# Contents

# Chapter 1

# Resolutions of the ground ring

`ResolutionAbelianGroup(L,n)` `ResolutionAbelianGroup(G,n)` Inputs a list $L := [m_1, m_2, ..., m_d]$ of nonnegativ
`ResolutionAlmostCrystalGroup(G,n)` Inputs a positive integer $n$ and an almost crystallographic pcp group $G$. It ret
`ResolutionAlmostCrystalQuotient(G,n,c)` `ResolutionAlmostCrystalQuotient(G,n,c,false)` An almost
`ResolutionArtinGroup(D,n)` Inputs a Coxeter diagram $D$ and an integer $n > 1$. It returns $n$ terms of a free $ZG$-resolu
`ResolutionAsphericalPresentation(F,R,n)` Inputs a free group $F$, a set $R$ of words in $F$ which constitute an asph
`ResolutionBieberbachGroup( G )`
`ResolutionBieberbachGroup( G, v )` Inputs a Bieberbach group $G$ (represented using AffineCrystGroupOnRight
`ResolutionDirectProduct(R,S)` `ResolutionDirectProduct(R,S,"internal")` Inputs a $ZG$-resolution $R$ and
`ResolutionExtension(g,R,S)` `ResolutionExtension(g,R, S,"TestFiniteness")` `ResolutionExtensio`
`ResolutionFiniteDirectProduct(R,S)` `ResolutionFiniteDirectProduct(R,S, "internal")` Inputs a $ZG$
`ResolutionFiniteExtension(gensE,gensG,R,n)` `ResolutionFiniteExtension(gensE,gensG,R,n,true)`
`ResolutionFiniteGroup(gens,n)` `ResolutionFiniteGroup(gens,n,true)` `ResolutionFiniteGroup(gens`
`ResolutionFiniteSubgroup(R,K)` `ResolutionFiniteSubgroup(R,gensG,gensK)` Inputs a $ZG$-resolution for a
`ResolutionGraphOfGroups(D,n)` `ResolutionGraphOfGroups(D,n,L)` Inputs a graph of groups $D$ and a positi
`ResolutionNilpotentGroup(G,n)` `ResolutionNilpotentGroup(G,n,"TestFiniteness")` Inputs a nilpotent g
`ResolutionNormalSeries(L,n)` `ResolutionNormalSeries(L,n,true)` `ResolutionNormalSeries(L,n,fa`
`ResolutionPrimePowerGroup(P,n)` `ResolutionPrimePowerGroup(G,n,p)` Inputs a $p$-group $P$ and integer $n>$(
`ResolutionSmallFpGroup(G,n)` `ResolutionSmallFpGroup(G,n,p)` Inputs a small finitely presented group $G$
`ResolutionSubgroup(R,K)` Inputs a $ZG$-resolution for an (infinite) group $G$ and a subgroup $K$ of finite index $|G:K|$.
`ResolutionSubnormalSeries(L,n)` Inputs a positive integer n and a list $L = [L_1, \ldots, L_k]$ of subgroups $L_i$ of a finite g
`TwistedTensorProduct(R,S,EhomG,GmapE,NhomE,NEhomN,EltsE,Mult,InvE)` Inputs a $ZG$-resolution $R$, a $ZN$-res

# Chapter 2

# Resolutions of modules

`ResolutionFpGModule(M,n)` Inputs an $FpG$-module $M$ and a positive integer $n$. It returns $n$ terms of a minimal free $F$

# Chapter 3

# Induced equivariant chain maps

`EquivariantChainMap(R,S,f)` Inputs a $ZG$-resolution $R$, a $ZG'$-resolution $S$, and a group homomorphism $f : G \longrightarrow$

