of bistellar moves in `SCIntFunc.SCChooseMove` (6.2.4) and `SCReduceComplexEx` (6.2.13) respectively.

1. `BaseRelaxation`: determines the length of the relaxation period. Default: 3
2. `BaseHeating`: determines the length of the heating period. Default: 4
3. `Relaxation`: value of the current relaxation period. Default: 0
4. `Heating`: value of the current heating period. Default: 0
5. `MaxRounds`: maximal over all number of bistellar flips that will be performed. Default: 500000
6. `MaxInterval`: maximal number of bistellar flips that will be performed without a change of the moved complex. Default: 100000

7. `Mode`: flip mode, 0=reducing, 1=comparing, 2=reduce as subcomplex. Default: 0
8. `LogLevel`: 0=no logging, 1=reduced logging, 2=full logging, information about every flip. Default: 2
9. `WriteLevel`: 0=no output, 1=storing of every vertex minimal complex to user library, 2=e-mail notification. Default: 1
10. `MailNotifyIntervall`: (minimum) number of seconds between two e-mail notifications. Default: $24 \cdot 60 \cdot 60$ (one day)
11. `MaxIntervalIsManifold`: maximal number of bistellar flips that will be performed without a change of a vertex link while trying to prove that the complex is a combinatorial manifold. Default: 500

Example

```
gap> SCBistellarOptions.BaseRelaxation;
3
gap> SCBistellarOptions.BaseHeating;
4
gap> SCBistellarOptions.Relaxation;
0
gap> SCBistellarOptions.Heating;
0
gap> SCBistellarOptions.MaxRounds;
500000
gap> SCBistellarOptions.MaxInterval;
100000
gap> SCBistellarOptions.Mode;
0
gap> SCBistellarOptions.LogLevel;
2
gap> SCBistellarOptions.WriteLevel;
1
gap> SCBistellarOptions.MailNotifyInterval;
86400
gap> SCBistellarOptions.MaxIntervalIsManifold;
5000
```

6.2.2 SCEquivalent

◊ `SCEquivalent( complex1, complex2 )` (function)

Returns: `true` / `false` upon success, `fail` otherwise.

Tests, whether the closed simplicial pseudomanifold `complex1` can be reduced via bistellar moves to `complex2`, i. e. whether `complex1` and `complex2` are *PL*-homeomorphic. Note, that in general the problem is undecidable.

It is recommended to use a minimal triangulation of `complex2` for the check if possible.

Internally calls `SCReduceComplexEx` (6.2.13) `(complex1,complex2,1,SCIntFunc.SCChooseMove);`

Example

```
# hexagon
gap> obj:=SC([[1,2],[2,3],[3,4],[4,5],[5,6],[6,1]]);;
```

```
# triangle as a (minimal) reference object
gap> refObj:=SCBdSimplex(2);;
gap> SCEquivalent(obj,refObj);
#I  round 0: [ 5, 5 ]
#I  round 1: [ 4, 4 ]
#I  round 2: [ 3, 3 ]
#I  SCReduceComplexEx: complexes are bistellarly equivalent.
true
```

6.2.3 SCExamineComplexBistellar

◊ `SCExamineComplexBistellar( complex )` (function)

Returns: simplicial complex passed as argument with additional properties upon success, `fail` otherwise.

Computes face lattice, f-vector, AS-determinant, dimension and maximal vertex label of `complex`.

Example

```
gap> obj:=SC([[1,2],[2,3],[3,4],[4,5],[5,6],[6,1]]);
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="unnamed complex n"
 Dim=1

/SimplicialComplex]
gap> SCExamineComplexBistellar(obj);
[SimplicialComplex

 Properties known: AltshulerSteinberg, Boundary, Chi, Dim, F, Faces, Facets,
                   HasBoundary, IsPM, IsPure, Name, VertexLabels.

 Name="unnamed complex 21"
 Dim=1
 Chi=0
 F=[ 6, 6 ]
 HasBoundary=false
 IsPM=true
 IsPure=true

/SimplicialComplex]
```

6.2.4 SCIntFunc.SCChooseMove

◊ `SCIntFunc.SCChooseMove( dim, moves )` (function)

Returns: a bistellar move, i. e. a pair of lists upon success, `fail` otherwise.

Since the problem of finding a bistellar flip sequence that reduces a simplicial complex is undecidable, we have to use an heuristic approach to choose the next move.

The implemented strategy SCIntFunc.SCChooseMove first tries to directly remove vertices, edges, *i*-faces in increasing dimension etc. If this is not possible it inserts high dimensional faces in decreasing co-dimension. To do this in an efficient way a number of parameters have to be adjusted, namely SCBistellarOptions.BaseHeating and SCBistellarOptions.BaseRelaxation. See SCBistellarOptions (6.2.1) for further options.

If this strategy does not work for you, just implement a customized strategy and pass it to SCReduceComplexEx (6.2.13).

See SCRMoves (6.2.10) for further information.

6.2.5 SCIsKStackedSphere

◊ SCIsKStackedSphere(complex, k) (function)

Returns: an integer value upon success, fail otherwise.

Checks, whether the given complex that must be a homology sphere is a k-stacked sphere by a randomized algorithm based on bistellar moves. Returns an integer less or equal to k, where a positive return value of i means that the complex is i-stacked, a zero return value means that the complex cannot be k-stacked and -1 if the question could not be decided.

Internally calls SCReduceComplexEx (6.2.13).

Example

```
gap> SCLib.SearchByName("S^4~S^1");
[ [ 204, "S^4~S^1 (VT)" ], [ 339, "S^4~S^1 (VT)" ], [ 341, "S^4~S^1 (VT)" ],
  [ 438, "S^4~S^1 (VT)" ], [ 493, "S^4~S^1 (VT)" ], [ 494, "S^4~S^1 (VT)" ],
  [ 495, "S^4~S^1 (VT)" ], [ 496, "S^4~S^1 (VT)" ], [ 497, "S^4~S^1 (VT)" ],
  [ 500, "S^4~S^1 (VT)" ], [ 501, "S^4~S^1 (VT)" ], [ 502, "S^4~S^1 (VT)" ] ]
gap> c:=SCLib.Load(204);;
gap> l:=c.Link(1);
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="lk(1) in S^4~S^1 (VT)"
 Dim=4

/SimplicialComplex]
gap> SCIsKStackedSphere(l,1);
#I  SCIsKStackedSphere: try 1/50
#I  round 0: [ 11, 40, 70, 65, 26 ]
#I  round 1: [ 10, 35, 60, 55, 22 ]
#I  round 2: [ 9, 30, 50, 45, 18 ]
#I  round 3: [ 8, 25, 40, 35, 14 ]
#I  round 4: [ 7, 20, 30, 25, 10 ]
#I  round 5: [ 6, 15, 20, 15, 6 ]
#I  SCReduceComplexEx: computed locally minimal complex after 6 rounds.
1
gap>
```

6.2.6 SCIsManifold

◊ `SCIsManifold( complex )` (function)

Returns: `true` / `false` upon success, `fail` otherwise.

Tries to prove that a closed simplicial d-pseudomanifold is a combinatorial manifold by reducing its vertex links to the boundary of the d-simplex.

`false` is only returned, if a minimal version of a reduced vertex link can be computed which is different from the boundary of the simplex. Internally calls `SCReduceComplexEx` (6.2.13)`(link,SCEmpty(),0,SCIntFunc.SCChooseMove);` for every link of `complex`. Note, that `false` is returned in case of a bounded manifold.

Example

```
gap> c:=SCBdCrossPolytope(3);;
gap> SCIsManifold(c);
#I  SCIsManifold: processing vertex link 1/6
#I  round 0: [ 3, 3 ]
#I  SCReduceComplexEx: computed locally minimal complex after 1 rounds.
#I  SCIsManifold: link is sphere.
...
#I  SCIsManifold: processing vertex link 6/6
#I  round 0: [ 3, 3 ]
#I  SCReduceComplexEx: computed locally minimal complex after 1 rounds.
#I  SCIsManifold: link is sphere.
true
```

6.2.7 SCIsMovableComplex

◊ `SCIsMovableComplex( complex )` (function)

Returns: `true` / `false` upon success, `fail` otherwise.

Checks, if `complex` can be modified by bistellar moves, i. e. if it is a closed combinatorial pseudomanifold.

Example

```
gap> c:=SCBdCrossPolytope(3);;
gap> SCIsMovableComplex(c);
true
```

Bounded pseudomanifold

Example

```
gap> c:=SC([[1,2],[2,3],[3,4],[3,1]]);;
gap> SCIsMovableComplex(c);
false
```

6.2.8 SCMove

◊ `SCMove( complex, move )` (function)

Returns: a new simplicial complex in standard labeling.

Applies the bistellar move `move` to `complex`. `move` is given as a $(r+1)$-tuple together with a $(d+1-r)$-tuple if d is the dimension of `complex` an if `move` is a r-move. See `SCRMoves` (6.2.10) for detailed information about bistellar r-Moves.

Note: `move` and `complex` should be given in standard labeling to ensure a correct result.

Example

```
gap> obj:=SC([[1,2],[2,3],[3,4],[4,1]]);
[SimplicialComplex

Properties known: Dim, Facets, Name, VertexLabels.

Name="unnamed complex n"
Dim=1

/SimplicialComplex]
gap> moves:=SCMoves(obj);
[[[[1, 2], []], [[1, 4], []],
    [[2, 3], []], [[3, 4], []]],
  [[[1], [2, 4]], [[2], [1, 3]],
    [[3], [2, 4]], [[4], [1, 3]]]]
gap> obj:=SCMove(obj,last[2][1]);
[SimplicialComplex

 Properties known: Chi, Dim, F, Faces, Facets, VertexLabels.

 Name="unnamed complex n"
 Dim=1
 Chi=0
 F=[3, 3]

/SimplicialComplex]
```

6.2.9 SCMoves

◊ `SCMoves( complex )` (function)

Returns: a list of list of pairs of lists upon success, `fail` otherwise.

See `SCRMoves` (6.2.10) for further information.

Example

```
gap> c:=SCBdCrossPolytope(3);;
gap> moves:=SCMoves(c);
[
# 0-moves
  [[[1, 3, 5], []], [[1, 3, 6], []], [[1, 4, 5], []],
    [[1, 4, 6], []], [[2, 3, 5], []], [[2, 3, 6], []],
    [[2, 4, 5], []], [[2, 4, 6], []]],
# 1-moves
  [[[1, 3], [5, 6]], [[1, 4], [5, 6]], [[1, 5], [3, 4]],
    [[1, 6], [3, 4]], [[2, 3], [5, 6]], [[2, 4], [5, 6]],
    [[2, 5], [3, 4]], [[2, 6], [3, 4]], [[3, 5], [1, 2]],
    [[3, 6], [1, 2]], [[4, 5], [1, 2]], [[4, 6], [1, 2]]],
# 2-moves
  []
```

```
]
```

6.2.10 SCRMoves

◊ `SCRMoves( complex, r )` (function)

Returns: a list of pairs of the form `[ list, list ]`.

A bistellar r-move of a d-dimensional combinatorial manifold `complex` is a r-face m_1 together with a $d-r$-tuple m_2 where m_1 is a common face of exactly $(d+1-r)$ facets and m_2 is not a face of `complex`.

The r-Move removes all facets containing m_1 and replaces them by the $(r+1)$ faces obtained by uniting m_2 with any subset of m_1 of order r.

The resulting complex has the same (PL-)topolgy as `complex`.

