

Handbook of Magma Functions

Part VIII: Modules

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

59 Introduction to Modules **1**
- 59.1 Overview . . . 1
- 59.2 General Modules . . . 1
- 59.3 The Presentation of Submodules . . . 2

60 Free Modules **3**
- 60.1 Introduction . . . 3
- 60.2 Definition of a Module . . . 4
- 60.3 Accessing Module Information . . . 5
- 60.4 Standard Constructions . . . 5
- 60.5 Construction of Elements . . . 6
- 60.6 Bases . . . 9
- 60.7 Submodules . . . 9
- 60.8 Quotient Modules . . . 11
- 60.9 Homomorphisms . . . 11

61 Modules over Dedekind Domains **19**
- 61.1 Introduction . . . 19
- 61.2 Creation of Modules . . . 19
- 61.3 Elementary Functions . . . 23
- 61.4 Predicates on Modules . . . 25
- 61.5 Arithmetic with Modules . . . 25
- 61.6 Basis of a Module . . . 27
- 61.7 Other Functions on Modules . . . 27
- 61.8 Homomorphisms between Modules . . . 29
- 61.9 Elements of Modules . . . 31
- 61.10 Pseudo Matrices . . . 32

62 Chain Complexes **35**
- 62.1 Complexes of Modules . . . 35
- 62.2 Chain Maps . . . 39

63 Multilinear Algebra **46**
- 63.1 Introduction . . . 46
- 63.2 Tensors . . . 48
- 63.3 Tensor Spaces . . . 97
- 63.4 Tensor Categories . . . 109
- 63.5 Linear Invariants of Tensors . . . 126
- 63.6 Some Extended Examples . . . 131
- 63.7 Bibliography . . . 135

References **137**

Index **138**

Chapter 59

Introduction to Modules

59.1 Overview

This section of the Handbook describes the Magma facilities for linear algebra and module theory. Since this topic is absolutely fundamental for much of algebra, it is important that the reader understand how linear algebra is presented in Magma. The structures covered under this heading include:

(i) Vector spaces;

(ii) Inner product spaces;

(iii) Modules defined over any ring or algebra

(iv) $R[G]$-modules, where R is a ring and G is a group;

(v) Linear transformations and R-module homomorphisms.

Although vector spaces are, of course, subsumed under general modules, we present a separate treatment of them, firstly because of their importance and secondly because their theory is somewhat cleaner than that of a general module. Magma users who are unfamiliar with the language of module theory will find a self-contained treatment of the vector space machinery in Chapter Vector Spaces. In the Magma universe, rectangular matrices are regarded as forming a module (actually a bimodule). We shall regard a rectangular matrix as the concrete realization of a linear transformation or R-module homomorphism. Thus, an $m \times n$ matrix over a ring R is considered to be an element of the module $\mathrm{Hom}_R(M, N)$. Reflecting the dual nature of matrices, the $\mathrm{Hom}_R(M, N)$ operations include the standard module-theoretic operations as well as operations that interpret an element of $\mathrm{Hom}_R(M, N)$ as a homomorphism.

59.2 General Modules

A module M is always regarded as a submodule or quotient module of the free module $S^{(n)}$, for some ring or algebra S. The types of module that are definable in the system fall into three classes:

(a) Abstract Modules: Given a ring R, a set M, and a mapping $\phi : R \times M \to M$, the pair (M, ϕ) will be referred to as an *abstract* R-module. Because of the very general nature of this construction, only the basic arithmetic operations may be applied to modules of this type.

(b) Modules with Scalar Action: Given a general ring R, an R-module with *scalar action* is a submodule or quotient module of the free R-module $R^{(n)}$, where the action is that of ring multiplication in R.

(c) Modules with Matrix Action: Let R be a PIR and suppose S is a R-algebra. Thus there exists a ring homomorphism $\phi : R \to S$, and so S is a (left) R-module with the R-action defined by $r * s = \phi(r) * s$. Indeed, any S-module M is a (left) S-module with action defined by $r * m = \phi(r) * m$. Furthermore, if $\phi(R)$ lies in the centre of S, then S acts on M as a ring of R-module endomorphisms. Consequently, M is an S-module. We take M to be the free R-module $R^{(n)}$, and so the action of S on M is given by the action of a subring of $M_n(R)$ on M. Thus, given an

R-algebra S, an S-module of the form $M = R^{(n)}$ may be specified by giving M together with a homomorphism of S into $M_n(R)$.

59.3 The Presentation of Submodules

Let N be a submodule of the module $M = R^{(m)}$. For simplicity, assume that N is free, and has dimension n where $n < m$. There are two ways in which N may be viewed:

(a) As a submodule embedded in M. Thus, though N is a module of dimension n, its elements are regarded as elements of M.

(b) As the module $R^{(n)}$ represented on a reduced basis, together with a morphism ϕ defining the inclusion of N into M.

The presentation (a) is the usual way submodules are regarded in elementary linear algebra. However, this presentation is inconvenient for more advanced applications. For example, many of the major functions available for studying an $R[G]$-module N expect that N is given relative to a reduced basis. We shall refer to a submodule presentation (a) as the *embedded presentation*, and (b) as the *reduced presentation*. To provide the user with the maximum flexibility, Magma supports both forms of submodule presentation for the important classes of modules. Usually, a module calculation commences with the definition of one or two modules from which further modules are created by the operations of forming submodules, quotient modules, and extensions. Let us call these latter modules *descendants* of the original module. Magma provides parallel creation functions which allow the user to choose the form of submodule presentation. Once that choice has been made, all descendants of the initial module(s) will follow the same presentation convention. The module creation functions that select the embedded form are usually of the form *Qualifier*`Space`, while those that adopt the standard form are usually of the form *Qualifier*`Module`. Thus in the case of vector spaces the function `KSpace(K, n)` constructs the n-dimensional vector space over the field K, where submodules are to be presented in embedded form. On the other hand, `RModule(K, n)` constructs the n-dimensional vector space over the field K, where submodules are to be presented in reduced form.

Chapter 60

Free Modules

60.1 Introduction

60.1.1 Free Modules

This chapter describes the facilities provided for modules in the following representations:

- *Tuple Modules:* Modules whose elements are n-tuples over a fixed ring R, i.e., modules $R^{(n)}$;
- *Matrix Modules:* Modules whose elements are homomorphisms of modules, i.e., $Hom_R(M, N)$. The elements of these modules are $m \times n$ matrices over the ring R;

The ring R acts on the right of the module element by scalar multiplication. If R is not an Euclidean Domain then, currently, only arithmetic with vectors is supported. In particular, the ability to work with submodules and quotient modules is restricted to situations where R is either a field or Euclidean Domain.

In the first part of the chapter we describe the operations that apply to modules generally, while in the second half we describe the creation of modules $Hom_R(M, N)$ together with the operations that are specific to them. Insofar as elementary module-theoretic operations are concerned, there is no real difference between tuple modules and matrix modules except for the input and display of elements. Many special operations provided for matrices are described in the chapter on matrices.

The reader is referred to the chapter on vector spaces for descriptions of the extensive functionality provided for modules over fields.

60.1.2 Module Categories

The family of all finitely generated modules over a given ring R forms a category, while the set of all finitely generated modules forms a family of categories indexed by the ring R. In this family of categories, objects are modules and the morphisms are module homomorphisms. The category name for modules is `ModRng`. We distinguish the following subcategories of `ModRng`:

- `ModTupFld` - the category of modules of n-tuples over a field;
- `ModMatFld` - the category of modules of $m \times n$ matrices over a field;
- `ModTupEd` - the category of modules of n-tuples over an euclidean domain.
- `ModTupRng` - the category of modules of n-tuples over a ring;
- `ModMatRng` - the category of modules of $m \times n$ matrices over a ring;

60.1.3 Presentation of Submodules

Let N be a free submodule of the R-module M. We have two alternative ways of presenting N. Firstly, we can present it on a set of generators that are elements of M; we call such a presentation an *embedded presentation*. Alternatively, given that N has rank r, we can present it as the module $S^{(r)}$, with appropriate action induced from the action of R on M. We call this presentation of N a *reduced presentation*. The user can control the method of submodule presentation at the time of creation of an initial module through selection of the appropriate creation function. Thus, the function `RModule` will create a module with the convention that it and all its submodules and quotient modules will have their submodules presented in reduced form. The use of `RSpace`, on the other hand, signifies that submodules are to be presented in embedded form.

60.1.4 Notation

Throughout this chapter, R will denote a ring (possibly a field) while K will denote a field. The letters M and N will denote modules, while U and V will denote vector spaces.

60.2 Definition of a Module

60.2.1 Construction of Modules of n-tuples

`RSpace(R, n): Rng, RngIntElt -> ModTupRng`

`RModule(R, n): Rng, RngIntElt -> ModRng`

Given a ring R and a non-negative integer n, create the free right R-module $R^{(n)}$, consisting of all n-tuples over R. The module is created with the standard basis, $e_1, \ldots, e_n$, where e_i $(i = 1, \ldots, n)$ is the vector containing a 1 in the i-th position and zeros elsewhere. The function `RModule` creates a module in reduced mode while `RSpace` creates a module in embedded mode.

`RSpace(R, n, F): Rng, RngIntElt, Mtrx -> ModTupRng`

Given a ring R, a non-negative integer n, and a square $n \times n$ symmetric matrix F, create the free right R-module $R^{(n)}$ (in embedded form), with inner product matrix F. This is the same as `RSpace(R, n)`, except that the functions `Norm` and `InnerProduct` (see below) will be with respect to the inner product matrix F.

Example: Module of 6-Tuples over the Integers `(ex-d42492)`

We construct the module consisting of 6-tuples over the integers.

```
> Z := IntegerRing();
> M := RModule(Z, 6);
> M;
RModule M of dimension 6 with base ring Integer Ring
```

60.2.2 Construction of Modules of $m \times n$ Matrices

`RMatrixSpace(R, m, n): Rng, RngIntElt, RngIntElt -> ModMatRng`

The module comprising all $m \times n$ matrices over the ring R.

60.2.3 Construction of a Module with Specified Basis

`RModuleWithBasis(Q): [ModFldElt] -> ModFld`

`RSpaceWithBasis(Q): [ModTupRngElt] -> ModTupRng`

`RSpaceWithBasis(a): AlgMatElt -> ModTupRng`

`RSpaceWithBasis(a): ModMatRngElt -> ModTupRng`

Given a sequence Q (or matrix a) of k independent vectors each lying in a module M, construct the submodule of M of dimension k whose basis is Q (or the rows of a). The basis is echelonized internally but all functions which depend on the basis of the space (e.g. `Coordinates`) will use the given basis.

`RMatrixSpaceWithBasis(Q): [ModTupRngElt] -> ModMatRng`

The module of $m \times n$ matrices whose basis is given by the linearly independent matrices of the sequence Q.

60.3 Accessing Module Information

`M . i: ModTupRng, RngIntElt -> ModElt`

Given an R-module M and a positive integer i, return the i-th generator of M. The integer i must lie in the range $[1, r]$, where r is the number of generators for M.

`CoefficientRing(M): ModTupRng -> Rng`

`BaseRing(M): ModTupRng -> Rng`

`CoefficientRing(M): ModRng -> Rng`

`BaseRing(M): ModRng -> Rng`

`CoefficientField(M): ModFld -> Fld`

`BaseField(M): ModFld -> Fld`

Given an R-module M which is defined as a submodule of $S^{(n)}$, return the ring S.

`Generators(M): ModTupRng -> { ModTupRngElt }`

The generators for the R-module M, returned as a set.

`OverDimension(M): ModTupRng -> RngIntElt`

Given an R-module M which is an embedded submodule of the module $S^{(n)}$, return n.

`OverDimension(u): ModTupRngElt -> RngIntElt`

Given an element u of an embedded submodule of the module $S^{(n)}$, return n.

`Moduli(M): ModTupRng -> [ RngElt ]`

The column moduli of the module M over a euclidean domain.

`Parent(u): ModTupRngElt -> ModRng`

Given an element u belonging to the R-module M, return M.

`Generic(M): ModRng -> ModRng`

Given an R-module M which is a submodule of the module $R^{(n)}$, return the module $R^{(n)}$ as an R-module.

60.4 Standard Constructions

Given one or more existing modules, various standard constructions are available to construct new modules.

60.4.1 Changing the Coefficient Ring

`ChangeRing(M, S): ModRng, Rng -> ModRng, Map`

Given a module M with base ring R, together with a ring S, construct the module N with base ring S obtained by coercing the components of elements of M into N, together with the homomorphism from M to N.

`ChangeRing(M, S, f): ModRng, Rng, Map -> ModRng, Map`

Given a module M with base ring R, together with a ring S, and a homomorphism $f : R \to S$, construct the module N with base ring S obtained by mapping the components of elements of M into N by f, together with the homomorphism from M to N.

`ChangeUniverse(~x, R): ModTupRngElt, Rng -> ModRng, Map`

Change the coefficient ring of x to be R.

60.4.2 Direct Sums

`DirectSum(M, N): ModRng, ModRng -> ModRng, Map, Map, Map, Map`

Given R-modules M and N, construct the direct sum D of M and N as an R-module. The embedding maps from M into D and from N into D respectively, and the projection maps from D onto M and from D onto N respectively are also returned.

`DirectSum(Q): [ ModRng ] -> [ ModRng ], [ Map ], [ Map ]`

Given a sequence Q of R-modules, construct the direct sum D of these modules. The embedding maps from each of the elements of Q into D and the projection maps from D onto each of the elements of Q are also returned.

60.5 Construction of Elements

`elt< M | a1, ..., an >: ModTupRng, List -> ModTupRngElt`

Given a module M with base module $S^{(n)}$, and elements $a_1, \ldots, a_n$ belonging to S, construct the element $m = (a_1, \ldots, a_n)$ of M. Note that if m is not an element of M, an error will result.

`M ! Q: ModTupRng, [RngElt] -> ModTupRngElt`

Given the module M with base module $S^{(n)}$, and elements $a_1, \ldots, a_n$ belonging to S, construct the element $m = (a_1, \ldots, a_n)$ of M. Note that if m is not an element of M, an error will result.

`CharacteristicVector(M, S): ModRng, { RngIntElt } -> ModRngElt`

Given a submodule M of the module $R^{(n)}$ together with a set S of integers lying in the interval $[1, n]$, return the characteristic number of S as a vector of R.

`Zero(M): ModRng -> ModRngElt`

`M ! 0: ModRng, RngIntElt -> ModRngElt`

The zero element for the R-module M.

`Random(M): ModRng -> ModRngElt`

Given a module M defined over a finite ring or field, return a random vector.

Example: Elements of a Free Module over Z[x] (ex-cd5034)

We create the module of 4-tuples over the polynomial ring $\mathbb{Z}[x]$ and define various elements.

```
> P<x> := PolynomialRing(IntegerRing());
> M := RModule(P, 4);
> a := elt< M | 1+x, -x, 2+x, 0 >;
> a;
(x + 1    -x x + 2     0)
> b := M ! [ 1+x+x^2, 0, 1-x^7, 2*x ];
> b;
(x^2 + x + 1           0    -x^7 + 1         2*x)
> zero := M ! 0;
> zero;
(0 0 0 0)
```

60.5.1 Deconstruction of Elements

`ElementToSequence(u): ModTupRngElt -> [RngElt]`

`Eltseq(u): ModTupRngElt -> [RngElt]`

Given an element u belonging to the R-module M, return u in the form of a sequence Q of elements of R. Thus, if u is an element of $R^{(n)}$, then $Q[i] = u[i], 1 \leq i \leq n$, while if u is an element of $R^{(m \times n)}$, then $Q[(i-1)n + j] = u[i, j], 1 \leq i \leq m, 1 \leq j \leq n$.

60.5.2 Operations on Module Elements

Arithmetic

`u + v: ModTupRngElt, ModTupRngElt -> ModTupRngElt`

Sum of the elements u and v, where u and v lie in the same R-module M.

`- u: ModTupRngElt -> ModTupRngElt`

Additive inverse of the element u.

`u - v: ModTupRngElt, ModTupRngElt -> ModTupRngElt`

Difference of the elements u and v, where u and v lie in the same R-module M.

`x * u: RngElt, ModTupRngElt -> ModTupRngElt`

Given an element x belonging to a ring R, and an element u belonging to the left R-module M, return the (left) scalar product $x * u$ as an element of M.

`u * x: ModTupRngElt, RngElt -> ModTupRngElt`

Given an element x belonging to a ring R, and an element u belonging to the right R-module M, return the (right) scalar product $u * x$ as an element of M.

`u / x: ModTupRngElt, RngElt -> ModTupRngElt`

Given a non-zero element x belonging to a field K, and an element u belonging to the right K-module M, return the scalar product $u * (1/x)$ as an element of M.

Indexing

`u[i]: ModTupRngElt, RngIntElt -> RngElt`

Given an element u belonging to a submodule M of the R-module $R^{(n)}$ and a positive integer i, $1 \leq i \leq n$, return the i-th component of u (as an element of the ring R).

`u[i] := x: ModTupRngElt, RngIntElt, RngElt -> ModTupRngElt`

Given an element u belonging to a submodule M of the R-module $T = R^{(n)}$, a positive integer i, $1 \leq i \leq n$, and an element x of the ring R, redefine the i-th component of u to be x. The parent of u is changed to T (since the modified element u need not lie in M).

Normalization

`Normalize(u): ModTupRngElt -> ModTupRngElt`

`Normalise(u): ModTupRngElt -> ModTupRngElt`

The element u must belong to an R-module, where R is either a field, the ring of integers, or a univariate polynomial ring over a field. Assume that the vector u is non-zero. If R is a field then `Normalize` returns $\frac{1}{a} * u$, where a is the first non-zero component of u. If R is the ring of integers, `Normalize` returns $\epsilon * u$, where ϵ is $+1$ if the first non-zero component of u is positive, and -1 otherwise. If R is the polynomial ring $K[x]$, K a field, then `Normalize` returns $\frac{1}{a} * u$, where a is the leading coefficient of the first non-zero (polynomial) component of u. If u is the zero vector, it is returned as the value of this function.

`Rotate(u, k): ModTupRngElt, RngIntElt -> ModTupRngElt`

Given a vector u, return the vector obtained from u by rotating by k coordinate positions.

`Rotate(~u, k): ModTupRngElt, RngIntElt`

Given a vector u, destructively rotate u by k coordinate positions.

Example: Arithmetic on Module Elements `(ex-8a5f2c)`

We illustrate the use of the arithmetic operators for module elements by applying them to elements of the module of 4-tuples over the polynomial ring $\mathbb{Z}[x]$.

```
> P<x> := PolynomialRing(IntegerRing());
> M := RModule(P, 4);
> a :=  M ! [ 1+x, -x, 2+x, 0 ];
> b := M ! [ 1+x+x^2, 0, 1-x^7, 2*x ];
> a + b;
(x^2 + 2*x + 2    -x    -x^7 + x + 3    2*x)
> -a;
(-x - 1      x -x - 2      0)
> a - b;
(       -x^2          -x x^7 + x + 1        -2*x)
> (1-x + x^2)*a;
(x^3 + 1    -x^3 + x^2 - x    x^3 + x^2 - x + 2    0)
> a*(1-x);
(    -x^2 + 1      x^2 - x -x^2 - x + 2            0)
> a[3];
x + 2
> a[3] := x - 2;
> a;
(x + 1    -x x - 2     0)
> ElementToSequence(a - b);
[
    -x^2,
    -x,
    x^7 + x - 3,
    -2*x
]
> Support(a);
{ 1, 2, 3 }
```

60.5.3 Properties of Vectors

IsZero(u): ModTupRngElt -> BoolElt

Returns **true** if the element u of the R-module M is the zero element.

Depth(v): ModTupRngElt -> RngIntElt

The index of the first non-zero entry of the vector v (0 if none such).

Support(u): ModTupRngElt -> { RngElt }

A set of integers giving the positions of the non-zero components of the vector u.

Weight(u): ModTupRngElt -> RngIntElt

The number of non-zero components of the vector u.

60.5.4 Inner Products

(u, v): ModTupRngElt, ModTupRngElt -> RngElt

InnerProduct(u, v): ModTupRngElt, ModTupRngElt -> RngElt

Return the inner product of the vectors u and v with respect to the inner product defined on the space. If an inner product matrix F is given when the space is created, then this is defined to be $u \cdot F \cdot v^{tr}$. Otherwise, this is simply $u \cdot v^{tr}$.

Norm(u): ModTupRngElt -> RngElt

Return the norm product of the vector u with respect to the inner product defined on the space. If an inner product matrix F is given when the space is created, then this is defined to be $u \cdot F \cdot u^{tr}$. Otherwise, this is simply $u \cdot u^{tr}$.

60.6 Bases

The application of the functions in this section is restricted either to vector spaces or to torsion-free modules over a Euclidean Domain. For a full description of the basis functions for a module defined over a field, the reader is referred to the chapter on vector spaces.

`Basis(M): ModTupRng -> [ModTupRngElt]`

The current basis for the free R-module M, R an ED, returned as a sequence of module elements.

`Rank(M): ModTupRng -> RngIntElt`

The rank of the free R-module M.

`Coordinates(M, u): ModTupRng, ModTupRngElt -> [RngElt]`

Given a vector u belonging to the rank r free R-module M, R an Euclidean Domain, with basis $u_1, \ldots, u_r$, return a sequence $[a_1, \ldots, a_r]$ giving the coordinates of u relative to the M-basis: $u = a_1 * u_1 + \cdots + a_r * u_r$.

60.7 Submodules

60.7.1 Construction of Submodules

Submodules may be defined for any type of module. However, functions that depend upon membership testing are only implemented for modules over Euclidean Domains (EDs). The conventions defining the presentations of submodules are as follows:

- If M has been created using the function `RSpace`, then every submodule of M is given in terms of a generating set consisting of elements of M, i.e. by means of an embedded generating set.
- If M has been created using the function `RModule`, then every submodule of M is given in terms of a reduced basis.

`sub<M | L>: ModTupRng, List -> ModTupRng`

`sub<M | L>: ModMatRng, List -> ModMatRng`

Given an R-module M, construct the submodule N generated by the elements of M specified by the list L. Each term L_i of the list L must be an expression defining an object of one of the following types:

(a) A sequence of n elements of R defining an element of M;

(b) A set or sequence whose terms are elements of M;

(c) A submodule of M;

(d) A set or sequence whose terms are submodules of M.

The generators stored for N consist of the elements specified by terms L_i together with the stored generators for submodules specified by terms of L_i. Repetitions of an element and occurrences of the zero element are removed (unless N is trivial).

The constructor returns the submodule N and the inclusion homomorphism $f : N \to M$.

Example: Submodule over a Rational Function Field `(ex-e4a021)`

We construct a submodule of the 4-dimensional vector space over the field of rational function $F[x]$, where F is $\mathbf{F}_5$.

```
> P := PolynomialRing(GF(5));
> R<x> := FieldOfFractions(P);
> M := RSpace(R, 4);
> N := sub< M | [1, x, 1-x, 0], [1+2*x-x^2, 2*x, 0, 1-x^4 ] >;
> N;
Vector space of degree 4, dimension 2 over Field of Fractions in x over
```

(continues on next page)

(continued from previous page)

```
Univariate Polynomial Algebra over GF(5)
Generators:
(1    x    4*x + 1    0)
(4*x^2 + 2*x + 1    2*x    0    4*x^4 + 1)
Echelonized basis:
(1  0  3/(x + 4)  (x^3 + x^2 + x + 1) / (x + 4))
(0  1  (4*x^2 + 2*x + 1) / (x^2 + 4*x)  (4*x^3 + 4*x^2 + 4*x + 4) / (x^2 + 4*x))
```

60.7.2 Operations on Submodules

60.7.3 Membership and Equality

The following operations are only available for submodules of $R^{(n)}$, $\mathrm{Hom}_R(M, N)$, and $R[G]$, where R is a Euclidean Domain. If the modules involved are $R[G]$-modules, the operators refer to the underlying R-module.

`u in M: ModTupRngElt, ModTupRng -> BoolElt`

`u in M: ModMatRngElt, ModMatRng -> BoolElt`

Returns `true` if the element u lies in the R-module M, where u and M belong to the same R-module.

`u notin M: ModTupRngElt, ModTupRng -> BoolElt`

`u notin M: ModMatRngElt, ModMatRng -> BoolElt`

Returns `true` if the element u does not lie in the R-module M, where u and M belong to the same R-module.

`N subset M: ModTupRng, ModTupRng -> BoolElt`

`N subset M: ModMatRng, ModMatRng -> BoolElt`

Returns `true` if the R-module N is contained in the R-module M, where M and N belong to a common R-module.

`N notsubset M: ModTupRng, ModTupRng -> BoolElt`

`N notsubset M: ModMatRng, ModMatRng -> BoolElt`

Returns `true` if the R-module N is not contained in the R-module M, where M and N belong to a common R-module.

`N eq M: ModTupRng, ModTupRng -> BoolElt`

`M eq N: ModMatRng, ModMatRng -> BoolElt`

Returns `true` if the R-modules N and M are equal, where N and M belong to a common R-module.

`N ne M: ModTupRng, ModTupRng -> BoolElt`

`M ne N: ModMatRng, ModMatRng -> BoolElt`

Returns `true` if the R-modules N and M are not equal, where N and M belong to a common R-module.

60.7.4 Operations on Submodules

The following operations are only available for submodules of $R^{(n)}$, $\mathrm{Hom}_R(M, N)$, and $R[G]$, where R is a Euclidean Domain. If the modules involved are $R[G]$-modules, the operators refer to the underlying R-module.

`M + N: ModTupRng, ModTupRng -> ModTupRng`

`M + N: ModMatRng, ModMatRng -> ModMatRng`

Sum of the submodules M and N, where M and N belong to a common R-module.

`M meet N: ModTupRng, ModTupRng -> ModTupRng`

`M meet N: ModMatRng, ModMatRng -> ModMatRng`

Intersection of the submodules M and N, where M and N belong to a common R-module.

60.8 Quotient Modules

60.8.1 Construction of Quotient Modules

`quo<M | L>: ModTupRng, List -> ModTupRng`

`quo<M | L>: ModMatRng, List -> ModMatRng`

Given an R-module M, construct the quotient module $P = M/N$, where N is the submodule generated by the elements of M specified by the list L. Each term L_i of the list L must be an expression defining an object of one of the following types:

(a) A sequence of n elements of R defining an element of M;

(b) A set or sequence whose terms are elements of M;

(c) A submodule of M;

(d) A set or sequence whose terms are submodules of M.

The generators constructed for N consist of the elements specified by terms L_i together with the stored generators for submodules specified by terms of L_i.

The constructor returns the quotient module P and the natural homomorphism $f : M \to P$.

60.9 Homomorphisms

Throughout this part of the chapter, when discussing the set of all R-homomorphisms from the R-module M into the R-module N, it will be assumed that R is a **commutative** ring. We further assume that M and N are free R-modules and that bases for these modules are present. The module $\mathrm{Hom}_R(M, N)$ will be identified with the module of $m \times n$ matrices over R. Thus, an element of $\mathrm{Hom}_R(M, N)$ is represented as a matrix relative to the bases of the generic modules corresponding to M and N. For this reason, we will refer to these modules as *matrix modules*. We remind the reader that submodules of $\mathrm{Hom}_R(M, N)$ are always presented in embedded form. If the user wishes to have submodules presented in reduced form then he/she should use the natural isomorphism between $R^{(m\times n)}$ and $R^{(mn)}$.

It should be noted that essentially operation defined for tuple modules, their elements, and submodules applies to matrix modules. Thus, all of the operations discussed earlier in this chapter apply to matrix modules. The modules M and N may themselves be matrix modules. In this case, the resulting matrix module has either a right or left action and an element belonging to it transforms a (homomorphism) element of M into a (homomorphism) element of N. The function `Reduce` may be used to construct for a matrix module H the matrix module H' equivalent to H whose elements are with respect to the actual bases of the domain and codomain of elements of H (not the generic bases of the domain and codomain).

60.9.1 $\mathrm{Hom}_R(M, N)$ for R-modules

`Hom(M, N): ModTupRng, ModTupRng -> ModMatRng`

If M is the tuple module $R^{(m)}$ and N is the tuple module $R^{(n)}$, create the module $\mathrm{Hom}_R(M, N)$ as the (R, R)-bimodule $R^{(m\times n)}$, represented as the set of all $m \times n$ matrices over R. The module is created with the standard basis, $\{E_{ij} \mid i = 1 \ldots, m, j = 1 \ldots, n\}$, where E_{ij} is the matrix having a 1 in the (i, j)-th position and zeros elsewhere.

`RMatrixSpace(R, m, n): Rng, RngIntElt, RngIntElt -> ModMatRng`

Given a ring R and positive integers m and n, construct $H = \mathrm{Hom}(M, N)$, where $M = R^{(m)}$ and $N = R^{(n)}$, as the free (R, R)-bimodule $R^{(m\times n)}$, consisting of all $m \times n$ matrices over R. The module is created with the standard basis, $\{E_{ij} \mid i = 1 \ldots, m, j = 1 \ldots, m\}$. Note that the modules M and N are created by this function and may be accessed as `Domain(H)` and `Codomain(H)`, respectively.

Example: Module of Homomorphisms of Vector Spaces `(ex-bf6e98)`

We construct the vector spaces V and W of dimensions 3 and 4, respectively, over the field of two elements and then define M to be the module of homomorphisms from V into W.

```
> F2 := GaloisField(2);
> V  := VectorSpace(F2, 3);
> W  := VectorSpace(F2, 4);
> M  := Hom(V, W);
> M;
Full KMatrixSpace of 3 by 4 matrices over GF(2)
```

60.9.2 $\mathrm{Hom}_R(M, N)$ for Matrix Modules

Hom(M, N, "right"): ModMatRng, ModMatRng, MonStgElt -> ModMatRng

Suppose M is a matrix module over the coefficient ring R whose elements are a by b matrices and have domain D and codomain C. Suppose also that N is a matrix module over the coefficient ring R whose elements are a by c matrices and have domain D and codomain C'. Then the homomorphism module $H = \mathrm{Hom}(M, N)$ with right multiplication action exists and consists of all b by c matrices over R which multiply an element of M on the right to yield an element of N. This function constructs H explicitly. The domain of elements of H is then M and the codomain of elements of H is N and the elements are b by c matrices over R which multiply an element of M on the right to yield an element of N. Note that if M and N are proper submodules of their respective generic modules, then H may be a proper submodule of its generic module, and the correct basis of H will be explicitly constructed.

Hom(M, N, "left"): ModMatRng, ModMatRng, MonStgElt -> ModMatRng

Suppose M is a matrix module over the coefficient ring R whose elements are a by c matrices and have domain D and codomain C. Suppose also that N is a matrix module over the coefficient ring R whose elements are b by c and have domain D' and codomain C. Then the homomorphism module $H = \mathrm{Hom}(M, N)$ with left multiplication action exists and consists of all b by a matrices over R which multiply an element of M on the left to yield an element of N. This function constructs H explicitly. The domain of elements of H is then M and the codomain of elements of H is N and the elements are b by a matrices over R which multiply an element of M on the right to yield an element of N. Note that if M and N are proper submodules of their respective generic modules, then H may be a proper submodule of its generic module, and the correct basis of H will be explicitly constructed.

Example: Homomorphisms Between Homomorphism Modules (ex-e3a29e)

We construct two homomorphism modules H_1 and H_2 over $\mathbb{Q}$ and then the homomorphism module $H = \mathrm{Hom}(H_1, H_2)$ with right matrix action.

```
> Q := RationalField();
> H1 := sub<RMatrixSpace(Q, 2, 3) | [1,2,3, 4,5,6], [0,0,1, 1,3,3]>;
> H2 := sub<RMatrixSpace(Q, 2, 4) | [6,5,7,1, 15,14,16,4], [0,0,0,0, 1,2,3,4]>;
> H := Hom(H1, H2, "right");
> H: Maximal;
KMatrixSpace of 3 by 4 matrices and dimension 1 over Rational Field
Echelonized basis:

[   1    2    3    4]
[-1/2   -1 -3/2   -2]
[   0    0    0    0]

> H1.1 * H.1;
[  0   0   0   0]
[3/2   3 9/2   6]
> H1.1 * H.1 in H2;
true
> Image(H.1): Maximal;
KMatrixSpace of 2 by 4 matrices and dimension 1 over Rational Field
```

(continues on next page)

(continued from previous page)

```
Echelonized basis:

[0 0 0 0]
[1 2 3 4]

> Kernel(H.1): Maximal;
KMatrixSpace of 2 by 3 matrices and dimension 1 over Rational Field
Echelonized basis:

[ 1  2  6]
[ 7 14 15]

> H1 := sub<RMatrixSpace(Q,2,3) | [1,2,3, 4,5,6]>;
> H2 := sub<RMatrixSpace(Q,3,3) | [1,2,3, 5,7,9, 4,5,6]>;
> H := Hom(H1, H2, "left");
> H: Maximal;
KMatrixSpace of 3 by 2 matrices and dimension 1 over Rational Field
Echelonized basis:

[1 0]
[1 1]
[0 1]
> Image(H.1);
KMatrixSpace of 3 by 3 matrices and dimension 1 over Rational Field
> Kernel(H.1);
KMatrixSpace of 2 by 3 matrices and dimension 0 over Rational Field
```

60.9.3 Modules $\mathrm{Hom}_R(M, N)$ with Given Basis

RMatrixSpaceWithBasis(Q): [ModMatRngElt] -> ModMatRng

Given a sequence Q of k independent matrices each lying in a matrix space $H = \mathrm{Hom}(M, N)$, where $M = R^{(m)}$ and $N = R^{(n)}$, construct the subspace of H of dimension k whose basis is Q. The basis is echelonized internally but all functions which depend on the basis of the matrix space (e.g. `Coordinates`) will use the given basis Q.

KMatrixSpaceWithBasis(Q): [ModMatRngElt] -> ModMatRng

Given a sequence Q of k independent matrices each lying in a matrix space $H = \mathrm{Hom}(M, N)$, where $M = K^{(m)}$ and $N = K^{(n)}$, with K a field, construct the subspace of H of dimension k whose basis is Q. The basis is echelonized internally but all functions which depend on the basis of the matrix space (e.g. `Coordinates`) will use the given basis Q.