# Chapter 4

# Functors

•

`HomToIntegers(X)` Inputs either a $ZG$-resolution $X = R$, or an equivariant chain map $X = (F : R \longrightarrow S)$. It returns th
`HomToIntegersModP(R)` Inputs a $ZG$-resolution $R$ and returns the cochain complex obtained by applying $HomZG(,Z$
`HomToIntegralModule(R,f)` Inputs a $ZG$-resolution $R$ and a group homomorphism $f : G \longrightarrow GL_n(Z)$ to the group o
`LowerCentralSeriesLieAlgebra(G)` `LowerCentralSeriesLieAlgebra(f)` Inputs a pcp group $G$. If each quo
`TensorWithIntegers(X)` Inputs either a $ZG$-resolution $X = R$, or an equivariant chain map $X = (F : R \longrightarrow S)$. It retu
`TensorWithIntegersModP(X,p)` Inputs either a $ZG$-resolution $X = R$, or an equivariant chain map $X = (F : R \longrightarrow S$
`TensorWithRationals(R)` Inputs a $ZG$-resolution $R$ and returns the chain complex obtained by tensoring with the tri

# Chapter 5

# Chain complexes

`ChevalleyEilenbergComplex(X,n)` Inputs either a Lie algebra $X = A$ (over the ring of integers $Z$ or over a field $K$)
`LeibnizComplex(X,n)` Inputs either a Lie or Leibniz algebra $X = A$ (over the ring of integers $Z$ or over a field $K$) or

# Chapter 6

# Homology and cohomology groups

`Cohomology(X)` Inputs either a cochain complex $X = C$ or a cochain map $X = (C \longrightarrow D)$ over the integers $Z$.If $X = C$

`CohomologyPrimePart(C,n,p)` Inputs a cochain complex $C$ in characteristic 0, a positive integer $n$, and a prime $p$. It re

`GroupCohomology(X,n)` `GroupCohomology(X,n,p)` Inputs a positive integer $n$ and either a finite group $X = G$ or a

`GroupHomology(X,n)`

`GroupHomology(X,n,p)` Inputs a positive integer $n$ and either a finite group $X = G$ or a Coxeter diagram $X = D$ represe

`Homology(X,n)` Inputs either a chain complex $X = C$ or a chain map $X = (C \longrightarrow D)$.If $X = C$ then the torsion coefficie

`HomologyPb(C,n)` This is a back-up function which might work in some instances where $Homology(C,n)$ fails. It is mo

`HomologyPrimePart(C,n,p)` Inputs a chain complex $C$ in characteristic 0, a positive integer $n$, and a prime $p$. It returns

`LeibnizAlgebraHomology(A,n)` Inputs a Lie or Leibniz algebra $X = A$ (over the ring of integers $Z$ or over a field $K$), t

`LieAlgebraHomology(A,n)` Inputs a Lie algebra $A$ (over the integers or a field) and a positive integer $n$. It returns the h

`PrimePartDerivedFunctor(G,R,F,n)` Inputs a finite group $G$, a positive integer $n$, at least $n+1$ terms of a $ZP$-resolut

`RankHomologyPGroup(G,n)` `RankHomologyPGroup(R,n)` `RankHomologyPGroup(G,n,"empirical")` Inputs a (sma

`RankPrimeHomology(G,n)` Inputs a (smallish) $p$-group $G$ together with a positive integer $n$. It returns a function $dim(k)$

# Chapter 7

# Poincare series

•

`EfficientNormalSubgroups(G)`
`EfficientNormalSubgroups(G,k)` Inputs a prime-power group $G$ and, optionally, a positive integer $k$. The default is $k$
`ExpansionOfRationalFunction(f,n)` Inputs a positive integer $n$ and a rational function $f(x) = p(x)/q(x)$ where the d
`PoincareSeries(G,n)` `PoincareSeries(R,n)`
`PoincareSeries(L,n)`
`PoincareSeries(G)` Inputs a finite $p$-group $G$ and a positive integer $n$. It returns a quotient of polynomials $f(x) = P($
`PoincareSeriesPrimePart(G,p,n)` Inputs a finite group $G$, a prime $p$, and a positive integer $n$. It returns a quotient o
`Prank(G)` Inputs a $p$-group $G$ and returns the rank of the largest elementary abelian subgroup.