Example

```
gap> c:=SCBdCrossPolytope(3);;
gap> moves:=SCRMoves(c,1);
[ [ [ 1, 3 ], [ 5, 6 ] ], [ [ 1, 4 ], [ 5, 6 ] ], [ [ 1, 5 ], [ 3, 4 ] ],
  [ [ 1, 6 ], [ 3, 4 ] ], [ [ 2, 3 ], [ 5, 6 ] ], [ [ 2, 4 ], [ 5, 6 ] ],
  [ [ 2, 5 ], [ 3, 4 ] ], [ [ 2, 6 ], [ 3, 4 ] ], [ [ 3, 5 ], [ 1, 2 ] ],
  [ [ 3, 6 ], [ 1, 2 ] ], [ [ 4, 5 ], [ 1, 2 ] ], [ [ 4, 6 ], [ 1, 2 ] ] ]
```

6.2.11 SCReduceAsSubcomplex

◊ `SCReduceAsSubcomplex( complex1, complex2 )` (function)

Returns: `SCBistellarOptions.WriteLevel=0`: a triple of the form `[ boolean, simplicial complex, rounds performed ]` upon termination of the algorithm.
`SCBistellarOptions.WriteLevel=1`: A library of simplicial complexes with all vertex minimal complexes and (upon termination) a triple of the form `[ boolean, simplicial complex, rounds performed ]`.
`SCBistellarOptions.WriteLevel=2`: A mail in case a smaller version of `complex1` was found, a library of simplicial complexes with all vertex minimal complexes and (upon termination) a triple of the form `[ boolean, simplicial complex, rounds performed ]` upon termination of the algorithm.
Returns `fail` upon an error.

Reduces a closed simplicial pseudomanifold `complex1` as a subcomplex of `complex2`.

Main application: Reduce a subcomplex of the cross polytope without introducing diagonals.

Internally calls `SCReduceComplexEx` (6.2.13) `(complex1,complex2,2,SCIntFunc.SCChooseMove);`

Example

```
gap> c:=SCFromFacets([[1,3],[3,5],[4,5],[4,1]]);;
gap> SCReduceAsSubcomplex(c,SCBdCrossPolytope(3));
#I  round 0, move: [ [ 2 ], [ 1, 4 ] ]
[ 3, 3 ]
#I  SCReduceComplexEx: computed locally minimal complex after 1 rounds.
[ true, [SimplicialComplex

     Properties known: Dim, Facets, Name, VertexLabels.

     Name="unnamed complex 9"
```

```
       Dim=1

      /SimplicialComplex], 1 ]
```

6.2.12 SCReduceComplex

◊ `SCReduceComplex( complex )` (function)

Returns: `SCBistellarOptions.WriteLevel=0`: a triple of the form `[ boolean, simplicial complex, rounds performed ]` upon termination of the algorithm.
`SCBistellarOptions.WriteLevel=1`: A library of simplicial complexes with all vertex minimal complexes and (upon termination) a triple of the form `[ boolean, simplicial complex, rounds performed ]`.
`SCBistellarOptions.WriteLevel=2`: A mail in case a smaller version of `complex1` was found, a library of simplicial complexes with all vertex minimal complexes and (upon termination) a triple of the form `[ boolean, simplicial complex, rounds performed ]` upon termination of the algorithm.
Returns `fail` upon an error..

Reduces a closed simplicial pseudomanifold via bistellar moves. Internally calls `SCReduceComplexEx` (6.2.13)`(complex,SCEmpty(),0,SCIntFunc.SCChooseMove);`

Example

```
# hexagon
gap> obj:=SC([[1,2],[2,3],[3,4],[4,5],[5,6],[6,1]]);;
gap> SCReduceComplex(obj);
#I  round 0, move: [ [ 6 ], [ 1, 5 ] ]
[ 5, 5 ]
#I  round 1, move: [ [ 4 ], [ 3, 5 ] ]
[ 4, 4 ]
#I  round 2, move: [ [ 3 ], [ 2, 5 ] ]
[ 3, 3 ]
#I  SCReduceComplexEx: computed locally minimal complex after 3 rounds.
[ true, [SimplicialComplex

     Properties known: Dim, Facets, Name, VertexLabels.

     Name="unnamed complex 6"
     Dim=1

    /SimplicialComplex], 3 ]
##
```

6.2.13 SCReduceComplexEx

◊ `SCReduceComplexEx( complex, refComplex, mode, choosemove )` (function)

Returns: `SCBistellarOptions.WriteLevel=0`: a triple of the form `[ boolean, simplicial complex, rounds ]` upon termination of the algorithm.
`SCBistellarOptions.WriteLevel=1`: A library of simplicial complexes with all vertex minimal complexes and (upon termination) a triple of the form `[ boolean, simplicial complex, rounds ]`.
`SCBistellarOptions.WriteLevel=2`: A mail in case a smaller version of `complex1` was found, a

library of simplicial complexes with all vertex minimal complexes and (upon termination) a triple of the form `[ boolean, simplicial complex, rounds ]` upon termination of the algorithm.
Returns `fail` upon an error.

Reduces a closed simplicial pseudomanifold (`complex`) via bistellar moves. Compares it ("mode"=1) to `refComplex` or reduces it as a subcomplex of `refComplex` (`mode`=2).

`choosemove` is a function containing a flip strategy for the flip sequence, see also `SCIntFunc.SCChooseMove` (6.2.4).

The currently minimal complex is stored to the variable `minComplex`, the currently minimal f-vector to `minF`. Note, that in general the algorithm will not stop until the maximum number of rounds is reached. You can adjust the maximum number of rounds via the property `SCBistellarOptions` (6.2.1). The number of rounds performed is returned in the third entry of the triple returned by this function.
This function is called by

1. `SCReduceComplex` (6.2.12),
2. `SCEquivalent` (6.2.2),
3. `SCReduceAsSubcomplex` (6.2.11),
4. `SCIsManifold` (6.2.6).

Please see `SCMailIsPending` (9.2.3) for further information about the email notification system in case `SCBistellarOptions.WriteLevel=2` is set.

Chapter 7

Functions for normal surfaces

7.1 Creating an `SCNormalSurface` object

This section contains functions to construct normal surfaces from facet lists or combinatorial 3-manifolds.

7.1.1 SCNSEmpty

◊ `SCNSEmpty( )` (function)

Returns: normal surface with empty facet list upon success, `fail` otherwise.
Generates an empty complex (of dimension -1).

Example

```
gap> SCNSEmpty();
[NormalSurface

 Properties known: Dim, Faces, Facets, Name, VertexLabels.

 Name="empty normal surface"
 Dim=-1

/NormalSurface]
gap>
```

7.1.2 SCNSFromFacets

◊ `SCNSFromFacets( facets )` (function)

Returns: A new object of type `SCNormalSurface` upon success, `fail` otherwise.
Constructor of a normal surface from a facet list, see `SCFromFacets` (4.1.1) for details.

Example

```
gap> ns:=SCNSFromFacets([[1,2,3],[1,2,4,5],[1,3,4,6],[2,3,5,6],[4,5,6]]);
[NormalSurface

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="unnamed normal surface n"
 Dim=2
```

```
/NormalSurface]
gap>
```

7.1.3 SCNS

◊ SCNS(facets) (function)

Returns: A new object of type SCNormalSurface upon success, fail otherwise.

Internally calls SCNSFromFacets (7.1.2).

Example

```
gap> ns:=SCNS([[1,2,3],[1,2,4,5],[1,3,4,6],[2,3,5,6],[4,5,6]]);
[NormalSurface

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="unnamed normal surface n"
 Dim=2

/NormalSurface]
gap>
```

7.1.4 SCNSSlicing

◊ SCNSSlicing(complex, slicing) (function)

Returns: a normal surface upon success, fail otherwise.

Computes a slicing defined by a partition slicing of the set of vertices of the 3-dimensional combinatorial pseudo manifold complex. slicing has to be of size 2.

Example

```
gap> SCLib.SearchByAttribute("F=[6,15,18,9]");
[ [ 4, "S^3 (VT)" ] ]
gap> c:=SCLib.Load(4);;
gap> ns:=SCNSSlicing(c,[[1,2,3],[4,5,6]]);
[NormalSurface

 Properties known: Chi, ConnectedComponents, Dim, F, Facets, Genus, IsConnect\
ed, Name, Oriented, Subdivision, TopologicalType, VertexLabels, Vertices.

 Name="slicing [ [ 1, 2, 3 ], [ 4, 5, 6 ] ] of S^3 (VT)"
 Dim=2
 Chi=2
 F=[ 9, 16, 4, 5 ]
 IsConnected=true
 TopologicalType="S^2"

/NormalSurface]
gap> ns.Facets;
[ [ [ 1, 4 ], [ 2, 4 ], [ 3, 4 ] ],
  [ [ 1, 6 ], [ 2, 6 ], [ 3, 6 ] ],
  [ [ 1, 4 ], [ 1, 5 ], [ 2, 4 ], [ 2, 5 ] ],
```

```
    [ [ 1, 5 ], [ 1, 6 ], [ 2, 5 ], [ 2, 6 ] ],
    [ [ 1, 4 ], [ 1, 6 ], [ 3, 4 ], [ 3, 6 ] ],
    [ [ 1, 4 ], [ 1, 5 ], [ 1, 6 ] ],
    [ [ 2, 4 ], [ 2, 5 ], [ 3, 4 ], [ 3, 5 ] ],
    [ [ 2, 5 ], [ 2, 6 ], [ 3, 5 ], [ 3, 6 ] ],
    [ [ 3, 4 ], [ 3, 5 ], [ 3, 6 ] ] ]
gap> ns:=SCNSSlicing(c,[[1,3,5],[2,4,6]]);
[NormalSurface

 Properties known: Chi, ConnectedComponents, Dim, F, Facets, Genus, IsConnect\
ed, Name, Oriented, Subdivision, TopologicalType, VertexLabels, Vertices.

 Name="slicing [ [ 1, 3, 5 ], [ 2, 4, 6 ] ] of S^3 (VT)"
 Dim=2
 Chi=0
 F=[ 9, 18, 0, 9 ]
 IsConnected=true
 TopologicalType="T^2"

/NormalSurface]
gap> ns.Facets;
[ [ [ 1, 2 ], [ 1, 4 ], [ 3, 2 ], [ 3, 4 ] ],
  [ [ 1, 2 ], [ 1, 6 ], [ 3, 2 ], [ 3, 6 ] ],
  [ [ 1, 2 ], [ 1, 4 ], [ 5, 2 ], [ 5, 4 ] ],
  [ [ 1, 2 ], [ 1, 6 ], [ 5, 2 ], [ 5, 6 ] ],
  [ [ 1, 4 ], [ 1, 6 ], [ 3, 4 ], [ 3, 6 ] ],
  [ [ 1, 4 ], [ 1, 6 ], [ 5, 4 ], [ 5, 6 ] ],
  [ [ 3, 2 ], [ 3, 4 ], [ 5, 2 ], [ 5, 4 ] ],
  [ [ 3, 2 ], [ 3, 6 ], [ 5, 2 ], [ 5, 6 ] ],
  [ [ 3, 4 ], [ 3, 6 ], [ 5, 4 ], [ 5, 6 ] ] ]
gap>
```

7.2 Generating new objects from normal surfaces

simpcomp provides the possibility to copy and / or triangulate normal surfaces. Note, that other constructions like the connected sum or the cartesian product do not make sense for (embedded) normal surfaces in general.

7.2.1 SCNSCopy

◊ `SCNSCopy( complex )` (function)

Returns: a normal surface upon success, `fail` otherwise.

Copies a `GAP` object of type `SCNormalSurface` (cf. `SCCopy` (3.1.2)).

