

Handbook of Magma Functions

Part XVIII: Topology

Version 2.30-1

Sydney

November 10, 2026

Editors

John Cannon *Wieb Bosma* *Claus Fieker* *Allan Steel*

Handbook Contributors

Eran Assaf *Geoff Bailey* *Wieb Bosma* *Gavin Brown* *Nils Bruin* *John Cannon* *Jon Carlson* *Scott Contini* *Edgar Costa* *Bruce Cox* *Brendan Creutz* *Steve Donnelly* *Tim Dokchitser* *Willem de Graaf* *Andreas-Stephan Elsenhans* *Claus Fieker* *Damien Fisher* *Volker Gebhardt* *Sergei Haller* *Michael Harrison* *Florian Hess* *Derek Holt* *David Howden* *Al Kasprzyk* *Markus Kirschmer* *David Kohel* *Axel Kohnert* *Dimitri Leemans* *Paulette Lieby* *Graham Matthews* *Scott Murray* *Eamonn O'Brien* *Dan Roozemond* *Ben Smith* *Bernd Souvignier* *William Stein* *Allan Steel* *Damien Stehlé* *Nicole Sutherland* *Don Taylor* *Bill Unger* *Alexa van der Waall* *Paul van Wamelen* *Helena Verrill* *John Voight* *Mark Watkins* *Greg White*

Production Editors

Wieb Bosma *Claus Fieker* *Allan Steel* *Nicole Sutherland*

HTML Production

Claus Fieker *Allan Steel*

Copyright © 1994–2026 The University of Sydney. All rights reserved.

Contents

153 Simplicial Homology **1**
153.1 Introduction 1
153.2 Simplicial Complexes 1
153.3 Homology Computation 9
153.4 Bibliography 13

References **14**

Index **15**

Chapter 153

Simplicial Homology

153.1 Introduction

This chapter presents the category of finite simplicial complexes.

We define an abstract simplicial complex K to be a subset of the power set of some set V of vertices, with the property that if $S \in K$ and $T \subset S$ then $T \in K$.

For detailed reading on simplicial complexes and their homology, we refer to [Hat02] and [Arm83].

Simplicial complexes may be defined over any `SetEnum`, however, many of the construction methods operate over `SetEnum[RngIntElt]`. The handbook refers to such simplicial complexes as *normalized*.

A simplicial complex carries the category name `SmpCpx`. Constructors and package internal functions guarantee that the closure under subsets relation is kept intact.

153.2 Simplicial Complexes

The module supports creation of simplicial complexes from lists of faces, as well as a few preprogrammed complex types. Furthermore, several standard techniques for modifying and recombining simplicial complexes are available.

`SimplicialComplex(f): SeqEnum[SetEnum] -> SmpCpx`

Constructs an abstract simplicial complex with the faces in the list f. The argument should be a sequence of sets. There is no requirement on the type of the elements in the face sets, however several of the constructions require the sets to have integer entries. See *Normalization* for automated renumbering of the face set elements.

Example: Constructing a Simplicial Complex (ex-083856)

We give a simplicial complex by listing the facets, or also redundant faces if we want to.

```
> sc := SimplicialComplex([{1,2,3},{1,2,4},{1,3,4},{2,3,4},{1,5},{2,5}]);
```

We can view the facets (faces not included in any other faces) of the complex, or by giving the `:Maximal` output flag, we can view all faces.

```
> sc;
Simplicial complex
[
    { 2, 3, 4 },
    { 1, 2, 3 },
    { 1, 2, 4 },
    { 2, 5 },
```

(continues on next page)

(continued from previous page)

```
    { 1, 5 },
    { 1, 3, 4 }
]

> sc:Maximal;
Simplicial complex
{
    {},
    { 4 },
    { 2, 3, 4 },
    { 2, 3 },
    { 3, 4 },
    { 1, 3 },
    { 3 },
    { 1 },
    { 1, 4 },
    { 1, 2, 3 },
    { 2 },
    { 1, 2 },
    { 1, 2, 4 },
    { 2, 5 },
    { 1, 5 },
    { 5 },
    { 2, 4 },
    { 1, 3, 4 }
}
```

SimplicialComplex(G): Grph -> SmpCpx

Constructs a 1-dimensional simplicial complex isomorphic to the graph G.