# Chapter 8

# Cohomology ring structure

`IntegralCupProduct(R,u,v,p,q)`
`IntegralCupProduct(R,u,v,p,q,P,Q,N)` (Various functions used to construct the cup product are also `CR`$_{f}$*unctio*
`IntegralRingGenerators(R,n)` Inputs at least $n+1$ terms of a $ZG$-resolution and integer $n>0$. It returns a minima
`ModPCohomologyGenerators(G,n)`
`ModPCohomologyGenerators(R)` Inputs either a $p$-group $G$ and positive integer $n$, or else $n$ terms of a minimal $Z_pG$-
`ModPCohomologyRing(G,n)`
`ModPCohomologyRing(G,n,level)`
`ModPCohomologyRing(R)`
`ModPCohomologyRing(R,level)` Inputs either a $p$-group $G$ and positive integer $n$, or else $n$ terms of a minimal $Z_pG$-
`ModPRingGenerators(A)` Inputs a mod $p$ cohomology ring $A$ (created using the preceeding function). It returns a mi

# Chapter 9

# Commutator and nonabelian tensor computations

- 

`BaerInvariant(G,c)` Inputs a nilpotent group $G$ and integer $c{>}0$. It returns the Baer invariant $M^(c)(G)$ defined as fo
`Coclass(G)` Inputs a group $G$ of prime-power order $p^n$ and nilpotency class $c$ say. It returns the integer $r = n - c$ .
`EpiCentre(G,N)`
`EpiCentre(G)` Inputs a finite group $G$ and normal subgroup $N$ and returns a subgroup $Z^*(G,N)$ of the centre of $N$. The
`NonabelianExteriorProduct(G,N)` Inputs a finite group $G$ and normal subgroup $N$. It returns a record $E$ with the fo
`NonabelianTensorProduct(G,N)` Inputs a finite group $G$ and normal subgroup $N$. It returns a record $E$ with the foll
`NonabelianTensorSquare(G)`
`NonabelianTensorSquare(G,m)` Inputs a finite or nilpotent infinite group $G$ and returns a record $T$ with the followin
`RelativeSchurMultiplier(G,N)` Inputs a finite group $G$ and normal subgroup $N$. It returns the homology group $H_2$
`TensorCentre(G)` Inputs a group $G$ and returns the largest central subgroup $N$ such that the induced homomorphism
`ThirdHomotopyGroupOfSuspensionB(G)`
`ThirdHomotopyGroupOfSuspensionB(G,m)` Inputs a finite or nilpotent infinite group $G$ and returns the abelian invar
`UpperEpicentralSeries(G,c)` Inputs a nilpotent group $G$ and an integer $c$. It returns the $c$-th term of the upper epic

Chapter 10

# Generators and relators of groups

•

CayleyGraphDisplay(G,X)
CayleyGraphDisplay(G,X,"mozilla") Inputs a finite group $G$ together with a subset $X$ of $G$. It displays the corresp
IsAspherical(F,R) Inputs a free group $F$ and a set $R$ of words in $F$. It performs a test on the 2-dimensional CW-spa
PresentationOfResolution(R) Inputs at least two terms of a reduced $ZG$-resolution $R$ and returns a record $P$ with
TorsionGeneratorsAbelianGroup(G) Inputs an abelian group $G$ and returns a generating set $[x_1,\ldots,x_n]$ where no p

# Chapter 11

# Orbit polytopes and fundamental domains

•

`FundamentalDomainAffineCrystGroupOnRight(v,G)` Inputs a crystallographic group G (represented using AffineC

`OrbitPolytope(G,v,L)` Inputs a permutation group or matrix group $G$ of degree $n$ and a rational vector $v$ of length $n$

The function uses Polymake software.