Example

```
gap> sl:=SCNSSlicing(SCBdSimplex(4),[[1],[2..5]]);
[NormalSurface

 Properties known: Chi, ConnectedComponents, Dim, F, Facets, Genus, IsConnect\
ed, Name, Oriented, Subdivision, TopologicalType, VertexLabels, Vertices.
```

```
 Name="slicing [ [ 1 ], [ 2, 3, 4, 5 ] ] of S^3_5"
 Dim=2
 Chi=2
 F=[ 4, 6, 4 ]
 IsConnected=true
 TopologicalType="S^2"

/NormalSurface]
gap> sl_2:=SCNSCopy(sl);
[NormalSurface

 Properties known: Chi, ConnectedComponents, Dim, F, Facets, Genus, IsConnect\
ed, Name, Oriented, Subdivision, TopologicalType, VertexLabels, Vertices.

 Name="slicing [ [ 1 ], [ 2, 3, 4, 5 ] ] of S^3_5"
 Dim=2
 Chi=2
 F=[ 4, 6, 4 ]
 IsConnected=true
 TopologicalType="S^2"

/NormalSurface]
gap> IsIdenticalObj(sl,sl_2);
false
gap>
```

7.2.2 SCNSSubdivision

◊ `SCNSSubdivision( normalsurface )` (function)

Returns: a simplicial complex upon success, `fail` otherwise.

Computes a simplicial subdivision of `normalsurface` without introducing new vertices. The subdivision is stored as a property of `normalsurface`. Note, that symmetry may be lost during the process of subdividing.

Example

```
gap> SCLib.SearchByAttribute("F=[6,15,18,9]");
[ [ 4, "S^3 (VT)" ] ]
gap> c:=SCLib.Load(4);;
gap> ns:=SCNSSlicing(c,[[1,3,5],[2,4,6]]);;
gap> ns.F;
[ 9, 18, 0, 9 ]
gap> sc:=SCNSSubdivision(ns);;
gap> sc.F;
[ 9, 27, 18 ]
gap>
```

7.3 Properties of `SCNormalSurface` objects

Although some properties of a normal surface can be computed by using the functions for simplicial complexes, there is a variety of properties needing specially designed functions. See below for a list.

7.3.1 SCNSDim

◊ `SCNSDim( normalsurface )` (function)

Returns: 2 if `normalsurface` is not empty, -1 otherwise.

Computes the dimension of a normal surface (which is always 2 in the non-empty case.

Example

```
gap> ns:=SCNSEmpty();;
gap> SCNSDim(ns);
-1
gap> ns:=SCNSFromFacets([[1,2,3],[1,2,4,5],[1,3,4,6],[2,3,5,6],[4,5,6]]);;
gap> SCNSDim(ns);
2
gap>
```

7.3.2 SCNSEulerCharacteristic

◊ `SCNSEulerCharacteristic( normalsurface )` (function)

Returns: an integer upon success, `fail` otherwise.

Computes the Euler characteristic of a normal surface.

Example

```
gap> SCLib.SearchByName("S^2xS^1");
[ [ 24, "S^2xS^1 (VT)" ], [ 33, "S^2xS^1 (VT)" ], [ 68, "S^2xS^1 (VT)" ],
[ 81, "S^2xS^1 (VT)" ], [ 82, "S^2xS^1 (VT)" ], [ 83, "S^2xS^1 (VT)" ], ...
]
gap> c:=SCLib.Load(24);;
gap> ns:=c.Slicing([[1..5],[6..10]]);;
gap> SCNSEulerCharacteristic(ns);
4
gap>
```

7.3.3 SCNSFVector

◊ `SCNSFVector( normalsurface )` (function)

Returns: a 1, 3 or 4 tuple of (non-negative) integer values upon success, `fail` otherwise.

Computes the f-vector of a normal surface, i. e. the number of vertices, edges, triangles and quadrilaterals of `normalsurface`.

Example

```
gap> SCLib.SearchByName("S^2xS^1");
[ [ 24, "S^2xS^1 (VT)" ], [ 33, "S^2xS^1 (VT)" ], [ 68, "S^2xS^1 (VT)" ],
[ 81, "S^2xS^1 (VT)" ], [ 82, "S^2xS^1 (VT)" ], [ 83, "S^2xS^1 (VT)" ], ...
]
gap> c:=SCLib.Load(24);;
gap> ns:=c.Slicing([[1..5],[6..10]]);;
gap> SCNSFVector(ns);
[ 20, 40, 16, 8 ]
```

```
gap>
```

7.3.4 SCNSFaceLattice

◊ `SCNSFaceLattice( complex )` (function)

Returns: a list of facet lists upon success, `fail` otherwise.

Computes the face lattice of a normal surface in original labeling. Triangles and quadrilaterals are stored separately (cf. `SCNSSkel` (7.3.9)).

Example

```
gap> c:=SCBdSimplex(4);;
gap> ns:=SCNSSlicing(c,[[1,2],[3..5]]);;
gap> SCNSFaceLattice(ns);
[ [ [ [ 1, 3 ] ], [ [ 1, 4 ] ], [ [ 1, 5 ] ], [ [ 2, 3 ] ], [ [ 2, 4 ] ],
      [ [ 2, 5 ] ] ],
  [ [ [ 1, 3 ], [ 1, 4 ] ], [ [ 1, 3 ], [ 1, 5 ] ], [ [ 1, 3 ], [ 2, 3 ] ],
    [ [ 1, 4 ], [ 1, 5 ] ], [ [ 1, 4 ], [ 2, 4 ] ], [ [ 1, 5 ], [ 2, 5 ] ],
    [ [ 2, 3 ], [ 2, 4 ] ], [ [ 2, 3 ], [ 2, 5 ] ], [ [ 2, 4 ], [ 2, 5 ] ] ],
  [ [ [ 1, 3 ], [ 1, 4 ], [ 1, 5 ] ], [ [ 2, 3 ], [ 2, 4 ], [ 2, 5 ] ] ],
  [ [ [ 1, 3 ], [ 1, 4 ], [ 2, 3 ], [ 2, 4 ] ],
    [ [ 1, 3 ], [ 1, 5 ], [ 2, 3 ], [ 2, 5 ] ],
    [ [ 1, 4 ], [ 1, 5 ], [ 2, 4 ], [ 2, 5 ] ] ] ]
gap> ns.F;
[ 6, 9, 2, 3 ]
gap>
```

7.3.5 SCNSFaceLatticeEx

◊ `SCNSFaceLatticeEx( complex )` (function)

Returns: a list of facet lists upon success, `fail` otherwise.

Computes the face lattice of a normal surface in standard labeling. Triangles and quadrilaterals are stored separately (cf. `SCNSSkelEx` (7.3.10)).

Example

```
gap> c:=SCBdSimplex(4);;
gap> ns:=SCNSSlicing(c,[[1,2],[3..5]]);;
gap> SCNSFaceLatticeEx(ns);
[ [ [ 1 ], [ 2 ], [ 3 ], [ 4 ], [ 5 ], [ 6 ] ],
  [ [ 1, 2 ], [ 1, 3 ], [ 1, 4 ], [ 2, 3 ], [ 2, 5 ],
    [ 3, 6 ], [ 4, 5 ], [ 4, 6 ], [ 5, 6 ] ],
  [ [ 1, 2, 3 ], [ 4, 5, 6 ] ],
  [ [ 1, 2, 4, 5 ], [ 1, 3, 4, 6 ], [ 2, 3, 5, 6 ] ] ]
gap> ns.F;
[ 6, 9, 2, 3 ]
gap>
```

7.3.6 SCNSFpBettiNumbers

◊ SCNSFpBettiNumbers(normalsurface, p) (function)

Returns: a list of non-negative integers upon success, fail otherwise

Computes the Betti numbers modulo p of normalsurface. Internally, normalsurface is subdivided and the Betti numbers are computed on the subdivision (cf. SCNSSubdivision (7.2.2)).

Example

```
gap> SCLib.SearchByName("(S^2xS^1)#20");
[ [ 688, "(S^2xS^1)#20" ] ]
gap> c:=SCLib.Load(688);;
gap> c.F;
[ 27, 298, 542, 271 ]
gap> ns:=c.Slicing([[1..13],[14..27]]);;
gap> SCNSFpBettiNumbers(ns,2);
[ 2, 14, 2 ]
gap>
```

7.3.7 SCNSGenus

◊ SCNSGenus(normalsurface) (function)

Returns: a non-negative integer upon success, fail otherwise.

Computes the genus of a normal surface.

Example

```
gap> SCLib.SearchByName("(S^2xS^1)#20");
[ [ 688, "(S^2xS^1)#20" ] ]
gap> c:=SCLib.Load(688);;
gap> c.F;
[ 27, 298, 542, 271 ]
gap> ns:=c.Slicing([[1..12],[13..27]]);;
gap> SCIsConnected(ns);
true
gap> SCNSGenus(ns);
7
gap>
```

7.3.8 SCNSHomology

◊ SCNSHomology(normalsurface) (function)

Returns: a list of homology groups upon success, fail otherwise

Computes the homology of normalsurface. Internally, normalsurface is subdivided and simplicial homology is computed on the subdivision (cf. SCNSSubdivision (7.2.2)).

Example

```
gap> SCLib.SearchByName("(S^2xS^1)#20");
[ [ 688, "(S^2xS^1)#20" ] ]
gap> c:=SCLib.Load(688);;
gap> c.F;
[ 27, 298, 542, 271 ]
gap> ns:=c.Slicing([[1..12],[13..27]]);;
gap> ns.Homology;
```

```
[ [ 0, [  ] ], [ 14, [  ] ], [ 1, [  ] ] ]
gap> ns:=c.Slicing([[1..13],[14..27]]);;
gap> ns.Homology;
[ [ 1, [  ] ], [ 14, [  ] ], [ 2, [  ] ] ]
gap>
```

7.3.9 SCNSSkel

◊ `SCNSSkel( normalsurface, dim )` (function)

Returns: a facet list of dimension `dim` upon success, `fail` otherwise.

Computes all faces of one type in original labeling: `dim` = 0 computes the vertices, `dim` = 1 computes the edges, `dim` = 2 computes the triangles, `dim` = 3 computes the quadrilaterals,

Example

```
gap> c:=SCBdSimplex(4);;
gap> ns:=SCNSSlicing(c,[[1],[2..5]]);;
gap> SCNSSkel(ns,1);
[ [ [ 1, 2 ], [ 1, 3 ] ], [ [ 1, 2 ], [ 1, 4 ] ], [ [ 1, 2 ], [ 1, 5 ] ],
  [ [ 1, 3 ], [ 1, 4 ] ], [ [ 1, 3 ], [ 1, 5 ] ], [ [ 1, 4 ], [ 1, 5 ] ] ]
gap>
```

Example

```
gap> c:=SCBdSimplex(4);;
gap> ns:=SCNSSlicing(c,[[1],[2..5]]);;
gap> SCNSSkel(ns,3);
[ ]
gap> ns:=SCNSSlicing(c,[[1,2],[3..5]]);;
gap> SCNSSkelEx(ns,3);
[ [ [ 1, 3 ], [ 1, 4 ], [ 2, 3 ], [ 2, 4 ] ],
  [ [ 1, 3 ], [ 1, 5 ], [ 2, 3 ], [ 2, 5 ] ],
  [ [ 1, 4 ], [ 1, 5 ], [ 2, 4 ], [ 2, 5 ] ] ]
gap>
```

7.3.10 SCNSSkelEx

◊ `SCNSSkelEx( normalsurface, dim )` (function)

Returns: a facet list of dimension `dim` upon success, `fail` otherwise.