FlagComplex(G): Grph -> SmpCpx

CliqueComplex(G): Grph -> SmpCpx

Constructs a simplicial complex with an n-simplex for each n-clique in the graph G.

Dimension(X): SmpCpx -> RngIntElt

Returns the dimension of the highest dimensional face of the simplicial complex X. Note that there is a difference between degrees and dimensions of faces. A face is said to have degree n if there are n elements in the face as a set, and it is said to have dimension n if there are $n+1$ elements in the face as a set.

Example: Dimension of a Simplicial Complex (ex-cb69ff)

The recently defined simplicial complex has top dimension 2.

```
> Dimension(sc);
2
```

Faces(X, d): SmpCpx, RngIntElt -> SeqEnum[SetEnum]

Returns the faces of one degree d of the simplicial complex X. Recall that the degree of a face is the number of elements in the face as a set, and one more than the dimension of the face.

The order of faces returned for each degree is also the correspondence between faces and the basis of the corresponding module in a chain complex constructed from the simplicial complex.

If the complex does not possess any faces of the requested degree, an empty sequence is returned.

Example: Faces of a Simplicial Complex (ex-36668a)

We can list faces of any degree in the defined simplicial complex.

```
> Faces(sc,2);
[
    { 2, 3 },
    { 3, 4 },
    { 1, 3 },
    { 1, 4 },
    { 1, 2 },
    { 2, 5 },
    { 1, 5 },
    { 2, 4 }
]
> Faces(sc,5);
[]
```

Facets(X): SmpCpx -> SeqEnum[SetEnum]

Returns the facets of the simplicial complex X. The facets of a simplicial complex are the faces that are not themselves subsets of another face. This is what the normal printing mode for a simplicial complex outputs.

Example: Facets of a Simplicial Complex (ex-4ea384)

We can read the facets both by fetching the sequence of facets and by printing the complex as such.

```
> Facets(sc);
[
    { 2, 3, 4 },
    { 1, 2, 3 },
    { 1, 2, 4 },
    { 2, 5 },
    { 1, 5 },
    { 1, 3, 4 }
]
> sc;
Simplicial complex
[
    { 2, 3, 4 },
    { 1, 2, 3 },
    { 1, 2, 4 },
    { 2, 5 },
    { 1, 5 },
    { 1, 3, 4 }
]
```

Normalization(X): SmpCpx -> SmpCpx

Normalization(~X): SmpCpx

Relabels the points (elements of the face sets) of the simplicial complex X using $1, 2, 3, \ldots$. This ensures that the simplicial complex is built on subsets of the integers, which is a prerequisite for several functions handling simplicial complexes. We call a simplicial complex on subsets of the integers *normalized* in this handbook.

Note the two calling signatures of this function. The first signature copies the simplicial complex and returns a new complex with the desired modification, whereas the second signature modifies the complex in place.

Shift(X, n): SmpCpx, RngIntElt -> SmpCpx

Shift(~X, n): SmpCpx, RngIntElt

Shifts all integer points in the normalized simplicial complex X by an offset given by n.

Note the two calling signatures of this function. The first signature copies the simplicial complex and returns a new complex with the desired modification, whereas the second signature modifies the complex in place.

Example: Normalizing and Shifting Labels (ex-0f49fd)

We define a simplicial complex using string labels, normalize it, and then shift the labels.

```
> cpx := Boundary(SimplicialComplex([{"a","b","c"}]));
> cpx;
Simplicial complex
[
    { c, a },
    { c, b },
    { b, a }
]

> Normalization(~cpx);
> cpx;
Simplicial complex
[
    { 1, 3 },
    { 2, 3 },
    { 1, 2 }
]

> Shift(~cpx,-2);
> cpx;
Simplicial complex
[
    { -1, 1 },
    { 0, 1 },
    { -1, 0 }
]
```

`Boundary(X): SmpCpx -> SmpCpx`

Returns the simplicial complex generated by all simplexes that lie in the boundary of the simplicial complex X. This can be used to easily acquire a simplicial complex representing the n-sphere.