`PolytopalComplex(G,v)`

`PolytopalComplex(G,v,n)`

Inputs a permutation group or matrix group $G$ of degree $n$ and a rational vector $v$ of length $n$. In both cases there is a natural action of $G$ on $v$. Let $P(G,v)$ be the convex polytope arising as the convex hull of the Euclidean points in the orbit of $v$ under the action of $G$. The cellular chain complex $C_* = C_*(P(G,v))$ is an exact sequence of (not necessarily free) $ZG$-modules. The function returns a component object $R$ with components:

- $R!.dimension(k)$ is a function which returns the number of $G$-orbits of the $k$-dimensional faces in $P(G,v)$. If each $k$-face has trivial stabilizer subgroup in $G$ then $C_k$ is a free $ZG$-module of rank $R.dimension(k)$.
- $R!.stabilizer(k,n)$ is a function which returns the stabilizer subgroup for a face in the $n$-th orbit of $k$-faces.
- If all faces of dimension $<k+1$ have trivial stabilizer group then the first $k$ terms of $C_*$ constitute part of a free $ZG$-resolution. The boundary map is described by the function $boundary(k,n)$ . (If some faces have non-trivial stabilizer group then $C_*$ is not free and no attempt is made to determine signs for the boundary map.)
- $R!.elements$, $R!.group$, $R!.properties$ are as in a $ZG$-resolution.

If an optional third input variable $n$ is used, then only the first $n$ terms of the resolution $C_*$ will be computed.

The function uses Polymake software.

`PolytopalGenerators(G,v)`

Inputs a permutation group or matrix group $G$ of degree $n$ and a rational vector $v$ of length $n$. In both cases there is a natural action of $G$ on $v$, and the vector $v$ must be chosen so that it has trivial stabilizer subgroup in $G$. Let $P(G,v)$ be the convex polytope arising as the convex hull of the Euclidean points in the orbit of $v$ under the action of $G$. The function returns a record $P$ with components:

- *P.generators* is a list of all those elements $g$ in $G$ such that $g \cdot v$ has an edge in common with $v$. The list is a generating set for $G$.
- *P.vector* is the vector $v$.
- *P.hasseDiagram* is the Hasse diagram of the cone at $v$.

The function uses Polymake software. The function is joint work with Seamus Kelly.

`VectorStabilizer(G,v)`

Inputs a permutation group or matrix group $G$ of degree $n$ and a rational vector of degree $n$. In both cases there is a natural action of $G$ on $v$ and the function returns the group of elements in $G$ that fix $v$.

# Chapter 12

# Cocycles

`CocycleCondition(R,n)` Inputs a resolution $R$ and an integer $n$>0. It returns an integer matrix $M$ with the following

`StandardCocycle(R,f,n)`

`StandardCocycle(R,f,n,q)` Inputs a $ZG$-resolution $R$ (with contracting homotopy), a positive integer $n$ and an integ

`Syzygy(R,g)` Inputs a $ZG$-resolution $R$ (with contracting homotopy) and a list $g = [g[1], \ldots, g[n]]$ of elements in $G$. It r

# Chapter 13

# Words in free $ZG$-modules

`AddFreeWords(v,w)` Inputs two words $v, w$ in a free $ZG$-module and returns their sum $v + w$. If the characteristic of $Z$
`AddFreeWordsModP(v,w,p)` Inputs two words $v, w$ in a free $ZG$-module and the characteristic $p$ of $Z$. It returns the su
`AlgebraicReduction(w)`
`AlgebraicReduction(w,p)` Inputs a word $w$ in a free $ZG$-module and returns a reduced version of the word in which
`Multiply Word(n,w)` Inputs a word $w$ and integer $n$. It returns the scalar multiple $n \cdot w$.
`Negate([i,j])` Inputs a pair $[i, j]$ of integers and returns $[-i, j]$.
`NegateWord(w)` Inputs a word $w$ in a free $ZG$-module and returns the negated word $-w$.
`PrintZGword(w,elts)` Inputs a word $w$ in a free $ZG$-module and a (possibly partial but sufficient) listing elts of the e
`TietzeReduction(S,w)` Inputs a set $S$ of words in a free $ZG$-module, and a word $w$ in the module. The function retu