60.9.4 The Endomorphism Ring

EndomorphismAlgebra(M): ModTupRng -> AlgMat

If M is the free R-module $R^{(m)}$, create the matrix algebra $\mathrm{Mat}_m(R)$. The algebra is created with the standard basis, $\{E_{ij} \mid i = 1 \ldots, m, j = 1 \ldots, m\}$, where E_{ij} is the matrix having a 1 in the (i, j)-th position and zeros elsewhere.

Example: Endomorphism Ring of a Vector Space (ex-7acde1)

We construct the endomorphism ring of the 4-dimensional vector space over the rational field.

```
> Q  := RationalField();
> R4 := RModule(Q, 4);
> M  := EndomorphismAlgebra(R4);
> M;
Full Matrix Algebra of degree 4 over Rational Field
```

60.9.5 The Reduced Form of a Matrix Module

`Reduce(H): ModMatRng -> ModMatRng, Map`

Suppose H is a matrix module whose elements have domain A and codomain B. Suppose first that A and B are tuple modules (R-spaces, R-modules, or RG-modules) and that A has degree a and dimension d while B has degree b and dimension e. (For the reduced cases (R-modules or RG-modules), a equals d and b equals e). The elements of H have the natural representation with respect to the standard embedded basis of the generic modules of A and B. Thus H has degree a by b. So one can multiply a 1 by a vector of A directly by an element h of H (in the natural matrix way) to get a 1 by b vector of B. Now suppose A and B are in the embedded form (R-space) and h is in H. Then h is an a by b matrix but there is a corresponding d by e matrix h' which gives the same transformation of h from A to B but is *with respect to the bases of A and B*. We call h' the *reduced* form of h. Also, there is the *reduced module* H' corresponding to H. This function constructs the reduced module H' corresponding to H, together with the epimorphism f from H onto H'. Note that if A and B are in reduced form, then H' is the same as H. Suppose secondly that A and B are matrix modules themselves. Suppose $A = \mathrm{Hom}(D_1, C_1)$, $B = \mathrm{Hom}(D_1, C_2)$, and $H = \mathrm{Hom}(A, B)$ with the right multiplication action. Suppose also that A has degree r by s and dimension d while B has degree r by t, and dimension e. Then H would have degree s by t so an element h of H would be s by t and would multiply a r by s element of A on the right to yield an r by t element of B. Then the reduced matrix h' corresponding to a matrix h of H would be a d by e matrix corresponding to the bases of A and B. This function similarly constructs the reduced module H' corresponding to H, together with the epimorphism f from H onto H'. Note also that in this case the domain and codomains of H' are the generic R-spaces (tuple modules) corresponding to A (of dimension d) and B (of dimension e). Similarly, for the left multiplication action there is the corresponding reduced module constructed in the obvious way. Note also that the kernel of the epimorphism f is the submodule of H which consists of all matrices which transform all elements of A to the zero element of B.

Example: Reduced Form for Maps Between Vector Spaces (ex-ecaba6)

We demonstrate the function `Reduce` for a homomorphism module from one vector space to another.

```
> V1 := sub<VectorSpace(GF(3), 3) | [1,0,1], [0,1,2]>;
> V2 := sub<VectorSpace(GF(3), 4) | [1,1,0,2], [0,0,1,2]>;
> H := Hom(V1, V2);
> H;
KMatrixSpace of 3 by 4 matrices and dimension 8 over GF(3)
> R, f := Reduce(H);
> R;
Full KMatrixSpace of 2 by 2 matrices over GF(3)
> H.1;
[1 0 0 0]
[0 1 0 2]
[0 1 0 2]
> f(H.1);
[1 0]
[0 0]
> V1.1;
(1 0 1)
> V1.1 * H.1;
(1 1 0 2)
> Coordinates(V2, V1.1 * H.1);
[ 1, 0 ]
> Coordinates(V2, V1.2 * H.1);
[ 0, 0 ]
> Kernel(f): Maximal;
KMatrixSpace of 3 by 4 matrices and dimension 4 over GF(3)
Echelonized basis:
```

(continues on next page)

(continued from previous page)

```
[1 0 0 0]
[2 0 0 0]
[2 0 0 0]

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

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

[0 0 0 1]
[0 0 0 2]
[0 0 0 2]
> R.1@@f;
[1 0 0 0]
[0 1 0 2]
[0 1 0 2]
```

Example: Reduced Form for Maps Between Hom Modules (ex-4c472d)

We demonstrate the function `Reduce` for a homomorphism module from one homomorphism module to another. Note that the reduced module has the same dimension as the original module but larger degrees!

```
> V1 := VectorSpace(GF(3), 2);
> V2 := VectorSpace(GF(3), 3);
> V3 := VectorSpace(GF(3), 4);
> H1 := Hom(V1, V2);
> H2 := Hom(V1, V3);
> H := Hom(H1, H2, "right");
> H1;
Full KMatrixSpace of 2 by 3 matrices over GF(3)
> H2;
Full KMatrixSpace of 2 by 4 matrices over GF(3)
> H;
Full KMatrixSpace of 3 by 4 matrices over GF(3)
> R,f := Reduce(H);
> R;
KMatrixSpace of 6 by 8 matrices and dimension 12 over GF(3)
> X := H.1;
> X;
[1 0 0 0]
[0 0 0 0]
[0 0 0 0]
> f(X);
[1 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]
> Domain(X);
Full KMatrixSpace of 2 by 3 matrices over GF(3)
```

(continues on next page)

(continued from previous page)

```
> Domain(f(X));
Full Vector space of degree 6 over GF(3)
> Image(X): Maximal;
KMatrixSpace of 2 by 4 matrices and dimension 2 over GF(3)
Echelonized basis:

[1 0 0 0]
[0 0 0 0]

[0 0 0 0]
[1 0 0 0]

> Image(f(X));
Vector space of degree 8, dimension 2 over GF(3)
Echelonized basis:
(1 0 0 0 0 0 0 0)
(0 0 0 0 1 0 0 0)
> Kernel(X): Maximal;
KMatrixSpace of 2 by 3 matrices and dimension 4 over GF(3)
Echelonized basis:

[0 1 0]
[0 0 0]

[0 0 1]
[0 0 0]

[0 0 0]
[0 1 0]

[0 0 0]
[0 0 1]

> Kernel(f(X)): Maximal;
Vector space of degree 6, dimension 4 over GF(3)
Echelonized basis:
(0 1 0 0 0 0)
(0 0 1 0 0 0)
(0 0 0 0 1 0)
(0 0 0 0 0 1)
```

60.9.6 Construction of a Matrix

`M ! Q: ModMatRng, [RngElt] -> ModMatRngElt`

Given the matrix bimodule M over the ring R, and the sequence $Q = [a_{11}, \ldots, a_{1n}, a_{21}, \ldots, a_{2n}, \ldots, a_{m1}, \ldots, a_{mn}]$ whose terms are elements of the ring R, construct the $m \times n$ matrix

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

as an element of M. In the context of the `sub` or `quo` constructors the coercion clause `M !` may be omitted.

Example: Hilbert Matrix as an Endomorphism `(ex-9b8959)`

We create the 4×4 Hilbert matrix $h4$ as an element of the endomorphism ring of the 4-dimensional

vector space over the rational field.

```
> Q  := RationalField();
> R4 := RModule(Q, 4);
> M  := EndomorphismAlgebra(R4);
> h4 := M ! [ 1/i :  i in [1 .. 16 ] ];
> h4;
[   1  1/2  1/3  1/4]
[ 1/5  1/6  1/7  1/8]
[ 1/9 1/10 1/11 1/12]
[1/13 1/14 1/15 1/16]
```

60.9.7 Element Operations

All operations that apply to elements of tuple modules also apply to elements of matrix modules. Here, we confine our discussion to those operations which are special to matrix modules. Throughout this section, M is a submodule of $R^{(m)}$, N is a submodule of $R^{(n)}$ and a is a homomorphism belonging to $\mathrm{Hom}_R(M, N)$, where R is a Euclidean Domain. See also the chapter on general matrices for many other functions applicable to matrices (e.g., `EchelonForm`).

`u * a: ModTupRngElt, ModMatRngElt -> ModTupRngElt`

`a(u): ModTupRngElt, ModMatRngElt -> ModTupRngElt`

Given an element u belonging to the module M, return the image of u under the homomorphism a as an element of the module N.

`a * b: ModMatRngElt, ModMatRngElt -> ModMatRngElt`

Given a homomorphism a belonging to a submodule of $\mathrm{Hom}(M, N)$, and a homomorphism b belonging to a submodule of $\mathrm{Hom}(N, P)$, return the composition of the homomorphisms a and b as an element of $\mathrm{Hom}(M, P)$. Note that if $\mathrm{Hom}(M, P)$ does not already exist, it will be created.

`a ^ -1: ModMatRngElt, RngIntElt -> ModMatRngElt`

Given a homomorphism a belonging to a submodule of $\mathrm{Hom}(M, N)$ with M and N having the same dimension, return the inverse of a as an element of $\mathrm{Hom}(N, M)$.

`Codomain(S): ModMatRng -> ModTupRng`

Given a submodule S of the module $\mathrm{Hom}(M, N)$, return the module N.

`Codomain(a): ModMatRngElt -> ModTupRng`

The codomain N of the homomorphism a belonging to $\mathrm{Hom}(M, N)$.

`Cokernel(a): ModMatRngElt -> ModTupRng`

The cokernel for the homomorphism a belonging to the module $\mathrm{Hom}(M, N)$.

`Domain(S): ModMatRng -> ModTupRng`

The domain M of the submodule S belonging to the module $\mathrm{Hom}(M, N)$.

`Domain(a): ModMatRngElt -> ModTupRng`

The domain M of the homomorphism a belonging to the module $\mathrm{Hom}(M, N)$.

`Image(a): ModMatRngElt -> ModTupRng`

The image of the homomorphism a belonging to the module $H = \mathrm{Hom}(M, N)$, returned as a submodule of N. Note that if the domain and codomain of a are matrix modules themselves, the image will be with respect to the appropriate action (right or left).

`Kernel(a): ModMatRngElt -> ModTupRng`

`NullSpace(a): ModMatRngElt -> ModTupRng`

The kernel of the homomorphism a belonging to the module $\mathrm{Hom}(M, N)$, returned as a submodule of M. Note that if the domain and codomain of a are matrix modules themselves, the kernel will be with respect to the appropriate action (right or left).

`Morphism(M, N): ModTupRng, ModTupRng -> ModMatRngElt`

Assuming the R-module M was created as a submodule of the module N, return the matrix defining the inclusion homomorphism $\phi : M \to N$ as an element of $\mathrm{Hom}_R(M, N)$. Thus ϕ gives the correspondence between elements of M (represented with respect to the standard basis of M) and elements for N.

`Rank(a): ModMatRngElt -> RngIntElt`

The dimension of the image of the homomorphism a, i.e. the rank of a.

`IsBijective(a): ModMatRngElt -> BoolElt`

Returns `true` if the homomorphism a belonging to the module $\mathrm{Hom}(M, N)$ is a bijective mapping.

`IsInjective(a): ModMatRngElt -> BoolElt`

Returns `true` if the homomorphism a belonging to the module $\mathrm{Hom}(M, N)$ is an injective mapping.

`IsSurjective(a): ModMatRngElt -> BoolElt`

Returns `true` if the homomorphism a belonging to the module $\mathrm{Hom}_R(M, N)$ is a surjective mapping.

Example: Operations in a Homomorphism Module (ex-85af3b)

We illustrate some of these operations in the context of the module $\mathrm{Hom}_R(M, N)$, where M and N are, respectively, the 4-dimensional and 3-dimensional vector spaces over $GF(8)$.

```
> K<w> := GaloisField(8);
> V3 := VectorSpace(K, 3);
> V4 := VectorSpace(K, 4);
> M := Hom(V4, V3);
> A := M ! [1, w, w^5, 0,  w^3, w^4, w, 1,  w^6, w^3, 1, w^4 ];
> A;
[  1   w w^5]
[  0 w^3 w^4]
[  w   1 w^6]
[w^3   1 w^4]
> Rank(A);
3
> Image(A);
Full Vector space of degree 3 over GF(2^3)
> Kernel(A);
Vector space of degree 4, dimension 1 over GF(2^3)
Echelonized basis:
(  1 w^5   1   1)
> Cokernel(A);
Vector space of degree 3, dimension 0 over GF(2^3)
```

Chapter 61

Modules over Dedekind Domains

61.1 Introduction

Since the structure theory for modules over arbitrary orders (which are in general not Dedekind domains) is very unsatisfactory, modules over orders in Magma are always modules over some maximal order of a number field or function field, they form a magma of type `ModDed`.

Let k be a number field or function field and $\mathcal{O}_k$ its ring of integers. Since $\mathcal{O}_k$ is a Dedekind domain, every finitely generated torsion free module M over $\mathcal{O}_k$ has a representation as a direct sum

$$M = \sum_{i=1}^{m} \mathcal{A}_i \alpha_i = \{\sum_{i=1}^{m} a_i \alpha_i \mid a_i \in \mathcal{A}_i\}$$

with (fractional) ideals $\mathcal{A}_i$ and elements $\alpha_i \in kM \cong k^r$.

A (not necessarily direct) sum $\sum_{i=1}^{m} \mathcal{A}_i \alpha_i$ will be represented as a pseudo–matrix $(\mathcal{A}|A)$ where $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_m)^t$ is a column vector of ideals and $A = (\alpha_1, \ldots, \alpha_m)^t \in k^{m \times r}$ is a matrix. The ideals $\mathcal{A}_i$ are called coefficient ideals.

This pseudo–matrix is called a pseudo–basis iff the sum is direct. A pseudo–matrix $(\mathcal{A}|A)$ is in Hermite normal form iff there are $s \leq m$, $1 \leq i_1 < i_2 < \cdots < i_s$ such that $A_{j,l} = 0$ $(1 \leq l < i_j)$, $A_{j,i_j} = 1$ and $A_{j,l}$ is reduced modulo $\mathcal{A}_j {\mathcal{A}_l}^{-1}$. For $j > s$ we have $A_{j,l} = 0$.

This normal form is unique if a suitable reduction is used.

As a consequence of this normalisation, usually $\alpha_i \notin M$. To be precise: $\alpha_i \in M$ iff $1 \in \mathcal{A}_i$.

All modules are in Hermite normal form, i.e. every module is represented by a pseudo–basis in Hermite normal form.

General (non torsion free) modules are represented as quotients of a torsion free module M and a submodule S. Elements of $Q := M/S$ are represented as elements of M, arithmetic in Q is reduced to arithmetic in M followed by a reduction modulo the pseudo–basis of S.

61.2 Creation of Modules

Modules over Dedekind domains can be created from orders of number fields and function fields and combinations of ideals and vector space elements. Submodules and quotient modules by submodules can also be created.

`Module(O, n): RngOrd, RngIntElt -> ModDed`

`Module(O, n): RngFunOrd, RngIntElt -> ModDed`

Create the free module O^n where O is a Dedekind domain.

`Module(O): RngOrd -> ModDed, Map`

`Module(O): RngFunOrd -> ModDed, Map`

Create the relative order O as a module over its coefficient ring. Also returns the map from the resulting module into O.

`Module(I): RngOrdFracIdl -> ModDed, Map`

`Module(I): RngFunOrdIdl -> ModDed, Map`

Create the ideal I of a relative order O as a module over the coefficient ring of the order. Also returns the map from the module into O.

`Module(S): SeqEnum[Tup] -> ModDed, Map`

Create a module from the sequence of tuples of ideals of a Dedekind domain and `ModElt`s with entries in the Dedekind domain or its field of fractions. The elements of the resulting module will be the sum of products of an element of an ideal and the corresponding `ModElt`. Also returns the map from the vector space into the module.

`Module(S): SeqEnum[RngOrdFracIdl] -> ModDed`

`Module(S): SeqEnum[RngFunOrdIdl] -> ModDed`

Create the module which is equal to the direct sum of the ideals in the sequence.

`Module(S): SeqEnum[ModElt] -> ModDed, Map`

Create the module which is freely generated by the elements of the sequence S. The elements of the sequence must be `ModElt`s with entries in a Dedekind domain or field of fractions of a Dedekind domain. Also returns the map from the vector space into the module.

Example: Creating Modules over Dedekind Domains (ex-5637aa)

The creation of some simple modules is shown.

```
> x := ext<Integers()|>.1;
> M := MaximalOrder(x^2 + 5);
> Module(M, 5);
Module over Maximal Equation Order with defining polynomial x^2 + 5 over Z
 generated by: (in echelon form)
Principal Ideal of M
Generator:
    M.1 * ( M.1 0 0 0 0 )
Principal Ideal of M
Generator:
    M.1 * ( 0 M.1 0 0 0 )
Principal Ideal of M
Generator:
    M.1 * ( 0 0 M.1 0 0 )
Principal Ideal of M
Generator:
    M.1 * ( 0 0 0 M.1 0 )
Principal Ideal of M
Generator:
    M.1 * ( 0 0 0 0 M.1 )
> I := 1/5*M;
> Module([I, I^3, I^8]);
Module over Maximal Equation Order with defining polynomial x^2 + 5 over Z
 generated by: (in echelon form)
Fractional Principal Ideal of M
Generator:
    1/5*M.1 * ( M.1 0 0 )
Fractional Principal Ideal of M
Generator:
    1/125*M.1 * ( 0 M.1 0 )
Fractional Principal Ideal of M
```

(continues on next page)

(continued from previous page)

```
Generator:
    1/390625*M.1 * ( 0 0 M.1 )
> V := RModule(M, 3);
> Module([<I, V![0, 1, 0]>, <I^4, V![2, 3, 5]>]);
Module over Maximal Equation Order with defining polynomial x^2 + 5 over Z
Fractional Principal Ideal of M
Generator:
    1/5*M.1 car Fractional Principal Ideal of M
Generator:
    1/125*M.1
```

The same can be done using orders of function fields.

```
> P<x> := PolynomialRing(Rationals());
> P<y> := PolynomialRing(P);
> F<c> := FunctionField(x^2 - y);
> M := MaximalOrderFinite(F);
> Module(M, 5);
Module over Maximal Equation Order of F over Univariate Polynomial Ring in x over
Rational Field
 generated by: (in echelon form)
Ideal of M
Generator:
1 * ( 1 0 0 0 0 )
Ideal of M
Generator:
1 * ( 0 1 0 0 0 )
Ideal of M
Generator:
1 * ( 0 0 1 0 0 )
Ideal of M
Generator:
1 * ( 0 0 0 1 0 )
Ideal of M
Generator:
1 * ( 0 0 0 0 1 )
> I := 1/5*M;
> Module([I, I^3, I^8]);
Module over Maximal Equation Order of F over Univariate Polynomial Ring in x over
Rational Field
 generated by: (in echelon form)
Ideal of M
Generator:
1/5 * ( 1 0 0 )
Ideal of M
Generator:
1/125 * ( 0 1 0 )
Ideal of M
Generator:
1/390625 * ( 0 0 1 )
> V := RModule(M, 3);
> Module([<I, V![0, 1, 0]>, <I^4, V![2, 3, 5]>]);
Integral Module over Maximal Equation Order of F over Univariate Polynomial Ring
in x over Rational
Field
Ideal of M
```

(continues on next page)

(continued from previous page)

```
Generator:
1/5 car Ideal of M
Generator:
1/125
```

sub<M | m>: ModDed, SeqEnum[ModDedElt] -> ModDed, Map

sub<M | m1, .., mn>: ModDed, ModDedElt, ..., ModDedElt -> ModDed, Map

Construct the submodule of the module M generated by the elements in the sequence or list of elements m. Also returns the inclusion map of the submodule into M.

quo<M | S>: ModDed, ModDed -> ModDed, Map

quo<M | m>: ModDed, SeqEnum[ModDedElt] -> ModDed, Map

quo<M | m1, .., mn>: ModDed, ModDedElt, ..., ModDedElt -> ModDed, Map

Construct the quotient of the module M by the submodule S or the submodule generated by the elements of the sequence or list of elements m. Also returns the inclusion map of the quotient module into M.

Example: Submodules and Quotient Modules (ex-90c9d8)

Use of the sub and quo constructors is illustrated below. Let M and V be as above when they were referring to number fields.

```
> Mod := Module([V|[0,1,0], [4,4,0]]);
> S1 := sub<Mod | >;
> S1;
Integral Module over Maximal Equation Order with defining polynomial x^2 + 5
over Z
(0)
> Q1 := quo<Mod | Mod>;
> Q1;
Quotient of Module over Maximal Equation Order with defining polynomial x^2 + 5
over Z
Principal Ideal of M
Generator:
    4/1*M.1 car Principal Ideal of M
Generator:
    M.1
by Integral Module over Maximal Equation Order with defining polynomial
x^2 + 5 over Z
Principal Ideal of M
Generator:
    4/1*M.1 car Principal Ideal of M
Generator:
    M.1
> S2 := sub<Mod | Mod.2>;
> S2;
Integral Module over Maximal Equation Order with defining polynomial x^2 + 5
over Z
Principal Ideal of M
Generator:
    M.1
> Q2 := quo<Mod | Mod.2>;
> Q2;
Quotient of Module over Maximal Equation Order with defining polynomial x^2 + 5
over Z
Principal Ideal of M
```

(continues on next page)

(continued from previous page)

```
Generator:
    4/1*M.1 car Principal Ideal of M
Generator:
    M.1
by Integral Module over Maximal Equation Order with defining polynomial
x^2 + 5 over Z
Principal Ideal of M
Generator:
    M.1
> S3 := sub<Mod | 4*Mod.1, Mod.2>;
> S3;
Integral Module over Maximal Equation Order with defining polynomial x^2 + 5
over Z
Principal Ideal of M
Generator:
    4/1*M.1 car Principal Ideal of M
Generator:
    M.1
> Q3 := quo<Mod | >;
> Q3;
Integral Module over Maximal Equation Order with defining polynomial x^2 + 5
over Z
Principal Ideal of M
Generator:
    4/1*M.1 car Principal Ideal of M
Generator:
    M.1
> Q4 := quo<Mod | S1>;
> Q4;
Quotient of Module over Maximal Equation Order with defining polynomial x^2 + 5
over Z
Principal Ideal of M
Generator:
    4/1*M.1 car Principal Ideal of M
Generator:
    M.1
by Integral Module over Maximal Equation Order with defining polynomial
x^2 + 5 over Z
(0)
```

61.3 Elementary Functions

Various simple properties of a module can be retrieved using the following functions.

BaseRing(M): ModDed -> Rng
CoefficientRing(M): ModDed -> Rng

The Dedekind domain which M is a module over.

Degree(M): ModDed -> RngIntElt

The dimension of the vector space the module M embeds into.

Ngens(M): ModDed -> RngIntElt
NumberOfGenerators(M): ModDed -> RngIntElt

The minimum number of vectors and ideals which generate the module M.

M . i: ModDed, RngIntElt -> ModTupRngElt

The vector of the ith vector and ideal pair generating the module M.

`Determinant(M): ModDed -> RngOrdIdl`

`Determinant(M): ModDed -> RngFunOrdIdl`

The determinant of the module M.

`Dimension(M): ModDed -> RngIntElt`

The dimension of the vector space spanned by the module M over its coefficient ring. This is the same as the number of generators of a pseudo basis of M.

`Contents(M): ModDed -> RngOrdFracIdl`

```
UseBasis: BoolElt                    Default: false
```

The contents of the module M, ie. the gcd of the ideals obtained by multiplying the coefficient ideals by the ideal generated by the coefficients in the corresponding generators. The parameter `UseBasis` decides whether a pseudo basis or pseudo generators are used.

`Simplify(M): ModDed -> ModDed`

```
UseBasis: BoolElt                    Default: false
```

Computes a module of contents 1 by scaling each coefficient ideal by the inverse of the contents of the module M. The parameter `UseBasis` determines if the operations are performed on the pseudo generators or the pseudo basis of M.

`EmbeddingSpace(M): ModDed -> Mod`

The canonical vector space containing the module M, ie.. M tensored with the field of fractions of the coefficient ring.

Example: Elementary Functions on a Module (ex-5e5c19)

The use of some elementary functions on a module is shown below.

```
> P<x> := PolynomialRing(Rationals());
> P<y> := PolynomialRing(P);
> F<c> := FunctionField(x^2 - y);
> M := MaximalOrderFinite(F);
> Vs := RModule(M, 2);
> s := [Vs | [1, 3], [2, 3]];
> Mods := Module(s);
> CoefficientRing(Mods);
Maximal Equation Order of F over Univariate Polynomial Ring in x over Rational
Field
> Mods.1;
(1 0)
> Determinant(Mods);
Ideal of M
Generator:
-3
> Vs := RSpace(M, 2);
> s := [Vs | [1, 3], [2, 3]];
> Mods := Module(s);
> sMods := sub<Mods | Mods!Vs![1, 3]>;
> qMods := quo<Mods | sMods>;
> Degree(Mods);
2
> Ngens(Mods);
2
> Ngens(sMods);
1
```

(continues on next page)

(continued from previous page)

```
> Degree(sMods);
2
> Degree(qMods);
2
> Ngens(qMods);
2
> Determinant(Mods);
Ideal of M
Basis:
[1]
> Determinant(sMods);
>>   Determinant(sMods);
                ^
Runtime error in `Determinant': Module must be square
> Determinant(qMods);
Ideal of M
Basis:
[1]
```

61.4 Predicates on Modules

Modules and potential elements can be tested against each other for a few properties.

`M eq N: ModDed, ModDed -> BoolElt`

Return `true` if M and N are equal as modules.

`x in M: Any, ModDed -> BoolElt`

Return `true` if x can be coerced into the module M.

`M subset N: ModDed, ModDed -> BoolElt`

Return `true` if M is a submodule of N. An embedding map of M in N can be returned by `IsSubmodule(M, N)`.

`IsFree(M): ModDed -> BoolElt`

Return `true` if M is a free module.

61.5 Arithmetic with Modules

Some arithmetic operations can be carried out involving modules and their elements and compatible ideals to gain more modules.

`I * M: RngOrdIdl, ModDed -> ModDed`

`I * M: RngFunOrdIdl, ModDed -> ModDed`

`M * I: ModDed, RngOrdIdl -> ModDed`

`M * I: ModDed, RngFunOrdIdl -> ModDed`

The module generated by the products of the ideals of the module M with I.

`M1 + M2: ModDed, ModDed -> ModDed`

The union of the modules $M1$ and $M2$.

`DirectSum(M1, M2): ModDed, ModDed -> ModDed, Map, Map, Map, Map`

`DirectSum(S): [ModDed] -> ModDed, [Map], [Map]`

The direct sum D of the modules $M1$ and $M2$ or the modules in the sequence S together with the embedding maps $M1 \to D, M2 \to D, \ldots$ and the projection maps $D \to M1, D \to M2, \ldots$ returned in sequences only when a sequence S is input.

`u * I: ModDedElt, RngOrdIdl -> ModDed`

`u * I: ModDedElt, RngFunOrdIdl -> ModDed`

`I * u: RngOrdIdl, ModDedElt -> ModDed`

`I * u: RngFunOrdIdl, ModDedElt -> ModDed`

The module containing elements which are products of the Dedekind module element u and an element lying in the ideal I.

Example: Module Arithmetic and Predicates (ex-79759b)

Some module predicates and arithmetic are shown below.

```
> P<x> := PolynomialRing(Integers());
> K :=  NumberField([x^5 + 3, x^2 + 2]);
> M := MaximalOrder(K);
> Vs := RModule(M, 2);
> s := [Vs | [1, 3], [2, 3]];
> Mods := Module(s);
> sMods := sub<Mods | Mods!Vs![1, 3]>;
> Mods eq sMods;
false
> [1, 0] in Mods;
true
> [1, 0] in sMods;
false
> sMods subset Mods;
true
> Vs := RSpace(M, 2);
> s := [Vs | [Random(M, 3), 3], [2, Random(M, 2)]];
> Mods := Module(s);
> sMods := sub<Mods | Mods!s[1]>;
> (7*M + 11*K.1*M)*sMods;
Module over Maximal Equation Order with defining polynomial x^5 + [3, 0] over
its ground order
 generated by:
Ideal of M
Two element generators:
    7/1*$.1*M.1
    11/1*$.1*M.2 * ( M.1 + (-$.1 - 2/1*$.2)*M.2 + $.2*M.3 + (2/1*$.1 +
2/1*$.2)*M.5 3/1*$.1*M.1 )
 in echelon form:
Ideal of M
Two element generators:
    21/1*$.1*M.1
    33/1*$.1*M.2 * ( 1/3*$.1*M.1 + (-1/3*$.1 - 2/3*$.2)*M.2 + 1/3*$.2*M.3 +
(2/3*$.1 + 2/3*$.2)*M.5 M.1 )
> Mods + sMods;
Module over Maximal Equation Order with defining polynomial x^5 + [3, 0] over
its ground order
 generated by: (in echelon form)
Ideal of M
Two element generators:
    148919257164/1*$.1*M.1
    (148919257048/1*$.1 + 26/1*$.2)*M.1 + (32/1*$.1 + 148919257015/1*$.2)*M.2 +
    (196/1*$.1 + 148919257117/1*$.2)*M.3 + (148919257163/1*$.1 + 76/1*$.2)*M.4 +
    (148919257077/1*$.1 + 148919257116/1*$.2)*M.5 * ( M.1 0 )
Ideal of M
Two element generators:
```

(continues on next page)

(continued from previous page)

```
    3/1*$.1*M.1
    ($.1 - $.2)*M.1 + -M.3 + M.4 + ($.1 + $.2)*M.5 * ( (-19866460521/1*$.1 -
33/1*$.2)*M.1 + (1/3*$.1 + 1/3*$.2)*M.4 + (1/3*$.1 + 1/3*$.2)*M.5 M.1 )
> 4*sMods;
Module over Maximal Equation Order with defining polynomial x^5 + [3, 0] over
its ground order
 generated by:
Principal Ideal of M
Generator:
    4/1*$.1*M.1 * ( (2/1*$.1 - 2/1*$.2)*M.1 + ($.1 + 2/1*$.2)*M.2 + (2/1*$.1 +
2/1*$.2)*M.3 + (2/1*$.1 - $.2)*M.5 3/1*$.1*M.1 )
 in echelon form:
Principal Ideal of M
Generator:
    12/1*$.1*M.1 * ( (2/3*$.1 - 2/3*$.2)*M.1 + (1/3*$.1 + 2/3*$.2)*M.2 +
(2/3*$.1 + 2/3*$.2)*M.3 + (2/3*$.1 - 1/3*$.2)*M.5 M.1 )
```

61.6 Basis of a Module

The basis of a module is given by vectors. However, more complete information can be supplied which includes the ideals.

Basis(M): ModDed -> SeqEnum

A sequence of vectors which correspond to a pseudo basis of the module M.

PseudoBasis(M): ModDed -> SeqEnum

A sequence of tuples containing ideals and vectors which generate the module M. The vectors are guaranteed to be linearly independent.

PseudoGenerators(M): ModDed -> SeqEnum

A sequence of tuples containing ideals and vectors which generate the module M. This will return the data used to define the module, so that in contrast to *PseudoBasis* the vectors will in general not be independent.

LocalBasis(M, p): ModDed, RngOrdIdl -> [ModTupFldElt]

```
Type: MonStgElt                    Default: ""
```

Returns a basis of a free module F that agrees with M at the completion at p. If Type is specified, it must either be "Submodule" or "Supermodule". In which case F is either a sub- or supermodule of M.

61.7 Other Functions on Modules

Intersections of modules can taken. Several other functions are also available.

M1 **meet** M2: ModDed, ModDed -> ModDed

Return the intersection of the modules $M1$ and $M2$.

Dual(M): ModDed -> ModDed

The module dual to M.

ElementaryDivisors(M, N): ModDed, ModDed -> SeqEnum

The elementary divisors (ideals) of the torsion part of the quotient R-module M/N: For $N \subseteq M$ we get

$$T(M/N) \cong \oplus_{i=1}^{n} R/\mathcal{A}_i$$

The $\mathcal{A}_i$ are unique if we require $R \subseteq \mathcal{A}_1 \subseteq \cdots \subseteq \mathcal{A}_n$. The $\mathcal{A}_i$ are called the elementary divisors (or elementary ideals) of M/N. This corresponds to the Smith normal form for integral matrices.

SteinitzClass(M): ModDed -> RngOrdIdl

The Steinitz class of the module M.

SteinitzForm(M): ModDed -> ModDed

The Steinitz (almost–free) form of the module M.