Example: Boundary of a Simplicial Complex (ex-c50a0b)

We can determine the boundary of our previously defined simplicial complex, or construct a 4-sphere.

```
> Boundary(sc);
Simplicial complex
[
    { 2, 3 },
    { 3, 4 },
    { 1, 3 },
    { 1, 4 },
    { 1, 2 },
    { 5 },
    { 2, 4 }
]
> sph4 := Boundary(SimplicialComplex([{1,2,3,4,5,6}]));
> sph4;
Simplicial complex
[
```

(continues on next page)

(continued from previous page)

```
    { 1, 2, 3, 5, 6 },
    { 2, 3, 4, 5, 6 },
    { 1, 3, 4, 5, 6 },
    { 1, 2, 3, 4, 5 },
    { 1, 2, 4, 5, 6 },
    { 1, 2, 3, 4, 6 }
]
```

`S + T: SmpCpx, SmpCpx -> SmpCpx`

Returns the topological sum, or disjoint union, of two simplicial complexes. Requires both complexes to be normalized using *Normalization*.

Example: Disjoint Union of Two Circles (ex-48ad79)

We can construct a disjoint union of two circles.

```
> sph2 := Boundary(SimplicialComplex([{1,2,3}]));
> sph2 + sph2;
Simplicial complex
[
    { 4, 6 },
    { 2, 3 },
    { 4, 5 },
    { 5, 6 },
    { 1, 3 },
    { 1, 2 }
]
```

`S eq T: SmpCpx, SmpCpx -> BoolElt`

Compares two simplicial complexes. Will not try to find an isomorphism, rather does the comparison by ensuring that the points are in the same universe and then doing a set comparison on the set of faces.

Example: Equality of Simplicial Complexes (ex-4ee2fb)

Shifting a complex, for instance, will break equality, since the labels differ.

```
> sc eq Shift(sc,2);
false
> sc eq Shift(Shift(sc,2),-2);
true
```

Furthermore, isomorphic complexes with different labels will not be equal.

```
> circ1 := Boundary(SimplicialComplex([{1,2,3}]));
> circ2 := Boundary(SimplicialComplex([{"a","b","c"}]));
> circ1 eq circ2;
false
```

`Product(S, T): SmpCpx, SmpCpx -> SmpCpx`

Returns the cartesian product of two simplicial complexes. This will work on any complexes, since the new points will be pairs of points from the component complexes.

Example: Torus as a Product of Circles (ex-80e5c8)

Using the two different circles from the last example, we can now construct a torus.

```
> Product(circ1,circ2);
Simplicial complex
```

(continues on next page)

(continued from previous page)

```
[
    { <3, b>, <1, b>, <3, c> },
    { <1, c>, <3, c>, <1, a> },
    { <2, a>, <3, b>, <3, a> },
    { <3, b>, <1, b>, <1, a> },
    { <1, b>, <2, b>, <2, c> },
    { <2, a>, <3, c>, <3, a> },
    { <1, c>, <1, b>, <3, c> },
    { <3, c>, <2, b>, <2, c> },
    { <1, c>, <1, b>, <2, c> },
    { <3, c>, <3, a>, <1, a> },
    { <2, a>, <3, c>, <2, c> },
    { <3, b>, <3, a>, <1, a> },
    { <2, a>, <1, a>, <2, b> },
    { <2, a>, <1, a>, <2, c> },
    { <1, c>, <1, a>, <2, c> },
    { <3, b>, <3, c>, <2, b> },
    { <1, b>, <1, a>, <2, b> },
    { <2, a>, <3, b>, <2, b> }
]
```

If we want something less unwieldy – especially after repeated cartesian products – we can always normalize the resulting complexes.

```
> line := SimplicialComplex([{1,2}]);
> square := Product(line,line);
> cube1 := Product(square,line);
> cube2 := Product(line,square);
> cube1 eq cube2;
false
> Normalization(cube1) eq Normalization(cube2);
false
> cube1;
Simplicial complex
[
    { <<1, 2>, 1>, <<1, 2>, 2>, <<1, 1>, 1>, <<2, 2>, 2> },
    { <<2, 1>, 1>, <<2, 2>, 1>, <<1, 1>, 1>, <<2, 2>, 2> },
    { <<1, 2>, 2>, <<1, 1>, 1>, <<1, 1>, 2>, <<2, 2>, 2> },
    { <<1, 2>, 1>, <<2, 2>, 1>, <<1, 1>, 1>, <<2, 2>, 2> },
    { <<1, 1>, 1>, <<1, 1>, 2>, <<2, 1>, 2>, <<2, 2>, 2> },
    { <<2, 1>, 1>, <<1, 1>, 1>, <<2, 1>, 2>, <<2, 2>, 2> }
]
> Normalization(cube1);
Simplicial complex
[
    { 3, 4, 5, 8 },
    { 4, 5, 6, 8 },
    { 1, 3, 4, 5 },
    { 2, 4, 5, 7 },
    { 1, 2, 4, 5 },
    { 4, 5, 6, 7 }
]
```