# Chapter 14

# $FpG$-modules

`DirectSumOfFpGModules(M,N)`
`DirectSumOfFpGModules([ M[1], M[2], ..., M[k] ]))` Inputs two $FpG$-modules $M$ and $N$ with common grou
`FpGModule(A,P)`
`FpGModule(A,G,p)` Inputs a $p$-group $P$ and a matrix $A$ whose rows have length a multiple of the order of $G$. It returns
`FpGModuleDualBasis(M)` Inputs an $FpG$-module $M$. It returns a record $R$ with two components:$R.freeModule$ is the
`FpGModuleHomomorphism(M,N,A)`
`FpGModuleHomomorphismNC(M,N,A)` Inputs $FpG$-modules $M$ and $N$ over a common $p$-group $G$. Also inputs a list $A$
`DesuspensionFpGModule(M,n)`
`DesuspensionFpGModule(R,n)` Inputs a positive integer $n$ and and FpG-module $M$. It returns an FpG-module $D^nM$
`RadicalOfFpGModule(M)` Inputs an $FpG$-module $M$ with $G$ a $p$-group, and returns the Radical of $M$ as an $FpG$-mod
`GeneratorsOfFpGModule(M)` Inputs an $FpG$-module $M$ and returns a matrix whose rows correspond to a minimal ge
`ImageOfFpGModuleHomomorphism(f)` Inputs an $FpG$-module homomorphism $f : M \longrightarrow N$ and returns its image $f($
`IntersectionOfFpGModules(M,N)` Inputs two $FpG$-modules $M,N$ arising as submodules in a common free module
`IsFpGModuleHomomorphismData(M,N,A)` Inputs $FpG$-modules $M$ and $N$ over a common $p$-group $G$. Also inputs a l
`MultipleOfFpGModule(w,M)` Inputs an $FpG$-module $M$ and a list $w := [g_1, ..., g_t]$ of elements in the group $G = M!.g$
`ProjectedFpGModule(M,k)` Inputs an $FpG$-module $M$ of ambient dimension $n|G|$, and an integer $k$ between 1 and $n$.
`RandomHomomorphismOfFpGModules(M,N)` Inputs two $FpG$-modules $M$ and $N$ over a common group $G$. It returns a
`Rank(f)` Inputs an $FpG$-module homomorphism $f : M \longrightarrow N$ and returns the dimension of the image of $f$ as a vector
`SumOfFpGModules(M,N)` Inputs two $FpG$-modules $M,N$ arising as submodules in a common free module $(FG)^n$ whe
`SumOp(f,g)` Inputs two $FpG$-module homomorphisms $f,g : M \longrightarrow N$ with common sorce and common target. It retu
`VectorsToFpGModuleWords(M,L)` Inputs an $FpG$-module $M$ and a list $L = [v_1, \ldots, v_k]$ of vectors in $M$. It returns a li

# Chapter 15

# Meataxe modules

•

`DesuspensionMtxModule(M)` Inputs a meataxe module $M$ over the field of $p$ elements and returns an FpG-module DM.
`FpG_to_MtxModule(M)` Inputs an FpG-module $M$ and returns an isomorphic meataxe module.
`GeneratorsOfMtxModule(M)` Inputs a meataxe module $M$ acting on, say, the vector space $V$. The function returns a m

# Chapter 16

# Coxeter diagrams and graphs of groups

`CoxeterDiagramComponents(D)` Inputs a Coxeter diagram $D$ and returns a list $[D_1, ..., D_d]$ of the maximal connected

`CoxeterDiagramDegree(D,v)` Inputs a Coxeter diagram $D$ and vertex $v$. It returns the degree of $v$ (i.e. the number of

`CoxeterDiagramDisplay(D)`

`CoxeterDiagramDisplay(D,"web browser")` Inputs a Coxeter diagram $D$ and displays it as a .gif file. It uses the M

`CoxeterDiagramFpArtinGroup(D)` Inputs a Coxeter diagram $D$ and returns the corresponding finitely presented Art