Example: Bases and Other Module Functions (ex-69678f)

Some bases and other functions are demonstrated below.

```
> P<x> := PolynomialRing(Rationals());
> P<y> := PolynomialRing(P);
> F<c> := FunctionField(y^3 - x^3*y^2 + y - x^7);
> M := MaximalOrderFinite(F);
> Vs := RSpace(M, 2);
> s := [Vs | [1, Random(M, 3)], [Random(M, 3), 3]];
> Mods := Module(s);
> qMods := quo<Mods | Mods!s[2]>;
> Basis(Mods);
[
    ([ 1, 0, 0 ] [ 2*x^2 - 3*x + 1/2, -2/3*x^2 - x + 1/3, 2/3*x^2 - 2*x + 1/2
        ]),
    ([ -x^2 + x + 2/3, -1/3*x^2 - 1, -3/2*x^2 - 2/3*x + 1 ] [ 3, 0, 0 ])
]
> Basis(qMods);
[
    ([ 1, 0, 0 ] [ 2*x^2 - 3*x + 1/2, -2/3*x^2 - x + 1/3, 2/3*x^2 - 2*x + 1/2
        ]),
    ([ -x^2 + x + 2/3, -1/3*x^2 - 1, -3/2*x^2 - 2/3*x + 1 ] [ 3, 0, 0 ])
]
> PseudoBasis(Mods) eq PseudoBasis(qMods);
> Vs := RModule(M, 2);
> s := [Vs | [Random(M, 3), Random(M, 3)], [2, 3]];
> Mods := Module(s);
> sMods := sub<Mods | Mods!s[1]>;
> Mods meet sMods;
Module over Maximal Equation Order of F over Univariate Polynomial Ring in x
over Rational Field
Ideal of M
Generator:
(x + 1)*c^2 + (-3/2*x^2 - 3/2*x)*c + 1/2*x^2 + 2/3*x + 1/2
> ElementaryDivisors(Mods, sMods);
[ Ideal of M
Basis:
[1 0 0]
[0 1 0]
[0 0 1], Ideal of M
Generator:
0 ]
> Dual(Mods);
Module over Maximal Equation Order of F over Univariate Polynomial Ring in x
over Rational Field
Fractional ideal of M
Generator:
(-3*x^4 + 21*x^3 + 20*x^2 + 80*x - 387)/(x^17 - 24*x^16 + 192*x^15 - 510*x^14 -
    94/3*x^13 + 304/3*x^12 + 902/3*x^11 - 3109/3*x^10 - 389/9*x^9 + 664/9*x^8 +
```

(continues on next page)

(continued from previous page)

```
    1094/3*x^7 - 1540/3*x^6 - 101/9*x^5 + 128/3*x^4 + 2000/27*x^3 - 4361/9*x^2 -
    427*x + 2056)*c^2 + (-3*x^9 + 48*x^8 - 189*x^7 - 24*x^6 + 3*x^5 - 48*x^4 +
    195*x^3 - 3*x^2 - x + 24)/(x^17 - 24*x^16 + 192*x^15 - 510*x^14 - 94/3*x^13
    + 304/3*x^12 + 902/3*x^11 - 3109/3*x^10 - 389/9*x^9 + 664/9*x^8 + 1094/3*x^7
    - 1540/3*x^6 - 101/9*x^5 + 128/3*x^4 + 2000/27*x^3 - 4361/9*x^2 - 427*x +
    2056)*c + (3*x^12 - 48*x^11 + 192*x^10 + 3*x^9 - 20*x^8 - 56*x^7 + 192*x^6 +
    3*x^5 - 5*x^4 - 8*x^3 + 272/3*x^2 + 128*x - 771)/(x^17 - 24*x^16 + 192*x^15
    - 510*x^14 - 94/3*x^13 + 304/3*x^12 + 902/3*x^11 - 3109/3*x^10 - 389/9*x^9 +
    664/9*x^8 + 1094/3*x^7 - 1540/3*x^6 - 101/9*x^5 + 128/3*x^4 + 2000/27*x^3 -
    4361/9*x^2 - 427*x + 2056) car Ideal of M
Generator:
1/3
> Dual(sMods);
Module over Maximal Equation Order of F over Univariate Polynomial Ring in x
over Rational Field
Fractional ideal of M
Generator:
(3/2*x^6 + 3*x^5 - 3/4*x^4 - 4*x^3 - 25/12*x^2 - 5/6*x - 1/2)/(x^17 + 3*x^16 +
    21/4*x^15 + 27/4*x^14 + 1/24*x^13 - 247/24*x^12 - 173/24*x^11 + 61/24*x^10 +
    305/72*x^9 + 17/72*x^8 - 4*x^7 - 53/12*x^6 - 1/8*x^5 + 23/6*x^4 +
    377/108*x^3 + 23/18*x^2 + 1/3*x + 1/8)*c^2 + (-5/2*x^9 - 5*x^8 - 1/4*x^7 +
    9/2*x^6 + 13/4*x^5 + 5/4*x^4 - 3/4*x^3 - 7/4*x^2 - 3/4*x)/(x^17 + 3*x^16 +
    21/4*x^15 + 27/4*x^14 + 1/24*x^13 - 247/24*x^12 - 173/24*x^11 + 61/24*x^10 +
    305/72*x^9 + 17/72*x^8 - 4*x^7 - 53/12*x^6 - 1/8*x^5 + 23/6*x^4 +
    377/108*x^3 + 23/18*x^2 + 1/3*x + 1/8)*c + (x^12 + 2*x^11 - 1/2*x^10 -
    7/2*x^9 - 8/3*x^8 - 5/12*x^7 + 11/4*x^6 + 19/4*x^5 - 1/4*x^4 - 25/6*x^3 -
    67/36*x^2 - 1/3*x - 1/4)/(x^17 + 3*x^16 + 21/4*x^15 + 27/4*x^14 + 1/24*x^13
    - 247/24*x^12 - 173/24*x^11 + 61/24*x^10 + 305/72*x^9 + 17/72*x^8 - 4*x^7 -
    53/12*x^6 - 1/8*x^5 + 23/6*x^4 + 377/108*x^3 + 23/18*x^2 + 1/3*x + 1/8)
> SteinitzClass(Mods) eq SteinitzClass(sMods);
false
> SteinitzForm(Mods);
Module over Maximal Equation Order of F over Univariate Polynomial Ring in x
over Rational Field
Ideal of M
Generator:
3 car Ideal of M
Generator:
1
> SteinitzForm(sMods);
Module over Maximal Equation Order of F over Univariate Polynomial Ring in x
over Rational Field
Ideal of M
Generator:
1
```

61.8 Homomorphisms between Modules

It is possible to create a homomorphism between two modules, take the image and kernel of such and verify that these are submodules of the codomain and domain respectively. The Hom–module can also be created as a module of a Dedekind domain.

hom<M -> N | T>: ModDed, ModDed, Map -> Map

hom<M -> N | T>: ModDed, ModDed, SeqEnum[ModTupRngElt] -> Map

hom<M -> N | T>: ModDed, ModDed, MtrxSpcElt -> Map

```
ModuleBasis: BoolElt                    Default: true
```

Return a homomorphism from the module M into the module N as specified by T from which the images of the generators can be inferred. T may be a map between the vector spaces of same degree as M and N, a matrix over the field of fractions or a sequence of vectors. If `ModuleBasis` is `true` then the matrix will be taken to be a transformation between the modules and as such will be expected to have size `Dimension(M)*Dimension(N)` otherwise it will be interpreted as a transformation between the corresponding vector spaces and will be expected to have size `Degree(M)*Degree(N)`.

`Hom(M, N): ModDed, ModDed -> ModDed, Map`

The module of homomorphisms between the module M and the module N and the map from the hom–module to the collection of maps from M to N, (such that given an element of the hom–module a homomorphism from M to N is returned). The module is over the same Dedekind domain as M and N.

`IsSubmodule(M, N): ModDed, ModDed -> BoolElt, Map`

Return `true` if M is a submodule of N and the map embedding M into N.

`Morphism(M, N): ModDed, ModDed -> Map`

The map giving the morphism from the module M to the module N. Either M is a submodule of N, in which case the embedding of M into N is returned, or N is a quotient module of M, in which case the natural epimorphism from M onto N is returned.

Example: Homomorphisms Between Modules over Dedekind Domains (ex-f6bf14)

This example demonstrates the use of homomorphisms between modules over Dedekind domains. Let `M` and `V` be as above referring to function fields.

```
> S := [V|[0,1,0], [4,4,0]];
> Mod := Module(S);
> W := KModule(FieldOfFractions(M), 4);
> S := [W|[3, 2, 1, 0]];
> N := Module(S);
> h := hom<Mod -> N | >;
>> h := hom<Mod -> N | >;
           ^
Runtime error in map< ... >: No images given
> h := hom<Mod -> N | V.1, V.2, V.3>;

>> h := hom<Mod -> N | V.1, V.2, V.3>;
           ^
Runtime error in map< ... >: An image for each generator is required
> h := hom<Mod -> N | W![3, 2, 1, 0], W![3*(M!F.1 + 1), 2*(M!F.1 + 1),
> M!F.1 + 1, 0] >;
> h(Mod!(4*V.1));
( 4 )
> h(Mod!V![0, 1, 0]);
( x^2 + 1 )
> I := Image(h);
> I;
Module over Maximal Equation Order of F over Univariate Polynomial Ring in x
over Rational Field
Ideal of M
Generator:
1
> K := Kernel(h);
> K;
Integral Module over Maximal Equation Order of F over Univariate Polynomial Ring
```

(continues on next page)

(continued from previous page)

```
in x over Rational Field
Ideal of M
Generator:
1
> IsSubmodule(K, Mod);
true Mapping from: ModDed: K to ModDed: Mod
> H, m := Hom(Mod, N);
> H; m;
Module over Maximal Equation Order of F over Univariate Polynomial Ring in x over
Rational Field
 generated by: (in echelon form)
Ideal of M
Generator:
1/4 * ( 1 0 )
Ideal of M
Generator:
1 * ( 0 1 )
Mapping from: ModDed: H to Power Structure of Map given by a rule [no inverse]
> m(H![5, 20]);
Mapping from: ModDed: Mod to ModDed: N
using
[5]
[20]
```

61.9 Elements of Modules

61.9.1 Creation of Elements

`M ! v: ModDed, SeqEnum -> ModDedElt`

`M ! v: ModDed, ModDedElt -> ModDedElt`

`M ! v: ModDed, ModElt -> ModDedElt`

Coerce v into an element of M. v can be a sequence of length dimension of M, a module element or vector or an element of another module over a Dedekind domain which is compatible with M.

Example: Coercion into Quotient Modules `(ex-2c7014)`

Let `Mod` and its submodules and quotient modules be as in the sub and quotient module example above.

```
> m := 4*Mod.1;
> m;
(4/1*M.1 0)
> Q1!m;
( 4/1*M.1 0 )
> Q2!m;
( 4/1*M.1 0 )
> m := Mod!m;
> Q3!m;
( 4/1*M.1 0 )
> Q4!m;
( 4/1*M.1 0 )
> S1!m;
>> S1!m;
     ^
Runtime error in `!': Illegal coercion
```

(continues on next page)

(continued from previous page)

```
LHS: ModDed
RHS: ModDedElt
> S1!Mod!V!0;
(  )
> S2!Mod!Mod.2;
( M.1 )
> S3!Mod!(4*Mod.1);
( 4/1*M.1 0 )
```

61.9.2 Arithmetic with Elements

Basic arithmetic can be performed with elements of a module over a Dedekind domain.

`x + y: ModDedElt, ModDedElt -> ModDedElt`

The sum of the module elements.

`x - y: ModDedElt, ModDedElt -> ModDedElt`

The difference of the module elements.

`u * c: ModDedElt, RngElt -> ModDedElt`

`c * u: RngElt, ModDedElt -> ModDedElt`

The product of the module element u and the ring element c.

`u / c: ModDedElt, RngElt -> ModDedElt`

The product of u and $1/c$ if it lies in the parent module of u.

`I * u: RngOrdIdl, ModDedElt -> ModDed`

`I * u: RngFunOrdIdl, ModDedElt -> ModDed`

`u * I: ModDedElt, RngOrdIdl -> ModDed`

`u * I: ModDedElt, RngFunOrdIdl -> ModDed`

The module containing elements which are products of u and an element lying in I.

61.9.3 Other Functions on Elements

Elements of modules over a Dedekind domain can be tested for equality and represented as a sequence.

`x eq y: ModDedElt, ModDedElt -> Bool`

Return `true` if x and y are the same element of a module.

`IsZero(a): ModDedElt -> BoolElt`

Returns whether the module element a is zero.

`ElementToSequence(a): ModDedElt -> SeqEnum`

`Eltseq(a): ModDedElt -> SeqEnum`

The module element a expressed as a sequence.

61.10 Pseudo Matrices

A pseudo matrix, ie. an object of type `PMat`, is a sequence of ideals together with a matrix. Pseudo matrices arise naturally in the (computational) theory of finitely generated torsion free modules over Dedekind domains, they are a natural extension of ordinary matrices which should be thought of as pseudo matrices where all the ideals are generated by 1. Pseudo matrices are generally used to represent the module that is generated by the rows of the matrix scaled by the elements of the corresponding ideal. Thus, if the matrix is regular, a linear combination of the rows lies in the module if and only if the coefficient of the ith row is a member of the ith ideal. The ideals are therefore called coefficient ideals.

61.10.1 Construction of a Pseudo Matrix

`PseudoMatrix(I, m): [RngOrdFracIdl], MtrxSpcElt -> PMat`

`PseudoMatrix(I, m): [RngFunOrdIdl], MtrxSpcElt -> PMat`

Construct the pseudo matrix with coefficient ideals the elements of the sequence I and matrix m.

`PseudoMatrix(m): Mtrx[FldOrd] -> PMat`

`PseudoMatrix(m): Mtrx[RngOrd] -> PMat`

`PseudoMatrix(m): Mtrx[FldFunOrd] -> PMat`

`PseudoMatrix(m): Mtrx[RngFunOrd] -> PMat`

Construct the pseudo matrix with trivial coefficient ideals and the matrix m.

`PseudoMatrix(M): ModDed -> PMat`

`Generators: BoolElt` Default: `false`

Construct the pseudo matrix described by the pseudo basis of the module M. If `Generators` is `true` the pseudo matrix described by the pseudo generators of M is returned.

`AbsoluteBasis(M): ModDed -> SeqEnum`

Returns a basis of the module M over $\mathbb{Z}$ or $\mathbb{F}_q[t]$.

61.10.2 Elementary Functions

`CoefficientIdeals(P): PMat -> SeqEnum`

Return the coefficient ideals of the pseudo matrix P.

`Matrix(P): PMat -> Mtrx`

Return the matrix of the pseudo matrix P.

`Order(pm): PMat -> Rng`

Return the order the pseudo matrix pm is over.

`Dimension(pm): PMat -> RngIntElt`

The dimension of the pseudo matrix pm. This is the number of columns of the matrix.

`Length(pm): PMat -> RngIntElt`

The length or dimension of the pseudo matrix pm. This is the number of coefficient ideals of pm.

61.10.3 Basis of a Pseudo Matrix

`Basis(P): PMat -> SeqEnum`

Return a list of sequences of ring elements corresponding to the entries of each row of the matrix of the pseudo matrix P.

61.10.4 Predicates

`p1 eq p2: PMat, PMat -> BoolElt`

Return whether the pseudo matrices $p1$ and $p2$ are equal, that is, whether they have the same matrix and the same sequence of coefficient ideals.

61.10.5 Operations with Pseudo Matrices

`Transpose(P): PMat -> PMat`

Return the pseudo matrix whose coefficient ideals are the same as those of P but whose matrix is the transpose of the matrix of P. This function requires the matrix to be square.

`HermiteForm(X): PMat -> PMat, AlgMatElt`

Return the Hermite normal form H of the pseudo matrix X together with a regular transformation matrix such that the module generated by X is the same as the one generated by H and such that (for the matrix parts) $H = TX$ holds.

`VerticalJoin(X, Y): PMat, PMat -> PMat`

Return the pseudo matrix whose matrix is the vertical join of the matrices of the pseudo matrices X and Y with coefficient ideals the concatenation of those of X and Y.

`X meet Y: PMat, PMat -> PMat`

Return the intersection of the pseudo matrices X and Y.

`Module(X): PMat -> ModDed`

Return the module $\sum C_i * m_i$ where C_i are the coefficient ideals of the pseudo matrix X and m_i are the rows of the matrix of X.

`I * X: RngOrdFracIdl, PMat -> PMat`

`I * X: RngFunOrdIdl, PMat -> PMat`

`X * I: PMat, RngOrdFracIdl -> PMat`

`X * I: PMat, RngFunOrdIdl -> PMat`

The pseudo matrix whose coefficient ideals are those of the pseudo matrix X multiplied by I and whose matrix is the matrix of X.

Chapter 62

Chain Complexes

62.1 Complexes of Modules

Complexes of modules are a fundamental object in homological algebra. Magma supports the type `ModCpx` representing a complex of modules. Conceptually, a complex is an infinite sequence of modules, indexed by integers, with maps between successive modules such that the composition of any two maps is zero. Complexes are often written

$$\ldots \xrightarrow{f_{n+1}} M_n \xrightarrow{f_n} M_{n-1} \xrightarrow{f_{n-1}} M_{n-2} \xrightarrow{f_{n-2}} \ldots$$

where the map f_n has domain M_n and codomain M_{n-1}. The indices on the modules and maps decrease to the right. In practice, Magma requires all but a finite number of the modules and maps to be zero.

The homomorphism from M_n to M_{n-1} in the complex is the n^{th} boundary map of the complex. The homology of the complex in degree n is the quotient of the kernel of the n^{th} boundary map by the image of the boundary map of degree $n-1$. Magma computes the homology only if both boundary maps are defined. A complex is said to be *exact* if the image of each map is equal to the kernel of the next.

Currently, there are two types of modules over which complexes are supported:

(a) Modules over a basic algebra A (see Chapter Basic Algebras);

(b) Modules over a multivariate polynomial ring over a field (see Chapter Modules over Multivariate Rings and in particular the function `FreeResolution(M): ModMPol → ModCpx, ModMPolHom`). Most of the functions in this chapter work for either type of module but exceptions are noted.

62.1.1 Creation

`Complex(L, d): List, RngIntElt -> ModCpx`

Given a list L of maps between successive A-modules create the corresponding complex. The last term of the complex has degree d. This function returns an error if the maps don't actually form a complex.

`Complex(f, d): Map, RngIntElt -> ModCpx`

Given a map f between A-modules M and N, form the complex consisting of the two term complex whose only map is f. The term N is in degree d.

`ZeroComplex(A, m, n): AlgBas, RngIntElt, RngIntElt -> ModCpx`

Given a basic algebra A and integers m and n such that $m > n$, create a complex of modules over the basic algebra A, starting with a term of degree m and ending with a term of degree n, where all the modules and maps are zero.

`Dual(C): ModCpx -> ModCpx`

`Dual(C, n): ModCpx, RngIntElt -> ModCpx`

The dual of the complex C over a basic algebra A as a complex over the opposite algebra of A. If an integer n is supplied then the last term of the dual complex is in degree n. Otherwise, the last term of the dual complex is in degree 0.

62.1.2 Subcomplexes and Quotient Complexes

Let C be a complex of A-modules, $M_m, \ldots, M_n$. A subcomplex S of C is a complex whose terms are submodules of the terms of C and whose maps are the restrictions of the maps of C to the terms of S.

`sub< C | Q >: ModCpx, SeqEnum[ModAlg] -> ModCpx, MapChn`

`sub< C | L >: ModCpx, List -> ModCpx, MapChn`

Given a complex C and a sequence Q of submodules of the terms of C, returns the smallest subcomplex whose terms contain the modules in Q. The input can also be a list L of sequences of elements of the successive terms of C. In this case the submodules generated by the elements is computed. The function also returns the chain map giving the inclusion of the subcomplex in C.

`RandomSubcomplex(C, Q): ModCpx, SeqEnum -> ModCpx, MapChn`

Given a chain complex C over a basic algebra in degrees a to $a-t+1$ and a sequence $Q = [q_1, \ldots, q_t]$ of natural numbers, the function creates the minimal chain complex whose term in degree $a-i+1$ is a submodule generated by q_i random elements of the term in degree $a-i+1$ of C. The function also returns the chain map that is the inclusion of the subcomplex into C.

`quo< C | D >: ModCpx, ModCpx -> ModCpx`

`quo< C | S >: ModCpx, SeqEnum[ModAlg] -> ModCpx`

`quo< C | L >: ModCpx, List -> ModCpx`

Given a complex C and a subcomplex D of C, returns the quotient complex C/D together with the natural quotient map. If given as a sequence S of submodules of the terms of C or a list L of sequences of elements of the terms of C, then the submodule generated by S or L is created and the quotient computed. The function also returns the chain map of C on the quotient.

62.1.3 Access Functions

`Degrees(C): ModCpx -> RngIntElt`

Returns the first and last degrees of the defined terms of the complex C.

`Algebra(C): ModCpx -> AlgBas`

Given a complex C over a basic algebra A, this function returns the algebra A.

`BoundaryMap(C, n): ModCpx, RngIntElt -> ModMatRngElt`

Returns the boundary map of the complex C from the term of degree n to the term of degree $n-1$.

`BoundaryMaps(C): ModCpx -> List`

The list of boundary maps of the complex C.

`DimensionsOfHomology(C): ModCpx -> SeqEnum`

Returns the list of the dimensions of the homology groups of the complex C.

`DimensionOfHomology(C, n): ModCpx, RngIntElt -> RngIntElt`

Returns the dimension of the homology group of the complex C in degree n.

`DimensionsOfTerms(C): ModCpx -> SeqEnum`

Returns the list of the dimensions of the terms of the complex C.

`Term(C, n): ModCpx, RngIntElt -> ModAlg`

The module in the complex C in degree n.

`Terms(C): ModCpx -> SeqEnum`

The sequence of terms of the complex C.

62.1.4 Elementary Operations

`DirectSum(C, D): ModCpx, ModCpx -> ModCpx`

Returns the direct sum of the complex C and the complex D.

`Homology(C): ModCpx -> SeqEnum`

`HomologyOfChainComplex(C): ModCpx -> SeqEnum`

The sequence of homology groups of the complex C as a sequence of modules.

`Homology(C, n): ModCpx, RngIntElt -> SeqEnum`

Returns the homology group in degree n of the complex C, as an A-module.

`Prune(C): ModCpx -> ModCpx`

Returns the complex that consists of the terms of C with the right end term (term of lowest degree) removed.

`Prune(C, n): ModCpx, RngIntElt -> ModCpx`

Returns the complex that consists of the terms of C with n terms removed from the right end.

`Preprune(C): ModCpx -> ModCpx`

Returns the complex that consists of the terms of C with the left end term (term of highest degree) removed.

`Preprune(C, n): ModCpx, RngIntElt -> ModCpx`

Returns the complex that consists of the terms of C with n terms removed from the left end.

`Shift(C, n): ModCpx, RngIntElt -> ModCpx`

Given the complex C in degrees r to s, returns the complex in shifted degrees $r + n$ to $s + n$. The integer n may be either positive or negative. The maps in the complex are all multiplied by the scalar $(-1)^n$.

`ShiftToDegreeZero(C): ModCpx -> ModCpx`

Given the complex C, returns the shift of C so that the last term, the term of lowest degree, is in degree 0.

`Splice(C, D): ModCpx, ModCpx -> ModCpx`

Given two complexes C and D over the same basic algebra, such that the end term of C coincides with the initial term of D, form the complex that corresponds to the concatenation of C and D (as sequences of maps) This function checks that the resulting sequence of maps forms a complex. The degrees of complex D remain unchanged while the degrees of the terms in complex C are changed to fit.

`Splice(C, D, f): ModCpx, ModCpx, ModMatRngElt -> ModCpx`

The splice of the complex C with the complex D along the map f from the last term of C to the first term of D. The degree of the last term of the splice is the same as the degree of the last term of the complex D.

62.1.5 Extensions

The following are elementary operations related to extending complexes

`LeftExactExtension(C): ModCpx -> ModCpx`

Given a complex C of modules over a basic algebra, returns the complex of length one greater that is obtained by adjoining the inclusion map from the kernel of the boundary map in highest degree to the term of highest degree in C.

`RightExactExtension(C): ModCpx -> ModCpx`

Given a complex C of modules over a basic algebra, returns the complex of length one greater that is obtained by adjoining the quotient map to the cokernel of the boundary map of lowest degree from the term of lowest degree in C.

`ExactExtension(C): ModCpx -> ModCpx`

Returns the left and right exact extensions of the complex C.

`LeftZeroExtension(C): ModCpx -> ModCpx`

`LeftZeroExtension(C, n): ModCpx, RngIntElt -> ModCpx`

Given a complex C of modules over a basic algebra, returns the complex of length one greater that is obtained by adjoining the zero map from the zero module to the term of highest degree in the complex. If a natural number n is included in the input then the operation is performed n times.

`RightZeroExtension(C): ModCpx -> ModCpx`

`RightZeroExtension(C, n): ModCpx, RngIntElt -> ModCpx`

Given a complex C of modules over a basic algebra, returns the complex of length one greater that is obtained by adjoining the zero map to the zero module from the term of lowest degree in the complex. If a natural number n is included in the input then the operation is performed n times.

`ZeroExtension(C): ModCpx -> ModCpx`

Returns the left and right zero extensions of the complex C.

`EqualizeDegrees(C, D): ModCpx, ModCpx -> ModCpx, ModCpx`

`EqualizeDegrees(Q): SeqEnum -> SeqEnum`

Given complexes C and D over the same algebra, the function returns the complexes obtained by taking zero extensions of C and D, if necessary, so that both complexes have the same degrees. The input can also be given as a sequence Q of complexes, in which case the function returns the sequence of complexes obtained by taking zero extensions of the elements of Q, if necessary, until all of the elements of the sequence have the same degrees.

`EqualizeDegrees(C, D, n): ModCpx, ModCpx, RngIntElt -> ModCpx, ModCpx`

The two complexes, C and D, with zero extension sufficient that the first and the shift by degree n of the second have the same degrees.

62.1.6 Predicates

The following functions return a Boolean value.

`IsExact(C): ModComplex -> BoolElt`

Returns `true` if and only if the complex C is an exact sequence, i.e. the image of each map in C is equal to the kernel of the succeeding map (with the obvious exceptions of the first and last maps of the complex). If the complex has only two terms then this is true trivially.

`IsShortExactSequence(C): ModCpx -> BoolElt, RngIntElt`

Returns `true` if the complex C consists of a short exact sequence together with other terms that are zero. The function returns also the degrees of the complex of nonzero terms.

`IsExact(C, n): ModCpx, RngIntElt -> BoolElt`

Returns `true` if and only if the complex C is an exact sequence in degree n, i.e. the image of the map in C into the term of degree n is equal to the kernel of the succeeding map.

`IsZeroComplex(C): ModCpx -> BoolElt`

Returns `true` if and only if the complex C is composed entirely of zero modules and maps.

`IsZeroMap(C, n): ModCpx, RngIntElt -> BoolElt`

Returns `true` if and only if the boundary map in degree n of the complex C is the zero map.

`IsZeroTerm(C, n): ModCpx, RngIntElt -> BoolElt`

Returns `true` if and only if the term in degree n of the complex C is the zero object.

Example: Complexes over a Quiver Algebra `(ex-bb43ae)`

We construct the quiver algebra of a quiver with three nodes and three arrows going from node 1 to node 2, from 2 to 1, and 2 to 3. The relation is that $(ab)^3a = 0$ where a is the first arrow and b is the second. Then we construct projective and injective resolutions of the first simple module.

```
> ff := GF(8);
> FA<e1,e2,e3,a,b,c> := FreeAlgebra(ff,6);
> rrr := [a*b*a*b*a*b*a];
> D := BasicAlgebra(FA,rrr,3,[<1,2>,<2,1>,<2,3>]);
> D;
Basic algebra of dimension 23 over GF(2^3)
Number of projective modules: 3
Number of generators: 6
> DimensionsOfProjectiveModules(D);
[ 10, 12, 1 ]
> DimensionsOfInjectiveModules(D);
[ 8, 7, 8 ]
```

Here is the opposite algebra:

```
> OD := OppositeAlgebra(D);
reverse trees
> OD;
Basic algebra of dimension 23 over GF(2^3)
Number of projective modules: 3
Number of generators: 6
> s1 := SimpleModule(D,1);
> P,mu := ProjectiveResolution(s1,7);
> P;
Basic algebra complex with terms of degree 7 down to 0
Dimensions of terms: 12 12 12 12 12 12 12 10
> Q,nu := InjectiveResolution(s1,7);
> Q;
Basic algebra complex with terms of degree 0 down to -7
Dimensions of terms: 8 7 0 0 0 0 0 0
```

Note that the projective and injective resolutions are complexes with the appropriate augmentation and coaugmentation maps. First we form the two term complex whose boundary map is the composition of the augmentation and coaugmentation maps.

```
> theta := MapToMatrix(hom<Term(P,0)-> Term(Q,0)|mu*nu>);
> E := Complex(theta,0);
```

Then we splice all of this together.

```
> R := Splice(P,E);
> S := Splice(R,Q);
> S;
Basic algebra complex with terms of degree 8 down to -7
Dimensions of terms: 12 12 12 12 12 12 12 10 8 7 0 0 0 0 0 0
```

62.2 Chain Maps

A chain map from complex C to complex D is a sequence of homomorphisms between the terms of C and the terms of D such that the maps commute with the boundary homomorphisms on the two complexes. The chain map has degree zero if the map on C_n has it's image in D_n. Otherwise the degree of the chain map expresses the extent to which the degrees of the terms are raised or lowered by the chain map.

Chain maps do not have to be defined in every degree for which either its domain or codomain is defined. On the other hand it must be defined wherever possible, i.e. for any degree i for which the term of the domain is defined and for which the term of the codomain is defined in degree $i+n$ where n is the degree of the chain map.

The freedom of definition can cause problems with constructions such as the kernel or cokernel. That is, the kernel of a chain complex may not be a complex. For this reason we allow the Kernel and Cokernel functions to work only for proper chain maps where "proper" is defined as follows. Suppose C is a chain complex in degrees a to b and D is a complex in degrees u to v. A chain map in degree n from C to D is proper provided $u \geq a + n$ and $v \geq b + n$.

62.2.1 Creation

`ChainMap(Q, C, D, n): SeqEnum, ModCpx, ModCpx, RngIntElt -> MapChn`

Returns the chain map given by the sequence of maps in Q. Each element in Q is a map from a term of the complex C to a corresponding term of the complex D. The integer n is the degree of the chain map. This means that for any i the term in degree i of C is mapped to the term of degree $n + i$ of D. There should be a map defined for any degree i in the range of degrees of C such that there is a corresponding term in degree $n + i$ for the complex D.

`ZeroChainMap(C, D): ModCpx, ModCpx -> MapChn`

Given chain complexes C and D, construct the chain map from C to D all of whose terms are zero.

62.2.2 Access Functions

`Degree(f): MapChn -> RngIntElt`

Given a chain map f, returns the degree of f. Note that $f : C \to D$ has degree n if it takes the term in degree i of the complex C to the term in degree $i + n$ of the complex D.

`ModuleMap(f, n): MapChn, RngIntElt -> ModMatRngElt`

Given a chain map $f : C \to D$ and an integer n, this function returns the map from the term in degree n of the complex C to the term in degree $n +$ `Degree`(f) of the complex D.

`Kernel(f): MapChn -> ModCpx, MapChn`

Returns the kernel complex of f and the inclusion of the kernel in the domain of f. The function is only defined for proper chain maps. If the chain map is not defined for a certain term of the domain then that term is equal to the term of the kernel complex. That is, if f is not defined on a term then the functions acts as if f were the zero map on that term.

`Cokernel(f): MapChn -> ModCpx, MapChn`

Returns the cokernel complex of f and the projection of the cokernel onto the codomain of f. The function is only defined for proper chain maps. If the chain map is not defined for a certain term of the codomain then that term is equal to the term of the cokernel complex. That is, if f is not defined on a term then the function acts as if f were the zero map on that term.

`Image(f): MapChn -> ModCpx, MapChn, MapChn`

Returns the image complex of f and the inclusion of the image in the codomain of f and the projection of the domain of f on to the image. The function is only defined for proper chain maps.

62.2.3 Elementary Operations

`f + g: MapChn, MapChn -> MapChn`

The sum of two chain maps with the same domain and codomain.

`a * g: RngElt, MapChn -> MapChn`

The product of the chain map g by the scalar a in the base ring of the algebra.

`f * g: MapChn, MapChn -> MapChn`

The composition of the two chain maps f and g.

62.2.4 Predicates

The following functions return a Boolean value.

`IsSurjective(f): MapChn -> BoolElt`
: Returns `true` if the chain map f is a surjection in every degree, `false` otherwise.

`IsInjective(f): MapChn -> BoolElt`
: Returns `true` if the chain map f is an injection in every degree, `false` otherwise.

`IsZero(f): MapChn -> BoolElt`
: Returns `true` if the chain map f is zero in every degree, `false` otherwise.

`IsIsomorphism(f): MapChn -> BoolElt`
: Returns `true` if the chain map f is an isomorphism of chain complexes, `false` otherwise.

`IsShortExactSequence(f, g): MapChn, MapChn -> BoolElt`
: Returns `true` if the sequence of chain complexes, $0 \to Domain(f) \to Domain(g) \to Codomain(g) \to 0$, where the internal maps are f and g, is exact.

`IsChainMap(L, C, D, n): List, ModCpx, ModCpx, RngIntElt -> BoolElt`
: Returns `true` if the list of maps L from the terms of complex C to the terms of the complex D is a chain map of degree n, i. e. it has the right length and the diagram commutes.

`IsChainMap(f): MapChn -> BoolElt`
: Returns `true` if the supposed chain map f really is a chain map, i. e. the diagrams commute.

`IsProperChainMap(f): MapChn -> BoolElt`
: Returns `true` if the chain map f is a proper chain map, a necessary condition for taking kernel and cokernel.

`HasDefinedModuleMap(C, n): ModCpx, RngIntElt -> BoolElt`
: Returns `true` if the module map in degree n of the complex C is defined.