Join(S, T): SmpCpx, SmpCpx -> SmpCpx

S * T: SmpCpx, SmpCpx -> SmpCpx

Constructs the join of two simplicial complexes. The join of two simplicial complexes is defined as the complex generated by faces on the form $\{x_1, \ldots, x_r, y_1, \ldots, y_s\}$ for $\{x_1, \ldots, x_r\} \in S$ and

$\{y_1, \ldots, y_s\} \in T$. Requires both simplicial complexes to be normalized.

Example: Join of Two Edges (ex-b9ce13)

The join of two edges is a 3-simplex.

```
> SimplicialComplex([{1,2}]) * SimplicialComplex([{1,2}]);
Simplicial complex
[
    { 1, 2, 3, 4 }
]
```

AddSimplex(X, s): SmpCpx, SetEnum -> SmpCpx

AddSimplex(~X, s): SmpCpx, SetEnum

AddSimplex(X, s): SmpCpx, SeqEnum -> SmpCpx

AddSimplex(~X, s): SmpCpx, SeqEnum

Adds either a single simplex s expressed as a set or a sequence of simplexes to a simplicial complex. The functional versions of the function call return a new complex with the added simplices included, and the procedural change the complex in place.

Prune(X, f): SmpCpx, SetEnum -> SmpCpx

Prune(~X, f): SmpCpx, SetEnum

Removes a face f and all faces containing f from a simplicial complex X. If f is not a face of X, then X is returned unchanged.

Note the two calling signatures of this function. The first signature copies the simplicial complex and returns a new complex with the desired modification, whereas the second signature modifies the complex in place.

Glue(X, e): SmpCpx, SeqEnum -> SmpCpx

Glue(~X, e): SmpCpx, SeqEnum

In the simplicial complex X, identify all points identified by pairs in the sequence e.

The sequence e should be a sequence of pairs of elements of the face sets of X. The identification will eliminate the first component of each pair, replacing it with the second component wherever it occurs in X.

Note the two calling signatures of this function. The first signature copies the simplicial complex and returns a new complex with the desired modification, whereas the second signature modifies the complex in place.

Example: Connected Sum of Two Tori (ex-ecfc35)

We can, for instance, construct the connected sum of two tori using `Product`, `Prune`, the topological sum, and `Glue`.

```
> circ := SimplicialComplex([{1,2},{2,3},{3,1}]);
> torus := Product(circ,circ);
> Normalization(~torus);
> torus;
Simplicial complex
[
    { 2, 3, 6 },
    { 1, 2, 9 },
    { 2, 6, 8 },
    { 4, 7, 8 },
    { 3, 5, 9 },
    { 4, 6, 7 },
    { 1, 8, 9 },
    { 1, 4, 8 },
```

(continues on next page)

(continued from previous page)

```
    { 6, 8, 9 },
    { 2, 7, 8 },
    { 2, 3, 9 },
    { 5, 6, 9 },
    { 1, 3, 5 },
    { 1, 4, 6 },
    { 3, 6, 7 },
    { 1, 5, 6 },
    { 1, 2, 7 },
    { 1, 3, 7 }
]

> oneholetorus := Prune(torus,{1,2,7});
> twoholetorus := Prune(Prune(torus,{1,2,7}),{6,8,9});
> threetori := oneholetorus + twoholetorus + oneholetorus;
> threetorus := Glue(threetori,[<1,10>,<2,11>,<7,16>,<15,19>,<17,20>,<18,25>]);
```

Note that we find the glue data by considering that the maximal point in the original torus had number 9, so for the second added torus, all points will be shifted by 9 and in the third all points will be shifted by 18. Thus, the excised facets $\{1,2,7\}$ and $\{6,8,9\}$ turn into $\{10,11,16\}$ and $\{15,17,18\}$, and the excised facet $\{1,2,7\}$ in the third torus turns into $\{19,20,25\}$.