Computes all faces of one type in standard labeling: `dim` = 0 computes the vertices, `dim` = 1 computes the edges, `dim` = 2 computes the triangles, `dim` = 3 computes the quadrilaterals,

Example

```
gap> c:=SCBdSimplex(4);;
gap> ns:=SCNSSlicing(c,[[1],[2..5]]);;
gap> SCNSSkelEx(ns,1);
[ [ 1, 2 ], [ 1, 3 ], [ 1, 4 ], [ 2, 3 ], [ 2, 4 ], [ 3, 4 ] ]
gap>
```

Example

```
gap> c:=SCBdSimplex(4);;
gap> ns:=SCNSSlicing(c,[[1],[2..5]]);;
```

```
gap> SCNSSkelEx(ns,3);
[ ]
gap> ns:=SCNSSlicing(c,[[1,2],[3..5]]);;
gap> SCNSSkelEx(ns,3);
[ [ 1, 2, 4, 5 ], [ 1, 3, 4, 6 ], [ 2, 3, 5, 6 ] ]
gap>
```

7.3.11 SCNSTopologicalType

◊ SCNSTopologicalType(ns) (function)

Returns: a string upon success, fail otherwise.

Determines the topological type of ns via the classification theorem for closed surfaces. If ns is not connected, the topological type of each connected component is computed.

Example

```
gap> SCLib.SearchByName("(S^2xS^1)#20");
[ [ 688, "(S^2xS^1)#20" ] ]
gap> c:=SCLib.Load(688);;
gap> c.F;
[ 27, 298, 542, 271 ]
gap> for i in [1..26] do
> ns:=SCNSSlicing(c,[[1..i],[i+1..27]]);
> Print(ns.TopologicalType,"\n");
> od;
S^2
S^2
S^2
S^2
S^2 U S^2
S^2 U S^2
S^2
(T^2)#3
(T^2)#5
(T^2)#4
(T^2)#3
(T^2)#7
(T^2)#7 U S^2
(T^2)#7 U S^2
(T^2)#7 U S^2
(T^2)#8 U S^2
(T^2)#7 U S^2
(T^2)#8
(T^2)#6
(T^2)#6
(T^2)#5
(T^2)#3
(T^2)#2
T^2
S^2
S^2
gap>
```

Chapter 8

Library and I/O

8.1 Simplicial complex library

simpcomp contains a library of simplicial complexes on few vertices, most of them (combinatorial) triangulations of manifolds. The user can load these known triangulations from the library in order to study their properties or to construct new triangulations out of the known ones. For exmaple, a user could determine the topological type of a given triangulation – which can be quite tedious if done by hand – by establishing a PL equivalence to a complex in the library.

Among other known triangulations, the library contains all of the vertex transitive triangulations of d-manifolds, $d \leq 11$ with few ($n \leq 13$ and $n \leq 15$ for $d = 2,3,9,10,11$) vertices classified by Frank Lutz that can be found on his "Manifold Page" `http://www.math.tu-berlin.de/diskregeom/stellar/`, along with some triangulations of sphere bundles and vertex transitive triangulations of pseudomanifolds.

See `SCLib` (8.1.2) for a naming convention used for the global library of simpcomp.

8.1.1 SCIsLibRepository

◊ `SCIsLibRepository( object )` (filter)

Returns: `true` / `false` upon success, `fail` otherwise.

Filter for the category of a library repository `SCIsLibRepository` used by the simpcomp library. The category `SCLibRepository` is derived from the category `SCPropertyObject`.

Example

```
gap> SCIsLibRepository(SCLib); #the global library is stored in SCLib
true
```

8.1.2 SCLib

◊ `SCLib` (global variable)

The global variable `SCLib` contains the library object of the global library of simpcomp through which the user can access the library. The path to the global library is `GAPROOT/pkg/simpcomp/complexes`.

The naming convention in the global library is the following: complexes are usually named by their topological type. As usual, ‘ Sˆd’ denotes a d-sphere, ‘T’ a torus, ‘x’ the cartesian product, ‘˜’ the twisted product and ‘#’ the connected sum. The Klein Bottle is denoted by ‘K’ or ‘Kˆ2’.

Example

```
gap> SCLib;
[Simplicial complex library. Properties:
 CalculateIndexAttributes=true
 Number of complexes in library=689
 IndexAttributes=["Name", "Date", "Dim", "F", "G", "H", "Chi", "Homology"]
 Loaded=true
 Path="GAPROOT/pkg/simpcomp/complexes/"
]
gap> SCLib.Size;
689
gap> SCLib.SearchByName("Sˆ4˜");
[ [ 204, "Sˆ4˜Sˆ1 (VT)" ], [ 339, "Sˆ4˜Sˆ1 (VT)" ], [ 341, "Sˆ4˜Sˆ1 (VT)" ],
  [ 438, "Sˆ4˜Sˆ1 (VT)" ], [ 493, "Sˆ4˜Sˆ1 (VT)" ], [ 494, "Sˆ4˜Sˆ1 (VT)" ],
  [ 495, "Sˆ4˜Sˆ1 (VT)" ], [ 496, "Sˆ4˜Sˆ1 (VT)" ], [ 497, "Sˆ4˜Sˆ1 (VT)" ],
  [ 500, "Sˆ4˜Sˆ1 (VT)" ], [ 501, "Sˆ4˜Sˆ1 (VT)" ], [ 502, "Sˆ4˜Sˆ1 (VT)" ] ]
gap> SCLib.Load(204);
[SimplicialComplex

 Properties known: AltshulerSteinberg, Boundary, Chi, ConnectedComponents,
                   Dim, DualGraph, F, Faces, Facets, G, H, HasBoundary,
                   HasInterior, Homology, Interior, IsConnected,
                   IsEulerianManifold, IsOrientable, IsPM, IsPure,
                   MinimalNonFaces, Name, Neighborliness, Orientation,
                   Reference, StronglyConnected, VertexLabels, Vertices.

 Name="Sˆ4˜Sˆ1 (VT)"
 Dim=5
 Chi=0
 F=[ 13, 78, 195, 260, 195, 65 ]
 G=[ 6, 21, -35 ]
 H=[ 7, 28, -7, 28, 7, 1 ]
 HasBoundary=false
 HasInterior=true
 Homology=[ [ 0, [ ] ], [ 1, [ ] ], [ 0, [ ] ], [ 0, [ ] ], [ 0, [ 2 ] ], [ 0,\
 [ ] ] ]
 IsConnected=true
 IsEulerianManifold=true
 IsOrientable=false
 IsPM=true
 IsPure=true
 Neighborliness=2

/SimplicialComplex]
```

8.1.3 SCLibAdd

◊ SCLibAdd(repository, complex[, name]) (function)

Returns: true upon success, fail otherwise.

Adds a given complex complex to a given repository repository. The complex is saved to a file with suffix .sc in the repositories base path, where the file name is either formed from the optional argument name and the current time or taken fom the name of the complex, if it is named.

Example

```
gap> myRepository:=SCLibInit("~/myrepository");
#I  SCLibInit: made directory "~/myrepository/" for user library.
#I  SCIntFunc.SCLibInit: index not found -- trying to reconstruct it.
#I  SCLibUpdate: rebuilding index for ~/myrepository/.
#I  SCLibUpdate: rebuilding index done.
gap> complex:=SCBdCrossPolytope(4);;
gap> SCLibAdd(myRepository,complex);
#I  SCLibAdd: saving complex to file "complex_Bd(beta4)_2009-10-29_17-12-36.sc\
".
true
gap> myRepository.Add(complex);; # alternative syntax
```

8.1.4 SCLibAllComplexes

◊ SCLibAllComplexes(repository) (function)

Returns: list of entries of the form [ID,NAME] upon success, fail otherwise.

Returns a list with entries of the form [ID,NAME] of all the complexes in the given repository repository.

Example

```
gap> all:=SCLibAllComplexes(SCLib);;
gap> all[1];
[ 1, "Moebius Strip" ]
gap> Length(all);
689
```

8.1.5 SCLibDelete

◊ SCLibDelete(repository, id) (function)

Returns: true upon success, fail otherwise.

Deletes the simplicial complex with the given id id from the given repository repository. Apart from deleting the complexes' index entry, the associated .sc file is also deleted.

Example

```
gap> myRepository:=SCLibInit("~/myrepository");
#I  SCLibInit: made directory "~/myrepository/" for user library.
#I  SCIntFunc.SCLibInit: index not found -- trying to reconstruct it.
#I  SCLibUpdate: rebuilding index for ~/myrepository/.
#I  SCLibUpdate: rebuilding index done.
gap> SCLibDelete(myRepository,1);
true
```

8.1.6 SCLibDetermineTopologicalType

◊ `SCLibDetermineTopologicalType( [repository,] complex )` (function)

Returns: a simplicial complex or a list of items integer ids upon success, `fail` otherwise.

Tries to determine the topological type of a given complex `complex` by first looking for complexes with matching homology in the library repository `repository` (if no repository is passed, the global repository `SCLib` is used) and either returns a simplicial complex object (that is combinatorially isomorphic to the complex given) or a list of library ids of complexes in the library with the same homology as the complex provided.
The ids obtained in this way can then be used to compare the complexes with `complex` by `SCEquivalent` (6.2.2).
If no complexes with matching homology can be found, the empty set is returned.

Example

```
gap> c:=SCFromFacets([[1,2,3],[1,2,6],[1,3,5],[1,4,5],[1,4,6],
                      [2,3,4],[2,4,5],[2,5,6],[3,4,6],[3,5,6]]);;
gap> SCLibDetermineTopologicalType(c);
[SimplicialComplex

 Properties known: AltshulerSteinberg, AutomorphismGroup,
                   AutomorphismGroupSize, AutomorphismGroupStructure,
                   AutomorphismGroupTransitivity, Boundary, Chi,
                   ConnectedComponents, Dim, DualGraph, F, Faces, Facets, G,
                   Generators, H, HasBoundary, HasInterior, Homology,
                   Interior, IsCentrallySymmetric, IsConnected,
                   IsEulerianManifold, IsOrientable, IsPM, IsPure,
                   MinimalNonFaces, Name, Neighborliness, Orientation,
                   Reference, StronglyConnected, VertexLabels, Vertices.

 Name="RP^2 (VT)"
 Dim=2
 AutomorphismGroupSize=60
 AutomorphismGroupStructure="A5"
 AutomorphismGroupTransitivity=2
 Chi=1
 F=[ 6, 15, 10 ]
 G=[ 2 ]
 H=[ 3, 6, 0 ]
 HasBoundary=false
 HasInterior=true
 Homology=[ [ 0, [ ] ], [ 0, [ 2 ] ], [ 0, [ ] ] ]
 IsCentrallySymmetric=false
 IsConnected=true
 IsEulerianManifold=true
 IsOrientable=false
 IsPM=true
 IsPure=true
 Neighborliness=2

/SimplicialComplex]
```

8.1.7 SCLibFlush

◊ SCLibFlush(repository, confirm) (function)

Returns: true upon success, fail otherwise.

Completely empties a given repository repository. The index and all simplicial complexes in this repository are deleted. The second argument, confirm, must be the string "yes" in order to confirm the deletion.

Example

```
gap> myRepository:=SCLibInit("~/repository");;
#I  SCLibInit: made directory "~/repository/" for user library.
#I  SCIntFunc.SCLibInit: index not found -- trying to reconstruct it.
#I  SCLibUpdate: rebuilding index for ~/repository/.
#I  SCLibUpdate: rebuilding index done.
gap> SCLibFlush(myRepository,"yes");
#I  SCLibUpdate: rebuilding index for /home/effenbfx/testrepo/.
#I  SCLibUpdate: rebuilding index done.
true
```

8.1.8 SCLibInit

◊ SCLibInit(dir) (function)

Returns: library repository of type SCLibRepository upon success, fail otherwise.

This function initializes a library repository object for the given directory dir and returns that library repository object in case of success. The returned object then provides a mean to access the library repository via the SCLib-functions of simpcomp. The global library repository of simpcomp is loaded automatically at startup and is stored in the variable SCLib. User repositories can be created by calling SCLibInit with a desired destination directory. Note, that each repository must reside in a different path since otherwise data may get lost. The function first tries to load the repository index for the given directory to rebuild it (by calling SCLibUpdate) if loading the index fails. The library index of a library repository is stored in its base path in the XML file complexes.idx, the complexes are stored in files with suffix .sc, also in XML format.