Example: Chain Maps over a Basic Algebra `(ex-f4e1ae)`

We from the basic algebra of the direct product of a cyclic group of order 3 with symmetric group on three letters over the field with three elements.

```
> FA<e1,e2,a,b> := FreeAlgebra(GF(3),4);
> MM:= [e1 +e2 - FA!1, a*b*a, b*a*b];
> BS3 := BasicAlgebra(FA, MM, 2, [<1,2>,<2,1>]);
> gg := CyclicGroup(3);
> BC3 := BasicAlgebra(gg,GF(3));
> A := TensorProduct(BS3,BC3);
> A;
Basic algebra of dimension 18 over GF(3)
Number of projective modules: 2
Number of generators: 6
```

Now we want the projective resolution of the second simple module as a complex. This we will manipulate and take a somewhat random complex.

```
> PR := ProjectiveResolution(SimpleModule(A,2),12);
> PR;
Basic algebra complex with terms of degree 12 down to 0
Dimensions of terms: 117 108 99 90 81 72 63 54 45 36 27 18 9
> PR := Prune(PR);
> PR := Prune(PR);
> PR := Prune(PR);
> PR;
```

(continues on next page)

(continued from previous page)

```
Basic algebra complex with terms of degree 12 down to 3
Dimensions of terms: 117 108 99 90 81 72 63 54 45 36
> PR := Prune(PR);
> PR := Prune(PR);
> PR;
Basic algebra complex with terms of degree 12 down to 5
Dimensions of terms: 117 108 99 90 81 72 63 54
> PR := ZeroExtension(PR);
> PR;
Basic algebra complex with terms of degree 13 down to 4
Dimensions of terms: 0 117 108 99 90 81 72 63 54 0
> PR := Shift(PR,-4);
> PR;
Basic algebra complex with terms of degree 9 down to 0
Dimensions of terms: 0 117 108 99 90 81 72 63 54 0
> S := [* *];
> for i := 1 to 10 do
>    S[i] := [Random(Term(PR,10-i)),Random(Term(PR,10-i))];
> end for;
> C,mu := Subcomplex(PR,S);
> C;
Basic algebra complex with terms of degree 9 down to 0
Dimensions of terms: 0 36 67 64 62 63 58 54 51 0
> Homology(C);
[
    AModule of dimension 4 over GF(3),
    AModule of dimension 6 over GF(3),
    AModule of dimension 5 over GF(3),
    AModule of dimension 3 over GF(3),
    AModule of dimension 2 over GF(3),
    AModule of dimension 1 over GF(3),
    AModule of dimension 4 over GF(3),
    AModule of dimension 26 over GF(3)
]
[
    Mapping from: AModule of dimension 4 over GF(3) to AModule of dimension 4
    over GF(3),
    Mapping from: AModule of dimension 38 over GF(3) to AModule of dimension 6
    over GF(3),
    Mapping from: AModule of dimension 34 over GF(3) to AModule of dimension 5
    over GF(3),
    Mapping from: AModule of dimension 33 over GF(3) to AModule of dimension 3
    over GF(3),
    Mapping from: AModule of dimension 31 over GF(3) to AModule of dimension 2
    over GF(3),
    Mapping from: AModule of dimension 33 over GF(3) to AModule of dimension 1
    over GF(3),
    Mapping from: AModule of dimension 29 over GF(3) to AModule of dimension 4
    over GF(3),
    Mapping from: AModule of dimension 51 over GF(3) to AModule of dimension 26
    over GF(3)
]
> D := Cokernel(mu);
> D;
Basic algebra complex with terms of degree 9 down to 0
Dimensions of terms: 0 81 41 35 28 18 14 9 3 0
```

(continues on next page)

(continued from previous page)

```
> Homology(D);
[
    AModule of dimension 64 over GF(3)
    AModule of dimension 5 over GF(3),
    AModule of dimension 3 over GF(3),
    AModule of dimension 2 over GF(3),
    AModule of dimension 1 over GF(3),
    AModule of dimension 4 over GF(3),
    AModule of dimension 3 over GF(3),
    AModule of dimension 3 over GF(3)
]
[
    Mapping from: AModule of dimension 64 over GF(3) to AModule of dimension 64
    over GF(3),
    Mapping from: AModule of dimension 22 over GF(3) to AModule of dimension 5
    over GF(3),
    Mapping from: AModule of dimension 22 over GF(3) to AModule of dimension 3
    over GF(3),
    Mapping from: AModule of dimension 15 over GF(3) to AModule of dimension 2
    over GF(3),
    Mapping from: AModule of dimension 14 over GF(3) to AModule of dimension 1
    over GF(3),
    Mapping from: AModule of dimension 8 over GF(3) to AModule of dimension 4
    over GF(3),
    Mapping from: AModule of dimension 9 over GF(3) to AModule of dimension 3
    over GF(3),
    Mapping from: AModule of dimension 3 over GF(3) to AModule of dimension 3
    over GF(3)
]
```

62.2.5 Maps on Homology

InducedMapOnHomology(f, n): MapChn, RngIntElt -> ModTupFldElt

The homomorphism induced on homology by the chain map f in degree n.

ConnectingHomomorphism(f, g, n): MapChn, MapChn, RngIntElt -> ModMatRngElt

The connecting homomorphism in degree n of the short exact sequence of chain complexes given by the chain maps f and g.

LongExactSequenceOnHomology(f, g): MapChn, MapChn -> ModCpx

The long exact sequence on homology for the exact sequence of complexes given by the chain maps f and g as a chain complex with the homology group in degree i for the Cokernel of the complex C appearing in degree $3i$.

Example: Long Exact Sequence in Homology (ex-4cd4be)

We create a basic algebra with two simple modules and four nonidempotent generators. The relations are given in the sequence `rrr`.

```
> ff := GF(3);
> p := Characteristic(ff);
> FA<e1, e2, y, x, a, b> := FreeAlgebra(ff,6);
> rrr := [y^p,x^p,x*y+y*x,x*a*b-a*b*x,y*a*b-a*b*y,(b*a)^2,(b*a)^2];
> A := BasicAlgebra(FA,rrr,2,[<1,1>,<1,1>,<1,2>,<2,1>]);
> A;
Basic algebra of dimension 81 over GF(3)
Number of projective modules: 2
```

(continues on next page)

(continued from previous page)

```
Number of generators: 6
> DimensionsOfProjectiveModules(A);
[ 45, 36 ]
```

Now we generate the simple modules and their projective covers. We want a module PP that is the direct sum of two copies of each of the projective indecomposable module.

```
> S1 := SimpleModule(A,1);
> PP := ProjectiveModule(A,[2,2]);
```

Now we are going to create a complex all of whose terms are isomorphic to PP. To make it interesting we will insist that the first homomorphism have its image in the second radical layer of PP.

```
> J1 := JacobsonRadical(PP);
> theta1 := Morphism(J1,PP);
> J2 := JacobsonRadical(J1);
> theta2 := Morphism(J2,J1);
> theta := theta2*theta1;
> HomPJ := AHom(PP, J2);
> HomPJ;
KMatrixSpace of 162 by 150 matrices and dimension 300 over GF(3)
> gamma := Random(HomPJ)*theta;
> LL := [* gamma *];
```

So we now have the first boundary map of the complex. The other boundary maps are generated randomly.

```
> for i := 1 to 15 do
>     K, phi := Kernel(gamma);
>     HomPK := AHom(PP,K);
>     gamma := Random(HomPK)*MapToMatrix(phi);
>     LL := [* gamma *] cat LL;
> end for;
```

Now we make the list into a chain complex.

```
> C := Complex(LL,0);
> C;
Basic algebra complex with terms of degree 16 down to 0
Dimensions of terms: 162 162 162 162 162 162 162 162 162 162 162 162 162 162
162 162 162
> DimensionsOfHomology(C);
[ 30, 23, 34, 32, 35, 32, 35, 42, 30, 34, 32, 33, 29, 42, 21 ]

Now we will get a random subcomplex.

> a,b := Degrees(C);
> S, mu := RandomSubcomplex(C,[2: i in [1 .. a-b+1]]);
> S;
Basic algebra complex with terms of degree 16 down to 0
Dimensions of terms: 162 155 162 162 162 154 142 162 162 148 154 162 162 152
155 152 162
> DimensionsOfHomology(S);
[ 27, 27, 34, 32, 35, 33, 48, 42, 28, 44, 38, 33, 29, 48, 24 ]
```

Now take the quotient.

```
> Q,nu := quo<C|S>;
> Q;
Basic algebra complex with terms of degree 16 down to 0
Dimensions of terms: 0 7 0 0 0 8 20 0 0 14 8 0 0 10 7 10 0
> DimensionsOfHomology(Q);
[ 7, 0, 0, 0, 6, 18, 0, 0, 12, 6, 0, 0, 8, 5, 10 ]
```

Now we check to see if this is a short exact sequence of chain maps.

```
> IsShortExactSequence(mu,nu);
true
> lll := LongExactSequenceOnHomology(mu,nu);
Basic algebra complex with terms of degree 47 down to 3
Dimensions of terms: 27 30 7 27 23 0 34 34 0 32 32 0 35 35 6 33 32 18 48 35 0
42 42 0 28 30 12 44 34 6 38 32 0 33 33 0 29 29 8 48 42 5 24 21 10
> IsExact(lll);
true
```

Chapter 63

Multilinear Algebra

63.1 Introduction

This chapter discusses Magma operations with multilinear algebra. The notation follows [FMW19] closely and can be viewed as companion literature. Wherever possible we follow the conventions in use in physics [Wey50a, Chapter V], differential geometry [Lee13, Chapter 10], and algebra [Lan12]. Necessary categorical formalism is drawn largely from [Wil13a]. The package covers:

(1) Tensors and multilinear functions and their associated groups and algebras.

(2) Spaces of tensors.

(3) Categories of tensors and tensor spaces.

(4) Linear invariants of tensors.

63.1.1 Overview

We set up some notation used throughout. Define K to be a commutative ring (typically a field), ν to be a nonnegative integer, and a sequence of right K-modules $U_0, \ldots, U_\nu$. Set $[\nu] = \{0, \ldots, \nu\}$, and for a subset $A \subset [\nu]$, let $\bar{A} = [\nu] - A$. Let $\oslash_{a\in[\nu]} U_a$ denote the set of multilinear maps $U_\nu \times \cdots U_1 \rightarrowtail U_0$. Here, we use $\rightarrowtail$ exclusively for multilinear maps.

A *tensor space* T is a K-module equipped with a monomorphism

$$\langle\cdot| : T \hookrightarrow \oslash_{a\in[\nu]} U_a.$$

Tensors are elements of T, $(U_\nu, \ldots, U_0)$ is the *frame*, and $\nu + 1$ is the valence. In Dirac styled $\langle t|$ "bra t" and $|u\rangle$ "ket u" notation, we evaluate $\langle t| \in \oslash_{a\in[\nu]} U_a$ on inputs $|u\rangle = |u_\nu, \ldots, u_1\rangle \in U_\nu \times \cdots \times U_1$ denote $\langle t|u\rangle$ or $\langle t|u_\nu, \ldots, u_1\rangle$.

The module U_0 is reserved for the codomain which in turn makes reverse indexing the simplest notation. Every tensor in Magma is treated as an element of a tensor space which is, by default, a universal tensor space:

$$\oslash_{a\in[\nu]} U_a = \hom_K(U_\nu, \ldots, \hom_K(U_1, U_0)\cdots).$$

In Magma, the tensor space determines the associated multilinear function of a given tensor T. Evaluation of T mimics the application of a map $U_v \times \cdots \times U_1 \to U_0$, for instance,

```
> <u_v, ..., u_1> @ T;
```

Special attention is given to bilinear maps $* : U_2 \times U_1 \rightarrowtail U_0$ including the ability to use infix notation `u\_2 * u\_1`. Tensor spaces have type `TenSpc` and behave like modules in that they have subspaces and quotient spaces. Tensors have type `TenSpcElt` and behave in a similar way to Magma matrices.

A library of commonly used exceptional tensors is provided. These include octonion algebras and exceptional Jordan algebras.

Tensor categories, type `TenCat`, provide Magma with the information needed to interpret the contents of a tensor space. For example, one tensor category may treat a $(d \times d)$-matrix F over a field K as a linear map $K^d \to K^d$, while another assigns the same matrix to a bilinear form $K^d \times K^d \rightarrowtail K$. Functors are provided to change tensor categories and to define standard categories.

63.1.2 Verbose Printing

The string `"TensorSpace"` and an integer, either 0 or 1, is now acceptable input to `SetVerbose`. Level 0 turns off printing, and level 1 turns it on.

Example: Verbose Printing for Tensor Spaces (ex-18db1e)

We demonstrate the verbose printing. Currently, we only have verbose printing when we solve linear systems. To turn on all the printing statements, set `"TensorSpace"` to 1.

```
> SetVerbose("TensorSpace", 1);
>
> t := RandomTensor(GF(2), [32, 32, 32]);
> D := DerivationAlgebra(t);
Construting a 3072 by 32768 matrix over Finite field of size 2.
Adding in possible fusion data.
Computing the nullspace of a 3072 by 32768 matrix.
```

`T : Magma;`

`S : Magma;`

`C : Magma;`

Prints strings that can be executed in Magma to construct the given object. If T is a black-box tensor where structure constants cannot be computed, then the printed string is the same as `print T`, which cannot be executed to construct T.

Example: Printing Tensors to Strings (ex-525a77)

We first construct a tensor category, tensor space, and a tensor contained in the tensor space.

```
> C := TensorCategory([1, 1, 0], {{0}, {1,2}});
> C;
Tensor category of valence 3 (->,->,==) ({ 0 },{ 1, 2 })
>
> T := KTensorSpace(GF(2), [10, 10, 2], C);
> T;
Tensor space of dimension 200 over GF(2) with valence 3
U2 : Full Vector space of degree 10 over GF(2)
U1 : Full Vector space of degree 10 over GF(2)
U0 : Full Vector space of degree 2 over GF(2)
>
> t := T![1..200];
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 10 over GF(2)
U1 : Full Vector space of degree 10 over GF(2)
U0 : Full Vector space of degree 2 over GF(2)
```

Now we will print at "Magma" level each of the objects we created.

```
> C:Magma;
TensorCategory([ 1, 1, 0 ], { PowerSet(IntegerRing()) |
{ IntegerRing() | 0 },
```

(continues on next page)

(continued from previous page)

```
{ IntegerRing() | 1, 2 }
})
>
> T:Magma;
TensorSpace([*VectorSpace(GF(2), 10), VectorSpace(GF(2), 10),
VectorSpace(GF(2),2)*], TensorCategory([ 1, 1, 0 ], {
PowerSet(IntegerRing()) |
{ IntegerRing() | 0 },
{ IntegerRing() | 1, 2 }
}))
>
> t:Magma;
Tensor(GF(2), [ 10, 10, 2 ], [ GF(2) | 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
0, 1, 0, 1, 0 ], TensorCategory([ 1, 1, 0 ], { PowerSet(IntegerRing()) |
{ IntegerRing() | 0 },
{ IntegerRing() | 1, 2 }
}))
```

63.2 Tensors

A tensor has the following information associated with it:-

- A commutative ring K of coefficients.
- A valence ν indicating the number of variables to include in its associated multilinear map.
- A list $[U_\nu, \ldots, U_0]$ of K-modules called the *frame*.
- A function $U_\nu \times \cdots \times U_1 \to U_0$ that is K-linear in each U_i.

Tensors have type `TenSpcElt` and are formally elements of a tensor space (type `TenSpc`). By default a tensor's parent space is a universal tensor space:

$$\hom_K(U_v, \ldots, \hom_K(U_1, U_0) \cdots) \cong \hom_K(U_v \otimes_K \cdots \otimes_K U_1, U_0).$$

The left hand module is primarily used because it avoids the need to work with the equivalence classes of a tensor product. Operations such as linear combinations of tensors take place within a tensor space. A tensor space has attributes such as coefficients, valence, and frame.

When necessary, the user may further direct the operations on tensors to appropriate tensor categories (type `TenCat`). For instance covariant and contravariant variables can be specified and symmetry conditions can be imposed. If no tensor category is prescribed then a default tensor category is selected based on the method of creation.

63.2.1 Creating Tensors

Black-box Tensors

A user can specify a tensor by a black-box function that evaluates the required multilinear map.

`Tensor(S, F): SeqEnum, UserProgram -> TenSpcElt, List`

`Tensor(S, F): List, UserProgram -> TenSpcElt, List`

`Tensor(S, F, Cat): SeqEnum, UserProgram, TenCat -> TenSpcElt, List`

`Tensor(S, F, Cat): List, UserProgram, TenCat -> TenSpcElt, List`

Returns a tensor T and a list of maps from the given frame into vector spaces of the returned frame. Note that T is a tensor over vector spaces—essentially forgetting all other structure. The last entry of S is assumed to be the codomain of the multilinear map. The user-defined function F should take as input a tuple of elements of the domain and return an element of the codomain. If no tensor category is provided, then Albert's homotopism category is used.

Example: Black-Box Dot Product Tensor (ex-cdbf81)

We demonstrate the black-box constructions by first constructing the dot product $\cdot : \mathbb{Q}^4 \times \mathbb{Q}^4 \rightarrowtail \mathbb{Q}$. The function used to evaluate our black-box tensor, `Dot`, must take exactly one argument. The argument will be a `Tup`, an element of the Cartesian product $U_\nu \times \cdots \times U_1$. Note that `x[i]` is the ith entry in the tuple and not the ith coordinate.

```
> Q := Rationals();
> U := VectorSpace(Q, 4);
> V := VectorSpace(Q, 4);
> W := VectorSpace(Q, 1);  // Vector space, not the field Q
> Dot := func< x | x[1]*Matrix(4, 1, Eltseq(x[2])) >;
```

Now we will construct the tensor from the data above. The first object returned is the tensor, and the second is a list of maps, mapping the given frame into the vector space frame. In this example, since the given frame consists of vector spaces, these maps are trivial. Note that the list of maps are not needed to work with the given tensor, we will demonstrate this later.

```
> Tensor([U, V, W], Dot);
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 1 over Rational Field
[*
    Mapping from: ModTupFld: U to ModTupFld: U given by a rule,
    Mapping from: ModTupFld: U to ModTupFld: U given by a rule,
    Mapping from: ModTupFld: W to ModTupFld: W given by a rule
*]
```

We will provide a tensor category for the dot product tensor, so that the returned tensor is not in the default homotopism category. We will use instead the $\{2,1\}$-adjoint category. While the returned tensor prints out the same as above, it does indeed live in a universe, see the section on Tensor Categories (`TenCat`).

```
> Cat := AdjointCategory(3, 2, 1);
> Cat;
Tensor category of valence 3 (<-,->,==) ({ 1 },{ 2 },{ 0 })
>
> t := Tensor([U, V, W], Dot, Cat);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 1 over Rational Field
>
> TensorCategory(t);
Tensor category of valence 3 (<-,->,==) ({ 1 },{ 2 },{ 0 })
```

Example: Black-Box Cross Product (ex-82ef8c)

We will construct the cross product $\times : \mathbb{R}^3 \times \mathbb{R}^3 \rightarrowtail \mathbb{R}^3$ and verify that $\mathbf{i} \times \mathbf{j} = \mathbf{k}$. However, to do this test, we will input integer sequences (specifically `[RngIntElt]`), and we will still be able to evaluate.

```
> K := RealField(5);
> V := VectorSpace(K, 3);
> CP := function(x)
>   return V![x[1][2]*x[2][3] - x[1][3]*x[2][2],
>     x[1][3]*x[2][1] - x[1][1]*x[2][3],
>     x[1][1]*x[2][2] - x[1][2]*x[2][1] ];
> end function;
> t := Tensor([V, V, V], CP);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 3 over Real field of precision 5
U1 : Full Vector space of degree 3 over Real field of precision 5
U0 : Full Vector space of degree 3 over Real field of precision 5
>
> // test that i x j = k
> <[1,0,0], [0,1,0]> @ t eq V.3;
true
```

Tensor(D, C, F): SeqEnum, Any, UserProgram -> TenSpcElt, List

Tensor(D, C, F): List, Any, UserProgram -> TenSpcElt, List

Tensor(D, C, F, Cat): SeqEnum, Any, UserProgram, TenCat -> TenSpcElt, List

Tensor(D, C, F, Cat): List, Any, UserProgram, TenCat -> TenSpcElt, List

Returns a tensor T and a list of maps from the given frame into vector spaces of the returned frame. Note that T is a tensor over vector spaces—essentially forgetting all other structure. The user-defined function F should take as input a tuple of elements of D and return an element of C. If no tensor category is provided, then Albert's homotopism category is used.

Example: Black-Box Triple Product (ex-e475ab)

Tensors make it easy to create algebras that do not fit into traditional categories, such as algebras with triple products. Here, we create a triple product $\langle\,\rangle : \mathrm{Mat}_{2\times3}(\mathrm{K}) \times \mathrm{Mat}_{2\times3}(\mathrm{K}) \times \mathrm{Mat}_{2\times3}(\mathrm{K}) \mapsto \mathrm{Mat}_{2\times3}(\mathrm{K})$, given by $\langle A, B, C\rangle = AB^tC$.

```
> K := GF(541);
> U := KMatrixSpace(K,2,3);
> my_prod := func< x | x[1]*Transpose(x[2])*x[3] >;
> t := Tensor([U,U,U,U], my_prod );
> t;
Tensor of valence 4, U3 x U2 x U1 >-> U0
U3 : Full Vector space of degree 6 over GF(541)
U2 : Full Vector space of degree 6 over GF(541)
U1 : Full Vector space of degree 6 over GF(541)
U0 : Full Vector space of degree 6 over GF(541)
```

Notice that the returned tensor is over vector spaces instead of the universe of `KMatrixSpace`. Tensors can still evaluate elements from the given frame, even though it prints out over vector spaces. However, the returned value form the tensor will be in the codomain of the tensor, so in this case, K^6.

```
> A := U![1,0,0,0,0,0];
> A;
[  1   0   0]
[  0   0   0]
> <A,A,A>@t;  // A is a generalized idempotent
(  1   0   0   0   0   0)
```

We can experiment to see if this triple product is left associative. To do this, we will construct five

random matrices $\{X_1, \ldots, X_5\} \subset \mathrm{Mat}_{2\times 3}(\mathrm{K})$, and then we test if

$$\langle\langle X_1, X_2, X_3\rangle, X_4, X_5\rangle = \langle X_1, \langle X_4, X_3, X_2\rangle, X_5\rangle.$$

Observe that the tuples have mixed entries, one from K^6 and two others from $\mathrm{Mat}_{2\times 3}(\mathrm{K})$.

```
> X := [Random(U) : i  in [1..5]];
> X;
[
    [485 378 385]
    [241 505 134],

    [141 531 245]
    [472 484 339],

    [377  85 170]
    [451 522 334],

    [211 340 409]
    [ 95 349 128],

    [264 372 144]
    [205  47 428]
]
>
> A := <X[1],X[2],X[3]> @ t;
> B := <X[4],X[3],X[2]> @ t;
> A, B;
(460 436 181 341 134 404)
(465 420 458 421 291 225)
>
> <A, X[4], X[5]> @ t eq <X[1], B, X[5]> @ t;
true
```

To confirm this product is left associative, we can create a new tensor for the left triple-associator and check that its image is 0. We will create a 6-tensor

$$\{\,\} : \prod_{k=1}^{5} \mathrm{Mat}_{2\times 3}(\mathrm{K}) \rightarrowtail \mathrm{Mat}_{2\times 3}(\mathrm{K})$$

where

$$\langle X_1, \ldots, X_5\rangle = \langle\langle X_1, X_2, X_3\rangle, X_4, X_5\rangle - \langle X_1, \langle X_4, X_3, X_2\rangle, X_5\rangle.$$

Therefore, if $\mathrm{im}(\langle\,\rangle) = 0$, then $\langle\,\rangle$ is left associative.

```
> l_asct := func< X | Eltseq(<<X[1], X[2], X[3]> @ t, X[4], X[5]> @ t
>     - <X[1], <X[4], X[3], X[2]> @ t, X[5]> @ t) >;
> Lt := Tensor([* U : i in [0..5] *], l_asct);
> Lt;
Tensor of valence 6, U5 x U4 x U3 x U2 x U1 >-> U0
U5 : Full Vector space of degree 6 over GF(541)
U4 : Full Vector space of degree 6 over GF(541)
U3 : Full Vector space of degree 6 over GF(541)
U2 : Full Vector space of degree 6 over GF(541)
U1 : Full Vector space of degree 6 over GF(541)
U0 : Full Vector space of degree 6 over GF(541)
>
```

(continues on next page)

(continued from previous page)

```
> I := Image(Lt);
> I;
Vector space of degree 6, dimension 0 over GF(541)
Generators:

>
> Dimension(I);
0
```

Observe that in `lasct` the function `Eltseq` is called. This is because t returns vectors in K^6 which is not naturally coercible into $\mathrm{Mat}_{2\times 3}(\mathrm{K})$. On the other hand, sequences can be coerced into $\mathrm{Mat}_{2\times 3}(\mathrm{K})$.

Tensors with Structure Constant Sequences

Most computations with tensors T will be carried out using structure constants $T_{j_\nu \cdots j_0} \in K$. Here T is framed by free K-modules $[U_\nu, \ldots, U_0]$ with each U_i having an ordered basis $\mathcal{B}_i = [e_{i1}, \ldots, e_{id_i}]$. The interpretation of structure constants is that the associated multilinear function $[x_\nu, \ldots, x_1]$ from $U_\nu \times \cdots \times U_1$ into U_0 is determined on bases $\mathcal{B}_i$ as follows:

$$[e_{\nu j_\nu}, \ldots, e_{1j_1}] = \sum_{k=1}^{d_0} T_{j_\nu \cdots j_0} e_{0k}.$$

Structure constants are input and stored as sequences S in K according to the following assignment. Set $f : \mathbb{Z}^{\nu+1} \to \mathbb{Z}$ to be:

$$f(j_\nu, \ldots, j_0) = 1 + \sum_{s=0}^{\nu} (j_s - 1) \prod_{t=0}^{s-1} d_t.$$

So $S[f(j_\nu, \ldots, j_0)] = T_{j_\nu \cdots j_0}$ specifies the structure constants as a sequence.

Notes.

- Magma does not currently support the notion of a sparse sequence of structure constants. A user can provide this functionality by specifying the multilinear function associated with a tensor by means of a user program rather than by structure constants.
- Some routines in Magma require structure constant sequences. If they are not provided, Magma may compute and store a structure constant representation with the tensor.
- Magma does not separate structure constant indices that are contravariant. Instead contravariant variables are signaled by tensor categories. So Ricci-styled tensors $T^{b_q \cdots b_1}_{a_p \cdots a_1}$ should be input as $T_{a_{p+q} \cdots a_{1+q} b_q \cdots b_1}$ and the tensor category changed to mark $\{q, \ldots, 1\}$ as contravariant. Intrinsics are provided to facilitate this approach. See Section *Tensor Categories* for more details on tensor categories.

`Tensor(D, S): [RngIntElt], SeqEnum -> TenSpcElt`

`Tensor(R, D, S): Rng, [RngIntElt], SeqEnum -> TenSpcElt`

`Tensor(D, S, Cat): [RngIntElt], SeqEnum, TenCat -> TenSpcElt`

`Tensor(R, D, S, Cat): Rng, [RngIntElt], SeqEnum, TenCat -> TenSpcElt`

Let R be a commutative unital ring and D a sequence $[d_v, \ldots, d_0]$ giving the dimensions of the R-modules $U_v, \ldots, U_0$. The intrinsic returns the tensor with frame $\{ U_v, \ldots, U_1\}$, codomain U_0, and multilinear map defined by the structure constant sequence S. If R is not specified then the parent ring of the first element of S is used. The default tensor category Cat is the homotopism category.

Example: Tensor from Structure Constants `(ex-3135bc)`

We will create structure constants sequence with all 0s and one 1 that occurs in the first entry. First, we will input this along with the dimensions of the tensor we are after, $2 \times 2 \times 2$. However,

since we did not specify a ring, it is assumed to be the parent ring of the first entry of the structure constants sequence. In this example, the ring is $\mathbb{Z}$.

```
> sc := [ 0 : i in [1..8] ];
> sc[1] := 1;
> Tensor([2, 2, 2], sc);
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full RSpace of degree 2 over Integer Ring
U1 : Full RSpace of degree 2 over Integer Ring
U0 : Full RSpace of degree 2 over Integer Ring
```

We do not want the underlying ring to be $\mathbb{Z}$, so we will input the ring we want: GF(64).

```
> K := GF(64);
> t := Tensor(K, [2, 2, 2], sc);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 2 over GF(2^6)
U1 : Full Vector space of degree 2 over GF(2^6)
U0 : Full Vector space of degree 2 over GF(2^6)
>
> Image(t);
Vector space of degree 2, dimension 1 over GF(2^6)
Generators:
(     1      0)
Echelonized basis:
(     1      0)
```

We can recover the structure constants by calling `StructureConstants` or `Eltseq`. We will also test that the tensor evaluates inputs correctly.

```
> StructureConstants(t);
[ 1, 0, 0, 0, 0, 0, 0, 0 ]
>
> <[1, 0], [K.1^3, 0]> @ t;
( K.1^3      0)
>
> <[K.1^29, 1], [0, K.1^2]> @ t;
(     0      0)
```

`StructureConstants(T): TenSpcElt -> SeqEnum`

`Eltseq(T): TenSpcElt -> SeqEnum`

Returns the sequence of structure constants of the given tensor T.

`Assign(T, ind, k): TenSpcElt, [RngIntElt], Any -> TenSpcElt`

`Assign(~T, ind, k): TenSpcElt, [RngIntElt], Any`

Returns the tensor T where the `ind` element (viewed as a multi-dimensional array) is replaced with k. For example, replacing the (a, b, c) entry of a 3-tensor, set `ind` $= [a, b, c]$.

Example: Structure Constants from a Black-Box Tensor `(ex-73e82c)`

We will construct the natural Lie module action for SL_2, but we will construct it as a *left* module. To do this, we construct a function that takes elements from $\mathrm{SL}_2 \times \mathrm{V}$ and returns an element that Magma can coerce into V. We run a quick test to make sure our function runs on the trivial example; this is the only check the intrinsic runs on black-box tensors. Since SL_2 and V are part of different universes in Magma, we must use the `List` environment when constructing this black-box tensor.

```
> SL2 := MatrixLieAlgebra("A1", GF(7));
> V := VectorSpace(GF(7), 2);
> left_action := func< x | x[2]*Transpose(Matrix(x[1])) >;
> left_action(<SL2!0, V!0>);
(0 0)
>
> SL2 := MatrixLieAlgebra("A1", GF(7));
> V := VectorSpace(GF(7), 2);
> left_action := func< x | x[2]*Transpose(Matrix(x[1])) >;
> left_action(<SL2!0, V!0>);
(0 0)
>
> t := Tensor([* SL2, V, V *], left_action);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 3 over GF(7)
U1 : Full Vector space of degree 2 over GF(7)
U0 : Full Vector space of degree 2 over GF(7)
```

Now we will extract the structure constants from this Lie module action. We will then construct a tensor with these structure constants and compare it with our first tensor above.

```
> StructureConstants(t);
[ 1, 0, 0, 6, 0, 0, 1, 0, 0, 1, 0, 0 ]
>
> s := Tensor([3, 2, 2], Eltseq(t));
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 3 over GF(7)
U1 : Full Vector space of degree 2 over GF(7)
U0 : Full Vector space of degree 2 over GF(7)
>
> t eq s;
true
```

Example: Stored Structure Constants (ex-80fd79)

The structure constants are convenient data structure for nearly all the algorithms using tensors. In fact, most computations require structure constants, so we store the structure constants sequence with the tensor. This means that after the initial structure constant sequence computation, every time `StructureConstants` or `Eltseq` is called, Magma retrieves what was previously computed.

We will demonstrate this on a large black-box example, so some time is spent computing the structure constants. Of course, the exact timing will vary by machine. We will construct a product of two subalgebras of $\mathrm{Mat}_{20}(\mathbf{F}_3)$, namely $* : \mathrm{SL}_{20}(\mathbf{F}_3) \times \mathrm{Mat}_4(\mathbf{F}_3) \rightarrowtail \mathrm{Mat}_{20}(\mathbf{F}_3)$.

```
> SL20 := MatrixLieAlgebra("A19", GF(3));
> M4 := MatrixAlgebra(GF(3), 4);
> Prod := func< x | Matrix(x[1])*DiagonalJoin(<x[2] : i in [1..5]>) >;
> t := Tensor([* SL20, M4 *], MatrixAlgebra(GF(3), 20), Prod);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 399 over GF(3)
U1 : Full Vector space of degree 16 over GF(3)
U0 : Full Vector space of degree 400 over GF(3)
```

We record the time it takes to initially compute the structure constants sequence, and then we record the time when we call the function again.

```
> time sc := StructureConstants(t);
Time: 58.670
>
> time sc := StructureConstants(t);
Time: 0.000
```

Bilinear Tensors

In the special case of bilinear maps $U_2 \times U_1 \rightarrowtail U_0$, the structure constant sequences may be presented as lists $[M_1, \ldots, M_a]$ of matrices. This can be considered as a left (resp. right) representation $U_2 \to \hom_K(U_1, U_0)$, (resp.$U_1 \to \hom_K(U_2, U_0)$). Alternatively, bilinear maps can be treated as *systems of bilinear forms* $[M_1, \ldots, M_a]$ where the matrices are the Gram matrices of bilinear forms $\phi_i : U_2 \times U_1 \rightarrowtail K$. Here the associated bilinear map $U_2 \times U_1 \rightarrowtail U_0$ is specified by

$$(u_2, u_1) \mapsto (\phi_1(u_2, u_1), \ldots, \phi_a(u_2, u_1)).$$

`Tensor(M, s, t): [Mtrx], RngIntElt, RngIntElt -> TenSpcElt`

`Tensor(M, s, t, C): [Mtrx], RngIntElt, RngIntElt, TenCat -> TenSpcElt`

Given a sequence of matrices M, the bilinear tensor specified by M is returned. The interpretation of the matrices as structure constant sequences is determined by the coordinates s and t which must be integers from the set $\{2, 1, 0\}$. Optionally a tensor category C can be specified.

Example: Symplectic Bilinear Form `(ex-be1f50)`

We will construct a symplectic bilinear form on $V = K^8$. It would be cumbersome to construct this tensor as a black-box tensor or by providing the structure constants sequence. Instead, we will provide a (Gram) matrix.

```
> K := GF(17);
> MS := KMatrixSpace(K, 2, 2);
> J := KroneckerProduct(IdentityMatrix(K, 4), MS![0, 1, -1, 0]);
> J;
[ 0  1  0  0  0  0  0  0]
[16  0  0  0  0  0  0  0]
[ 0  0  0  1  0  0  0  0]
[ 0  0 16  0  0  0  0  0]
[ 0  0  0  0  0  1  0  0]
[ 0  0  0  0 16  0  0  0]
[ 0  0  0  0  0  0  0  1]
[ 0  0  0  0  0  0 16  0]
>
> t := Tensor(J, 2, 1);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 8 over GF(17)
U1 : Full Vector space of degree 8 over GF(17)
U0 : Full Vector space of degree 1 over GF(17)
>
> IsAlternating(t);
true
```

Now we will construct the symplectic form using the black-box construction and verify that the two tensors are the same.

```
> V := VectorSpace(K, 8);
> symp := func< x | x[1]*J*Matrix(8, 1, Eltseq(x[2])) >;
```

(continues on next page)

(continued from previous page)

```
> s := Tensor([V, V], VectorSpace(K, 1), symp);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 8 over GF(17)
U1 : Full Vector space of degree 8 over GF(17)
U0 : Full Vector space of degree 1 over GF(17)
>
> SystemOfForms(s);
[
    [ 0  1  0  0  0  0  0  0]
    [16  0  0  0  0  0  0  0]
    [ 0  0  0  1  0  0  0  0]
    [ 0  0 16  0  0  0  0  0]
    [ 0  0  0  0  0  1  0  0]
    [ 0  0  0  0 16  0  0  0]
    [ 0  0  0  0  0  0  0  1]
    [ 0  0  0  0  0  0 16  0]
]
```

AsMatrices(T, s, t): TenSpcElt, RngIntElt, RngIntElt -> SeqEnum

SystemOfForms(T): TenSpcElt -> SeqEnum

For a tensor T with frame $[K^{d_v}, \ldots, K^{d_0}]$, a list $[M_1, \ldots, M_d]$, $d = (d_v \cdots d_0)/d_s d_t$, of $(d_s \times d_t)$-matrices in K representing the tensor as an element of $\hom_K(K^{d_s} \otimes_K K^{d_t}, K^d)$ is returned. For the intrinsic **SystemOfForms**, T must have valence 2 and the implied values are $s = 2$ and $t = 1$.