`BarycentricSubdivision(X): SmpCpx -> SmpCpx`

Constructs the barycentric subdivision of the simplicial complex X. Abstractly, this is a simplicial complex whose faces are chains $X_1 \subset \cdots \subset X_n$ of faces from X. The new complex has more faces but the same homotopy type as the old complex.

`Skeleton(X, q): SmpCpx, RngIntElt -> SmpCpx`

Returns the q-skeleton of the simplicial complex X. This is the complex consisting of all faces of X of dimension at most q.

`UnderlyingGraph(X): SmpCpx -> GrphUnd, GrphVertSet, GrphEdgeSet`

Constructs a graph isomorphic, as a graph, to the 1-skeleton of the simplicial complex X.

`Cone(X): SmpCpx -> SmpCpx`

Constructs a cone over the simplicial complex X. The cone is generated by all faces of the complex, with an additional vertex included into each face. Any cone is acyclic, in other words all homology groups vanish. Requires a normalized simplicial complex.

`Suspension(X): SmpCpx -> SmpCpx`

Constructs the suspension, or double cone, over the normalized simplicial complex X. The suspension has the added property that all the homology groups occur in it, shifted up by one dimension.

Example: Cone and Suspension (ex-d8fe8c)

For computation of the relevant homology, and a demonstration of the stated facts about the cone and suspension operations, please refer to examples in the Section on homology computation.

```
> circ := Boundary(SimplicialComplex([{1,2,3}]));
> Cone(circ);
Simplicial complex
[
    { 1, 3, 4 },
    { 2, 3, 4 },
    { 1, 2, 4 }
]

> Suspension(circ);
```

(continues on next page)

(continued from previous page)

```
Simplicial complex
[
    { 1, 3, 5 },
    { 1, 3, 4 },
    { 1, 2, 5 },
    { 2, 3, 4 },
    { 1, 2, 4 },
    { 2, 3, 5 }
]
```

153.2.1 Standard Topological Objects

`Simplex(n): RngIntElt -> SmpCpx`

Returns the n-dimensional simplex as a simplicial complex.

`Sphere(n): RngIntElt -> SmpCpx`

Returns a simplicial complex triangulating the n-sphere.

`KleinBottle() -> SmpCpx`

Returns a triangulation of the Klein bottle as an abstract simplicial complex.

`Torus() -> SmpCpx`

Returns a triangulation of a torus as an abstract simplicial complex.

`Cylinder() -> SmpCpx`

Returns a triangulation of a cylinder as an abstract simplicial complex.

`MoebiusStrip() -> SmpCpx`

Returns a triangulation of the Moebius strip as an abstract simplicial complex.

`LensSpace(p): RngIntElt -> SmpCpx`

Constructs the lens space $L(p, 1)$. This has a 1-dimensional homology generator of order p.

`SimplicialProjectivePlane() -> SmpCpx`

Constructs a triangulation of the projective plane.

Examples for the usage of `LensSpace` and `SimplicialProjectivePlane` will be given in Section *Homology Computation* on homology computation.

153.3 Homology Computation

The code computes exclusively reduced homology of the given simplicial complexes. If you want the non-reduced homology, just add a single free rank to dimension 0 and let it be generated by any single point in the complex.

`Homology(X): SmpCpx -> SeqEnum, SeqEnum`

`Homology(~X): SmpCpx`

`Homology(X, A): SmpCpx, Rng -> SeqEnum, SeqEnum`

`Homology(~X, A): SmpCpx, Rng`

Calculates the reduced homology of a simplicial complex `X` with coefficients in the ring A. The procedural form of this command caches the results of the calculation in the simplicial complex object. If no ring is given, then the function defaults to integer coefficients.