Example

```
gap> myRepository:=SCLibInit("~/myrepository");
#I  SCLibInit: made directory "~/myrepository/" for user library.
#I  SCIntFunc.SCLibInit: index not found -- trying to reconstruct it.
#I  SCLibUpdate: rebuilding index for ~/myrepository/.
#I  SCLibUpdate: rebuilding index done.
[Simplicial complex library. Properties:
CalculateIndexAttributes=true
Number of complexes in library=0
IndexAttributes=["Name", "Date", "Dim", "F", "G", "H", "Chi", "Homology"]
Loaded=true
Path="~/myrepository/"
]
```

8.1.9 SCLibIsLoaded

◊ SCLibIsLoaded(repository) (function)

Returns: true or false upon succes, fail otherwise.

Returns `true` when a given library repository `repository` is in loaded state. This means that the directory of this repository is accessible and a repository index file for this repository exists in the repositories' path.

Example

```
gap> SCLibIsLoaded(SCLib);
true
gap> SCLib.IsLoaded;
true
```

8.1.10 SCLibSearchByAttribute

◊ `SCLibSearchByAttribute( repository, expr )` (function)

Returns: A list of items of the form `[ID, NAME]` upon success, `fail` otherwise.

Searches a given repository `repository` for complexes for which the boolean expression `expr`, passed as string, evaluates to `true` and returns a list of complexes with entries of the form `[ID, NAME]` or `fail` upon error. The expression may use all GAP functions and can access all the indexed attributes of the complexes in the given repository for the query. The standard attributes are: Dim (Dimension), F (f-vector), G (g-vector), H (h-vector), Chi (Euler characteristic), Homology, Name. See `SCLib` for the set of indexed attributes of the global library of simpcomp.

Example

```
#search for all 3-neighborly complexes of dimension 4 in the global library
gap> SCLibSearchByAttribute(SCLib,"Dim=4 and F[3]=Binomial(F[1],3)");
[ [ 17, "CP^2 (VT)" ], [ 584, "K3 surface" ] ]
# alternative syntax
gap> SCLib.SearchByAttribute("Dim=4 and F[3]=Binomial(F[1],3)");
[ [ 17, "CP^2 (VT)" ], [ 584, "K3 surface" ] ]
```

8.1.11 SCLibSearchByName

◊ `SCLibSearchByName( repository, name )` (function)

Returns: A list of items of the form `[ID, NAME]` upon success, `fail` otherwise.

Searches a given repository `repository` for complexes that contain the string `name` as a substring of their name attribute and returns a list of the complexes found with entries of the form `[ID, NAME]`. See `SCLib` (8.1.2) for a naming convention used for the global library of simpcomp.

Example

```
gap> SCLibSearchByName(SCLib,"K3");
[ [ 584, "K3 surface" ] ]
gap> SCLib.SearchByName("K3"); #alternative syntax
[ [ 584, "K3 surface" ] ]
gap> SCLib.SearchByName("S^4x"); #search for products with S^4
[ [ 291, "S^4xS^1 (VT)" ], [ 340, "S^4xS^1 (VT)" ], [ 342, "S^4xS^1 (VT)" ],
  [ 571, "S^4xS^2" ], [ 627, "S^4xS^3" ], [ 655, "S^4xS^4" ] ]
```

8.1.12 SCLibSize

◊ `SCLibSize( repository )` (function)

Returns: integer upon success, `fail` otherwise.

Returns the number of complexes contained in the given repository `repository`. Fails if the library repository was not previously loaded with `SCLibInit`.

Example

```
gap> SCLibSize(SCLib); #SCLib is the repository of the global library
689
```

8.1.13 SCLibUpdate

◊ `SCLibUpdate( repository, OR, path[, recalc] )` (function)

Returns: library repository of type `SCLibRepository` upon success, fail otherwise.

Recreates the index of a given repository (either via a repository object `repository` or a base path `path` of a repository) by scanning the base path for all `.sc` files containing simplicial complexes of the repository. Returns a repository object with the newly created index on success of fail in case of an error. The optional boolean argument `recalc` forces simpcomp to recompute all the indexed properties (such as f-vector, homology, etc.) of the simplicial complexes in the repository if set to `true`.

Example

```
gap> myRepository:=SCLibInit("~/repository");;
#I  SCLibInit: made directory "~/repository/" for user library.
#I  SCIntFunc.SCLibInit: index not found -- trying to reconstruct it.
#I  SCLibUpdate: rebuilding index for ~/repository/.
#I  SCLibUpdate: rebuilding index done.
gap> SCLibUpdate(myRepository);
#I  SCLibUpdate: rebuilding index for ~/repository/.
#I  SCLibUpdate: rebuilding index done.
[Simplicial complex library. Properties:
CalculateIndexAttributes=true
Number of complexes in library=0
IndexAttributes=["Name", "Date", "Dim", "F", "G", "H", "Chi", "Homology"]
Loaded=true
Path="~/myrepository/"
]
```

8.1.14 SCLibStatus

◊ `SCLibStatus( repository )` (function)

Returns: the library repository object of type `SCLibRepository` passed as argument.

Lets GAP print the status of a given library repository `repository`. `IndexAttributes` is the list of attributes indexed for this repository. If `CalculateIndexAttributes` is true, the index attributes for a complex added to the library are calculated automatically upon addition of the complex, otherwise this is left to the user and only precalculated attributes are indexed.

Example

```
gap> SCLibStatus(SCLib);
[Simplicial complex library. Properties:
CalculateIndexAttributes=true
Number of complexes in library=N
IndexAttributes=["Name", "Date", "Dim", "F", "G", "H", "Chi", "Homology"]
Loaded=true
```

```
 Path="GAPROOT/pkg/simpcomp/complexes/"
]
```

8.2 simpcomp input / output functions

This section contains a description of the input/output-functionality provided by simpcomp. The package provides the functionality to save and load simplicial complexes (and their known properties) to, respectively from files in XML format. Furthermore it provides the user with functions to export simplicial complexes into polymake format (for this format there also exists rudimentary import functionality), as JavaView geometry or in form of a latex table. For importing more complex polymake data the package polymaking [RÖ07] can be used.

8.2.1 SCLoad

◊ `SCLoad( filename )` (function)

Returns: simplicial complex of type `SCSimplicialComplex` if successful, fail otherwise.

Loads a simplicial complex stored in XML format from a file specified in `filename` (as string). If the filename provided does not end in `.sc`, this suffix is appended to the file name.

Example

```
gap> c:=SCBdSimplex(3);;
gap> SCSave(c,"bddelta3");
true
gap> c:=SCLoad("bddelta3");
[SimplicialComplex

 Properties known: AutomorphismGroup, AutomorphismGroupOrder,
                   AutomorphismGroupStructure, AutomorphismGroupTransitivity,
                   Chi, Dim, F, Facets, Generators, HasBoundary, Homology,
                   IsConnected, IsStronglyConnected, Name, TopologicalType,
                   VertexLabels.

 Name="S^2_4"
 Dim=2
 TopologicalType="S^2"
 Chi=2
 F=[4, 6, 4]
 Homology=[[0, []], [0, []], [1, []]]
 AutomorphismGroupStructure="S4"

/SimplicialComplex]
```

8.2.2 SCSave

◊ `SCSave( complex, filename )` (function)

Returns: true if successful, fail otherwise.

Saves a simplicial complex `complex` to a file specified by `filename` (as string) in XML format. If the file name provided does not end in `.sc`, this suffix is appended to the file name.

Example

```
gap> c:=SCBdSimplex(3);;
gap> SCSave("bddelta3");
true
```

8.2.3 SCExportPolymake

◊ `SCExportPolymake( complex, filename )` (function)

Returns: true upon success, fail otherwise.

Exports the facet list with vertex labels of a given simplicial complex `complex` in `polymake` format to a file specified by `filename`. Currently, only the export in the format of `polymake` version 2.3 is supported.

Example

```
gap> c:=SCBdCrossPolytope(4);;
gap> SCExportPolymake(c,"bdbeta4.poly");
true
```

8.2.4 SCImportPolymake

◊ `SCImportPolymake( filename )` (function)

Returns: simplicial complex of type `SCSimplicialComplex` upon success, fail otherwise.

Imports the facet list of a topaz polymake file (discarding any vertex labels) specified by `filename` and creates a simplicial complex object from these facets.

Example

```
gap> c:=SCImportPolymake("3dimcomplex.poly");
[SimplicialComplex

Properties known: Chi, Dim, Facets, VertexLabels.

 Name="unnamed complex n"
 Dim=3

/SimplicialComplex]
```

8.2.5 SCExportLatexTable

◊ `SCExportLatexTable( complex, filename, itemsperline )` (function)

Returns: true on success, fail otherwise.

Exports the facet list of a given simplicial complex `complex` in form of a LaTeX table to a file specified by `filename`. The argument `itemsperline` specifies, how many columns the exported table should have. Facets are exported in the format $\langle v_1, \ldots, v_{dim-1} \rangle$.

Example

```
gap> c:=SCBdSimplex(5);;
gap> SCExportLatexTable(c,"bd5simplex.tex",5);
true
gap> c.ExportLatexTable("bd5simplex.tex",5);
```

```
true
```

8.2.6 SCExportJavaView

◊ `SCExportJavaView( complex, file, coords )` (function)

Returns: true on success, fail otherwise.

Exports the facet list of the given simplicial complex `complex` in `JavaView` format (file name suffix `.jvx`) to a file specified by `filename` (as string). The list `coords` must contain a 3-tuple of real coordinates for each vertex of the complex, either as tuple of length three containing the coordinates or as string of the form `"x.x y.y z.z"` with decimal numbers `x.x`, `y.y`, `z.z` for the three coordinates (i.e. `"1.0 0.0 0.0"`). Warning: as GAP only has rudementary support for floating point values, currently only integer numbers can be used as coordinates.

Example

```
gap> coords:=[[1,0,0],[0,1,0],[0,0,1]];;
gap> SCExportJavaView(SCBdSimplex(2),"triangle.jvx",coords);
true
```

Chapter 9

Miscellaneous functions

The bahaviour of simpcomp can be changed by setting global options, functions for which will be described in the following.

9.1 simpcomp error handling

simpcomp can be configured to either enter a break loop when encountering an error or to just log the error and return `fail` in the called function. Examples for errors would be a computation failing or a `SCSimplicialComplex` object having an inconsistent internal state

9.1.1 SCErrorBreak

◊ `SCErrorBreak( flag )` (function)

Returns: `true` upon success, `fail` otherwise.

Sets whether simpcomp enters a break loop when encountering an error or just returns `fail` in the function called.

Example

```
gap> SCErrorBreak(true);
true
```

9.1.2 SCErrorMail

◊ `SCErrorMail( flag )` (function)

Returns: `true` upon success, `fail` otherwise.

Sets whether simpcomp should send an email to the address specified by `SCMailSetAddress` (9.2.6) when encountering an error.

Example

```
gap> SCErrorMail(true);
true
```

9.2 Email notification system

simpcomp comes with an email notification system that can be used for being notified of the progress of lengthy computations (such as reducing a complex via bistellar flips). See below for a descriptions of the mail notification related functions.

9.2.1 SCMailClearPending

◊ `SCMailClearPending( )` (function)

Returns: nothing.

Clears a pending mail message.

Example

```
gap> SCMailClearPending();
```

9.2.2 SCMailIsEnabled

◊ `SCMailIsEnabled( )` (function)

Returns: `true` or `false` upon success, `fail` otherwise.

Returns `true` when the mail notification system of simpcomp is enabled, `false` otherwise. Default setting is `false`.