Example: Trilinear Tensor from Matrices (ex-1d87b3)

We construct the associator of $\mathrm{SL}_2(\mathbb{Q})$ where

$$\langle\,\rangle : \prod_{k=1}^{3} \mathrm{SL}_2 \rightarrowtail \mathrm{SL}_2$$

given by

$$\langle x, y, z\rangle = [[x, y], z] - [x, [y, z]].$$

It can be hard to understand some of the features of this trilinear map by only looking at the structure constants sequence. The function **AsMatrices** slices the sequence and presents the data as a sequence of matrices.

```
> K := Rationals();
> L := LieAlgebra("A1", K);
> t := AssociatorTensor(L);
> t;
Tensor of valence 4, U3 x U2 x U1 >-> U0
U3 : Full Vector space of degree 3 over Rational Field
U2 : Full Vector space of degree 3 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 3 over Rational Field
>
> Eltseq(t);
[ 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, -2, 0, 0,
0,-2, 0, 0, 0, 0, 0, 0, 0, 2, 0, -1, 0, 0, 0, 0, 0, 0, 0, -1, 0, 2, 0, 0, 0,
0, 0, 0, 0, -2, 0, 0, 0, -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 2,
0, 0, 0, 0, 0, 0 ]
```

Calling **AsMatrices(t, 3, 1)** returns a sequence of nine matrices, but we only show the first four. As explained in the documentation above, this sequence of matrices can be interpreted as

the system of bilinear forms for the 3-tensor $\circ_{31} : V_3 \times V_1 \rightarrowtail \hom_K(V_2, V_0)$ given by

$$x \circ_{31} z = \langle x, -, z \rangle = [[x, -], z] - [x, [-, z]].$$

```
> AsMatrices(t, 3, 1)[1..4];
[
    [ 0  0  2]
    [ 0  0  0]
    [-2  0  0],

    [ 0  0  0]
    [ 0  0  2]
    [ 0 -2  0],

    [0 0 0]
    [0 0 0]
    [0 0 0],

    [ 0  1  0]
    [-1  0  0]
    [ 0  0  0]
]
```

Tensors from Algebraic Objects

A natural and important source of tensors come from algebraic objects with a distributive property. One main source is from algebras, where $* : A \times A \rightarrowtail A$ is given by multiplication in A. Like with the previous sections on tensor constructions, all tensors will be constructed over vector spaces. The user can still input elements from the original algebra, but map(s) will also be returned. Furthermore, each tensor is assigned a category relevant to its origin, see the section on tensor categories for more information.

`Tensor(A): Alg -> TenSpcElt, Map`

Given an algebra A with an element product that satisfies an appropriate distributive law, return the bilinear tensor given by the product in A.

Example: Lie Bracket Tensor of D4(11) `(ex-17021d)`

We want to get the Lie bracket from $D_4(11)$. Tensors created from algebras will have a homotopism category, but with $U_2 = U_1 = U_0$. This forces the operators acting to be the same on all the coordinates; in other words, $\Omega = \mathrm{End}(\mathrm{U}_2)$ instead of $\Omega = \mathrm{End}(\mathrm{U}_2) \times \mathrm{End}(\mathrm{U}_1) \times \mathrm{End}(\mathrm{U}_0)$.

```
> L := LieAlgebra("D4", GF(11));
> t := Tensor(L);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 28 over GF(11)
U1 : Full Vector space of degree 28 over GF(11)
U0 : Full Vector space of degree 28 over GF(11)
> IsAlternating(t);
true
> TensorCategory(t);
Tensor category of valence 3 (->,->,->) ({ 0, 1, 2 })
```

If we compute the derivation algebra of L, our operators will act in the same way on each coordinate. This is the standard definition of the derivation algebra of a ring.

```
> D := DerivationAlgebra(t);
> Dimension(D);
```

(continues on next page)

(continued from previous page)

```
28
> SemisimpleType(D);
D4
```

Now we will change the category to the standard homotopism category, where we do *not* fuse U_2, U_1, and U_0.

```
> ChangeTensorCategory(~t, HomotopismCategory(3));
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 28 over GF(11)
U1 : Full Vector space of degree 28 over GF(11)
U0 : Full Vector space of degree 28 over GF(11)
> TensorCategory(t);
Tensor category of valence 3 (->,->,->) ({ 1 },{ 2 },{ 0 })
```

We compare the same computation of derivation algebra. This time, the theory tells us that there will be a solvable radical. In this example, $\mathrm{Rad}(D) = K^2$.

```
> D := DerivationAlgebra(t);
> Dimension(D);
30
> R := SolvableRadical(D);
> SemisimpleType(D/R);
D4
```

Tensor(Q): RngUPolRes -> TenSpcElt, Map

Returns the bilinear tensor given by the product in quotient polynomial ring Q.

Example: Tensor of the Witt Algebra (ex-3ead54)

The Witt algebra, over a finite field of characteristic p, is isomorphic to the derivation algebra of $K[x]/(x^p)$. The Witt algebra is a simple Lie algebra with dimension p and a trivial Killing form. First, we will construct the tensor from the ring $\mathbf{F}_5[x]/(x^5)$. Note that, like with algebras, the tensor category will fuse U_2, U_1, and U_0, so that the operators act the same way on every coordinate.

```
> p := 5;
> R<x> := PolynomialRing(GF(p));
> I := ideal< R | x^p >;
> Q := quo< R | I >;
> Q;
Univariate Quotient Polynomial Algebra in $.1 over Finite field of size
5 with modulus $.1^5
> t := Tensor(Q);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 5 over GF(5)
U1 : Full Vector space of degree 5 over GF(5)
U0 : Full Vector space of degree 5 over GF(5)
> TensorCategory(t);
Tensor category of valence 3 (->,->,->) ({ 0, 1, 2 })
```

Now we will construct a Lie representation of the Witt algebra from the tensor t.

```
> D := DerivationAlgebra(t);
> IsSimple(D);
```

(continues on next page)

(continued from previous page)

```
true
> Dimension(D);
5
> KillingForm(D);
[0 0 0 0 0]
[0 0 0 0 0]
[0 0 0 0 0]
[0 0 0 0 0]
[0 0 0 0 0]
```

`CommutatorTensor(A): Alg -> TenSpcElt, Map`

`AnticommutatorTensor(A): Alg -> TenSpcElt, Map`

Given an algebra A return the tensor corresponding to the bilinear commutator map $[a, b] = ab - ba$ or the anticommutator map $\langle a, b\rangle = ab + ba$ of the algebra A. This should not be used to get the tensor given by the Lie or Jordan product in a Lie or Jordan algebra; instead use `Tensor`.

Example: Commutator Tensor of a Matrix Algebra (ex-ad0163)

We will construct the commutator tensor from $\mathrm{Mat}_4(\mathbb{Q})$.

```
> A := MatrixAlgebra(Rationals(), 4);
> t := CommutatorTensor(A);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 16 over Rational Field
U1 : Full Vector space of degree 16 over Rational Field
U0 : Full Vector space of degree 16 over Rational Field
> IsAlternating(t); // [X, X] = 0?
true
```

With this tensor, we will compute the dimension of the centralizer of the diagonal matrix M with diagonal entries $(1, 1, -1, -1)$ in $\mathrm{Mat}_4(\mathbb{Q})$. To do this, we will subtract the dimension of the image of $[M, A]$ from the dimension of A.

```
> M := A![1,0,0,0,0,1,0,0,0,0,-1,0,0,0,0,-1];
> M;
[ 1  0  0  0]
[ 0  1  0  0]
[ 0  0 -1  0]
[ 0  0  0 -1]
> Dimension(A) - Dimension(<M, A> @ t);
8
```

Example: Jordan Product of a Matrix Algebra (ex-b964df)

This time, we will obtain a Jordan product from $\mathrm{Mat}_4(\mathbb{Q})$. That is, we will construct the bilinear map $* : \mathrm{Mat}_4(\mathbb{Q}) \times \mathrm{Mat}_4(\mathbb{Q}) \rightarrowtail \mathrm{Mat}_4(\mathbb{Q})$ where $A * B = (AB + BA)/2$.

```
> A := MatrixAlgebra(Rationals(), 4);
> t := AnticommutatorTensor(A);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 16 over Rational Field
U1 : Full Vector space of degree 16 over Rational Field
U0 : Full Vector space of degree 16 over Rational Field
> SystemOfForms(t)[1];
[2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

(continues on next page)

(continued from previous page)

```
[0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0]
[0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
> A.1*t*A.1;
(2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
```

From the documentation on `AnticommutatorTensor`, we have to scale our tensor above t by $1/2$ to get what we want. Of course this won't affect the proceeding tests though.

```
> s := (1/2)*t;
> A.1*s*A.1;
(1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
```

Now we will confirm that s is a Jordan product. First, we check that s is commutative, and then we check that it satisfies the Jordan identity: $(xy)(xx) = x(y(xx))$.

```
> IsSymmetric(s);
true
> JordanID := func< x, y | (x*s*y)*s*(x*s*x) - x*s*(y*s*(x*s*x)) >;
> forall{ <x,y> : x in Basis(A), y in Basis(A) |
>     JordanID(x, y) eq Codomain(s)!0 };
true
```

`AssociatorTensor(A): Alg -> TenSpcElt, Map`

Given an algebra A return the tensor corresponding to the trilinear associator map $[a, b, c] = (ab)c - a(bc)$ on the algebra A.

Example: Associator Tensor of the Octonions (ex-4acc3b)

Do three random octonions satisfy the associate law? Hardly ever.

```
> O := OctonionAlgebra ( GF (1223) , -1 , -1 , -1);
> t := AssociatorTensor ( O );
> t;
Tensor of valence 4, U3 x U2 x U1 >-> U0
U3 : Full Vector space of degree 8 over GF(1223)
U2 : Full Vector space of degree 8 over GF(1223)
U1 : Full Vector space of degree 8 over GF(1223)
U0 : Full Vector space of degree 8 over GF(1223)
> <Random(O),Random(O),Random(O)> @ t eq O!0;
false
```

However, for all $a, b \in O$, $(aa)b = a(ab)$ as octonions are alternative algebras.

```
> a := Random ( O );
> b := Random ( O );
> <a, a, b> @ t eq O!0;
true
> IsAlternating ( t );
true
```

pCentralTensor(G, p, s, t): Grp, RngIntElt, RngIntElt, RngIntElt -> TenSpcElt, List

pCentralTensor(G, s, t): GrpPC, RngIntElt, RngIntElt -> TenSpcElt, List

pCentralTensor(G, s, t): Grp, RngIntElt, RngIntElt -> TenSpcElt, List

pCentralTensor(G): Grp -> TenSpcElt, List

Returns the bilinear map of commutation from the associated graded Lie algebra of the lower exponent-p central series η of G. The bilinear map pairs η_s/η_{s+1} with η_t/η_{t+1} into η_{s+t}/η_{s+t+1}. If $s = t$ the tensor category is set to force $U_2 = U_1$; otherwise it is the general homotopism category. In addition, maps from the subgroups into the vector spaces are returned as a list. If p, s, and t are not given, it is assumed G is a p-group and $s = t = 1$.

Example: Recovering a Ring from a p-Group (ex-94b48f)

Groups have a single binary operation. So even when groups are built from rings it can be difficult to recover the ring from the group operations. Tensors supply one approach for that task. We will get the p-central tensor of $G = \mathrm{SL}(3, 125)$; however, we will lose the fact that there is a field $\mathbf{F}_{125}$. The tensor we will get back is $[,] : K^6 \times K^6 \rightarrowtail K^3$, where $K = \mathbf{F}_5$.

```
> P := ClassicalSylow(SL(3,125), 5);
> Q := PCGroup(P);     // The group Q won't have explicit knowledge of GF(125).
> Q;
GrpPC : Q of order 1953125 = 5^9
PC-Relations:
    Q.4^Q.1 = Q.4 * Q.7^4,
    Q.4^Q.2 = Q.4 * Q.8^4,
    Q.4^Q.3 = Q.4 * Q.9^4,
    Q.5^Q.1 = Q.5 * Q.8^4,
    Q.5^Q.2 = Q.5 * Q.9^4,
    Q.5^Q.3 = Q.5 * Q.7^3 * Q.8^3,
    Q.6^Q.1 = Q.6 * Q.9^4,
    Q.6^Q.2 = Q.6 * Q.7^3 * Q.8^3,
    Q.6^Q.3 = Q.6 * Q.8^3 * Q.9^3
> t := pCentralTensor(Q);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 6 over GF(5)
U1 : Full Vector space of degree 6 over GF(5)
U0 : Full Vector space of degree 3 over GF(5)
```

Knowing that G is defined over $\mathbf{F}_{125}$, we know that the commutator is really just the alternating form $\cdot : \mathbf{F}_{125}^2 \times \mathbf{F}_{125}^2 \rightarrowtail \mathbf{F}_{125}$. This information can be extracted from the centroid of t above, and we can rewrite t over the field $\mathbf{F}_{125}$.

```
> F := Centroid(t);  // Recover GF(125)
> Dimension(F);
3
> IsSimple(F);
true
> IsCommutative(F);
true
> s := TensorOverCentroid(t);
```

(continues on next page)

(continued from previous page)

```
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 2 over GF(5^3)
U1 : Full Vector space of degree 2 over GF(5^3)
U0 : Full Vector space of degree 1 over GF(5^3)
```

MatrixTensor(K, S): Fld, [RngIntElt] -> TenSpcElt, List

Given a field K and a sequence of positive integers $S = [s_1, \ldots, s_\nu]$, return the tensor

$$\mathrm{Mat}_{s_1 \times s_2}(K) \times \cdots \times \mathrm{Mat}_{s_{\nu-1} \times s_\nu}(K) \rightarrowtail \mathrm{Mat}_{s_1 \times s_\nu}(K),$$

given by matrix multiplication. A list of maps from the matrix spaces to the vector spaces is given as well even though the given tensor will evaluate matrices as well.

Polarisation(f): MPolElt -> TenSpcElt, MPolElt

Polarisation(f): RngUPolElt -> TenSpcElt

Polarization(f): MPolElt -> TenSpcElt, MPolElt

Polarization(f): RngUPolElt -> TenSpcElt

Returns the polarization of the homogeneous multivariate polynomial f as a tensor and as a multivariate polynomial. Note that the intrinsic does not normalize by $1/d!$, where d is the degree of f.

Example: Polarization of a Binary Cubic Form (ex-794950)

The polarization intrinsic is applied to the polynomial $f(x, y) = x^2 y$. Because f is homogeneous of degree 3 with 2 variables, it is expected that the polarization will have 6 variables and that the corresponding multilinear form will be $K^2 \times K^2 \times K^2 \rightarrowtail K$. The polarization of f is given by $P(x_1, x_2, y_1, y_2, z_1, z_2) = 2(x_1 y_1 z_2 + x_1 y_2 z_1 + x_2 y_1 z_1)$.

```
> R<x,y> := PolynomialRing(Rationals(), 2);
> T, p := Polarization(x^2*y);
> p;
2*$.1*$.3*$.6 + 2*$.1*$.4*$.5 + 2*$.2*$.3*$.5
> T;
Tensor of valence 4, U3 x U2 x U1 >-> U0
U3 : Full Vector space of degree 2 over Rational Field
U2 : Full Vector space of degree 2 over Rational Field
U1 : Full Vector space of degree 2 over Rational Field
U0 : Full Vector space of degree 1 over Rational Field
> <[1,0],[1,0],[1,0]> @ T;
(0)
> <[1,0],[1,0],[0,1]> @ T;
(2)
```

New Tensors from Old

A number of new tensors may be constructed from a given tensor.

AlternatingTensor(T): TenSpcElt -> TenSpcElt

Returns the alternating tensor induced by the given tensor T. If the tensor is already alternating, then the given tensor is returned.

AntisymmetricTensor(T): TenSpcElt -> TenSpcElt

Returns the antisymmetric tensor induced by the given tensor T. If the tensor is already antisymmetric, then the given tensor is returned.

SymmetricTensor(T): TenSpcElt -> TenSpcElt

Returns the symmetric tensor induced by the given tensor T. If the tensor is already symmetric, then the given tensor is returned.

Example: Alternating Tensor of a Lie Algebra (ex-3641e3)

Tensors coming from Lie algebras are alternating. If we call `AlternatingTensor` on a tensor from a Lie algebra, nothing will be changed.

```
> L := LieAlgebra("A3", GF(3));
> t := Tensor(L);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 15 over GF(3)
U1 : Full Vector space of degree 15 over GF(3)
U0 : Full Vector space of degree 15 over GF(3)
> AlternatingTensor(t) eq t;
true
```

Example: Symmetrizing a Matrix Algebra Tensor (ex-73d5eb)

We will make the tensor coming from the product in $\mathrm{Mat}_3(\mathbb{Q})$ symmetric.

```
> A := MatrixAlgebra(Rationals(), 3);
> t := Tensor(A);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 9 over Rational Field
U1 : Full Vector space of degree 9 over Rational Field
U0 : Full Vector space of degree 9 over Rational Field
> SystemOfForms(t)[1];
[1 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0]
```

The first matrix in the sequence of bilinear forms of t is not symmetric, so t is not symmetric (of course matrix multiplication is not commutative also). We will construct a symmetric version of t and inspect the first matrix of the bilinear forms.

```
> s := SymmetricTensor(t);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 9 over Rational Field
U1 : Full Vector space of degree 9 over Rational Field
U0 : Full Vector space of degree 9 over Rational Field
> SystemOfForms(s)[1];
[2 0 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 0 1 0 0]
[0 1 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0]
```

Shuffle(T, g): TenSpcElt, GrpPermElt -> TenSpcElt

Shuffle(T, g): TenSpcElt, SeqEnum -> TenSpcElt

Given a tensor T in $\hom(U_\nu, \ldots, \hom(U_1, U_0)\cdots)$, this intrinsic generates the representation of T in

$$\hom(U_{\nu^g}, \ldots, \hom(U_{1^g}, U_{0^g})\ldots),$$

where g is a permutation of $\{0, \ldots, \nu\}$. The permutation g may be given as an element of a permutation group or as a sequence. Both the image and pre-image of 0 under g will be replaced by their K-dual space. For cotensors, g must permute $\{1, \ldots, \nu\}$. Sequences $[a_1, \ldots, a_{\nu+1}]$ will be interpreted as a permutation in one-line notation.

Example: Transposing a Form by Shuffling (ex-e67343)

We will shuffle the alternating form $\mathbb{Q}^2 \times \mathbb{Q}^2 \rightarrowtail \mathbb{Q}$ as a means of performing a transpose on the Gram matrix. To do this, we need to shuffle by the transposition $(1, 2)$ in $\mathrm{Sym}(\{0, 1, 2\})$.

```
> t := Tensor(Rationals(), [2, 2, 1], [0, 1, -1, 0]);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 2 over Rational Field
U1 : Full Vector space of degree 2 over Rational Field
U0 : Full Vector space of degree 1 over Rational Field
> SystemOfForms(t);
[
    [ 0  1]
    [-1  0]
]
>
> s := Shuffle(t, [0,2,1]);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 2 over Rational Field
U1 : Full Vector space of degree 2 over Rational Field
U0 : Full Vector space of degree 1 over Rational Field
> SystemOfForms(s);
[
    [ 0 -1]
    [ 1  0]
]
```

Example: Shuffling a Random 5-Tensor (ex-ae928f)

We will generate a random 5-tensor and shuffle it with $(0, 2, 4, 1, 3)$.

```
> t := RandomTensor(GF(2), [5,4,3,2,1]);
> t;
Tensor of valence 5, U4 x U3 x U2 x U1 >-> U0
U4 : Full Vector space of degree 5 over GF(2)
U3 : Full Vector space of degree 4 over GF(2)
U2 : Full Vector space of degree 3 over GF(2)
U1 : Full Vector space of degree 2 over GF(2)
U0 : Full Vector space of degree 1 over GF(2)
>
> G := Sym({0..4});
> g := G![2, 3, 4, 0, 1];
> g;
(0, 2, 4, 1, 3)
>
> s := Shuffle(t, g);
```

(continues on next page)

(continued from previous page)

```
> s;
Tensor of valence 5, U4 x U3 x U2 x U1 >-> U0
U4 : Full Vector space of degree 2 over GF(2)
U3 : Full Vector space of degree 1 over GF(2)
U2 : Full Vector space of degree 5 over GF(2)
U1 : Full Vector space of degree 4 over GF(2)
U0 : Full Vector space of degree 3 over GF(2)
```

`TensorProduct(t, s): TenSpcElt, TenSpcElt -> TenSpcElt`

Given K-tensors $t : U_\nu \times \cdots \times U_1 \rightarrowtail U_0$ and $s : V_\nu \times \cdots \times V_1 \rightarrowtail V_0$, returns the tensor $t \otimes s : U_\nu \otimes V_\nu \times \cdots \times U_1 \otimes V_1 \rightarrowtail U_0 \otimes V_0$. This is like a generalized Kronecker product.

63.2.2 Operations with Tensors

Magma supports two perspectives for operations with tensors. Firstly, tensors determine multilinear maps and so behave as functions. Secondly, tensors are elements of a tensor space and so behave as elements of a module.

Elementary Operations

Treating the tensor space as a K-module, we have the standard operations.

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

`S - T: TenSpcElt, TenSpcElt -> TenSpcElt`

`k * T: RngElt, TenSpcElt -> TenSpcElt`

`-T: TenSpcElt -> TenSpcElt`

Given tensors S and T belonging to the same tensor space V, and an element k belonging to the coefficient ring K, these intrinsics return the sum, difference, scalar multiple, or additive inverse of S and T as elements of V. The corresponding multilinear maps are the sum, difference, scalar multiple, or additive inverse of the multilinear maps.

Example: Module Operations on Tensors `(ex-ce9931)`

As tensors are elements of a tensor space, it inherits module operations. We will demonstrate them all here. First, here are the tensors we will operate with.

```
> K := Rationals();
> t := Tensor(K, [2, 2, 2], [1..8]);
> s := Tensor(K, [2, 2, 2], &cat[[2, -1] : i in [1..4]]);
> SystemOfForms(t);
[
    [1 3]
    [5 7],

    [2 4]
    [6 8]
]
> SystemOfForms(s);
[
    [2 2]
    [2 2],

    [-1 -1]
    [-1 -1]
]
```

Now we perform the module operations.

```
> SystemOfForms(-t);
[
    [-1 -3]
    [-5 -7],

    [-2 -4]
    [-6 -8]
]
> SystemOfForms((1/3)*s);
[
    [2/3 2/3]
    [2/3 2/3],

    [-1/3 -1/3]
    [-1/3 -1/3]
]
> SystemOfForms(t+s);
[
    [3 5]
    [7 9],

    [1 3]
    [5 7]
]
> SystemOfForms(t-2*s);
[
    [-3 -1]
    [ 1  3],

    [ 4  6]
    [ 8 10]
]
```

AssociatedForm(T): TenSpcElt -> TenSpcElt

For a tensor T with frame $U_\nu \times \cdots \times U_1 \rightarrowtail U_0$, this intrinsic creates the associated multilinear form $U_\nu \times \cdots \times U_1 \times U_0^* \rightarrowtail K$. The valence is increased by 1.

Compress(T): TenSpcElt -> TenSpcElt

Compress(~T): TenSpcElt

Returns the compression of the tensor T. This removes all 1-dimensional spaces in the domain.

Example: Associated Form of a Lie Algebra Tensor (ex-9d2ed1)

We will construct the associated form of the tensor from the Lie algebra $B_3(5)$. The codomain of the original tensor gets moved (and dualed) to the domain.

```
> L := LieAlgebra("B3", GF(5));
> t := Tensor(L);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 21 over GF(5)
U1 : Full Vector space of degree 21 over GF(5)
U0 : Full Vector space of degree 21 over GF(5)
> s := AssociatedForm(t);
> s;
Tensor of valence 4, U3 x U2 x U1 >-> U0
U3 : Full Vector space of degree 21 over GF(5)
U2 : Full Vector space of degree 21 over GF(5)
```

(continues on next page)

(continued from previous page)

```
U1 : Full Vector space of degree 21 over GF(5)
U0 : Full Vector space of degree 1 over GF(5)
> <L.2, L.11> @ t;
(0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
> <L.2, L.11, L.2> @ s;
(4)
> <L.2, L.11, L> @ s;
Full Vector space of degree 1 over GF(5)
Generators:
(4)
```

We shuffle the associated form by the permutation $(0, 3)$ and compress it. The result is just the shuffle of the original bilinear map t by $(0, 2, 1)$.

```
> shf := Shuffle(s, [3,1,2,0]);
> shf;
Tensor of valence 4, U3 x U2 x U1 >-> U0
U3 : Full Vector space of degree 1 over GF(5)
U2 : Full Vector space of degree 21 over GF(5)
U1 : Full Vector space of degree 21 over GF(5)
U0 : Full Vector space of degree 21 over GF(5)
> cmp := Compress(shf);
> cmp;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 21 over GF(5)
U1 : Full Vector space of degree 21 over GF(5)
U0 : Full Vector space of degree 21 over GF(5)
> cmp eq Shuffle(t, [2, 0, 1]);
true
```

General Properties

We provide basic intrinsics to retrieve data stored in the `TenSpcElt` object in Magma. Most of these functions are already stored as attributes. The ones that are not initially stored at construction are stored immediately after the initial computation, such as `Image`.

`Parent(T): TenSpcElt -> TenSpc`

Returns the tensor space that contains the tensor T. The default space is the universal tensor space.

`Domain(T): TenSpcElt -> List`

Returns the domain of the tensor T as a list of modules.

`Codomain(T): TenSpcElt -> Any`

Returns the codomain of the tensor T.

`Valence(T): TenSpcElt -> RngIntElt`

Returns the valence of the tensor T.

`Frame(T): TenSpcElt -> List`

Returns the modules in the frame of the tensor T in the form of a list containing the domain modules and the codomain.

`BaseRing(T): TenSpcElt -> Rng`

`BaseField(T): TenSpcElt -> Fld`

Returns the base ring or field of the tensor T.

Example: Basic Properties of a Tensor (ex-7020de)

We demonstrate how to get basic properties of a tensor and what to expect as an output. We will construct a tensor $* : \mathrm{Mat}_{2\times 3}(\mathbb{Q}) \times \mathbb{Q}^3 \rightarrowtail \mathbb{Q}^2$ given by multiplication. Nearly all of this information is displayed when printing a tensor.

```
> K := Rationals();
> U2 := KMatrixSpace(K, 2, 3);
> U1 := VectorSpace(K, 3);
> U0 := VectorSpace(K, 2);
> mult := func< x | Eltseq(x[1]*Matrix(3,1,Eltseq(x[2]))) >;
> t := Tensor([* U2, U1, U0 *], mult);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 6 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
>
> Parent(t);
Tensor space of dimension 36 over Rational Field with valence 3
U2 : Full Vector space of degree 6 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
>
> Domain(t);
[*
    Full Vector space of degree 6 over Rational Field,

    Full Vector space of degree 3 over Rational Field
*]
>
> Codomain(t);
Full Vector space of degree 2 over Rational Field
>
> Valence(t);
3
>
> Frame(t);
[*
    Full Vector space of degree 6 over Rational Field,

    Full Vector space of degree 3 over Rational Field,

    Full Vector space of degree 2 over Rational Field
*]
> BaseRing(t);
Rational Field
```

TensorCategory(T): TenSpcElt -> TenCat

Returns the underlying tensor category of tensor T.

ChangeTensorCategory(T, C): TenSpcElt, TenCat -> TenSpcElt

ChangeTensorCategory(~T, C): TenSpcElt, TenCat

Returns the tensor T with the given category.

IsCovariant(T): TenSpcElt -> BoolElt

IsContravariant(T): TenSpcElt -> BoolElt

Returns **true** if the underlying category of the tensor T is covariant or contravariant.

Example: Category of a Right Module Tensor (ex-bef887)

We will construct a tensor from a right module, $* : \mathbb{Q}^2 \times \mathrm{Mat}_{2\times 2}(\mathbb{Q}) \rightarrowtail \mathbb{Q}^2$.

```
> K := Rationals();
> U := KMatrixSpace(K, 2, 2);
> V := VectorSpace(K, 2);
> mult := func< x | x[1]*x[2] >;
> t := Tensor([* V, U, V *], mult);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 2 over Rational Field
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
```

Because this tensor comes from a module, we want the tensor category to reflect this. Currently, the tensor category is the default homotopism category. We will keep everything about the category the same, except we will fuse coordinates 2 and 0. These changes could easily go unnoticed if no operators are constructed.

```
> TensorCategory(t);
Tensor category of valence 3 (->,->,->) ({ 1 },{ 2 },{ 0 })
> Cat := TensorCategory([1, 1, 1], {{2,0},{1}});
> Cat;
Tensor category of valence 3 (->,->,->) ({ 1 },{ 0, 2 })
> ChangeTensorCategory(~t, Cat);
> TensorCategory(t);
Tensor category of valence 3 (->,->,->) ({ 1 },{ 0, 2 })
> IsCovariant(t);
true
```

`NondegenerateTensor(T): TenSpcElt -> TenSpcElt, Hmtp`

Returns the nondegenerate tensor associated to T along with a homotopism from the given tensor to the returned nondegenerate tensor.

`IsNondegenerate(T): TenSpcElt -> BoolElt`

`IsDegenerate(T): TenSpcElt -> BoolElt`

Returns `true` if T is a degenerate or nondegenerate multilinear map.

Example: Nondegeneracy of a Tensor (ex-f6f9ff)

An important property for tensors is nondegeneracy: all radicals are trivial. First we create a tensor with degeneracy.

```
> K := GF(541);
> V := VectorSpace(K, 10);
> U := VectorSpace(K, 5);
> mult := function(x)
>   M := Matrix(3, 3, Eltseq(x[1])[2..10]);
>   v := VectorSpace(K, 3)!(Eltseq(x[2])[[1,3,5]]);
>   return Eltseq(v*M) cat [0,0];
> end function;
> t := Tensor([V, U, U], mult);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 10 over GF(541)
U1 : Full Vector space of degree 5 over GF(541)
U0 : Full Vector space of degree 5 over GF(541)
```

In this example, both $U_{\widehat{2}}^{\perp}$ and $U_{\widehat{1}}^{\perp}$ are nontrivial. We will construct the associated nondegenerate tensor s, which is given by $U_2/U_{\widehat{2}}^{\perp} \times U_1/U_{\widehat{1}}^{\perp} \rightarrowtail U_0$.

```
> IsNondegenerate(t);
false
> s, H := NondegenerateTensor(t);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 9 over GF(541)
U1 : Full Vector space of degree 3 over GF(541)
U0 : Full Vector space of degree 5 over GF(541)
> H;
Maps from U2 x U1 >-> U0 to V2 x V1 >-> V0.
U2 -> V2: Mapping from: Full Vector space of degree 10 over GF(541) to
Full Vector space of degree 9 over GF(541)
U1 -> V1: Mapping from: Full Vector space of degree 5 over GF(541) to
Full Vector space of degree 3 over GF(541)
U0 -> V0: Mapping from: Full Vector space of degree 5 over GF(541) to
Full Vector space of degree 5 over GF(541)
```

Image(T): TenSpcElt -> ModTupRng

Returns the image of the tensor T.

FullyNondegenerateTensor(T): TenSpcElt -> TenSpcElt, Hmtp

Returns the fully nondegenerate multilinear map associated to tensor T along with a cohomotopism from the given tensor to the returned tensor.

IsFullyNondegenerate(T): TenSpcElt -> BoolElt

Returns `true` if the tensor T is a fully nondegenerate multilinear map.

Example: Fully Nondegenerate Tensors (ex-f9b7b8)

We use the same tensor as the previous example illustrating the use of `NondegenerateTensor`.

```
> K := GF(541);
> V := VectorSpace(K, 10);
> U := VectorSpace(K, 5);
> mult := function(x)
>   M := Matrix(3, 3, Eltseq(x[1])[2..10]);
>   v := VectorSpace(K, 3)!(Eltseq(x[2])[[1,3,5]]);
>   return Eltseq(v*M) cat [0,0];
> end function;
> t := Tensor([V, U, U], mult);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 10 over GF(541)
U1 : Full Vector space of degree 5 over GF(541)
U0 : Full Vector space of degree 5 over GF(541)
```

Here, we want to construct a fully nondegenerate tensor. Of course, the tensor form the previous example is not fully nondegenerate as it is not degenerate, but we check that the image is not isomorphic to the codomain.

```
> IsFullyNondegenerate(t);
false
> Image(t);
Vector space of degree 5, dimension 3 over GF(541)
Generators:
(  1   0   0   0   0)
```

(continues on next page)

(continued from previous page)

```
(  0   1   0   0   0)
(  0   0   1   0   0)
Echelonized basis:
(  1   0   0   0   0)
(  0   1   0   0   0)
(  0   0   1   0   0)
```

Now we will construct the associated fully nondegenerate tensor: $U_2/U_{\widehat{2}}^{\perp} \times U_1/U_{\widehat{1}}^{\perp} \rightarrowtail U_2 * U_1$. Notice that the morphism between the original tensor and the fully nondegenerate tensor is a cohomotopism.