Example: Homology of a Simplicial Complex `(ex-a2198f)`

The resulting modules are stored in falling dimension, always including the dimension -1 vanishing homology module at the very end.

```
> circ := Boundary(SimplicialComplex([{1,2,3}]));
> Homology(circ,Integers());
[
    Full Quotient RSpace of degree 1 over Integer Ring
    Column moduli:
    [ 0 ],

    Full Quotient RSpace of degree 0 over Integer Ring
    Column moduli:
    [ ],

    Full Quotient RSpace of degree 0 over Integer Ring
    Column moduli:
    [ ]
]
[
    Mapping from: RSpace of degree 3, dimension 1 over Integer Ring to Full
    Quotient RSpace of degree 1 over Integer Ring
    Column moduli:
    [ 0 ],
    Mapping from: RSpace of degree 3, dimension 2 over Integer Ring to Full
    Quotient RSpace of degree 0 over Integer Ring
    Column moduli:
    [ ],
    Mapping from: Full RSpace of degree 1 over Integer Ring to Full Quotient
    RSpace of degree 0 over Integer Ring
    Column moduli:
    [ ]
]
> lens3 := LensSpace(3);
> Homology(~lens3,Integers());
> Homology(lens3,Integers())[3];
Full Quotient RSpace of degree 1 over Integer Ring
Column moduli:
[ 3 ]
```

HomologyGroup(X, q): SmpCpx, RngIntElt -> ModRng

HomologyGroup(X, q, A): SmpCpx, RngIntElt, Rng -> ModRng

Calculates and returns the qth homology group of X with coefficients in A. If no ring is given, the function defaults to integer coefficients. If the homology is cached in X, the cached results are returned. This function will not compute the entire homology in order to return one homology group.

BettiNumber(X, q): SmpCpx, RngIntElt -> RngIntElt

BettiNumber(X, q, A): SmpCpx, RngIntElt, Rng -> RngIntElt

Returns the qth Betti number, computed as the free rank of the q-dimensional homology group, with coefficients in A. If no ring is given, then the function will default to integer coefficients.

Note that the Betti number computations compensate for the homology computations being reduced. Thus, `BettiNumber(X,0) eq Rank(HomologyGroup(X,0)) + 1`.

TorsionCoefficients(X, q): SmpCpx, RngIntElt -> SeqEnum[RngElt]

TorsionCoefficients(X, q, A): SmpCpx, RngIntElt, Rng -> SeqEnum[RngElt]

Returns the torsion coefficients of the qth homology group of X with coefficients in A. If no ring is given, then the function will default to integer coefficients.

EulerCharacteristic(X): SmpCpx -> RngIntElt

Computes the Euler characteristic of the complex. If homology is cached, this is used for computation, and else the characteristic is computed using the ranks of the chain groups.

`BoundaryMatrix(X, q, A): SmpCpx, RngIntElt, Rng -> Mtrx`

Returns the qth boundary matrix of the corresponding chain complex to the simplicial complex X with coefficients in A.

`ChainComplex(X, A): SmpCpx, Rng -> ModCpx`

Constructs a reduced chain complex of free A-modules corresponding to the abstract simplicial complex X.

Note that the produced complex includes one extra rank 1 module on each end, with the zero map leading to it, to simulate the maps to and from the zero module that would end a chain complex constructed from a simplicial complex in ordinary cases.

Example: Chain Complex of a Simplicial Complex (ex-dacb67)

```
> ChainComplex(SimplicialComplex([{1}]),Integers());
Chain complex with terms of degree 2 down to -1
Dimensions of terms: 1 1 1 1
> BoundaryMaps(ChainComplex(SimplicialComplex([{1}]),Integers()));
[*
    [0],

    [-1],

    [0]
*]
> ChainComplex(SimplicialComplex([{1,2,3}]),GF(3));
Chain complex with terms of degree 4 down to -1
Dimensions of terms: 1 1 3 3 1 1
> BoundaryMaps(ChainComplex(SimplicialComplex([{1,2,3}]),GF(3)));
[*
    [0],

    [1 2 2],

    [1 0 2]
    [0 1 2]
    [1 2 0],

    [2]
    [2]
    [2],

    [0]
*]
```

`HomologyGenerators(X): SmpCpx`

`HomologyGenerators(X, A): SmpCpx, Rng`

`HomologyGenerators(H, M, X): SeqEnum, SeqEnum, SmpCpx`

Prints generators of the homology groups of the simplicial complex X with coefficients in A together with their order, in order of dimension. The latter calling form expects H, M to be the result from `H,M := ChainComplex(A,smp);`. This function will recalculate homology each time unless the homology is already cached in the simplicial complex using `Homology(A, smp)`.