Example

```
gap> SCMailIsEnabled();
false
```

9.2.3 SCMailIsPending

◊ `SCMailIsPending( )` (function)

Returns: `true` or `false` upon success, `fail` otherwise.

Returns `true` when an email of the simpcomp email notification system is pending, `false` otherwise.

Example

```
gap> SCMailIsPending();
false
```

9.2.4 SCMailSend

◊ `SCMailSend( message[, starttime][,][forcesend] )` (function)

Returns: `true` when the message was sent, `false` if it was not send, `fail` upon an error.

Tries to send an email to the address specified by `SCMailSetAddress` (9.2.6) using the unix program `mail`. The optional parameter `starttime` specifies the starting time (as the integer unix timestamp) a calculation was started (then the duration of the calculation is included in the email), the optional boolean parameter `forcewrite` can be used to force the sending of an email, even if this violates the minimal email sending interval, see `SCMailSetMinInterval` (9.2.8).

Example

```
gap> SCMailIsEnabled();
true
gap> SCMailSend("Hello, this is simpcomp.");
true
```

9.2.5 SCMailSendPending

◊ `SCMailSendPending( )` (function)

Returns: `true` upon success, `fail` otherwise.

Tries to send a pending email of the simpcomp email notification system. Returns `true` on success or if there was no mail pending.

Example

```
gap> SCMailSendPending();
true
```

9.2.6 SCMailSetAddress

◊ `SCMailSetAddress( address )` (function)

Returns: `true` upon success, `fail` otherwise.

Sets the email address that should be used to send notification messages and enables the mail notification system by calling `SCMailSetEnabled` (9.2.7)(`true`).

Example

```
gap> SCMailSetAddress("johndoe@somehost");
true
```

9.2.7 SCMailSetEnabled

◊ `SCMailSetEnabled( flag )` (function)

Returns: `true` upon success, `fail` otherwise.

Enables or disables the mail notification system of simpcomp. By default it is disabled. Returns `fail` if no email message was previously set with `SCMailSetAddress` (9.2.6).

Example

```
gap> SCMailSetAddress("johndoe@somehost"); #enables mail notification
true
gap> SCMailSetEnabled(false);
true
```

9.2.8 SCMailSetMinInterval

◊ `SCMailSetMinInterval( interval )` (function)

Returns: `true` upon success, `fail` otherwise.

Sets the minimal time interval in seconds that mail messages can be sent by simpcomp. This prevents a flooding of the specified email address with messages sent by simpcomp. Default is 3600, i.e. one hour.

Example

```
gap> SCMailSetMinInterval(7200);
true
```

9.3 Testing the functionality of simpcomp

simpcomp makes use of the GAP internal testing mechanisms and provides the user with a function to test the functionality of the package.

9.3.1 SCRunTest

◊ `SCRunTest( )` (function)

Returns: `true` upon success, `fail` otherwise.

Test whether the package simpcomp is functional by calling `ReadTest("GAPROOT/pkg/simpcomp/tst/simpcomp.tst");`.

Example

```
gap> SCRunTest();
+ test simpcomp package, version 1.1.9
+ GAP4stones: 69988
true
```

Chapter 10

Property handlers

As explained in chapter 3, an object of type `SCSimplicialComplex` or `SCNormalSurface` provides a set of property handlers for easily accessing and computing its properties using the .-operator.
For example, the f-vector of a simplicial complex / normal surface `c` that is stored as `SCSimplicialComplex` object can be accessed via the statement `c.F;`. See below for a list of all properties supported by a `SCSimplicialComplex` and `SCNormalSurface` object.

10.1 Property handlers of a `SCSimplicialComplex`

This section contains a table of all property handlers of a `SCSimplicialComplex` object.

PROPERTY HANDLER	FUNCTION CALLED
ASDet	`SCAltshulerSteinberg` (4.5.1)
AlexanderDual	`SCAlexanderDual` (4.6.1)
AutomorphismGroup	`SCAutomorphismGroup` (4.5.2)
AutomorphismGroupInternal	`SCAutomorphismGroupInternal` (4.5.3)
Boundary	`SCBoundary` (4.5.4)
BoundaryOperatorMatrix	`SCBoundaryOperatorMatrix` (5.1.1)
Chi	`SCEulerCharacteristic` (4.5.7)
CoboundaryOperatorMatrix	`SCCoboundaryOperatorMatrix` (5.2.1)
Cohomology	`SCCohomology` (5.2.2)
CohomologyBasis	`SCCohomologyBasis` (5.2.3)
CohomologyBasisAsSimplices	`SCCohomologyBasisAsSimplices` (5.2.4)
CollapseGreedy	`SCCollapseGreedy` (4.6.2)
Cone	`SCCone` (4.6.3)
ConnectedComponents	`SCConnectedComponents` (4.3.3)
Copy	`SCCopy` (3.1.2)
CupProduct	`SCCupProduct` (5.2.5)
DeletedJoin	`SCDeletedJoin` (4.6.4)
DetermineTopologicalType	`SCLibDetermineTopologicalType` (8.1.6)
Difference	`SCDifference` (4.6.5)
Dim	`SCDim` (4.5.5)
DualGraph	`SCDualGraph` (4.5.6)
Equivalent	`SCEquivalent` (6.2.2)
ExportJavaView	`SCExportJavaView` (8.2.6)
ExportLatexTable	`SCExportLatexTable` (8.2.5)
ExportPolymake	`SCExportPolymake` (8.2.3)

Join	SCJoin (4.6.12)
LabelMax	SCLabelMax (4.4.1)
LabelMin	SCLabelMin (4.4.2)
Labels	SCLabels (4.4.3)
Link	SCLink (4.6.13)
Links	SCLinks (4.6.14)
Load	SCLoad (8.2.1)
MinimalNonFaces	SCMinimalNonFaces (4.5.39)
MinimalNonFacesEx	SCMinimalNonFacesEx (4.5.40)
Move	SCMove (6.2.8)
Moves	SCMoves (6.2.9)
Name	SCName (4.5.41)
Neighborliness	SCNeighborliness (4.5.42)
Neighbors	SCNeighbors (4.6.15)
NeighborsEx	SCNeighborsEx (4.6.16)
Orientation	SCOrientation (4.5.43)
RMoves	SCRMoves (6.2.10)
Recalc	SCForceRecalc (3.1.4)
Reduce	SCReduceComplex (6.2.12)
ReduceAsSubcomplex	SCReduceAsSubcomplex (6.2.11)
ReduceEx	SCReduceComplexEx (6.2.13)
Relabel	SCRelabel (4.4.4)
RelabelStandard	SCRelabelStandard (4.4.5)
RelabelTransposition	SCRelabelTransposition (4.4.6)
Rename	SCRename (4.4.7)
Save	SCSave (8.2.2)
Shelling	SCShelling (4.6.17)
ShellingExt	SCShellingExt (4.6.18)
Shellings	SCShellings (4.6.19)
Skel	SCSkel (4.5.44)
SkelEx	SCSkelEx (4.5.45)
Slicing	SCNSSlicing (7.1.4)
SortComplex	SCSortComplex (4.4.8)
Span	SCSpan (4.6.20)
SpanningTree	SCSpanningTree (4.5.46)
Star	SCStar (4.6.21)
Stars	SCStars (4.6.22)
StronglyConnectedComponents	SCStronglyConnectedComponents (4.3.11)
Suspension	SCSuspension (4.6.23)
Union	SCUnion (4.6.24)
UnlabelFace	SCUnlabelFace (4.4.9)
VertexIdentification	SCVertexIdentification (4.6.25)
Vertices	SCVertices (4.5.47)
VerticesEx	SCVerticesEx (4.5.48)
Wedge	SCWedge (4.6.26)

10.2 Property handlers of a `SCNormalSurface`

This section contains a table of all property handlers of a `SCNormalSurface` object.

PROPERTY HANDLER	FUNCTION CALLED
Betti	SCNSFpBettiNumbers
Chi	SCNSEulerCharacteristic
ConnComp	SCConnectedComponents
Connected	SCIsConnected
Copy	SCNSCopy
Dim	SCNSDim
F	SCNSFVector
FaceLattice	SCNSFaceLattice
Faces	SCNSSkel
Facets	SCFacets
Genus	SCNSGenus
Homology	SCNSHomology
IsEmpty	SCIsEmpty
Labels	SCLabels
Name	SCName
Triangulation	SCNSSubdivision
Type	SCNSTopologicalType
Vertices	SCVertices

Chapter 11

A demo session with SimpComp

This chapter contains the transcript of a demo session with SimpComp that is intended to give an insight into what things can be done with this package. Of course this only scratches the surface of the functions provided by SimpComp, see chapters 4 through 9 for further functions provided by SimpComp.

11.1 Creating a `SCSimplicialComplex` object

Simplicial complex objects can either be created from a facet list, from a function generating triangulations of standard complexes or by expanding difference cycles. There are also functions creating new simplicial complexes from old, see chapter 4 which will be described in the next sections.

Example

```
gap> #first run functionality test on simpcomp
gap> SCRunTest();
+ test simpcomp package, version 1.1.9
+ GAP4stones: 69988
true
gap> #all ok
gap> c1:=SCFromFacets([[1,2],[2,3],[3,1]]);
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="unnamed complex 1"
 Dim=1

/SimplicialComplex]
gap> c2:=SCBdSimplex(2);
[SimplicialComplex

 Properties known: AutomorphismGroup, AutomorphismGroupOrder,
                   AutomorphismGroupStructure, AutomorphismGroupTransitivity,
                   Chi, Dim, F, Facets, Generators, HasBounday, Homology,
                   IsConnected, IsStronglyConnected, Name, TopologicalType,
                   VertexLabels.

 Name="S^1_3"
```

```
 Dim=1
 AutomorphismGroupStructure="S3"
 AutomorphismGroupTransitivity=3
 Chi=0
 F=[ 3, 3 ]
 Homology=[ [ 0, [ ] ], [ 1, [ ] ] ]
 IsConnected=true
 IsStronglyConnected=true
 TopologicalType="S^1"

/SimplicialComplex]
gap> c3:=SCFromDifferenceCycles([[1,1,6],[3,3,2]]);
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="complex from diffcycles [ [ 1, 1, 6 ], [ 3, 3, 2 ] ]"
 Dim=2

/SimplicialComplex]
gap>
```

11.2 Working with a `SCSimplicialComplex` object

As described in section 3.2 there are two several ways of accessing an object of type `SCSimplicialComplex`. An example for the two equivalent ways is given below. The preference will be given to the object oriented notation in this demo session. The code listed below

Example

```
gap> c:=SCBdSimplex(3);; # create a simplicial complex object
gap> SCFVector(c);
[ 4, 6, 4 ]
gap> SCSkel(c,0);
[ [ 1 ], [ 2 ], [ 3 ], [ 4 ] ]
gap>
```

is equivalent to

Example

```
gap> c:=SCBdSimplex(3);; # create a simplicial complex object
gap> c.F;
[ 4, 6, 4 ]
gap> c.Skel(0);
[ [ 1 ], [ 2 ], [ 3 ], [ 4 ] ]
gap>
```

11.3 Calculating properties of a `SCSimplicialComplex` object

SimpComp provides a variety of functions for calculating properties of simplicial complexes, see chapter 4. All these properties are only calculated once and stored in the simplicial complex object.