```
> s, H := FullyNondegenerateTensor(t);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 9 over GF(541)
U1 : Full Vector space of degree 3 over GF(541)
U0 : Vector space of degree 5, dimension 3 over GF(541)
Generators:
(  1   0   0   0   0)
(  0   1   0   0   0)
(  0   0   1   0   0)
Echelonized basis:
(  1   0   0   0   0)
(  0   1   0   0   0)
(  0   0   1   0   0)
> H;
Maps from U2 x U1 >-> U0 to V2 x V1 >-> V0.
U2 -> V2: Mapping from: Full Vector space of degree 10 over GF(541) to
Full Vector space of degree 9 over GF(541)
U1 -> V1: Mapping from: Full Vector space of degree 5 over GF(541) to
Full Vector space of degree 3 over GF(541)
U0 <- V0: Mapping from: Full Vector space of degree 5 over GF(541) to
Full Vector space of degree 5 over GF(541)
Composition of Mapping from: Full Vector space of degree 5 over GF(541)
to Full Vector space of degree 5 over GF(541) and
Mapping from: Vector space of degree 5, dimension 3 over GF(541) to Full
Vector space of degree 5 over GF(541)
```

IsAlternating(T): TenSpcElt -> BoolElt

Returns **true** if the tensor T is an alternating tensor.

IsAntisymmetric(T): TenSpcElt -> BoolElt

Returns **true** if the tensor T is an antisymmetric tensor.

IsSymmetric(T): TenSpcElt -> BoolElt

Returns **true** if the tensor T is a symmetric tensor.

Example: Symmetric Form from a Ternary Cubic (ex-6f9baa)

We will construct the multilinear form given by polarizing the homogeneous polynomial $f(x, y, z) = x^3 + y^3 + z^3 + xyz$. Since f is a symmetric polynomial, its multilinear form is also symmetric.

```
> K := Rationals();
> R<x,y,z> := PolynomialRing(K, 3);
> f := x^3 + y^3 + z^3 + x*y*z;
> t, p := Polarization(f);
> p;
6*$.1*$.4*$.7 + $.1*$.5*$.9 + $.1*$.6*$.8 + $.2*$.4*$.9 +
```

(continues on next page)

(continued from previous page)

```
    6*$.2*$.5*$.8 + $.2*$.6*$.7 + $.3*$.4*$.8 + $.3*$.5*$.7 +
    6*$.3*$.6*$.9
> t;
Tensor of valence 4, U3 x U2 x U1 >-> U0
U3 : Full Vector space of degree 3 over Rational Field
U2 : Full Vector space of degree 3 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 1 over Rational Field
```

The resulting homogeneous polynomial from polarizing is

$$p(x_1, x_2, x_3, y_1, y_2, y_3, z_1, z_2, z_3) = \sum_{\sigma \in S_3} (x_{1^\sigma} y_{2^\sigma} z_{3^\sigma} + 3x_{1^\sigma} y_{1^\sigma} z_{1^\sigma}).$$

```
> IsSymmetric(t);
true
> AsMatrices(t, 3, 1) eq AsMatrices(t, 2, 1);
true
> AsMatrices(t, 3, 1) eq AsMatrices(t, 3, 2);
true
> AsMatrices(t, 3, 1);
[
    [6 0 0]
    [0 0 1]
    [0 1 0],

    [0 0 1]
    [0 6 0]
    [1 0 0],

    [0 1 0]
    [1 0 0]
    [0 0 6]
]
```

Because the underlying field is $\mathbb{Q}$ and because t is symmetric, we know that t is not alternating nor antisymmetric.

```
> IsAlternating(t);
false
> IsAntisymmetric(t);
false
```

Tensors As Multilinear Maps

Regarding tensors as multilinear maps makes it possible to define the operations of composition and evaluation for tensors.

`x @ T: Tup, TenSpcElt -> Any`

Evaluates the tensor T at $x \in U_\nu \times \cdots \times U_1$. The entries can be elements from the vector space U_i or sequences that Magma can naturally coerce into the vector space U_i. In some circumstances, tensors come from algebraic objects (e.g. algebras), and in these cases the entries of x can be contained in the original algebraic object as well.

Example: Evaluating the Jacobi Identity Tensor `(ex-be5d59)`

Here we create the 4-tensor $\langle\,\rangle$ of an algebra A given by the Jacobi identity:

$$(x, y, z) \mapsto (xy)z + (yz)x + (zx)y.$$

Therefore, the algebra satisfies the Jacobi identity if $\langle A, A, A \rangle = 0$. We will also change the tensor category so that all the coordinates are fused together.

```
> A := MatrixAlgebra(GF(3), 3);
> JacobiID := func< x | x[1]*x[2]*x[3]+x[2]*x[3]*x[1]+x[3]*x[1]*x[2] >;
> Cat := TensorCategory([1 : i in [0..3]], {{0..3}});
> t, maps := Tensor([A : i in [0..3]], JacobiID, Cat);
> t;
Tensor of valence 4, U3 x U2 x U1 >-> U0
U3 : Full Vector space of degree 9 over GF(3)
U2 : Full Vector space of degree 9 over GF(3)
U1 : Full Vector space of degree 9 over GF(3)
U0 : Full Vector space of degree 9 over GF(3)
> TensorCategory(t);
Tensor category of valence 4 (->,->,->,->) ({ 0 .. 3 })
```

Even though our tensor originated over the algebra $A = \mathrm{Mat}_3(\mathbf{F}_3)$, the returned tensor is over vector spaces $\mathbf{F}_3^9$. However, the tensor t can still evaluate elements from $\mathrm{Mat}_3(\mathbf{F}_3)$ as well as $\mathbf{F}_3^9$. Observe that the out of t will be a vector regardless of the input. The second output at construction, the `List` of maps, can be used to map the vectors to $\mathrm{Mat}_3(\mathbf{F}_3)$.

```
> x := <A.1, A.2, A.2^2>;
> x;
<
    [1 0 0]
    [0 0 0]
    [0 0 0],

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

    [0 0 1]
    [1 0 0]
    [0 1 0]
>
> x @ t;
(2 0 0 0 1 0 0 0 0)
>
> phi := maps[1];
> x := <A.1 @ phi, A.2 @ phi, (A.2^2) @ phi>;
> x;
<(1 0 0 0 0 0 0 0 0), (0 1 0 0 0 1 1 0 0), (0 0 1 1 0 0 0 1 0)>
> x @ t;
(2 0 0 0 1 0 0 0 0)
```

Because `@` takes `Tup` as input, t can evaluate mixed tuples as well: where some entries are contained in $\mathrm{Mat}_3(\mathbf{F}_3)$ and other entries are contained in $\mathbf{F}_3^9$.

```
> x := <A.1, A.2 @ phi, Eltseq(A.2^2)>;
> x;
<
    [1 0 0]
    [0 0 0]
    [0 0 0],

    (0 1 0 0 0 1 1 0 0),
```

(continues on next page)

(continued from previous page)

```
    [ 0, 0, 1, 1, 0, 0, 0, 1, 0 ]
>
> <Type(i) : i in x>;
<AlgMatElt, ModTupFldElt, SeqEnum>
> x @ t;
(2 0 0 0 1 0 0 0 0)
```

`T * f: TenSpcElt, Map -> TenSpcElt`

`t * s: TenSpcElt, TenSpcElt -> TenSpcElt`

The tensor formed by composing the tensor T with the map f is constructed. If a tensor s is used instead of a `Map`, s must have valence ≤ 1.

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

Returns `true` if the tensors S and T are the same. Two tensors are equivalent if, and only if, they have the same tensor category, base ring, frame, and structure constants.

Example: Composing Tensors with Linear Maps (ex-0a17aa)

We start with the same tensor as the previous example: the tensor given by the Jacobi identity on the algebra $A = \mathrm{Mat}_3(\mathbf{F}_3)$.

```
> A := MatrixAlgebra(GF(3), 3);
> JacobiID := func< x | x[1]*x[2]*x[3]+x[2]*x[3]*x[1]+x[3]*x[1]*x[2] >;
> Cat := TensorCategory([1 : i in [0..3]], {{0..3}});
> t, maps := Tensor([A : i in [0..3]], JacobiID, Cat);
> t;
Tensor of valence 4, U3 x U2 x U1 >-> U0
U3 : Full Vector space of degree 9 over GF(3)
U2 : Full Vector space of degree 9 over GF(3)
U1 : Full Vector space of degree 9 over GF(3)
U0 : Full Vector space of degree 9 over GF(3)
> TensorCategory(t);
Tensor category of valence 4 (->,->,->,->) ({ 0 .. 3 })
```

The maps in `Maps` are vector space isomorphisms and map A to V. Suppose $\phi : A \to V$ is a vector space isomorphism. If we compose t with ϕ^{-1}, the returned tensor is *still* over vector spaces. The codomain is *not* A; this is because all tensors are over vector spaces. In fact, the returned tensor is exactly the same as t.

```
> phi := maps[1];
> t * (phi^-1);
Tensor of valence 4, U3 x U2 x U1 >-> U0
U3 : Full Vector space of degree 9 over GF(3)
U2 : Full Vector space of degree 9 over GF(3)
U1 : Full Vector space of degree 9 over GF(3)
U0 : Full Vector space of degree 9 over GF(3)
> t * (phi^-1) eq t;
true
```

Let $\mathcal{E} \subset A$ be an orthogonal frame—a set of primitive, orthogonal, idempotents. We will compose t with the linear transformation

$$a \mapsto \sum_{e \in \mathcal{E}} eae.$$

Because the codomain of t is a vector space, we will precompose this map by ϕ^{-1}.

```
> E := [A.1, A.2^-1*A.1*A.2, A.2^-2*A.1*A.2^2];
> E;
[
    [1 0 0]
    [0 0 0]
    [0 0 0],

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

    [0 0 0]
    [0 0 0]
    [0 0 1]
]
> f := map< A -> A | x :-> &+[ E[i]*x*E[i] : i in [1..3] ] >;
> f;
Mapping from: AlgMat: A to AlgMat: A given by a rule [no inverse]
> s := t*(phi^-1*f);
> s;
Tensor of valence 4, U3 x U2 x U1 >-> U0
U3 : Full Vector space of degree 9 over GF(3)
U2 : Full Vector space of degree 9 over GF(3)
U1 : Full Vector space of degree 9 over GF(3)
U0 : Full Vector space of degree 9 over GF(3)
> s eq t;
false
```

We can also wrap f as a 2-tensor and compose it with t.

```
> g := Tensor([A, A], func< x | x[1]@f >);
> g;
Tensor of valence 2, U1 >-> U0
U1 : Full Vector space of degree 9 over GF(3)
U0 : Full Vector space of degree 9 over GF(3)
> t * g eq s;
true
```

Operations with Bilinear Maps

Tensors of valence 3, also known as bilinear tensors, or as bilinear maps, are commonly described as distributive products. Examples include the product of two elements of an algebra, the action of a ring on a module, or an inner product. Magma supports these interpretations in two ways: by permitting an infix $x * t * y$ notation for a 3-tensor t, and a product $x * y$ notation for the evaluation of bilinear tensors. For the latter, this is achieved by creating special types `BmpU[Elt]`, `BmpV[Elt]`, and `BmpW[Elt]` for the terms of the frame of a bilinear tensor. For bilinear maps, we refer to the modules in the frame as $U \times V \rightarrowtail W$.

`x * T: Any, TenSpcElt -> Any`

`T * y: TenSpcElt, Any -> Any`

Given a bilinear tensor T framed by $[U, V, W]$, $x * t$ returns the action on the right as a linear map $L : V \to W$ given by $vL = x * v$ if x is an element of U. If x is a subspace of U, then this returns a subspace of the tensor space T with frame $V \rightarrowtail W$. For the left action use $T * y$ instead. If T is valence 1, then the image of either x or y is returned. Therefore, the possible outputs are a tensor space `TenSpc`, a tensor `TenSpcElt`, or a vector `ModTupFld`.

Related to this intrinsic is the following: using tensor spaces with the infix notation.

`x * T: Any, TenSpc -> Any`

`T * y: TenSpc, Any -> Any`

Given a subspace of bilinear tensors, return the subspace generated by all $x * t$, for $t \in T$. This is either a tensor space or a vector space.

Example: Infix Notation for Bilinear Maps (ex-4ade75)

We demonstrate the infix notation by constructing the tensor in $A = \mathrm{Mat}_2(\mathbb{Q})$ given by multiplication.

```
> A := MatrixAlgebra(Rationals(), 2);
> t := Tensor(A);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 4 over Rational Field
```

Like tensor evaluation, the infix notation will accept elements of a vector space (or objects that Magma can easily coerce into a vector space) or elements from the original algebraic object. We will use the infix notation to construct the 2-tensor $* : A \rightarrowtail A$, where $X \mapsto MX$, for some matrix M.

```
> M := A![0, 1, 0, 0];
> M;
[0 1]
[0 0]
> s := M*t;
> s;
Tensor of valence 2, U1 >-> U0
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 4 over Rational Field
> AsMatrices(s, 1, 0);
[
    [0 0 0 0]
    [0 0 0 0]
    [1 0 0 0]
    [0 1 0 0]
]
```

From the structure constants above, evaluating `M*t` at `V.3` should output `W.1`. Furthermore, the image of `M*t` is 2-dimensional in W.

```
> M*t*[0, 0, 1, 0];
(1 0 0 0)
> M*t*VectorSpace(Rationals(), 4);
Vector space of degree 4, dimension 2 over Rational Field
Generators:
(1 0 0 0)
(0 1 0 0)
Echelonized basis:
(1 0 0 0)
(0 1 0 0)
```

If we switch the order and multiply t by U on the left first, we will get a tensor space T. Because that tensor space came from t, which originally came from algebraic objects, we can use the returned tensor space to evaluate M. An arbitrary tensor space, however, would not evaluate M unless it is an appropriate vector.

```
> T := VectorSpace(Rationals(), 4)*t;
> T;
Tensor space of dimension 4 over Rational Field with valence 2
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 4 over Rational Field
> T*M;
Vector space of degree 4, dimension 2 over Rational Field
Generators:
(0 1 0 0)
(0 0 0 1)
Echelonized basis:
(0 1 0 0)
(0 0 0 1)
```

The next style of notation we support is the product notation, $x * y$. In order to use this style, the user needs to coerce both x and y into the `LeftDomain` and `RightDomain` respectively.

`x * y: BmpUElt, BmpVElt -> Any`

`x * y: BmpU, BmpV -> Any`

`x * y: BmpUElt, BmpV -> Any`

`x * y: BmpU, BmpVElt -> Any`

If x and y are associated to the bilinear map B, these operations return `<x,y> @ B`.

`LeftDomain(B): TenSpcElt -> BmpU`

The left domain U of the bilinear map B with frame $[U, V, W]$ is returned. This operation is used when enabling the use of infix notation.

`RightDomain(B): TenSpcElt -> BmpV`

The right domain U of the bilinear map B with frame $[U, V, W]$ is returned. This operation is used when enabling the use of infix notation.

`IsCoercible(U, x): BmpU, Any -> BoolElt, BmpUElt`

`IsCoercible(V, x): BmpV, Any -> BoolElt, BmpVElt`

`U ! x: BmpU, Any -> BmpUElt`

`V ! x: BmpV, Any -> BmpVElt`

Returns `true` if x can be coerced into the left domain (right domain) U (V) of a bilinear map. If successful it returns the coerced element.

Example: Product Notation for Bilinear Maps `(ex-90f3f7)`

We demonstrate the product notation for tensors of valence 3 using a tensor derived from a p-group. Suppose G is a p-group and $[,] : U \times V \rightarrowtail W$ is the tensor given by commutation where $U = V = G/\eta_2$ and $W = \eta_2/\eta_3$, where η_i denotes the ith term of the exponent-p central series of G.

```
> G := SmallGroup(512, 10^6);
> t := pCentralTensor(G);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 5 over GF(2)
U1 : Full Vector space of degree 5 over GF(2)
U0 : Full Vector space of degree 4 over GF(2)
> U := LeftDomain(t);
> V := RightDomain(t);
> U;
Bimap space U: Full Vector space of degree 5 over GF(2)
> V;
Bimap space V: Full Vector space of degree 5 over GF(2)
```

Like with the other styles of notation (infix and tuple), users can evaluate elements from the original algebraic object, vectors from the vector spaces, and even sequences that Magma can easily coerce into vector spaces. To use the product notation, coerce elements into the `LeftDomain` and `RightDomain`.

```
> x := U!(G.1*G.2*G.4);
> y := V![1,0,0,0,0];
> x;
Bimap element of U: (1 1 0 1 0)
> y;
Bimap element of V: (1 0 0 0 0)
> x*y;
(1 0 0 1)
```

We can take this further and evaluate subspaces of U or V.

```
> H := sub< G | G.2,G.4 >;
> U!H * V!G.1;
Vector space of degree 4, dimension 2 over GF(2)
Generators:
(0 0 0 1)
(1 0 0 0)
Echelonized basis:
(1 0 0 0)
(0 0 0 1)
> U!H * V;
Vector space of degree 4, dimension 3 over GF(2)
Generators:
(1 0 0 0)
(0 0 1 0)
(0 0 0 1)
(1 0 0 0)
(1 0 0 0)
Echelonized basis:
(1 0 0 0)
(0 0 1 0)
(0 0 0 1)
```

Parent(x): BmpUElt -> BmpU

Parent(x): BmpVElt -> BmpV

The parent space of the bilinear map element x is returned.

Parent(X): BmpU -> TenSpcElt

Parent(X): BmpV -> TenSpcElt

The original bilinear map from which the space X originated is returned.

u1 **eq** u2: BmpUElt, BmpUElt -> BoolElt

Returns **true** if the elements $u1$ and $u2$ from the left domain of a bilinear map are equal.

v1 **eq** v2: BmpUElt, BmpUElt -> BoolElt

Returns **true** if the elements $v1$ and $v2$ from the right domain of a bilinear map are equal.

U1 **eq** U2: BmpU, BmpU -> BoolElt

Returns **true** if the subspaces $U1$ and $U2$ of the left domain of a bilinear map are equal.

V1 **eq** V2: BmpV, BmpV -> BoolElt

Returns **true** if the subspaces $V1$ and $V2$ of the right domain of a bilinear map are equal.

Example: Domains of a Bilinear Map (ex-77d198)

We will construct the same tensor as the previous example.

```
> G := SmallGroup(512, 10^6);
> t := pCentralTensor(G);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 5 over GF(2)
U1 : Full Vector space of degree 5 over GF(2)
U0 : Full Vector space of degree 4 over GF(2)
> U := LeftDomain(t);
> V := RightDomain(t);
> U;
Bimap space U: Full Vector space of degree 5 over GF(2)
> V;
Bimap space V: Full Vector space of degree 5 over GF(2)
```

The product notation has some basic functions for comparing objects and retrieving information.

```
> V!G.1 eq V![1,0,0,0,0];
true
> Parent(U);
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 5 over GF(2)
U1 : Full Vector space of degree 5 over GF(2)
U0 : Full Vector space of degree 4 over GF(2)
> Parent(U) eq t;
true
```

Manipulating Tensor Data

The data associated with a tensor can be accessed in different ways. In the case of tensors given by structure constants this access is the multidimensional analog of choosing a row or column of a matrix. Other operations are a generalization of the transpose of a matrix. Magma does these operations with a degree of efficiency, that is, it may not physically move the values in a structure constant sequence but instead permute the lookup coordinates of the values.

`Slice(T, grid): TenSpcElt, [SetEnum] -> SeqEnum`

`InducedTensor(T, grid): TenSpcElt, [SetEnum] -> TenSpcElt`

Returns the slice of the structure constants running through the given grid. For a tensor T framed by free modules $[U_\nu, \ldots, U_0]$ with $d_i = \dim U_i$, a *grid* is a sequence $[G_\nu, \ldots, G_0]$ of subsets $G_i \subseteq \{1, \ldots, d_i\} \cup \{-d_i, \ldots, -1\}$. If an entry $g \in G_i$ is negative, it will be taken to mean $d_i + g + 1$, so -1 would be equivalent to d_i. The slice is the list of entries in the structure constants of the tensor indexed by $G_\nu \times \cdots \times G_0$. The intrinsic `Slice` returns the structure constants whereas `InducedTensor` produces a tensor with these structure constants.

Example: Slicing a Tensor (ex-ff7268)

We will construct a tensor $* : \mathbb{Q}^4 \times \mathbb{Q}^3 \rightarrowtail \mathbb{Q}^2$ with a structure constants sequence equal to $[1, \ldots, 24]$. If every $G_i = \{1, \ldots, d_i\}$, then the result is the same as `Eltseq`.

```
> U := VectorSpace(Rationals(),4);
> V := VectorSpace(Rationals(),3);
> W := VectorSpace(Rationals(),2);
> T := TensorSpace([U, V, W]);
> t := T![1..24];
> t;
Tensor of valence 3, U2 x U1 >-> U0
```

(continues on next page)

(continued from previous page)

```
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
> Slice(t, [{1..4},{1..3},{1..2}]) eq Eltseq(t);
true
```

Now we will slice with the following grid $[\{1,\ldots,4\},\{2\},\{1\}]$. Compare this with the product $U * v_2$.

```
> [ U.i*t*V.2 : i in [1..4]];
[
    (3 4),
    ( 9 10),
    (15 16),
    (21 22)
]
> Slice(t, [{1..4},{2},{1}]);
[ 3, 9, 15, 21 ]
```

If, instead, we slice W at its last basis vector, we get the following.

```
> Slice(t, [{1..4},{2},{-1}]);
[ 4, 10, 16, 22 ]
```

Notice that if we use -1 and 2 in the last set of the grid we get the same output we got above.

```
> Slice(t, [{1..4},{2},{-1,2}]);
[ 4, 10, 16, 22 ]
```

Now we will compare `Slice` and `InducedTensor`. `InducedTensor` is basically a `Tensor` wrapping the `Slice` function.

```
> s := InducedTensor(t, [{1..4}, {2}, {1,2}]);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 1 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
> s2 := Tensor([4, 1, 2], Slice(t, [{1..4}, {2}, {1,2}]));
> s2;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 1 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
>
> s eq s2;
true
> Eltseq(s);
[ 3, 4, 9, 10, 15, 16, 21, 22 ]
```

`SliceAsMatrices(T, grid, a, b): TenSpcElt, [SetEnum], RngIntElt, RngIntElt -> SeqEnum`

Given a tensor T, a set and integers a and b, returns a sequence of matrices whose output is equivalent to composing `InducedTensor` and `AsMatrices`. This intrinsic will be slightly faster than actually composing those two functions together as a tensor is not constructed with `SliceAsMatrices`.

Example: Slices as Matrices (ex-436f03)

We will create the same tensor as the previous example.

```
> t := Tensor(Rationals(), [4,3,2], [1..24]);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
> AsMatrices(t, 1, 0);
[
    [1 2]
    [3 4]
    [5 6],

    [ 7  8]
    [ 9 10]
    [11 12],

    [13 14]
    [15 16]
    [17 18],

    [19 20]
    [21 22]
    [23 24]
]
```

Now we will slice up this sequence of matrices. We will remove the second row and the second and third matrix from the sequence above.

```
> SliceAsMatrices(t, [{1,-1}, {1,-1}, {1,2}], 1, 0);
[
    [1 2]
    [5 6],

    [19 20]
    [23 24]
]
```

`Foliation(T, i): TenSpcElt, RngIntElt -> Mtrx`

If T is a tensor with frame $\hom(U_\nu \otimes \cdots \otimes U_1, U_0)$, and i is the index of some term of the frame of T, this intrinsic returns the matrix representing the linear map $U_i \to \hom(\bigotimes_{j \neq i} U_i, U_0)$ using the bases of each U_j. If $i = 0$, then the returned matrix is given by the representation $U_0^* \to \hom(\bigotimes U_i, K)$.

Example: Exfoliating and Foliating a Tensor `(ex-c6ce4b)`

We will, again, construct the same tensor, $* : \mathbb{Q}^4 \times \mathbb{Q}^3 \rightarrowtail \mathbb{Q}^2$, from the previous two examples whose structure constants sequence is $[1, \ldots, 24]$.

```
> K := Rationals();
> Forms := [Matrix(K, 3, 2, [6*i+1..6*(i+1)]) : i in [0..3]];
> t := Tensor(Forms, 1, 0);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
```

We will use `Foliation` to compute the 2-radical of t. (This is essentially what `Radical` does in Magma.) That is, we will compute the subspace $U_{\hat{2}}^{\perp} \leq \mathbb{Q}^4$ such that $U_{\hat{2}}^{\perp} * \mathbb{Q}^3 = 0$. This computation can be regarded as a nullspace computation.

```
> F2 := Foliation(t, 2);
> F2;
[ 1  2  3  4  5  6]
[ 7  8  9 10 11 12]
[13 14 15 16 17 18]
[19 20 21 22 23 24]
> R := Nullspace(F2);
> R;
Vector space of degree 4, dimension 2 over Rational Field
Echelonized basis:
( 1  0 -3  2)
( 0  1 -2  1)
```

We claim this is the 2-radical of t. Our claim is verified if `R*t` is a 0-dimensional subspace of the tensor space T with frame $\mathbb{Q}^3 \rightarrowtail \mathbb{Q}^2$. In other words, `R*t` is the trivial 2-tensor.

```
> R*t;
Tensor space of dimension 0 over Rational Field with valence 2
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
> Dimension(R*t);
0
```

`AsTensorSpace(T, i): TenSpcElt, RngIntElt -> TenSpc, Mtrx`

If T is a tensor contained in $\hom(U_v \otimes \cdots \otimes U_1, U_0)$, and i is the index of some term of the frame of T, this intrinsic returns the associated tensor space of T at $i > 0$ together with a matrix given by the foliation of T at i. The returned tensor space is framed by $U_v \times \cdots \times U_{i+1} \times U_{i-1} \times \cdots \times U_1 \rightarrowtail U_0$ and is generated by the tensors T_u for each u in the basis of U_i. For $i = 0$, use `AsCotensorSpace`.

`AsCotensorSpace(T): TenSpcElt -> TenSpc, Mtrx`

Given a tensor T, this intrinsic returns the associated cotensor space of T together with a matrix given by the foliation of T at 0. The returned cotensor space has frame $U_\nu \times \cdots \times U_1 \rightarrowtail K$ and is generated by the tensors T_f for each f in the basis of U_0^*. In the case that T is a bilinear map, this is equivalent to the cotensor space generated by the `SystemOfForms`.

Example: Tensor Space at a Coordinate (ex-8523ef)

We begin by creating a 4-tensor and constructing the associated tensor space at the third coordinate. `AsCotensorSpace` works similarly but when $a = 0$.

```
> t := Tensor(Rationals(), [5,4,3,2], [1..120]);
> t;
Tensor of valence 4, U3 x U2 x U1 >-> U0
U3 : Full Vector space of degree 5 over Rational Field
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
> T := AsTensorSpace(t, 3);
> T;
Tensor space of dimension 2 over Rational Field with valence 3
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
```

The dimension of T cannot be larger than 5 because that is the dimension of U_3. However, T is 2-dimensional. Slicing t as a sequence of matrices will illuminate why T is 2-dimensional.

```
> F := [SliceAsMatrices(t, [{k},{1..4},{1..3},{1,2}], 2, 1) :
>     k in [1..5]];
```

Here, F is a sequence of a system of forms for t, one for each basis vector in U_3. If T were 5-dimensional, the systems of forms in F would be linearly independent. However, it is not, and evidently, three of the five systems of forms are linear combinations two systems of forms. We will determine the linear combinations. The first two systems of forms are independent.

```
> F[1];
[
    [ 1  3  5]
    [ 7  9 11]
    [13 15 17]
    [19 21 23],

    [ 2  4  6]
    [ 8 10 12]
    [14 16 18]
    [20 22 24]
]
> F[2];
[
    [25 27 29]
    [31 33 35]
    [37 39 41]
    [43 45 47],

    [26 28 30]
    [32 34 36]
    [38 40 42]
    [44 46 48]
]
```

So `F[3]` is `2*F[2]-F[1]`, and the rest of the linear combinations are filled in.

```
> F[3];
[
    [49 51 53]
    [55 57 59]
    [61 63 65]
    [67 69 71],

    [50 52 54]
    [56 58 60]
    [62 64 66]
    [68 70 72]
]
> Tensor(F[3], 2, 1) eq 2*Tensor(F[2], 2, 1) - Tensor(F[1], 2, 1);
true
> Tensor(F[4], 2, 1) eq 3*Tensor(F[2], 2, 1) - 2*Tensor(F[1], 2, 1);
true
> Tensor(F[5], 2, 1) eq 4*Tensor(F[2], 2, 1) - 3*Tensor(F[1], 2, 1);
true
```

So, indeed, T is the tensor space generated by the tensors in `F`. Note that the dimension of the 3-radical of t is 3.

```
> SystemOfForms(T.1) eq F[1];
true
> SystemOfForms(T.2) eq F[2];
true
```

(continues on next page)

(continued from previous page)

```
> Radical(t, 3);
Vector space of degree 5, dimension 3 over Rational Field
Echelonized basis:
( 1  0  0 -4  3)
( 0  1  0 -3  2)
( 0  0  1 -2  1)

[ 0  0  0  0  1]
[ 0  0  0  1  0]
[ 0  0  1 -2  1]
[ 0  1  0 -3  2]
[ 1  0  0 -4  3]
```

AsTensor(S): TenSpc -> TenSpcElt

Given a tensor space S, this intrinsic returns a tensor belonging to S. If S is contravariant, then the returned tensor has the frame $U_\nu \times \cdots \times U_1 \rightarrowtail S$, where S is thought of as a free K-module.. If S is covariant, then the returned tensor has the frame $S \times U_\nu \times \cdots \times U_1 \rightarrowtail U_0$. Note that **AsTensor** is "inverse" to **AsCotensorSpace** and **AsTensorSpace** when $i = \nu$.

Example: Recovering a Tensor from Its Tensor Space (ex-a38c03)

We will construct the same tensor as the previous example and turn it into a tensor space at the third coordinate.

```
> t := Tensor(Rationals(), [5,4,3,2], [1..120]);
> t;
Tensor of valence 4, U3 x U2 x U1 >-> U0
U3 : Full Vector space of degree 5 over Rational Field
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
> T := AsTensorSpace(t, 3);
> T;
Tensor space of dimension 2 over Rational Field with valence 3
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
```

Now we are going to "recover" t by creating a tensor from T. However, it is not equal to t because the 3-radical of the new tensor is trivial.

```
> s := AsTensor(T);
> s;
Tensor of valence 4, U3 x U2 x U1 >-> U0
U3 : Full Vector space of degree 2 over Rational Field
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
> Radical(s, 3);
Vector space of degree 2, dimension 0 over Rational Field

[0 1]
[1 0]
```

We can even see the same sequences of matrices in s.

```
> AsMatrices(s, 2, 1);
[
    [ 1  3  5]
    [ 7  9 11]
    [13 15 17]
    [19 21 23],

    [ 2  4  6]
    [ 8 10 12]
    [14 16 18]
    [20 22 24],

    [25 27 29]
    [31 33 35]
    [37 39 41]
    [43 45 47],

    [26 28 30]
    [32 34 36]
    [38 40 42]
    [44 46 48]
]
```

63.2.3 Invariants of Tensors

In this subsection, we detail functions to construct invariants of tensors. To access the projections or the objects acting on a specific factor U_i, the following function(s) should be used.

Induce(X, i): AlgMat, RngIntElt -> Map, AlgMat

Induce(X, i): AlgMatLie, RngIntElt -> Map, AlgMatLie

Induce(X, i): GrpMat, RngIntElt -> Map, GrpMat

Returns the projection from the given object to the induced sub-object on the ith coordinate and the induced sub-object of the associated tensor.

Example: Inducing Derivations of a Symplectic Form (ex-4be7ac)

To demonstrate how to Induce, we construct the 2-dimensional symplectic form on $K = \mathrm{GF}(3)$.

```
> t := Tensor(GF(3), [2, 2, 1], [0, 1, 2, 0]);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 2 over GF(3)
U1 : Full Vector space of degree 2 over GF(3)
U0 : Full Vector space of degree 1 over GF(3)
> IsAlternating(t);
true
```

This tensor has a nontrivial derivation algebra, isomorphic to $K^2 \oplus \mathrm{SL}_2(3)$. However, Der(t) is represented in $\mathrm{End}(\mathrm{U}_2) \times \mathrm{End}(\mathrm{U}_1) \times \mathrm{End}(\mathrm{U}_0)$. We will induce the action on the 1st coordinate.

```
> D := DerivationAlgebra(t);
> D.1;
[1 0 0 0 0]
[0 1 0 0 0]
[0 0 1 0 0]
[0 0 0 1 0]
[0 0 0 0 2]
```

(continues on next page)

(continued from previous page)

```
> D.2;
[1 0 0 0 0]
[2 0 0 0 0]
[0 0 2 0 0]
[0 0 2 1 0]
[0 0 0 0 2]
> pi, D1 := Induce(D, 1);
> D1;
Matrix Lie Algebra of degree 2 over Finite field of size 3
> pi;
Mapping from: AlgMatLie: D to AlgMatLie: D1 given by a rule [no inverse]
```

Now we can see that have the action of D on U_1.

```
> D1.1;
[1 0]
[0 1]
> D1.2;
[2 0]
[2 1]
```

We include a partner intrinsic to `Include` and that is the following procedure.

`DerivedFrom(~X, t, C, RC : parameters): Any, TenSpcElt, {RngIntElt}, {RngIntElt}`

```
Fused: BoolElt                    Default: true
```

Includes the following tensor data in the matrix object X: the tensor t, the relevant coordinates which are in the set $C \subseteq \{0, \dots, \nu\}$, and the coordinates $RC \subseteq C$ for which the object is represented on. The reason for the subset C is so that we know the coordinates which can be induced on, see `Induce`. Currently, this only works for objects of type `AlgMat`, `AlgMatLie`, `GrpMat`, and `ModMatFld`. It is assumed that X is block diagonal, whose blocks starting from the top left go in decreasing order in the coordinates—in the same way the coordinates of the frame decrease from left to right for a tensor.

Standard Invariants

We integrate the invariant theory associated to bilinear and multilinear maps into the realm of tensors.

`Radical(T, s): TenSpcElt, RngIntElt -> ModTupRng`

Returns the sth-radical of T as a subspace of U_s and a matrix that splits the radical in V_s. This is the subspace

$$U_{\bar{s}}^{\perp} = \{u_s \in U_s : \forall\, |u_{\bar{s}}\rangle\,,\ \langle T|u\rangle = 0\}\,.$$

`Radical(T): TenSpcElt -> Tup`

Returns the tuple of all the s-radicals for each $s \in \{1, \dots, \nu\}$.

`Coradical(T): TenSpcElt -> ModTupRng, Map`

Returns the coradical of T and a surjection from the codomain to the coradical. This is the quotient $U_0/\langle T|U_\nu, \dots, U_1\rangle$.

Example: Radicals of the Commutator Tensor `(ex-fef917)`

We will construct the tensor for multiplication in $\mathrm{GL}_3(\mathbb{Q})$, or equivalently, the commutator tensor of $\mathrm{Mat}_3(\mathbb{Q})$. Because $\mathrm{GL}_3(\mathbb{Q})$ contains the center of $\mathrm{Mat}_3(\mathbb{Q})$, there will be a 2- and 1-radical.