If no ring is given, the function defaults to integer coefficients.

Example: Homology Generators (ex-1e8d54)

This function gives a condensed form of the actual bases of the homology groups, as well as mappings back to an actual chain representative for each homology class.

```
> HomologyGenerators(threetorus,Integers());

*** dimension 2 ***
inf: { 21, 23, 27 } - { 4, 6, 16 } - { 20, 24, 26 } + { 20, 25, 26 } -
  { 14, 19, 25 } + { 12, 14, 25 } + { 3, 6, 11 } - { 10, 14, 19 } +
  { 19, 20, 27 } - { 3, 9, 11 } - { 10, 13, 20 } + { 10, 13, 19 } -
  { 10, 12, 14 } - { 12, 16, 19 } - { 11, 19, 20 } - { 13, 16, 20 } +
  { 21, 24, 25 } + { 24, 26, 27 } - { 20, 21, 24 } - { 19, 22, 26 } -
  { 3, 10, 16 } - { 4, 6, 10 } - { 22, 25, 26 } + { 11, 16, 20 } +
  { 4, 8, 16 } - { 5, 6, 9 } + { 4, 8, 10 } - { 11, 12, 19 } - { 3, 5, 9 } +
  { 11, 12, 25 } + { 8, 9, 10 } + { 20, 21, 27 } + { 3, 6, 16 } +
  { 10, 11, 25 } + { 19, 22, 24 } - { 9, 10, 11 } + { 3, 5, 10 } +
  { 10, 20, 25 } - { 8, 11, 16 } - { 19, 26, 27 } - { 19, 23, 24 } +
  { 10, 12, 16 } - { 19, 21, 23 } - { 6, 8, 9 } + { 13, 16, 19 } +
  { 19, 21, 25 } + { 23, 24, 27 } - { 22, 24, 25 } + { 5, 6, 10 } +
  { 6, 8, 11 }


*** dimension 1 ***
inf: -1*{ 10, 13 } + { 3, 5 } - { 8, 9 } - { 8, 16 } - { 19, 22 } +
  { 20, 26 } + { 9, 11 } - { 3, 16 } + { 5, 10 } + { 22, 26 } + { 11, 20 } +
  2*{ 10, 20 } + { 13, 19 } - 2*{ 10, 11 }

inf: { 3, 9 } - { 3, 5 } - { 9, 11 } - { 5, 10 } + { 10, 11 }

inf: -1*{ 26, 27 } - { 11, 19 } + 2*{ 19, 22 } - { 20, 26 } -
  2*{ 22,  26 } + { 11, 20 } + { 19, 21 } - { 21, 27 }

inf: { 10, 13 } + { 11, 19 } - { 11, 20 } - { 10, 20 } - { 13, 19 }

inf: -1*{ 11, 19 } + { 19, 22 } - { 20, 26 } - { 22, 26 } + { 11, 20 }

inf: { 11, 20 } + { 10, 20 } - { 10, 11 }
```

The six found generators are the generators of each of the contained torus homology groups.

Notice that each generator is printed out with a prefix. This gives the order of the generator - so that for instance torsion elements of homology may be recognized. Thus, we see with the projective plane:

```
> HomologyGenerators(SimplicialProjectivePlane(),Integers());

*** dimension 1 ***
  2: { 3, 6 } + { 2, 3 } - { 2, 6 }
```

We can further take this opportunity to verify the claims about `Cone` and `Suspension` with regard to the homology.

```
> HomologyGenerators(Cone(SimplicialProjectivePlane()),Integers());
Complex is acyclic.

> HomologyGenerators(Suspension(SimplicialProjectivePlane()),Integers());

*** dimension 2 ***
  2: { 1, 5, 7 } - { 1, 4, 7 } + { 1, 2, 5 } + { 1, 5, 8 } - { 2, 3, 8 } -
```

(continues on next page)