Example

```
gap> c1.F;
[ 3, 3 ]
gap> c1.FaceLattice;
[ [ [ 1 ], [ 2 ], [ 3 ] ], [ [ 1, 2 ], [ 1, 3 ], [ 2, 3 ] ] ]
gap> c1.AutomorphismGroup;
S3
gap> c1.Generators;
[ [ [ 1, 2 ], 3 ] ]
gap> c3.Facets;
[ [ 1, 2, 3 ], [ 1, 2, 8 ], [ 1, 3, 6 ], [ 1, 4, 6 ], [ 1, 4, 7 ],
  [ 1, 7, 8 ], [ 2, 3, 4 ], [ 2, 4, 7 ], [ 2, 5, 7 ], [ 2, 5, 8 ],
  [ 3, 4, 5 ], [ 3, 5, 8 ], [ 3, 6, 8 ], [ 4, 5, 6 ], [ 5, 6, 7 ],
  [ 6, 7, 8 ] ]
gap> c3.F;
[ 8, 24, 16 ]
gap> c3.G;
[ 4 ]
gap> c3.H;
[ 5, 11, -1 ]
gap> c3.ASDet;
186624
gap> c3.Chi;
0
gap> c3.Generators;
[ [ [ 1, 2, 3 ], 16 ] ]
gap> c3.HasBoundary;
false
gap> c3.IsConnected;
true
gap> c3.IsCentrallySymmetric;
true
gap> c3.Vertices;
[ 1, 2, 3, 4, 5, 6, 7, 8 ]
gap> c3.ConnectedComponents;
[ [SimplicialComplex

     Properties known: Dim, Facets, Name, VertexLabels.

     Name="Connected component #1 of complex from diffcycles [ [ 1, 1, 6 ], [ \
3, 3, 2 ] ]"
     Dim=2

    /SimplicialComplex] ]
gap> c3.UnknownProperty;
#I  SCPropertyObject: unhandled property 'UnknownProperty'. Handled properties\
 are [ "Equivalent", "IsKStackedSphere", "IsManifold", "IsMovable", "Move",
  "Moves", "RMoves", "ReduceAsSubcomplex", "Reduce", "ReduceEx", "Copy",
  "Recalc", "ASDet", "AutomorphismGroup", "AutomorphismGroupInternal",
  "Boundary", "ConnectedComponents", "Dim", "DualGraph", "Chi", "F",
  "FaceLattice", "FaceLatticeEx", "Faces", "FacesEx", "Facets", "FacetsEx",
  "FpBetti", "FundamentalGroup", "G", "Generators", "GeneratorsEx", "H",
  "HasBoundary", "HasInterior", "Homology", "Incidences", "IncidencesEx",
```

```
  "Interior", "IsCentrallySymmetric", "IsConnected", "IsEmpty",
  "IsEulerianManifold", "IsHomologySphere", "IsInKd", "IsKNeighborly",
  "IsOrientable", "IsPM", "IsPure", "IsShellable", "IsStronglyConnected",
  "MinimalNonFaces", "MinimalNonFacesEx", "Name", "Neighborliness",
  "Orientation", "Skel", "SkelEx", "SpanningTree",
  "StronglyConnectedComponents", "Vertices", "VerticesEx",
  "BoundaryOperatorMatrix", "HomologyBasis", "HomologyBasisAsSimplices",
  "HomologyInternal", "CoboundaryOperatorMatrix", "Cohomology",
  "CohomologyBasis", "CohomologyBasisAsSimplices", "CupProduct",
  "IntersectionForm", "IntersectionFormParity",
  "IntersectionFormDimensionality", "Load", "Save", "ExportPolymake",
  "ExportLatexTable", "ExportJavaView", "LabelMax", "LabelMin", "Labels",
  "Relabel", "RelabelStandard", "RelabelTransposition", "Rename",
  "SortComplex", "UnlabelFace", "AlexanderDual", "CollapseGreedy", "Cone",
  "DeletedJoin", "Difference", "HandleAddition", "Intersection",
  "IsIsomorphic", "IsSubcomplex", "Isomorphism", "IsomorphismEx", "Join",
  "Link", "Links", "Neighbors", "NeighborsEx", "Shelling", "ShellingExt",
  "Shellings", "Span", "Star", "Stars", "Suspension", "Union",
  "VertexIdentification", "Wedge", "DetermineTopologicalType", "Dim",
  "Facets", "VertexLabels", "Name", "Vertices", "IsConnected",
  "ConnectedComponents" ].

fail
gap>
```

11.4 Creating new complexes from a `SCSimplicialComplex` object

Example

```
gap> c4:=c3+c3;
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels, Vertices.

 Name="complex from diffcycles [ [ 1, 1, 6 ], [ 3, 3, 2 ] ]#+-complex from dif\
fcycles [ [ 1, 1, 6 ], [ 3, 3, 2 ] ]"
 Dim=2

/SimplicialComplex]
gap> SCRename(c4,"T^2#T^2");
true
gap> SCLink(c4,1);
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels.

 Name="lk(1) in T^2#T^2"
 Dim=1

/SimplicialComplex]
gap> SCStar(c4,[1,2]);
[SimplicialComplex
```

```
 Properties known: Dim, Facets, Name, VertexLabels.

 Name="star([ 1, 2 ]) in T^2#T^2"
 Dim=2

/SimplicialComplex]
gap> SCRename(c3,"T^2");
true
gap> SCConnectedProduct(c3,4);
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels, Vertices.

 Name="T^2#+-T^2#+-T^2#+-T^2"
 Dim=2

/SimplicialComplex]
gap> SCCartesianProduct(c2,c2);
[SimplicialComplex

 Properties known: Dim, Facets, Name, TopologicalType, VertexLabels.

 Name="S^1_3xS^1_3"
 Dim=2
 TopologicalType="S^1xS^1"

/SimplicialComplex]
gap> SCCartesianPower(c2,3);
[SimplicialComplex

 Properties known: Dim, Facets, Name, TopologicalType, VertexLabels.

 Name="(S^1_3)^3"
 Dim=3
 TopologicalType="(S^1)^3"

/SimplicialComplex]
gap>
```

11.5 Homology related calculations

SimpComp relies on the GAP package homology [DHSW04] for its homology computations but provides further (co-)homology related functions, see chapter 5.

Example

```
gap> s2s2:=SCCartesianProduct(SCBdSimplex(3),SCBdSimplex(3));
[SimplicialComplex

 Properties known: Dim, Facets, Name, TopologicalType, VertexLabels.

 Name="S^2_4xS^2_4"
 Dim=4
```

```
 TopologicalType="S^2xS^2"

/SimplicialComplex]
gap> SCHomology(s2s2);
[ [ 0, [  ] ], [ 0, [  ] ], [ 2, [  ] ], [ 0, [  ] ], [ 1, [  ] ] ]
gap> SCHomologyInternal(s2s2);
[ [ 0, [  ] ], [ 0, [  ] ], [ 2, [  ] ], [ 0, [  ] ], [ 1, [  ] ] ]
gap> SCHomologyBasis(s2s2,2);
[ [ 1, [ [ 1, 70 ], [ -1, 12 ], [ 1, 2 ], [ -1, 1 ] ] ],
  [ 1, [ [ 1, 143 ], [ -1, 51 ], [ 1, 29 ], [ -1, 25 ] ] ] ]
gap> SCHomologyBasisAsSimplices(s2s2,2);
[ [ 1,
      [ [ 1, [ 2, 3, 4 ] ], [ -1, [ 1, 3, 4 ] ], [ 1, [ 1, 2, 4 ] ], [ -1, [ 1
                    , 2, 3 ] ] ] ],
  [ 1, [ [ 1, [ 5, 9, 13 ] ], [ -1, [ 1, 9, 13 ] ], [ 1, [ 1, 5, 13 ] ],
          [ -1, [ 1, 5, 9 ] ] ] ] ]
gap> SCCohomologyBasis(s2s2,2);
[ [ 1,
      [ [ 1, 122 ], [ 1, 115 ], [ 1, 112 ], [ 1, 111 ], [ 1, 93 ], [ 1, 90 ],
          [ 1, 89 ], [ 1, 84 ], [ 1, 83 ], [ 1, 82 ], [ 1, 46 ], [ 1, 43 ],
          [ 1, 42 ], [ 1, 37 ], [ 1, 36 ], [ 1, 35 ], [ 1, 28 ], [ 1, 27 ],
          [ 1, 26 ], [ 1, 25 ] ] ],
  [ 1, [ [ 1, 213 ], [ 1, 201 ], [ 1, 192 ], [ 1, 189 ], [ 1, 159 ],
          [ 1, 150 ], [ 1, 147 ], [ 1, 131 ], [ 1, 128 ], [ 1, 125 ],
          [ 1, 67 ], [ 1, 58 ], [ 1, 55 ], [ 1, 39 ], [ 1, 36 ], [ 1, 33 ],
          [ 1, 10 ], [ 1, 7 ], [ 1, 4 ], [ 1, 1 ] ] ] ]
gap> SCCohomologyBasisAsSimplices(s2s2,2);
[ [ 1, [ [ 1, [ 4, 8, 12 ] ], [ 1, [ 3, 8, 12 ] ], [ 1, [ 3, 7, 12 ] ],
          [ 1, [ 3, 7, 11 ] ], [ 1, [ 2, 8, 12 ] ], [ 1, [ 2, 7, 12 ] ],
          [ 1, [ 2, 7, 11 ] ], [ 1, [ 2, 6, 12 ] ], [ 1, [ 2, 6, 11 ] ],
          [ 1, [ 2, 6, 10 ] ], [ 1, [ 1, 8, 12 ] ], [ 1, [ 1, 7, 12 ] ],
          [ 1, [ 1, 7, 11 ] ], [ 1, [ 1, 6, 12 ] ], [ 1, [ 1, 6, 11 ] ],
          [ 1, [ 1, 6, 10 ] ], [ 1, [ 1, 5, 12 ] ], [ 1, [ 1, 5, 11 ] ],
          [ 1, [ 1, 5, 10 ] ], [ 1, [ 1, 5, 9 ] ] ] ],
  [ 1, [ [ 1, [ 13, 14, 15 ] ], [ 1, [ 9, 14, 15 ] ], [ 1, [ 9, 10, 15 ] ],
          [ 1, [ 9, 10, 11 ] ], [ 1, [ 5, 14, 15 ] ], [ 1, [ 5, 10, 15 ] ],
          [ 1, [ 5, 10, 11 ] ], [ 1, [ 5, 6, 15 ] ], [ 1, [ 5, 6, 11 ] ],
          [ 1, [ 5, 6, 7 ] ], [ 1, [ 1, 14, 15 ] ], [ 1, [ 1, 10, 15 ] ],
          [ 1, [ 1, 10, 11 ] ], [ 1, [ 1, 6, 15 ] ], [ 1, [ 1, 6, 11 ] ],
          [ 1, [ 1, 6, 7 ] ], [ 1, [ 1, 2, 15 ] ], [ 1, [ 1, 2, 11 ] ],
          [ 1, [ 1, 2, 7 ] ], [ 1, [ 1, 2, 3 ] ] ] ] ]
gap> PrintArray(SCIntersectionForm(s2s2));
[ [  0,  1 ],
  [  1,  0 ] ]
gap> c:=s2s2+s2s2;
[SimplicialComplex

 Properties known: Dim, Facets, Name, VertexLabels, Vertices.

 Name="S^2_4xS^2_4#+-S^2_4xS^2_4"
 Dim=4

/SimplicialComplex]
```

```
gap> PrintArray(SCIntersectionForm(c));
[ [   0,  -1,   0,   0 ],
  [  -1,   0,   0,   0 ],
  [   0,   0,   0,  -1 ],
  [   0,   0,  -1,   0 ] ]
gap>
```