```
> K := Rationals();
> A := MatrixAlgebra(K, 3);
> t, phi := CommutatorTensor(A);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 9 over Rational Field
U1 : Full Vector space of degree 9 over Rational Field
U0 : Full Vector space of degree 9 over Rational Field
>
> R2 := Radical(t, 2);
> R2.1 @@ phi;
[1 0 0]
[0 1 0]
[0 0 1]
> Radical(t);
<
    Vector space of degree 9, dimension 1 over Rational Field
    Echelonized basis:
    (1 0 0 0 1 0 0 0 1),

    Vector space of degree 9, dimension 1 over Rational Field
    Echelonized basis:
    (1 0 0 0 1 0 0 0 1)
>
```

Similarly, the image of t will be 8-dimensional in $\mathbb{Q}^9$, so the coradical is $\mathbb{Q}$.

```
> Image(t);
Vector space of degree 9, dimension 8 over Rational Field
Generators:
( 1  0  0  0  0  0  0  0 -1)
( 0  1  0  0  0  0  0  0  0)
( 0  0  1  0  0  0  0  0  0)
( 0  0  0  1  0  0  0  0  0)
( 0  0  0  0  1  0  0  0 -1)
( 0  0  0  0  0  1  0  0  0)
( 0  0  0  0  0  0  1  0  0)
( 0  0  0  0  0  0  0  1  0)
Echelonized basis:
( 1  0  0  0  0  0  0  0 -1)
( 0  1  0  0  0  0  0  0  0)
( 0  0  1  0  0  0  0  0  0)
( 0  0  0  1  0  0  0  0  0)
( 0  0  0  0  1  0  0  0 -1)
( 0  0  0  0  0  1  0  0  0)
( 0  0  0  0  0  0  1  0  0)
( 0  0  0  0  0  0  0  1  0)
> Coradical(t);
Full Vector space of degree 1 over Rational Field
Mapping from: Full Vector space of degree 9 over Rational Field to Full
Vector space of degree 1 over Rational Field
```

We include some well-known polynomial invariants for bilinear maps.

`Discriminant(B): TenSpcElt -> RngMPolElt`

Returns the discriminant of the bilinear map.

Example: Discriminant of the Octonions (ex-8de961)

We will compute the discriminant of the tensor $\cdot : O \times O \rightarrowtail O$. The discriminant of this tensor is homogeneous of degree 8 with 330 terms.

```
> A := OctonionAlgebra(GF(7), -1, -1, -1);
> t := Tensor(A);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 8 over GF(7)
U1 : Full Vector space of degree 8 over GF(7)
U0 : Full Vector space of degree 8 over GF(7)
> R<a,b,c,d,e,f,g,h> := PolynomialRing(GF(7), 8);
> disc := R!Discriminant(t);
> Degree(disc);
8
> IsHomogeneous(disc);
true
> #Terms(disc);
330
```

However, if we factor the discriminant, then we can see that `disc` $= \left(x_1^2 + \cdots x_8^2\right)^4$.

```
> Factorization(disc);
[
    <a^2 + b^2 + c^2 + d^2 + e^2 + f^2 + g^2 + h^2, 4>
]
```

`Pfaffian(B): TenSpcElt -> RngMPolElt`

Returns the Pfaffian of the antisymmetric bilinear map.

Example: Pfaffians of Genus 2 Groups `(ex-4cc2d2)`

In this demonstration, we pull from [BMW17]. A p-group G has genus 2, if the image of the exponent-p central tensor of G is 2-dimensional over the centroid. One of the algorithms in [BMW17] to decide isomorphism of such groups depends on the Pfaffian of the tensor.

First, we create two 3-groups with genus 2. The first group we create as a quotient of the Sylow 3-subgroup of $\mathrm{GL}(3, \mathrm{GF}(3^5))$.

```
> P := ClassicalSylow(GL(3, 3^5), 3);
> P := PCPresentation(UnipotentMatrixGroup(P));
> Z := Center(P);
> N := sub< Z | [Random(Z) : i in [1..3]] >;
> G := P/N;
```

The second group we create will be a quotient of the Sylow 3-subgroup of the of $\mathrm{GL}(3, \mathrm{GF}(9)) \times \mathrm{GL}(3, \mathrm{GF}(27))$.

```
> A := ClassicalSylow(GL(3, 9), 3);
> B := ClassicalSylow(GL(3, 27), 3);
> A := PCPresentation(UnipotentMatrixGroup(A));
> B := PCPresentation(UnipotentMatrixGroup(B));
> Q, inc := DirectProduct(A, B);
> ZA := Center(A);
> ZB := Center(B);
> gens := [(ZA.i@inc[1])*(ZB.i@inc[2])^-1 : i in [1..2]]
>     cat [ZB.3@inc[2]];
> M := sub< Q | gens >;
> H := Q/M;
```

Now we will construct the exponent-p central tensors of G and H. From the way we have created the groups, $G \cong H$ if, and only if, their tensors are pseudo-isometric.

```
> t := pCentralTensor(G);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 10 over GF(3)
U1 : Full Vector space of degree 10 over GF(3)
U0 : Full Vector space of degree 2 over GF(3)
>
> s := pCentralTensor(H);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 10 over GF(3)
U1 : Full Vector space of degree 10 over GF(3)
U0 : Full Vector space of degree 2 over GF(3)
```

An implication of the main algorithm in [BMW17] is that if the splitting behavior of the Pfaffians are different, then the groups are not isomorphic. Therefore, because the Pfaffian of G has a different splitting behavior from the Pfaffian of H, we conclude that $G \ncong H$.

```
> R<x,y> := PolynomialRing(GF(3), 2);
> f := R!Pfaffian(t);
> g := R!Pfaffian(s);
> f;
x^5 + x^3*y^2 + 2*x^2*y^3 + 2*x*y^4 + y^5
> g;
x^5 + 2*x^4*y + x^3*y^2 + 2*x^2*y^3 + x*y^4 + y^5
> Factorization(f), Factorization(g);
[
    <x^5 + x^3*y^2 + 2*x^2*y^3 + 2*x*y^4 + y^5, 1>
]
[
    <x^2 + x*y + 2*y^2, 1>,
    <x^3 + x^2*y + x*y^2 + 2*y^3, 1>
]
```

63.2.4 Exporting Tensors

In the previous sections, we used groups, rings, and algebras to define tensors, but tensors can be used to define algebraic structures as well. In this section, we describe ways to build groups and algebras from tensors. Currently, all intrinsics in this section only support 3-tensors.

`HeisenbergAlgebra(T): TenSpcElt -> AlgGen`

Returns the Heisenberg algebra A induced by the bilinear tensor $T : U \times V \rightarrowtail W$. The algebra A depends on the tensor category of T. If the tensor category forces equality between U, V, and W, then $A \cong U$ as vector spaces and is a nonassociative algebra (i.e. not necessarily associative). If the tensor category forces equality between U and V, then $A \cong U \oplus W$ as vector spaces and is a nilpotent algebra where $C(A) \geq W$ and $A^2 \leq W$. Otherwise, $A \cong U \oplus V \oplus W$ as vector spaces, and using $*$ for T,

$$(u, v, w) \cdot (u', v', w') = (0, 0, u * v').$$

Example: Heisenberg Algebras from Tensors `(ex-182a22)`

We will demonstrate some of the nuances of `HeisenbergAlgebra` and how it interacts with the tensor category of the given tensor. We will use the same tensor throughout but changing the categories. The frame of the tensor is $\mathbb{Q}^3 \times \mathbb{Q}^3 \rightarrowtail \mathbb{Q}^3$.

```
> t := Tensor(Rationals(), [3, 3, 3], [i : i in [1..27]]);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 3 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 3 over Rational Field
> SystemOfForms(t);
[
    [ 1  4  7]
    [10 13 16]
    [19 22 25],

    [ 2  5  8]
    [11 14 17]
    [20 23 26],

    [ 3  6  9]
    [12 15 18]
    [21 24 27]
]
> Radical(t);
<
    Vector space of degree 3, dimension 1 over Rational Field
    Echelonized basis:
    ( 1 -2  1),

    Vector space of degree 3, dimension 1 over Rational Field
    Echelonized basis:
    ( 1 -2  1)
>
> Image(t);
Vector space of degree 3, dimension 2 over Rational Field
Generators:
( 1  0 -1)
( 0  1  2)
Echelonized basis:
( 1  0 -1)
( 0  1  2)
```

The default tensor category of t above is the homotopism category where none of the modules are fused together. Therefore, if we construct the Heisenberg algebra of t, the resulting algebra will be 9-dimensional. Looking at the system of forms, the first 6 matrices will be zero, and the last 3 matrices will contain the above system of forms as a 3×3 block, starting at the $(1, 4)$ entry.

```
> A := HeisenbergAlgebra(t);
> A;
Algebra of dimension 9 with base ring Rational Field
> Center(A);
Algebra of dimension 5 with base ring Rational Field
> SystemOfForms(Tensor(A))[7..9];
[
    [ 0  0  0  1  4  7  0  0  0]
    [ 0  0  0 10 13 16  0  0  0]
    [ 0  0  0 19 22 25  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
```

(continues on next page)

(continued from previous page)

```
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0],

    [ 0  0  0  2  5  8  0  0  0]
    [ 0  0  0 11 14 17  0  0  0]
    [ 0  0  0 20 23 26  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0],

    [ 0  0  0  3  6  9  0  0  0]
    [ 0  0  0 12 15 18  0  0  0]
    [ 0  0  0 21 24 27  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
]
```

Now we will fuse U and V together with a new category. The resulting algebra will then be 6-dimensional. We will also look at the system of forms of A to see the structure constants of A.

```
> NilCat := TensorCategory([1, 1, 1], {{2,1},{0}});
> NilCat;
Tensor category of valence 3 (->,->,->) ({ 0 },{ 1, 2 })
> ChangeTensorCategory(~t, NilCat);
> A := HeisenbergAlgebra(t);
> A;
Algebra of dimension 6 with base ring Rational Field
> Center(A);
Algebra of dimension 4 with base ring Rational Field
> A^2;
Algebra of dimension 2 with base ring Rational Field
> SystemOfForms(Tensor(A))[4..6];
[
    [ 1  4  7  0  0  0]
    [10 13 16  0  0  0]
    [19 22 25  0  0  0]
    [ 0  0  0  0  0  0]
    [ 0  0  0  0  0  0]
    [ 0  0  0  0  0  0],

    [ 2  5  8  0  0  0]
    [11 14 17  0  0  0]
    [20 23 26  0  0  0]
    [ 0  0  0  0  0  0]
    [ 0  0  0  0  0  0]
    [ 0  0  0  0  0  0],

    [ 3  6  9  0  0  0]
```

(continues on next page)

(continued from previous page)

```
    [12 15 18  0  0  0]
    [21 24 27  0  0  0]
    [ 0  0  0  0  0  0]
    [ 0  0  0  0  0  0]
    [ 0  0  0  0  0  0]
]
```

Finally, we will put a tensor category on t where U, V, and W are all fused together. The Heisenberg algebra from this tensor is 3-dimensional, and the product is exactly the tensor t.

```
> AlgCat := TensorCategory([1, 1, 1], {{2,1,0}});
> AlgCat;
Tensor category of valence 3 (->,->,->) ({ 0, 1, 2 })
> ChangeTensorCategory(~t, AlgCat);
> A := HeisenbergAlgebra(t);
> A;
Algebra of dimension 3 with base ring Rational Field
> SystemOfForms(Tensor(A));
[
    [ 1  4  7]
    [10 13 16]
    [19 22 25],

    [ 2  5  8]
    [11 14 17]
    [20 23 26],

    [ 3  6  9]
    [12 15 18]
    [21 24 27]
]
```

HeisenbergLieAlgebra(T): TenSpcElt -> AlgLie

Returns the Heisenberg Lie algebra L with Lie bracket given by the 3-tensor $T : U \times V \rightarrowtail W$. If the tensor category of T forces equality only between U and V and T is alternating, then the Heisenberg Lie algebra will have structure constants equal to T. In this case, $L \cong U \oplus W$ as vector spaces. Otherwise, $L \cong U \oplus V \oplus W$ as vector spaces, and, using $*$ in place for T, the product in L is given by

$$(u, v, w) * (u', v', w') = (0, 0, uv' - u'v).$$

Example: Lie Algebras from Tensors (ex-19a3f6)

We will construct an alternating tensor with frame $\mathbb{Q}^3 \times \mathbb{Q}^3 \rightarrowtail \mathbb{Q}^2$.

```
> t := Tensor(Rationals(), [3, 3, 2], &cat[[1,-1,0] : i in [1..6]]);
> t := AlternatingTensor(t);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 3 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
> SystemOfForms(t);
[
    [ 0 -1 -2]
    [ 1  0 -1]
```

(continues on next page)

(continued from previous page)

```
    [ 2  1  0],

    [ 0  2  1]
    [-2  0 -1]
    [-1  1  0]
]
```

First, we will just construct the Heisenberg Lie algebra from t as is. Because the tensor category of t does not fuse together U, V, and W, the Lie algebra will be 8-dimensional.

```
> L := HeisenbergLieAlgebra(t);
> L;
Lie Algebra of dimension 8 with base ring Rational Field
> SystemOfForms(Tensor(L))[7..8];
[
    [ 0  0  0  0 -1 -2  0  0]
    [ 0  0  0  1  0 -1  0  0]
    [ 0  0  0  2  1  0  0  0]
    [ 0 -1 -2  0  0  0  0  0]
    [ 1  0 -1  0  0  0  0  0]
    [ 2  1  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0],

    [ 0  0  0  0  2  1  0  0]
    [ 0  0  0 -2  0 -1  0  0]
    [ 0  0  0 -1  1  0  0  0]
    [ 0  2  1  0  0  0  0  0]
    [-2  0 -1  0  0  0  0  0]
    [-1  1  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0]
]
```

Now if we fuse U and V together, the resulting Lie algebra will be 5-dimensional, and the Lie bracket will essentially be equal to t.

```
> NilCat := TensorCategory([1, 1, 1], {{2,1},{0}});
> NilCat;
Tensor category of valence 3 (->,->,->) ({ 0 },{ 1, 2 })
> ChangeTensorCategory(~t, NilCat);
> L := HeisenbergLieAlgebra(t);
> L;
Lie Algebra of dimension 5 with base ring Rational Field
> SystemOfForms(Tensor(L))[4..5];
[
    [ 0 -1 -2  0  0]
    [ 1  0 -1  0  0]
    [ 2  1  0  0  0]
    [ 0  0  0  0  0]
    [ 0  0  0  0  0],

    [ 0  2  1  0  0]
    [-2  0 -1  0  0]
    [-1  1  0  0  0]
    [ 0  0  0  0  0]
```

(continues on next page)

(continued from previous page)

```
    [ 0  0  0  0  0]
]
```

HeisenbergGroup(T): TenSpcElt -> GrpMat

HeisenbergGroupPC(T): TenSpcElt -> GrpPC

Returns the class 2, exponent p, Heisenberg p-group with commutator given by the bilinear tensor $T : U \times V \rightarrowtail W$ over a finite field. If T is alternating and the tensor category of T forces equality between U and V, then the group returned is an extension of V by W, so $|G| = |V| \cdot |W|$. Otherwise, the group returned is an extension of $U \oplus V$ by W, so $|G| = |U| \cdot |V| \cdot |W|$. If T is not full, then G is an extension of $U \oplus V \oplus W/\mathrm{im(T)}$ (or $V \oplus W/\mathrm{im(T)}$) by $\mathrm{im(T)}$.

Example: p-Groups from Tensors (ex-3bc793)

Here we demonstrate how to export a group from a tensor. We will start with a tensor that is alternating and compare the different outputs based on the category of the tensor.

```
> t := KTensorSpace(GF(5), [5, 5, 4])!0;
> for i in [1..4] do
>   Assign(~t, [i, i+1, i], 1);   // 1s above diag
>   Assign(~t, [i+1, i, i], -1);  // 4s below diag
> end for;
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 5 over GF(5)
U1 : Full Vector space of degree 5 over GF(5)
U0 : Full Vector space of degree 4 over GF(5)
> IsAlternating(t);
true
```

In this example, we will focus on the more straight-forward case, when t is full. The next example demonstrates a tensor that is not full.

```
> SystemOfForms(t);
[
    [0 1 0 0 0]
    [4 0 0 0 0]
    [0 0 0 0 0]
    [0 0 0 0 0]
    [0 0 0 0 0],

    [0 0 0 0 0]
    [0 0 1 0 0]
    [0 4 0 0 0]
    [0 0 0 0 0]
    [0 0 0 0 0],

    [0 0 0 0 0]
    [0 0 0 0 0]
    [0 0 0 1 0]
    [0 0 4 0 0]
    [0 0 0 0 0],

    [0 0 0 0 0]
    [0 0 0 0 0]
    [0 0 0 0 0]
    [0 0 0 0 1]
    [0 0 0 4 0]
```

(continues on next page)

(continued from previous page)

```
]
> IsFullyNondegenerate(t);
true
```

Even though t is alternating, the category does not fuse together U and V. Therefore, the Heisenberg group H is an extension of $U \oplus V$ by W, and hence, $|H| = 5^{5+5+4}$.

```
> H := HeisenbergGroup(t);
> LMGOrder(LMGCenter(H)) eq 5^4;
true
> s := pCentralTensor(H, 5, 1, 1);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 10 over GF(5)
U1 : Full Vector space of degree 10 over GF(5)
U0 : Full Vector space of degree 4 over GF(5)
```

Now we will fuse the 2 and 1 coordinate, so that $|H| = 5^{5+4}$.

```
> PgrpCat := TensorCategory([1, 1, 1], {{2,1},{0}});
> PgrpCat;
Tensor category of valence 3 (->,->,->) ({ 0 },{ 1, 2 })
> ChangeTensorCategory(~t, PgrpCat);
> H := HeisenbergGroup(t);
> LMGOrder(LMGCenter(H)) eq 5^4;
true
> s := pCentralTensor(H, 5, 1, 1);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 5 over GF(5)
U1 : Full Vector space of degree 5 over GF(5)
U0 : Full Vector space of degree 4 over GF(5)
```

Note that t and s are not the same tensor. However, they are pseudo-isometric (isotopic in the category with $U_2 = U_1$). This effect is exaggerated with `HeisenbergGroupPC` because of how PC-generators are organized.

```
> SystemOfForms(s);
[
    [0 0 0 0 0]
    [0 0 0 0 1]
    [0 0 0 0 0]
    [0 0 0 0 0]
    [0 4 0 0 0],

    [0 0 0 0 0]
    [0 0 0 0 0]
    [0 0 0 0 4]
    [0 0 0 0 0]
    [0 0 1 0 0],

    [0 0 0 0 0]
    [0 0 0 0 0]
    [0 0 0 1 0]
    [0 0 4 0 0]
    [0 0 0 0 0],
```

(continues on next page)

(continued from previous page)

```
    [0 0 0 4 0]
    [0 0 0 0 0]
    [0 0 0 0 0]
    [1 0 0 0 0]
    [0 0 0 0 0]
]
```

Example: p-Groups from Non-Full Tensors (ex-ce8a17)

We will start with an alternating tensor that is not full with the frame $\mathbf{F}_3^4 \times \mathbf{F}_3^4 \rightarrowtail \mathbf{F}_3^3$.

```
> t := Tensor(GF(3), [4, 4, 3], &cat[[1,0,0,1] : i in [1..12]]);
> t := AlternatingTensor(t);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over GF(3)
U1 : Full Vector space of degree 4 over GF(3)
U0 : Full Vector space of degree 3 over GF(3)
> Radical(t);
<
    Vector space of degree 4, dimension 1 over GF(3)
    Echelonized basis:
    (1 2 1 2),

    Vector space of degree 4, dimension 1 over GF(3)
    Echelonized basis:
    (1 2 1 2)
>
> Image(t);
Vector space of degree 3, dimension 2 over GF(3)
Generators:
(1 0 2)
(0 1 0)
Echelonized basis:
(1 0 2)
(0 1 0)
> SystemOfForms(t);
[
    [0 0 2 2]
    [0 0 2 2]
    [1 1 0 0]
    [1 1 0 0],

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

    [0 0 1 1]
    [0 0 1 1]
    [2 2 0 0]
    [2 2 0 0]
]
```

Because the tensor category of t does not force equality between U and V, the group H created from t will have order 3^{11} and be 8-generated. Furthermore, the tensor has a 2-dimensional image, so the tensor from the exponent-p central series of H will have frame $\mathbf{F}_3^{4+4+1} \times \mathbf{F}_3^{4+4+1} \rightarrowtail \mathbf{F}_3^2$. We

display the system of forms of the exponent-p central tensor of H to show how different it looks compared to the original.

```
> G := HeisenbergGroup(t);
> LMGOrder(G) eq 3^11;
true
> s := pCentralTensor(G);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 9 over GF(3)
U1 : Full Vector space of degree 9 over GF(3)
U0 : Full Vector space of degree 2 over GF(3)
> SystemOfForms(s);
[
    [0 0 0 0 0 0 0 0 1]
    [0 0 0 0 0 0 2 2 0]
    [0 0 0 0 0 0 0 0 1]
    [0 0 0 0 0 0 1 1 2]
    [0 0 0 0 0 0 0 0 0]
    [0 0 0 0 0 0 0 0 0]
    [0 1 0 2 0 0 0 0 0]
    [0 1 0 2 0 0 0 0 0]
    [2 0 2 1 0 0 0 0 0],

    [0 0 0 0 0 0 1 0 0]
    [0 0 0 0 0 0 1 0 0]
    [0 0 0 0 0 0 0 2 2]
    [0 0 0 0 0 0 0 2 2]
    [0 0 0 0 0 0 0 0 0]
    [0 0 0 0 0 0 0 0 0]
    [2 2 0 0 0 0 0 0 0]
    [0 0 1 1 0 0 0 0 0]
    [0 0 1 1 0 0 0 0 0]
]
```

63.3 Tensor Spaces

In Magma a tensor space is a parent type for tensors. It behaves as a module but also maintains an interpretation of its elements as multilinear maps. Each tensor space further maintains a tensor category which is assigned to its tensors.

63.3.1 Constructions of Tensor and Cotensor Spaces

Universal Tensor Spaces

Construction of universal tensor spaces is modeled after construction of free modules and matrix spaces. For efficiency reasons, the actual representation may vary based on the parameters, e.g. it may be a space of structure constants, black-box functions, or systems of forms. So access to the tensors in these tensor space should be made through the provided functions.

`KTensorSpace(K, S): Fld, [RngIntElt] -> TenSpc`

`KTensorSpace(K, S, C): Fld, [RngIntElt], TenCat -> TenSpc`

For a field K and sequence $S = [d_\nu, \cdots, d_0]$, returns the universal tensor space $\oslash_{a\in[\nu]} K^{d_a}$ with covariant tensor category given by C. The default category is the homotopism category.

`RTensorSpace(R, S): Rng, [RngIntElt] -> TenSpc`

`RTensorSpace(R, S, C): Rng, [RngIntElt], TenCat -> TenSpc`

For a commutative ring R and sequence $S = [d_\nu, \cdots, d_0]$, returns the universal tensor space

$\oslash_{a\in[\nu]} R^{d_a}$ with covariant tensor category given by C. The default category is the homotopism category.

Example: Universal Tensor Space over a Field (ex-9c4d34)

We demonstrate how to construct universal tensor spaces from a field $K = \mathbb{Q}$ and a sequence of nonnegative integers $[6, 5, 4, 3]$. The resulting tensor space is isomorphic to the space of multilinear maps with frame

$$\mathbb{Q}^6 \times \mathbb{Q}^5 \times \mathbb{Q}^4 \rightarrowtail \mathbb{Q}^3.$$

```
> K := Rationals();
> S := [6, 5, 4, 3];
> T := KTensorSpace(K, S);
> T;
Tensor space of dimension 360 over Rational Field with valence 4
U3 : Full Vector space of degree 6 over Rational Field
U2 : Full Vector space of degree 5 over Rational Field
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 3 over Rational Field
```

Because tensor spaces act like modules, we can construct tensors in the same way we construct vectors.

```
> t := T![Random([-1, 0, 1]) : i in [1..Dimension(T)]];
> t;
Tensor of valence 4, U3 x U2 x U1 >-> U0
U3 : Full Vector space of degree 6 over Rational Field
U2 : Full Vector space of degree 5 over Rational Field
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 3 over Rational Field
> Parent(t);
Tensor space of dimension 360 over Rational Field with valence 4
U3 : Full Vector space of degree 6 over Rational Field
U2 : Full Vector space of degree 5 over Rational Field
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 3 over Rational Field
```

`TensorSpace(S): SeqEnum -> TenSpc, List`

`TensorSpace(S): List -> TenSpc, List`

`TensorSpace(S, C): SeqEnum, TenCat -> TenSpc, List`

`TensorSpace(S, C): List, TenCat -> TenSpc, List`

Given a sequence $S = [U_\nu, \ldots, U_0]$ of R-modules returns a universal tensor space equivalent to $\oslash_{a\in[\nu]} U_a$ with covariant tensor category given by C and a list of maps into the vector spaces in the frame. The default category is the homotopism category.

Example: Universal Tensor Space from Modules (ex-d8700c)

We will construct a similar tensor space as the previous example. However, the objects will be different, according to Magma.

```
> R := Integers();
> S := [* RMatrixSpace(R, 2, 3), RSpace(R, 5), RMatrixSpace(R, 2, 2),
>     RSpace(R, 3) *];
> T := TensorSpace(S);
> T;
Tensor space of dimension 360 over Integer Ring with valence 4
U3 : Full RSpace of degree 6 over Integer Ring
```

(continues on next page)

(continued from previous page)

```
U2 : Full RSpace of degree 5 over Integer Ring
U1 : Full RSpace of degree 4 over Integer Ring
U0 : Full RSpace of degree 3 over Integer Ring
```

Even though the frame of T does not include matrix spaces, like with tensors, it can still evaluate matrices. For example, we evaluate T at $(0, 0, 0)$, which is the trivial subspace in $\mathbb{Z}^3$.

```
> x := < X!0 : X in S[1..3] >;
> x;
<
    [0 0 0]
    [0 0 0],

    (0 0 0 0 0),

    [0 0]
    [0 0]
>
> x @ T;
RSpace of degree 3, dimension 0 over Integer Ring
Generators:
```

TensorSpace(V, p, q): ModTupFld, RngIntElt, RngIntElt -> TenSpc

TensorSpace(K, d, p, q): Fld, RngIntElt, RngIntElt, RngIntElt -> TenSpc

Given a field K and a dimension d or a vector space $V = K^d$ for which the first p indices are covariant and the last q indices are contravariant, returns the signatured (p, q)-tensor space over V. This is functionally equivalent to creating a universal tensor space from the sequence $[V, \ldots,_p V, V^*, \ldots,_q V^*, K]$ and the tensor category with arrows $[1, \ldots,_p 1, -1, \ldots,_q -1, 0]$ and duplicates $\{\{p + q, \ldots, 1 + q\}, \{q, \ldots, 1\}, \{0\}\}$. The valence of the returned tensor space will be $p + q + 1$.

Example: Signatured Tensor Space (ex-170743)

Here we simply demonstrate the nuances of the signatured tensor space constructor. We set $V = \mathbf{F}_5^4$, $p = 3$, and $q = 2$, and the tensor space we will construct will have frame

$$V \times V \times V \times V^* \times V^* \rightarrowtail \mathbf{F}_5.$$

Notice this is equivalent to the frame $V \times V \times V \rightarrowtail V \times V$.

```
> K := GF(5);
> T := TensorSpace(K, 4, 3, 2);
> T;
Tensor space of dimension 1024 over GF(5) with valence 6
U5 : Full Vector space of degree 4 over GF(5)
U4 : Full Vector space of degree 4 over GF(5)
U3 : Full Vector space of degree 4 over GF(5)
U2 : Full Vector space of degree 4 over GF(5)
U1 : Full Vector space of degree 4 over GF(5)
U0 : Full Vector space of degree 1 over GF(5)
> S := KTensorSpace(K, [4, 4, 4, 4, 4, 1]);
> S;
Tensor space of dimension 1024 over GF(5) with valence 6
U5 : Full Vector space of degree 4 over GF(5)
U4 : Full Vector space of degree 4 over GF(5)
U3 : Full Vector space of degree 4 over GF(5)
U2 : Full Vector space of degree 4 over GF(5)
```

(continues on next page)

(continued from previous page)

```
U1 : Full Vector space of degree 4 over GF(5)
U0 : Full Vector space of degree 1 over GF(5)
```

However, the subtleties of this construction lies in the tensor category. On the surface, the spaces T and S look the same, but probing the category reveals their differences.

```
> TensorCategory(S); // default category
Tensor category of valence 6 (->,->,->,->,->,->) ({ 1 },{ 2 },{ 0 },{ 3 },{ 4
},{ 5 })
> TensorCategory(T);
Tensor category of valence 6 (<-,<-,<-,->,->,==) ({ 0 },{ 3 .. 5 },{ 1 .. 2 })
> S eq T;
false
```

Universal Cotensor Spaces

Currently, we only consider cotensor spaces over fields.

`KCotensorSpace(K, S): Fld, [RngIntElt] -> TenSpc`

`KCotensorSpace(K, S, C): Fld, [RngIntElt], TenCat -> TenSpc`

For a field K and sequence $S = [d_\nu, \ldots, d_1]$ returns the universal cotensor space $\bigotimes_{a\in\bar{0}} K^{d_a}$ with the given contravariant tensor category C. The default category is the homotopism category. Notice the sequence S indices do *not* include 0.

`CotensorSpace(S): SeqEnum -> TenSpc`

`CotensorSpace(S): List -> TenSpc, List`

`CotensorSpace(S, C): SeqEnum, TenCat -> TenSpc`

`CotensorSpace(S, C): List, TenCat -> TenSpc, List`

Given a sequence $S = [U_\nu, \ldots, U_1]$ of K-vector spaces returns the universal tensor space equivalent to $\bigotimes_{a\in\bar{0}} U_a$ with contravariant tensor category given by C and a list of maps into the vector spaces of the frame. The default category is the homotopism category. Notice the sequence S indices do *not* include 0.

Example: Universal Cotensor Spaces (ex-a0d0fa)

Constructing cotensor spaces is nearly the same as constructing tensor spaces. When providing a sequence of modules or a sequence of dimensions, we do not include the entry for U_0. It is automatically set to $U_0 = K$. Furthermore, the black-box construction forgets all other structure of the vector spaces and only builds a frame out of vector spaces. The maps are returned as a `List` in case they are needed.

```
> K := Rationals();
> S := [* KSpace(K, 3), MatrixAlgebra(K, 3), KMatrixSpace(K, 2, 3) *];
> S;
[*
    Full Vector space of degree 3 over Rational Field,
    Full Matrix Algebra of degree 3 over Rational Field,
    Full KMatrixSpace of 2 by 3 matrices over Rational Field
*]
> T := CotensorSpace(S);
> T;
Cotensor space of dimension 162 over Rational Field with valence 4
U3 : Full Vector space of degree 3 over Rational Field
U2 : Full Vector space of degree 9 over Rational Field
U1 : Full Vector space of degree 6 over Rational Field
```

Like with tensor spaces, cotensors are elements of cotensor spaces, and for many purposes in Magma, cotensor and tensor spaces have similar functionality.

```
> t := T.3;
> t;
Cotensor of valence 4, U3 x U2 x U1 >-> K
U3 : Full Vector space of degree 3 over Rational Field
U2 : Full Vector space of degree 9 over Rational Field
U1 : Full Vector space of degree 6 over Rational Field
> Eltseq(t);
[ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0 ]
```

Some Standard Constructions

We include some subspaces generated by well-known tensors.

`AlternatingSpace(T): TenSpc -> TenSpc, Map`

Returns the sub(co-)tensor space generated by all the alternating (co-)tensors contained in the given (co-)tensor space.

`AntisymmetricSpace(T): TenSpc -> TenSpc, Map`

Returns the sub(co-)tensor space generated by all the antisymmetric (co-)tensors contained in the given (co-)tensor space.

`SymmetricSpace(T): TenSpc -> TenSpc, Map`

Returns the sub(co-)tensor space generated by all the symmetric (co-)tensors contained in the given (co-)tensor space.

Example: Standard Tensor Subspaces `(ex-4fe5c3)`

We will construct some standard tensor subspaces of the universal tensor space with frame $\mathbf{F}_3^4 \times \mathbf{F}_3^4 \rightarrowtail \mathbf{F}_3^4$.

```
> K := GF(3);
> T := KTensorSpace(K, [4, 4, 4]);
> T;
Tensor space of dimension 64 over GF(5) with valence 3
U2 : Full Vector space of degree 4 over GF(5)
U1 : Full Vector space of degree 4 over GF(5)
U0 : Full Vector space of degree 4 over GF(5)
```

First, we will construct the subspace of T containing all alternating bilinear maps. We expect the subspace to be 24-dimensional in T.

```
> Alt := AlternatingSpace(T);
> Alt;
Tensor space of dimension 24 over GF(3) with valence 3
U2 : Full Vector space of degree 4 over GF(3)
U1 : Full Vector space of degree 4 over GF(3)
U0 : Full Vector space of degree 4 over GF(3)
> t := Random(Alt);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over GF(3)
U1 : Full Vector space of degree 4 over GF(3)
```

(continues on next page)

(continued from previous page)

```
U0 : Full Vector space of degree 4 over GF(3)
> IsAlternating(t);
true
```

Now we will construct the symmetric subspace of the alternating subspace, which is trivial.

```
> S := SymmetricSpace(Alt);
> S;
Tensor space of dimension 0 over GF(3) with valence 3
U2 : Full Vector space of degree 4 over GF(3)
U1 : Full Vector space of degree 4 over GF(3)
U0 : Full Vector space of degree 4 over GF(3)
```

`ExteriorCotensorSpace(V, n): ModTupFld, RngIntElt -> TenSpc`

Returns the cotensor space given by the nth exterior power of the vector space V.

`SymmetricCotensorSpace(V, n): ModTupFld, RngIntElt -> TenSpc`

Returns the cotensor space given by the nth symmetric power of the vector space V.

Example: Standard Cotensor Subspaces `(ex-cd38a4)`

We set $V = \mathbf{F}_5^6$ and we construct the cotensor space of the symmetric square of V, $V \wedge V$. The frame is $V \times V \rightarrowtail \mathbf{F}_5$, so the cotensor space is 15-dimensional.