`CoxeterDiagramFpCoxeterGroup(D)` Inputs a Coxeter diagram $D$ and returns the corresponding finitely presented C

`CoxeterDiagramIsSpherical(D)` Inputs a Coxeter diagram $D$ and returns "true" if the associated Coxeter groups is

`CoxeterDiagramMatrix(D)` Inputs a Coxeter diagram $D$ and returns a matrix representation of it. The matrix is give

`CoxeterSubDiagram(D,V)` Inputs a Coxeter diagram $D$ and a subset $V$ of its vertices. It returns the full sub-diagram

`CoxeterDiagramVertices(D)` Inputs a Coxeter diagram $D$ and returns its set of vertices.

`EvenSubgroup(G)` Inputs a group $G$ and returns a subgroup $G^+$. The subgroup is that generated by all products $xy$ wh

`GraphOfGroupsDisplay(D)`

`GraphOfGroupsDisplay(D,"web browser")` Inputs a graph of groups $D$ and displays it as a .gif file. It uses the Mo

`GraphOfGroupsTest(D)` Inputs an object $D$ and itries to test whether it is a Graph of Groups. However, it DOES NO

# Chapter 17

# Some functions for accessing basic data

`BoundaryMap(C)` Inputs a resolution, chain complex or cochain complex $C$ and returns the function $C!.boundary$.
`BoundaryMatrix(C,n)` Inputs a chain or cochain complex $C$ and integer $n>0$. It returns the $n$-th boundary map of $C$
`Dimension(C)`
`Dimension(M)` Inputs a resolution, chain complex or cochain complex $C$ and returns the function $C!.dimension$ . Alte
`EvaluateProperty(X,"name")` Inputs a component object $X$ (such as a $ZG$-resolution or chain map) and a string "n
`GroupOfResolution(R)` Inputs a $ZG$-resolution $R$ and returns the group $G$.
`Length(R)` Inputs a resolution $R$ and returns its length (i.e. the number of terms of $R$ that HAP has computed).
`Map(f)` Inputs a chain map, or cochain map or equivariant chain map $f$ and returns the mapping function (as opposed
`Source(f)` Inputs a chain map, or cochain map, or equivariant chain map, or $FpG$-module homomorphism $f$ and retu
`Target(f)` Inputs a chain map, or cochain map, or equivariant chain map, or $FpG$-module homomorphism $f$ and retu

# Chapter 18

# Parallel Computation - Core Functions

`ChildProcess()`
`ChildProcess("computer.ac.wales")` This starts a GAP session as a child process and returns a stream to the child p

- open a shell on thishost
- cd .ssh
- ls
-¿ if id$_d sa, id_r saetcexists, skipthenexttwosteps$!
- ssh-keygen -t rsa
- ssh-keygen -t dsa
- scp *.pub user@remotehost: /
- ssh remotehost -l user
- cat id$_r sa.pub >> .ssh/authorized_k eys$
- cat id$_d sa.pub >> .ssh/authorized_k eys$
- rm id$_r sa.pubid_d sa.pub$
- exit

You should now be able to connect from ”thishost” to ”remotehost” without a password prompt.)
`ChildClose(s)` This closes the stream s to a child GAP process.
`ChildCommand("cmd;",s)` This runs a GAP command ”cmd;” on the child process accessed by the stream s. Here ”cm
`NextAvailableChild(L)` Inputs a list *L* of child processes and returns a child in *L* which is ready for computation (as
`IsAvailableChild(s)` Inputs a child process *s* and returns true if s is currently available for computations, and false o
`ChildPut(A,"B",s)` This copies a GAP object A on the parent process to an object B on the child process s. (The cop
`ChildGet("A",s)` This functions copies a GAP object A on the child process s and returns it on the parent process. (T