11.6 Bistellar flips

Example

```
gap> beta4:=SCBdCrossPolytope(4);;
gap> s3:=SCBdSimplex(4);;
gap> SCEquivalent(beta4,s3);
#I  round 0, move: [ [ 2, 6, 7 ], [ 3, 4 ] ]
[ 8, 25, 34, 17 ]
#I  round 1, move: [ [ 2, 7 ], [ 3, 4, 5 ] ]
[ 8, 24, 32, 16 ]
#I  round 2, move: [ [ 2, 5 ], [ 3, 4, 8 ] ]
[ 8, 23, 30, 15 ]
#I  round 3, move: [ [ 2 ], [ 3, 4, 6, 8 ] ]
[ 7, 19, 24, 12 ]
#I  round 4, move: [ [ 6, 8 ], [ 1, 3, 4 ] ]
[ 7, 18, 22, 11 ]
#I  round 5, move: [ [ 8 ], [ 1, 3, 4, 5 ] ]
[ 6, 14, 16, 8 ]
#I  round 6, move: [ [ 5 ], [ 1, 3, 4, 7 ] ]
[ 5, 10, 10, 5 ]
#I  SCReduceComplexEx: complexes are bistellarly equivalent.
true
gap> SCBistellarOptions.WriteLevel;
0
gap> SCBistellarOptions.WriteLevel:=1;
1
gap> SCEquivalent(beta4,s3);
#I  SCLibInit: made directory "~/PATH" for user library.
#I  SCIntFunc.SCLibInit: index not found -- trying to reconstruct it.
#I  SCLibUpdate: rebuilding index for ~/PATH.
#I  SCLibUpdate: rebuilding index done.

#I  round 0, move: [ [ 2, 4, 6 ], [ 7, 8 ] ]
[ 8, 25, 34, 17 ]
#I  round 1, move: [ [ 2, 4 ], [ 5, 7, 8 ] ]
[ 8, 24, 32, 16 ]
#I  round 2, move: [ [ 4, 5 ], [ 1, 7, 8 ] ]
[ 8, 23, 30, 15 ]
#I  round 3, move: [ [ 4 ], [ 1, 6, 7, 8 ] ]
[ 7, 19, 24, 12 ]
#I  SCLibAdd: saving complex to file "complex_ReducedComplex_7_vertices_3_2009\
-10-27_11-40-00.sc".
#I  round 4, move: [ [ 2, 6 ], [ 3, 7, 8 ] ]
[ 7, 18, 22, 11 ]
#I  round 5, move: [ [ 2 ], [ 3, 5, 7, 8 ] ]
```

```
[ 6, 14, 16, 8 ]
#I  SCLibAdd: saving complex to file "complex_ReducedComplex_6_vertices_5_2009\
-10-27_11-40-00.sc".
#I  round 6, move: [ [ 5 ], [ 1, 3, 7, 8 ] ]
[ 5, 10, 10, 5 ]
#I  SCLibAdd: saving complex to file "complex_ReducedComplex_5_vertices_6_2009\
-10-27_11-40-00.sc".
#I  SCLibAdd: saving complex to file "complex_ReducedComplex_5_vertices_7_2009\
-10-27_11-40-00.sc".
#I  SCReduceComplexEx: complexes are bistellarly equivalent.
true
gap> myLib:=SCLibInit("~/PATH"); # copy path from above
[Simplicial complex library. Properties:
CalculateIndexAttributes=true
Number of complexes in library=4
IndexAttributes=[ "Name", "Date", "Dim", "F", "G", "H", "Chi", "Homology" ]
Loaded=true
Path="/home/spreerjn/reducedComplexes/2009-10-27_11-40-00/"
]
gap> s3:=myLib.Load(3);
[SimplicialComplex

 Properties known: Chi, Date, Dim, F, Faces, Facets, G, H, Homology,
                   IsConnected, Name, VertexLabels.

 Name="ReducedComplex_5_vertices_6"
 Dim=3
 Chi=0
 F=[ 5, 10, 10, 5 ]
 G=[ 0, 0 ]
 H=[ 1, 1, 1, 1 ]
 Homology=[ [ 0, [ ] ], [ 0, [ ] ], [ 0, [ ] ], [ 1, [ ] ] ]
 IsConnected=true

/SimplicialComplex]
gap> s3:=myLib.Load(2);
[SimplicialComplex

 Properties known: Chi, Date, Dim, F, Faces, Facets, G, H, Homology,
                   IsConnected, Name, VertexLabels.

 Name="ReducedComplex_6_vertices_5"
 Dim=3
 Chi=0
 F=[ 6, 14, 16, 8 ]
 G=[ 1, 0 ]
 H=[ 2, 2, 2, 1 ]
 Homology=[ [ 0, [ ] ], [ 0, [ ] ], [ 0, [ ] ], [ 1, [ ] ] ]
 IsConnected=true

/SimplicialComplex]
gap> t2:=SCCartesianProduct(SCBdSimplex(2),SCBdSimplex(2));;
gap> t2.F;
```

```
[ 9, 27, 18 ]
gap> SCBistellarOptions.WriteLevel:=0;
0
gap> SCBistellarOptions.LogLevel:=0;
0
gap> mint2:=SCReduceComplex(t2);
[ true, [SimplicialComplex

     Properties known: Dim, Facets, Name, VertexLabels.

     Name="unnamed complex 85"
     Dim=2

    /SimplicialComplex], 32 ]
gap>
```

Chapter 12

simpcomp internals

The package simpcomp works with simplicial complexes for which a GAP object type `SCSimplicialComplex` is defined and calculates properties of these objects via so called property handlers. This chapter describes how to extend simpcomp by writing own property handlers.

By writing own property handlers, simpcomp can be extended in its functionality. If you extended simpcomp and want to share your extension with other users please send your extension to one of the authors and we will consider including it (of course with giving credit) in a future release of simpcomp.

12.1 Example of a common property handler

In this section we will have a look at the property handler `SCFVector` in oder to explain the inner workings of property handlers. This is the code of the property handler for calculating the f-vector of a complex in simpcomp:

Example

```
InstallGlobalFunction(SCFVector,
        function(complex)

        local f, faces;

        if(not SCIsSimplicialComplex(complex)) then
                SCError()("SCFVector: first argument must be of type
                SCSimplicialComplex.\n");
                return fail;
        fi;

        f:=SCPropertyByName(complex,"F");
        if f<>fail and not SCIntFunc.ForceRecalcEnabled(complex) and  IsList(f)
                and        ForAll(f,x->IsInt(x) and x>=0) then
                return f;
        else
                SCPropertyDrop(complex,"F");
                faces:=SCFaceLatticeEx(complex);
                if faces=fail then
                        return fail;
                fi;
                f:=List(faces,x->Size(x));
```

```
                SCPropertySet(complex,"F",f);
                return f;
        fi;
end);
```

When looking at the code one already see the structure that such a handler needs to have:

1. The arguments are checked for validity – in case of invalid arguments an error is signaled with `SCError()("Error message")`. `SCError` then displays the error message and optionally enters a break loop.
2. It is checked whether the property was computed already with the function `SCPropertyByName` – it takes a complex (as `SCPropertyObject`) and a property name (as `String`) as arguments and returns the property or `fail` if that property does not exist.
3. If `SCPropertyByName` did not return `fail`, `SCIntFunc.ForceRecalcEnabled(complex)` did return false and the cached property passes a validity check the cached property is returned.
4. Otherwise the property is first dropped via the function `SCPropertyDrop` – it takes a complex (as `SCPropertyObject`) and a property name (as `String`) as arguments – then recomputed and finally written back to the complex via the function `SCPropertySet` (that takes a complex, a property name and a property as arguments) and returned.

12.2 Writing a property handler

This section provides the skeleton of a property handler that can be used when writing own property handlers:

Example

```
DeclareGlobalFunction("SCMyPropertyHandler");

InstallGlobalFunction(SCMyPropertyHandler,
        function(complex[, further arguments])

        local myprop,....;

        # test arguments
        if(not SCIsSimplicialComplex(complex)) then
                SCError()("SCMyPropertyHandler: first argument must be of type
                                        SCSimplicialComplex.\n");
                return fail;
        fi;

        # check if the property that you want to compute is cached
        myprop:=SCPropertyByName(complex,"MyProperty");

        if myprop<>fail and not SCIntFunc.ForceRecalcEnabled(complex) and
                [check for validity of myprop here] then
                #return the cached property
                return myprop;
        else
                # first drop the property
```

```
                SCPropertyDrop(complex,"MyProperty");

                # now (re)compute the property
                [ do property computation here]

                # set the newly created property and return it
                SCPropertySet(complex,"MyProperty",myprop);
                return myprop;
        fi;
end);
```

References

[BD08] Bhaskar Bagchi and Basudeb Datta. On Walkup's class $\mathcal{K}(d)$ and a minimal triangulation of a 4-manifold. arXiv:0804.2153v1 [math.GT], Preprint, 8 pages, 2008. 68

[BL00] Anders Björner and Frank H. Lutz. Simplicial manifolds, bistellar flips and a 16-vertex triangulation of the Poincaré homology 3-sphere. *Experiment. Math.*, 9(2):275–289, 2000. 15

[Dat07] Basudeb Datta. Minimal triangulations of manifolds. *J. Indian Inst. Sci.*, 87(4):429–449, 2007. 12

[DHSW04] J.-G. Dumas, F. Heckenbach, B. D. Saunders, and V. Welker. Simplicial Homology, v. 1.4.2. http://www.cis.udel.edu/~dumas/Homology/, 2004. 2, 9, 56, 67, 71, 77, 80, 130

[DKT08] Mathieu Desbrun, Eva Kanso, and Yiying Tong. Discrete differential forms for computational modeling. In *Discrete differential geometry*, volume 38 of *Oberwolfach Semin.*, pages 287–324. Birkhäuser, Basel, 2008. 82, 83

[GJ00] Ewgenij Gawrilow and Michael Joswig. polymake: a framework for analyzing convex polytopes. In *Polytopes—combinatorics and computation (Oberwolfach, 1997)*, volume 29 of *DMV Sem.*, pages 43–73. Birkhäuser, Basel, 2000. 9, 21

[Grü03] Branko Grünbaum. *Convex polytopes*, volume 221 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, second edition, 2003. Prepared and with a preface by Volker Kaibel, Victor Klee and Günter M. Ziegler. 12

[Hud69] J. F. P. Hudson. *Piecewise linear topology*. University of Chicago Lecture Notes prepared with the assistance of J. L. Shaneson and J. Lees. W. A. Benjamin, Inc., New York-Amsterdam, 1969. 12

[Küh95] Wolfgang Kühnel. *Tight polyhedral submanifolds and tight triangulations*, volume 1612 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1995. 16

[Lut] Frank H. Lutz. The Manifold Page. http://www.math.tu-berlin.de/diskregeom/stellar. 2, 66

[Lut03] Frank H. Lutz. *Triangulated Manifolds with Few Vertices and Vertex-Transitive Group Actions*. PhD thesis, TU Berlin, 2003. 2, 88

[Lut05] Frank H. Lutz. Triangulated Manifolds with Few Vertices: Combinatorial Manifolds. arXiv:math/0506372v1 [math.CO], Preprint, 37 pages, 2005. 12

[McK84] Brendan McKay. The nauty page. http://cs.anu.edu.au/people/bdm/nauty/, 1984. 2, 46

[Pac87] Udo Pachner. Konstruktionsmethoden und das kombinatorische Homöomorphieproblem für Triangulierungen kompakter semilinearer Mannigfaltigkeiten. *Abh. Math. Sem. Uni. Hamburg*, 57:69–86, 1987. 61, 74, 75, 88

[RÖ7] M. Röder. GAP package polymaking. http://www.gap-system.org/Packages/polymaking.html, 2007. 114

[RS72] Colin P. Rourke and Brian J. Sanderson. *Introduction to piecewise-linear topology*. Springer-Verlag, New York, 1972. Ergebnisse der Mathematik und ihrer Grenzgebiete, Band 69. 12

[Soi06] Leonard H. Soicher. GRAPE - GRaph Algorithms using PErmutation groups. http://www.gap-system.org/Packages/grape.html, 2006. Version 4.3. 2, 9, 46

[Zie95] Günter M. Ziegler. *Lectures on polytopes*, volume 152 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1995. 12, 61, 74, 75

Index