(continued from previous page)

```
{ 3, 5, 7 } - { 1, 2, 8 } + { 5, 6, 8 } + { 1, 4, 5 } - { 4, 6, 8 } +
{ 3, 4, 7 } + { 2, 3, 5 } - { 4, 5, 6 } - { 3, 4, 8 }
```

153.4 Bibliography

- [Arm83] Mark Anthony Armstrong. *Basic topology.* Undergraduate Texts in Mathematics. Springer-Verlag, New York, 1983. ISBN 0-387-90839-0. Corrected reprint of the 1979 original.
- [Hat02] Allen Hatcher. *Algebraic topology.* Cambridge University Press, Cambridge, 2002. ISBN 0-521-79160-X; 0-521-79540-0.

References

[Arm83] Mark Anthony Armstrong. *Basic topology*. Undergraduate Texts in Mathematics. Springer-Verlag, New York, 1983. ISBN 0-387-90839-0. Corrected reprint of the 1979 original.

[Hat02] Allen Hatcher. *Algebraic topology*. Cambridge University Press, Cambridge, 2002. ISBN 0-521-79160-X; 0-521-79540-0.

Index

A

AddSimplex(~X, s) (*procedure*), 7
AddSimplex(X, s) (*function*), 7

B

BarycentricSubdivision(X) (*function*), 8
BettiNumber(X, q) (*function*), 10
BettiNumber(X, q, A) (*function*), 10
Boundary of a Simplicial Complex (*example*), 4
Boundary(X) (*function*), 4
BoundaryMatrix(X, q, A) (*function*), 11

C

Chain Complex of a Simplicial Complex (*example*), 11
ChainComplex(X, A) (*function*), 11
CliqueComplex(G) (*function*), 2
Cone and Suspension (*example*), 8
Cone(X) (*function*), 8
Connected Sum of Two Tori (*example*), 7
Constructing a Simplicial Complex (*example*), 1
Cylinder() (*function*), 9

D

Dimension of a Simplicial Complex (*example*), 2
Dimension(X) (*function*), 2
Disjoint Union of Two Circles (*example*), 5

E

Equality of Simplicial Complexes (*example*), 5
EulerCharacteristic(X) (*function*), 10

F

Faces of a Simplicial Complex (*example*), 2
Faces(X, d) (*function*), 2
Facets of a Simplicial Complex (*example*), 3
Facets(X) (*function*), 3
FlagComplex(G) (*function*), 2

G

Glue(~X, e) (*procedure*), 7
Glue(X, e) (*function*), 7

H

Homology Generators (*example*), 11
Homology of a Simplicial Complex (*example*), 9
Homology(~X) (*procedure*), 9
Homology(~X, A) (*procedure*), 9
Homology(X) (*function*), 9
Homology(X, A) (*function*), 9
HomologyGenerators(H, M, X) (*procedure*), 11
HomologyGenerators(X) (*procedure*), 11
HomologyGenerators(X, A) (*procedure*), 11
HomologyGroup(X, q) (*function*), 10
HomologyGroup(X, q, A) (*function*), 10

J

Join of Two Edges (*example*), 7
Join(S, T) (*function*), 6

K

KleinBottle() (*function*), 9

L

LensSpace(p) (*function*), 9

M

MoebiusStrip() (*function*), 9

N

Normalization(~X) (*procedure*), 3
Normalization(X) (*function*), 3
Normalizing and Shifting Labels (*example*), 4

P

Product(S, T) (*function*), 5
Prune(~X, f) (*procedure*), 7
Prune(X, f) (*function*), 7

S

S * T (*operation*), 6
S + T (*operation*), 5
S eq T (*operation*), 5
Shift(~X, n) (*procedure*), 3
Shift(X, n) (*function*), 3
Simplex(n) (*function*), 9
SimplicialComplex(f) (*function*), 1
SimplicialComplex(G) (*function*), 2
SimplicialProjectivePlane() (*function*), 9
Skeleton(X, q) (*function*), 8
Sphere(n) (*function*), 9
Suspension(X) (*function*), 8

T

`TorsionCoefficients(X, q)` (*function*), 10
`TorsionCoefficients(X, q, A)` (*function*), 10
`Torus as a Product of Circles` (*example*), 5
`Torus()` (*function*), 9

U

`UnderlyingGraph(X)` (*function*), 8