# Chapter 19

# Parallel Computation - Extra Functions

`ChildFunction("function(arg);",s)` This runs the GAP function "function(arg);" on a child process accessed by th
`ChildRead(s)` This returns, as a string, the output of the last application of $ChildFunction("function(arg);",s)$.
`ChildReadEval(s)` This returns, as an evaluated string, the output of the last application of $ChildFunction("function(a$
`ParallelList(I,fn,L)` Inputs a list $I$, a function $fn$ such that $fn(x)$ is defined for all $x$ in $I$, and a list of children $L$. It

## Chapter 20

# Topological Data Analysis

`MatrixToTopologicalSpace(A,n)` Inputs an integer matrix $A$ and an integer $n$. It returns a 2-dimensional topological
`ReadImageAsTopologicalSpace("file.png",n)` `ReadImageAsTopologicalSpace("file.png",[m,n])` Reads
`ReadImageAsMatrix("file.png")` Reads an image file ("file.png", "file.eps", "file.bmp" etc) and returns an integer m
`WriteTopologicalSpaceAsImage(T,"filename","ext")` Inputs a 2-dimensional topological space T, and a filenam
`ViewTopologicalSpace(T)` `ViewTopologicalSpace(T,"mozilla")` Inputs a topological space T, and optionally a
`BettiNumbers(T,n)` `BettiNumbers(T)` Inputs a topological space $T$ and a non-negative integer $n$. It returns the n-th
`PathComponent(T,n)` Inputs a topological space $T$ and an integer $n$ in the rane 0, ..., BettiNumbers(T,0) . It returns the
`SingularChainComplex(A)` Inputs a topological space $T$ and returns a (usually very large) integral chain complex tha
`ContractTopologicalSpace(T)` Inputs a topological space $T$ of dimension $d$ and removes $d$-dimensional cells from T
`BoundaryTopologicalSpace(T)` Inputs a topological space $T$ and returns its boundars as a topological space.
`BoundarySingularities(T)` Inputs a topological space $T$ and returns the subspace of points in the boundary where th
`ThickenedTopologicalSpace(T)` `ThickenedTopologicalSpace(T,n)` Inputs a topological space $T$ and returns a
`ComplementTopologicalSpace(T)` Inputs a topological space $T$ and returns a topological space $S$. A euclidean point
`ConcatenatedTopologicalSpace(L)` Inputs a list $L$ of topological spaces whose underlying arrays of numbers all hav

# Chapter 21

# Pseudo lists

`Add(L,x)` Let $L$ be a pseudo list of length $n$, and $x$ an object compatible with the entries in $L$. If $x$ is not in $L$ then this o
`Append(L,K)` Let $L$ be a pseudo list and $K$ a list whose objects are compatible with those in $L$. This operation applies A
`ListToPseudoList(L)` Inputs a list $L$ and returns the pseudo list representation of $L$.

# Chapter 22

# Miscellaneous

`BigStepLCS(G,n)` Inputs a group $G$ and a positive integer $n$. It returns a subseries $G = L_1 > L_2 > \ldots L_k = 1$ of the lowe

`Compose(f,g)` Inputs two $FpG$-module homomorphisms $f : M \longrightarrow N$ and $g : L \longrightarrow M$ with $Source(f) = Target(g)$ .

`HAPcopyright()` This function provides details of HAP'S GNU public copyright licence.

`IsLieAlgebraHomomorphism(f)` Inputs an object $f$ and returns true if $f$ is a homomorphism $f : A \longrightarrow B$ of Lie alge

`IsSuperperfect(G)` Inputs a group $G$ and returns "true" if both the first and second integral homology of $G$ is trivial

`MakeHAPManual()` This function creates the manual for HAP from an XML file.

`PermToMatrixGroup(G,n)` Inputs a permutation group $G$ and its degree $n$. Returns a bijective homomorphism $f : G$

`SolutionsMatDestructive(M,B)` Inputs an $m \times n$ matrix $M$ and a $k \times n$ matrix $B$ over a field. It returns a k$\times$*mmatri*

`TestHap()` This runs a representative sample of HAP functions and checks to see that they produce the correct output

# Index