```
> V := VectorSpace(GF(5), 6);
> T := ExteriorCotensorSpace(V, 2);
> T;
Cotensor space of dimension 15 over GF(5) with valence 3
U2 : Full Vector space of degree 6 over GF(5)
U1 : Full Vector space of degree 6 over GF(5)
```

The cotensor space T is generated by all alternating tensors $\langle t| : V \times V \rightarrowtail \mathbf{F}_5$. We will demonstrate by constructing a random cotensor from T.

```
> t := Random(T);
> t;
Cotensor of valence 3, U2 x U1 >-> K
U2 : Full Vector space of degree 6 over GF(5)
U1 : Full Vector space of degree 6 over GF(5)
> SystemOfForms(t);
[
    [0 2 2 2 4 1]
    [3 0 4 1 0 0]
    [3 1 0 3 1 1]
    [3 4 2 0 2 0]
    [1 0 4 3 0 2]
    [4 0 4 0 3 0]
]
> IsAlternating(t);
true
```

Notice this construction is not $V \times V \rightarrowtail V \wedge V$. However, if we want this tensor, we can use the intrinsic `AsTensor`.

```
> s := AsTensor(T);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 6 over GF(5)
```

(continues on next page)

(continued from previous page)

```
U1 : Full Vector space of degree 6 over GF(5)
U0 : Full Vector space of degree 15 over GF(5)
> IsAlternating(s);
true
```

63.3.2 Operations on Tensor Spaces

Membership and Comparison with Tensor Spaces

We define some intuitive functions for tensor spaces, similar to those found for modules.

`T in TS: TenSpcElt, TenSpc -> BoolElt`

Returns `true` if T is contained in the tensor space TS.

`TS ! T: TenSpc, TenSpcElt -> TenSpcElt`

If the tensor T can be coerced into the tensor space TS, the tensor is returned as an element of TS.

`TS ! S: TenSpc, SeqEnum -> TenSpcElt`

If the sequence S can be coerced into the tensor space TS as a tensor, the corresponding tensor is returned.

`T ! n: TenSpc, RngIntElt -> TenSpcElt`

This is a shortcut designed to only work when $n = 0$, and thus, returns `true` and the zero tensor from the tensor space. Any other integer will yield an error.

`IsCoercible(TS, x): TenSpc, Any -> BoolElt, TenSpcElt`

Returns `true` if x can be coerced into the tensor space TS and the corresponding tensor, otherwise `false`.

Example: Coercion into a Tensor Space (ex-e939dc)

We illustrate that using `!` is the same as creating the tensor from scratch.

```
> T := KTensorSpace( GF(2), [2,3,2] );
> T;
Tensor space of dimension 12 over GF(2) with valence 3
U2 : Full Vector space of degree 2 over GF(2)
U1 : Full Vector space of degree 3 over GF(2)
U0 : Full Vector space of degree 2 over GF(2)
>
> S := [ 1 : i in [1..12] ];
> t := T!S;
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 2 over GF(2)
U1 : Full Vector space of degree 3 over GF(2)
U0 : Full Vector space of degree 2 over GF(2)
>
> t eq Tensor(GF(2), [2,3,2], S);
true
>
> T!0 in T;
true
> SystemOfForms(T!0);
[
    [0 0 0]
    [0 0 0],
```

(continues on next page)

(continued from previous page)

```
    [0 0 0]
    [0 0 0]
]
```

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

Decides if the tensor spaces S and T are equal.

`S subset T: TenSpc, TenSpc -> BoolElt`

Decides if S is a subset of the tensor space T.

`IsCoercible(T, S): TenSpc, Any -> BoolElt`

`T ! S: TenSpc, TenSpc -> TenSpc`

Decides if the tensor space S can be coerced into the tensor space T as a subspace. If so, the corresponding subspace is returned.

Example: Containment of Tensor Spaces (ex-812352)

We construct the universal tensor space T with the frame $\mathbf{F}_2^4 \times \mathbf{F}_2^4 \rightarrowtail \mathbf{F}_2^4$. We also create a tensor space T_2 with frame $\mathbf{F}_2^4 \times \mathbf{F}_2 \times \mathbf{F}_2^4 \times \mathbf{F}_2 \rightarrowtail \mathbf{F}_2^4$.

```
> T := KTensorSpace(GF(2), [4,4,4]);
> T;
Tensor space of dimension 64 over GF(2) with valence 3
U2 : Full Vector space of degree 4 over GF(2)
U1 : Full Vector space of degree 4 over GF(2)
U0 : Full Vector space of degree 4 over GF(2)
>
> T2 := KTensorSpace(GF(2), [4,1,4,1,4]);
> T2;
Tensor space of dimension 64 over GF(2) with valence 5
U4 : Full Vector space of degree 4 over GF(2)
U3 : Full Vector space of degree 1 over GF(2)
U2 : Full Vector space of degree 4 over GF(2)
U1 : Full Vector space of degree 1 over GF(2)
U0 : Full Vector space of degree 4 over GF(2)
>
> S := sub< T | T.2, T.4, T.8 >;
> S;
Tensor space of dimension 3 over GF(2) with valence 3
U2 : Full Vector space of degree 4 over GF(2)
U1 : Full Vector space of degree 4 over GF(2)
U0 : Full Vector space of degree 4 over GF(2)
```

We verify that S is a subset of T and not T_2. Since the tensors of T naturally embed into T_2, the subspace S can be coerced into T_2, which we label as S_2. We then verify that S_2 is a subset of T_2 and not T.

```
> S subset T2;
false
> S2 := T2!S;
> S2 subset T2;
true
> S2 subset T;
false
```

Tensor Spaces as Modules

We view a tensor space as a K-module, so we have notions of generators, dimension (if it is free), and cardinality.

Basis(T): TenSpc -> SeqEnum

Generators(T): TenSpc -> SeqEnum

Returns a basis for the tensor space T.

T . i: TenSpc, RngIntElt -> TenSpcElt

Returns the ith basis tensor of the tensor space T.

NumberOfGenerators(T): TenSpc -> RngIntElt

Ngens(T): TenSpc -> RngIntElt

Returns the number of generators of the tensor space T.

Dimension(T): TenSpc -> RngIntElt

Returns the dimension of the tensor space T as a free K-module.

T: TenSpc -> RngIntElt

Returns the size of the tensor space, provided T is finite.

Example: Module Functions on Tensor Spaces (ex-d57848)

We demonstrate the basic module functions of tensor spaces. We construct the universal tensor space T with frame $\mathbf{F}_8^3 \times \mathbf{F}_8^5 \rightarrowtail \mathbf{F}_8^7$.

```
> K := GF(8);
> T := KTensorSpace(K, [3,5,7]);
> T;
Tensor space of dimension 105 over GF(2^3) with valence 3
U2 : Full Vector space of degree 3 over GF(2^3)
U1 : Full Vector space of degree 5 over GF(2^3)
U0 : Full Vector space of degree 7 over GF(2^3)
```

Now we obtain module properties of T.

```
> Dimension(T);
105
> #Basis(T);
105
> T.100 in Basis(T);
true
> T.100;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 3 over GF(2^3)
U1 : Full Vector space of degree 5 over GF(2^3)
U0 : Full Vector space of degree 7 over GF(2^3)
> #T eq 8^(3*5*7);
true
```

Random(T): TenSpc -> TenSpcElt

Provided the base ring has a random algorithm in Magma, it returns a random element of the tensor space T.

RandomTensor(R, S): Rng, [RngIntElt] -> TenSpcElt

RandomTensor(R, S, C): Rng, [RngIntElt], TenCat -> TenSpcElt

RandomCotensor(K, S): Fld, [RngIntElt] -> TenSpcElt

Provided R has a random algorithm in Magma, it returns a random (co)tensor from the universal (co)tensor space $\oslash_{s\in S} R^s$, with category C. The default category is the homotopism category.

Example: Random Tensors (ex-377558)

We create some random tensors, which works similarly to `RandomMatrix`. There are two main ways to obtain a random tensor: from a given tensor space or from a ring and a dimension sequence (the universal tensor space). First we will construct the alternating tensor space of the universal tensor space with frame $\mathbf{F}_3^4 \times \mathbf{F}_3^4 \rightarrowtail \mathbf{F}_3^2$.

```
> T := KTensorSpace(GF(3), [4,4,2]);
> T;
Tensor space of dimension 32 over GF(3) with valence 3
U2 : Full Vector space of degree 4 over GF(3)
U1 : Full Vector space of degree 4 over GF(3)
U0 : Full Vector space of degree 2 over GF(3)
> S := AlternatingSpace(T);
> S;
Tensor space of dimension 12 over GF(3) with valence 3
U2 : Full Vector space of degree 4 over GF(3)
U1 : Full Vector space of degree 4 over GF(3)
U0 : Full Vector space of degree 2 over GF(3)
```

Now we construct a random alternating tensor from S.

```
> t := Random(S);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over GF(3)
U1 : Full Vector space of degree 4 over GF(3)
U0 : Full Vector space of degree 2 over GF(3)
> SystemOfForms(t);
[
    [0 2 0 0]
    [1 0 0 0]
    [0 0 0 2]
    [0 0 1 0],

    [0 2 1 0]
    [1 0 1 2]
    [2 2 0 1]
    [0 1 2 0]
]
```

RandomAlternatingTensor(R, d, n, c): Rng, RngIntElt, RngIntElt, RngIntElt -> TenSpcElt

RandomAlternatingTensor(R, S): Rng, [RngIntElt] -> TenSpcElt

Returns a random alternating tensor from the universal tensor space

$$\prod_{k=1}^{n} R^d \rightarrowtail R^c.$$

If S is given instead, then we assume $S = [d, \ldots, d, c]$. The returned tensor has the homotopism category but fuses every module in the domain. An error is raised if R does not have a random algorithm in Magma.

RandomAntisymmetricTensor(R, d, n, c): Rng, RngIntElt, RngIntElt, RngIntElt -> TenSpcElt

RandomAntisymmetricTensor(R, S): Rng, [RngIntElt] -> TenSpcElt

Returns a random antisymmetric tensor from the universal tensor space

$$\prod_{k=1}^{n} R^d \rightarrowtail R^c.$$

If S is given instead, then we assume $S = [d, \ldots, d, c]$. The returned tensor has the homotopism category but fuses every module in the domain. An error is raised if R does not have a random algorithm in Magma.

`RandomSymmetricTensor(R, d, n, c): Rng, RngIntElt, RngIntElt, RngIntElt -> TenSpcElt`

`RandomSymmetricTensor(R, S): Rng, [RngIntElt] -> TenSpcElt`

Returns a random symmetric tensor from the universal tensor space

$$\prod_{k=1}^{n} R^d \rightarrowtail R^c.$$

If S is given instead, then we assume $S = [d, \ldots, d, c]$. The returned tensor has the homotopism category but fuses every module in the domain. An error is raised if R does not have a random algorithm in Magma.

Example: Random Symmetric Tensors (ex-f99231)

We just demonstrate how to use the `RandomSymmetricTensor`. The other two intrinsics are the same. This is almost equivalent to constructing a random tensor whose category fuses all the modules in the domain and then applying `SymmetricTensor`. In characteristic 2, all entries on the diagonal will be 0, but with `RandomSymmetricTensor` this is not the case.

```
> t := RandomSymmetricTensor(GF(2), 4, 2, 3);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over GF(2)
U1 : Full Vector space of degree 4 over GF(2)
U0 : Full Vector space of degree 3 over GF(2)
>
> TensorCategory(t);
Tensor category of valence 3 (->,->,->) ({ 0 },{ 1 .. 2 })
>
> SystemOfForms(t);
[
    [1 1 1 1]
    [1 0 1 0]
    [1 1 1 1]
    [1 0 1 0],

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

    [1 0 0 0]
    [0 0 1 1]
    [0 1 0 0]
    [0 1 0 1]
]
> IsSymmetric(t);
true
```

Properties of Tensor Spaces

We define some functions to access basic properties of tensor spaces.

`Valence(T): TenSpc -> RngIntElt`

Returns the valence of the tensor space.

Frame(T): TenSpc -> List

Returns the list of modules in the frame of the tensor space.

BaseRing(T): TenSpc -> Rng

BaseField(T): TenSpc -> Fld

Returns the base ring (or field) of the tensor space.

Example: Properties of Tensor Spaces (ex-40631c)

As with tensors, we can obtain the basic tensor properties from a tensor space.

```
> T := KTensorSpace(Rationals(), [7,5,3,2]);
> T;
Tensor space of dimension 210 over Rational Field with valence 4
U3 : Full Vector space of degree 7 over Rational Field
U2 : Full Vector space of degree 5 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
> Valence(T);
4
> Frame(T);
[*
    Full Vector space of degree 7 over Rational Field,

    Full Vector space of degree 5 over Rational Field,

    Full Vector space of degree 3 over Rational Field,

    Full Vector space of degree 2 over Rational Field
*]
> BaseRing(T);
Rational Field
```

TensorCategory(T): TenSpc -> TenCat

Returns the underlying tensor category of the tensor space.

IsCovariant(T): TenSpc -> BoolElt

IsContravariant(T): TenSpc -> BoolElt

Decides if the underlying tensor category is covariant or contravariant.

ChangeTensorCategory(T, C): TenSpc, TenCat -> TenSpc

ChangeTensorCategory(~T, C): TenSpc, TenCat

Returns the tensor category with the given tensor category.

Example: Categories of Tensor Spaces (ex-066785)

Again, like with tensors, we can obtain categorical information from tensor spaces.

```
> T := KTensorSpace(Rationals(), [4,4,4]);
> T;
Tensor space of dimension 64 over Rational Field with valence 3
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 4 over Rational Field
>
> TensorCategory(T);
Tensor category of valence 3 (->,->,->) ({ 1 },{ 2 },{ 0 })
> C := TensorCategory([1,1,-1], {{0},{1,2}});
> C;
```

(continues on next page)

(continued from previous page)

```
Tensor category of valence 3 (->,->,<-) ({ 0 },{ 1, 2 })
>
> ChangeTensorCategory(~T, C);
> T;
Tensor space of dimension 64 over Rational Field with valence 3
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 4 over Rational Field
> TensorCategory(T);
Tensor category of valence 3 (->,->,<-) ({ 0 },{ 1, 2 })
```

IsAlternating(T): TenSpc -> BoolElt

Decides if every tensor in the tensor space is an alternating tensor.

IsAntisymmetric(T): TenSpc -> BoolElt

Decides if every tensor in the tensor space is an antisymmetric tensor.

IsSymmetric(T): TenSpc -> BoolElt

Decides if every tensor in the tensor space is a symmetric tensor.

UniversalTensorSpace(T): TenSpc -> TenSpc

UniversalCotensorSpace(T): TenSpc -> TenSpc

Generic(T): TenSpc -> TenSpc

Returns the universal (co-)tensor space with the same frame and category as T.

Example: Subspace of Alternating Tensors (ex-c56f94)

We construct the subspace of alternating tensors S of the tensor space T with the frame $\mathbb{Q}^6 \times \mathbb{Q}^6 \rightarrowtail \mathbb{Q}^2$. We verify that the space is in fact alternating (i.e. every tensor is alternating).

```
> T := KTensorSpace(Rationals(), [6,6,2]);
> S := AlternatingSpace(T);
> S;
Tensor space of dimension 30 over Rational Field with valence 3
U2 : Full Vector space of degree 6 over Rational Field
U1 : Full Vector space of degree 6 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
> IsAlternating(S);
true
```

From S, we construct the universal tensor space U, which in this example is equal to T.

```
> U := UniversalTensorSpace(S);
> U;
Tensor space of dimension 72 over Rational Field with valence 3
U2 : Full Vector space of degree 6 over Rational Field
U1 : Full Vector space of degree 6 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
> U eq T;
true
```

63.4 Tensor Categories

Magma allows tensors and tensor spaces to change categories. Unless a user specifies otherwise, all tensors are assigned a category that is natural to the method by which it was created. For example a tensor created from an algebra will be assigned an algebra category, whereas a tensor created by structure constants will be assigned the Albert homotopism category [Alb42]. Tensor categories influence the

behavior of commands such as kernels and images as well as the algebraic invariants such as derivation algebras of a tensor.

Our conventions follow [Wil13a]. In particular, given a tensor T framed by $[U_\nu, \ldots, U_0]$ then a tensor category for T will specify a function $A : [\nu] \to \{-1, 0, 1\}$ along with a partition $\mathcal{P}$ of $[\nu]$ such that the following rules apply to the tensors and morphisms in the category.

(1) for each tensor T framed by $[U_\nu, \ldots, U_0]$, if $X \in \mathcal{P}$, then

$$\forall i, j \in X, \quad U_i = U_j.$$

(2) Given a second tensor S framed by $[V_v, \ldots, V_0]$, a morphism $f : T \to S$ (Magma type `Hmtp`) will be a list $[f_\nu, \ldots, f_0]$ of homomorphisms as follows:

- (Covariant) if $A(i) = 1$ then $f_i : U_i \to V_i$;
- (Constant) if $A(i) = 0$ then $U_i = V_i$ and $f_i = 1_{U_i}$; or else
- (Contravariant) $A(i) = -1$ and $f_i : U_i \leftarrow V_i$. So if $A(0) = 1$ then

$$\left\langle \sum_{i \in A^{-1}(1)} u_i f_i + \sum_{j \notin A^{-1}(-1)} v_j \right\rangle_S = \left\langle \sum_{i \in A^{-1}(1)} u_i + \sum_{j \notin A^{-1}(-1)} v_j f_j \right\rangle_T f_0;$$

if $A(0) = 0$ then

$$\left\langle \sum_{i \in A^{-1}(1)} u_i f_i + \sum_{j \notin A^{-1}(-1)} v_j \right\rangle_S = \left\langle \sum_{i \in A^{-1}(1)} u_i + \sum_{j \notin A^{-1}(-1)} v_j f_j \right\rangle_T ;$$

else $A(0) = -1$ and

$$\left\langle \sum_{i \in A^{-1}(1)} u_i f_i + \sum_{j \notin A^{-1}(-1)} v_j \right\rangle_S f_0 = \left\langle \sum_{i \in A^{-1}(1)} u_i + \sum_{j \notin A^{-1}(-1)} v_j f_j \right\rangle_T .$$

Magma manages internally the differences between vectors and covectors and more generally tensors and cotensors. Both types are issued the Magma type `TenSpcElt`. For operations sensitive to the difference, Magma stores a value of co/contra-variance of the tensor as a property of the tensor category. This is the third general property stored in Magma's tensor category type `TenCat`.

We use the phrase tensor category exclusively for categories that describe tensors and tensor spaces. In other words, the data structure of a tensor category is a function $A : [\nu] \to \{-1, 0, 1\}$ and a partition $\mathcal{P}$ of $[\nu]$. It is useful to distinguish from tensors and cotensors at the categorical level, so a tensor category is either covariant or contravariant as well (in the latter case, referred to as a cotensor category).

63.4.1 Constructing Tensor Categories

`TensorCategory(A, P): [RngIntElt], {SetEnum} -> TenCat`

`TensorCategory(A, P): Map, {SetEnum} -> TenCat`

Sets up a covariant tensor space category with specified direction of arrows A, and a partition $\mathcal{P}$ indicating variables to be treated as equivalent. The fiber $A^{-1}(1)$ denotes the covariant variables, $A^{-1}(0)$ identifies the constant variables, and $A^{-1}(-1)$ marks the contra-variant variables.

`CotensorCategory(A, P): [RngIntElt], {SetEnum} -> TenCat`

`CotensorCategory(A, P): Map, {SetEnum} -> TenCat`

Sets up a contra-variant tensor space category with specified direction of arrows A, and a partition $\mathcal{P}$ indicating variables to be treated as equivalent. The fiber $A^{-1}(1)$ denotes the covariant variables, $A^{-1}(0)$ identifies the constant variables, and $A^{-1}(-1)$ marks the contra-variant variables.

Example: Basic Tensor Category Construction (ex-2873fe)

We demonstrate the basic tensor category constructor. The difference between `TensorCategory` and `CotensorCategory` is only that the former is covariant and the latter is contravariant.

```
> C := TensorCategory([1,0,-1], {{0},{1},{2}});
> C;
Tensor category of valence 3 (->,==,<-) ({ 1 },{ 2 },{ 0 })
> IsCovariant(C);
true
>
> arrows := map< {1..5} -> {1} | x :-> 1 >;
> C := CotensorCategory(arrows, {{1..5}});
> C;
Cotensor category of valence 6 (->,->,->,->,->,==) ({ 0 },{ 1 .. 5 })
> IsContravariant(C);
true
```

`HomotopismCategory(v : parameters): RngIntElt -> TenCat`

```
Contravariant: BoolElt                    Default: false
```

Returns Albert's homotopism category – all modules categories are covariant and no duplicates considered. Set the optional parameter `Contravariant` to `true` to make it a cotensor category.

`CohomotopismCategory(v): RngIntElt -> TenCat`

Returns the cohomotopism category – all domain modules categories are covariant, the codomain is contravariant, and no duplicates considered.

`AdjointCategory(v, s, t): RngIntElt, RngIntElt, RngIntElt -> TenCat`

`LinearCategory(v, s, t): RngIntElt, RngIntElt, RngIntElt -> TenCat`

Returns the tensor category where all modules are constant except in position s and t. Both s and t are in $[v]$. Position s is covariant, position t is contravariant.

Example: Special Tensor Category Constructors (ex-f54041)

Now we look at a few special tensor category constructors. The default tensor category is the homotopism category, so we construct the homotopism category using `TensorCategory` and verify they are equivalent.

```
> C := TensorCategory([1,1,1,1], {{i} : i in [0..3]});
> C;
Tensor category of valence 4 (->,->,->,->) ({ 1 },{ 2 },{ 0 },{ 3 })
> HomotopismCategory(4) eq C;
true
```

The other special tensor categories can be constructed using `TensorCategory` as well, but we just construct a few to show their properties.

```
> CohomotopismCategory(3);
Tensor category of valence 3 (->,->,<-) ({ 1 },{ 2 },{ 0 })
>
> AdjointCategory(5, 4, 1);
Tensor category of valence 5 (<-,==,==,->,==) ({ 1 },{ 0, 2, 3 },{ 4 })
```

63.4.2 Operations on Tensor Categories

In this section the basic operations for tensor categories are described.

`C1 eq C2: TenCat, TenCat -> BoolElt`

Returns `true` if the tensor categories are the same.

`Valence(C): TenCat -> RngIntElt`

Returns the valence of the tensor category.

`Arrows(C): TenCat -> SeqEnum`

Returns the sequence of arrows of the tensor category. A -1 signifies a contravariant index, a 0 signifies a constant index, and a 1 signifies a covariant index.

`RepeatPartition(C): TenCat -> SetEnum`

Returns the repeat partition for the tensor category.

`IsCovariant(C): TenCat -> BoolElt`

`IsContravariant(C): TenCat -> BoolElt`

Returns `true` if the tensor category is covariant or contravariant.

Example: Properties of Tensor Categories (ex-d61054)

We obtain basic properties of tensor categories.

```
> C := CotensorCategory([1,0,-1,1],{{4,3},{1},{2}});
> C;
Cotensor category of valence 5 (->,==,<-,->,==) ({ 1 },{ 2 },{ 0 },{ 3, 4 })
>
> Valence(C);
5
> Arrows(C);
[ 1, 0, -1, 1 ]
> IsContravariant(C);
true
> RepeatPartition(C);
{
    { 1 },
    { 2 },
    { 3, 4 }
}
```

63.4.3 Categorical Operations

In this section, we define subtensors, local ideals, ideals, and quotients of tensors. For the following definitions fix a tensor $t \in T$, with frame $\oslash_{a\in[\nu]}U_a$—that is $U_\nu \times \cdots \times U_1 \rightarrowtail U_0$.

A tensor $s : V_\nu \times \cdots \times V_1 \rightarrowtail V_0$ is a *subtensor* of t if for all a, $V_a \leq U_a$.

For $A \subseteq [\nu] - 0$, a tensor $s : V_\nu \times \cdots \times V_1 \rightarrowtail V_0$ is an *A-local ideal* of t if s is a subtensor of t and for each $a \in A$,

$$\langle s \,|\, V_\nu, \ldots, V_{a+1}, U_a, V_{a-1}, \ldots, V_1\rangle \leq V_0.$$

A tensor s is an *ideal* of t if it is a $\{1, \ldots, \nu\}$-local ideal of t.

The *A-local quotient* of a tensor t by an A-local ideal s is the tensor $q : U_\nu/V_\nu \times \cdots \times U_1/V_1 \rightarrowtail U_0/V_0$ where for all $|\overline{\nu}\rangle$,

$$\langle q \,|\, \overline{\nu}\rangle \equiv \langle q \,|\, \nu\rangle \mathrm{mod} \mathrm{V}_0.$$

The *quotient* of a tensor t by an ideal s is the $\{1, \ldots, \nu\}$-local quotient of t by s.

Categorical Operations on Tensors

We include functions defined for the category of tensors. Most functions are currently defined only for the homotopism category.

`Subtensor(T, S): TenSpcElt, List -> TenSpcElt`

`Subtensor(T, S): TenSpcElt, SeqEnum -> TenSpcElt`

Returns the smallest submap of T containing S.

`Subtensor(T, D, C): TenSpcElt, List, Any -> TenSpcElt`

`Subtensor(T, D, C): TenSpcElt, SeqEnum, Any -> TenSpcElt`

Returns the smallest submap of T containing D in the domain and C in the codomain.

`IsSubtensor(T, S): TenSpcElt, TenSpcElt -> BoolElt`

Decides whether S is a subtensor of T.

Example: Subtensors of Octonion Multiplication `(ex-b3fa65)`

We construct the tensor t given by octonion multiplication. The quaternions H are a subalgebra of $A = O$ generated by the first four basis elements. However, H cannot be coerced into A because of how Magma organizes algebras.

```
> A := OctonionAlgebra(Rationals(), -1, -1, -1);
> t := Tensor(A);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 8 over Rational Field
U1 : Full Vector space of degree 8 over Rational Field
U0 : Full Vector space of degree 8 over Rational Field
> H := sub< A | A.1, A.2, A.3, A.4 >;
> H;
Algebra of dimension 4 with base ring Rational Field
```

There are multiple ways to get the subtensor of multiplication from H. We will create $H \times H \rightarrowtail H$ as a subtensor of $A \times A \rightarrowtail A$.

```
> H_gens := [A.i : i in [1..4]];
> s := Subtensor(t, [*H_gens, H_gens, A!0*]);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Vector space of degree 8, dimension 4 over Rational Field
Generators:
(1 0 0 0 0 0 0 0)
(0 1 0 0 0 0 0 0)
(0 0 1 0 0 0 0 0)
(0 0 0 1 0 0 0 0)
Echelonized basis:
(1 0 0 0 0 0 0 0)
(0 1 0 0 0 0 0 0)
(0 0 1 0 0 0 0 0)
(0 0 0 1 0 0 0 0)
U1 : Vector space of degree 8, dimension 4 over Rational Field
Generators:
(1 0 0 0 0 0 0 0)
(0 1 0 0 0 0 0 0)
(0 0 1 0 0 0 0 0)
(0 0 0 1 0 0 0 0)
Echelonized basis:
(1 0 0 0 0 0 0 0)
```

(continues on next page)

(continued from previous page)

```
(0 1 0 0 0 0 0 0)
(0 0 1 0 0 0 0 0)
(0 0 0 1 0 0 0 0)
U0 : Vector space of degree 8, dimension 4 over Rational Field
Generators:
(1 0 0 0 0 0 0 0)
(0 1 0 0 0 0 0 0)
(0 0 1 0 0 0 0 0)
(0 0 0 1 0 0 0 0)
Echelonized basis:
(1 0 0 0 0 0 0 0)
(0 1 0 0 0 0 0 0)
(0 0 1 0 0 0 0 0)
(0 0 0 1 0 0 0 0)
```

Note that because H cannot be coerced into A (i.e. `A!H` produces an error), a subtensor of `AlgGen` cannot be done by `Subtensor(t, [H, H, H])`. Now we will construct the tensor straight from H. There is a subtle difference between the subtensor from A and the tensor from H—namely, the frame is different.

```
> s2 := Tensor(H);
> s2;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 4 over Rational Field
> s eq s2;
false
> Eltseq(s) eq Eltseq(s2);
true
```

`LocalIdeal(T, S, I): TenSpcElt, List, {RngIntElt} -> TenSpcElt`

`LocalIdeal(T, S, I): TenSpcElt, SeqEnum, {RngIntElt} -> TenSpcElt`

Returns the local ideal of T at I containing S.

`LocalIdeal(T, D, C, I): TenSpcElt, List, Any, {RngIntElt} -> TenSpcElt`

`LocalIdeal(T, D, C, I): TenSpcElt, SeqEnum, Any, {RngIntElt} -> TenSpcElt`

Returns the local ideal of T at I containing D in the domain and C in the codomain.

`LocalIdeal(T, S, I): TenSpcElt, TenSpcElt, {RngIntElt} -> TenSpcElt`

Returns the local ideal of T at I containing S as a submap.

`IsLocalIdeal(T, S, I): TenSpcElt, TenSpcElt, {RngIntElt} -> BoolElt`

Decides if S is a local ideal of T at I.

Example: Local Ideals of a Tensor `(ex-96befc)`

We use the same tensor t as the previous example, multiplication in $A = O$, and we construct the subtensor t_2 given by multiplication in H. We construct a subtensor s of t as the submap containing $\langle A_2 \rangle \times \langle A_1, A_4 \rangle \rightarrowtail \langle 0 \rangle$, which is equal to $\langle A_2 \rangle \times \langle A_1, A_4 \rangle \rightarrowtail \langle A_2A_1, A_2A_4 \rangle$.

```
> A := OctonionAlgebra(Rationals(), -1, -1, -1);
> t := Tensor(A);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 8 over Rational Field
U1 : Full Vector space of degree 8 over Rational Field
```

(continues on next page)

(continued from previous page)

```
U0 : Full Vector space of degree 8 over Rational Field
> H_gens := [A.i : i in [1..4]];
> t2 := Subtensor(t, [*H_gens, H_gens, H_gens*]);
> s := Subtensor(t, [* A.2, [A.1, A.4], A!0 *]);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Vector space of degree 8, dimension 1 over Rational Field
Generators:
(0 1 0 0 0 0 0 0)
Echelonized basis:
(0 1 0 0 0 0 0 0)
U1 : Vector space of degree 8, dimension 2 over Rational Field
Generators:
(1 0 0 0 0 0 0 0)
(0 0 0 1 0 0 0 0)
Echelonized basis:
(1 0 0 0 0 0 0 0)
(0 0 0 1 0 0 0 0)
U0 : Vector space of degree 8, dimension 2 over Rational Field
Generators:
(0 1 0 0 0 0 0 0)
(0 0 1 0 0 0 0 0)
Echelonized basis:
(0 1 0 0 0 0 0 0)
(0 0 1 0 0 0 0 0)
```

The quaternions are a subalgebra of A generated by $\{A_1, A_2, A_3, A_4\}$. Therefore, the $\{2\}$-local ideal of s in t must contain A in the codomain. However, the $\{2\}$-local ideal of s in t_2 must only contain $\langle A_1, A_2, A_3, A_4\}$ in the codomain.

```
> s1 := LocalIdeal(t, s, {2});
> Codomain(s1);
Full Vector space of degree 8 over Rational Field
Generators:
(1 0 0 0 0 0 0 0)
(0 1 0 0 0 0 0 0)
(0 0 1 0 0 0 0 0)
(0 0 0 1 0 0 0 0)
(0 0 0 0 1 0 0 0)
(0 0 0 0 0 1 0 0)
(0 0 0 0 0 0 1 0)
(0 0 0 0 0 0 0 1)
> s2 := LocalIdeal(t2, s, {2});
> Codomain(s2);
Vector space of degree 8, dimension 4 over Rational Field
Generators:
(1 0 0 0 0 0 0 0)
(0 1 0 0 0 0 0 0)
(0 0 1 0 0 0 0 0)
(0 0 0 1 0 0 0 0)
Echelonized basis:
(1 0 0 0 0 0 0 0)
(0 1 0 0 0 0 0 0)
(0 0 1 0 0 0 0 0)
(0 0 0 1 0 0 0 0)
```

Ideal(T, S): TenSpcElt, List -> TenSpcElt

`Ideal(T, S): TenSpcElt, SeqEnum -> TenSpcElt`

Returns the ideal of T containing S.

`Ideal(T, D, C): TenSpcElt, List, Any -> TenSpcElt`

`Ideal(T, D, C): TenSpcElt, SeqEnum, Any -> TenSpcElt`

Returns the ideal of T containing D in the domain and C in the codomain.

`Ideal(T, S): TenSpcElt, TenSpcElt -> TenSpcElt`

Returns the ideal of T containing S as a submap.

`IsIdeal(T, S): TenSpcElt, TenSpcElt -> BoolElt`

Decides if S is an ideal of T.

Example: Ideals of the Tensor of an Algebra (ex-fad333)

First we will construct the tensor from the $\mathbb{Q}$-algebra, $\mathbb{Q}^5$.

```
> T := KTensorSpace(Rationals(), [5,5,5]);
> A := VectorSpace(Rationals(), 5);
> t := T!0;
> for i in [1..5] do
>   Assign(~t, [i,i,i], 1);
> end for;
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 5 over Rational Field
U1 : Full Vector space of degree 5 over Rational Field
U0 : Full Vector space of degree 5 over Rational Field
> SystemOfForms(t);
[
    [1 0 0 0 0]
    [0 0 0 0 0]
    [0 0 0 0 0]
    [0 0 0 0 0]
    [0 0 0 0 0],

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

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

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

    [0 0 0 0 0]
    [0 0 0 0 0]
    [0 0 0 0 0]
    [0 0 0 0 0]
```

(continues on next page)

(continued from previous page)

```
    [0 0 0 0 1]
]
```

Now we will construct the ideal tensor from the subtensor containing $\langle A_1 \rangle \times \langle A_2 \rangle \rightarrowtail \langle A_3 \rangle$. Note that the $\{2\}$-local ideal must include $\langle A_2, A_3 \rangle$ in the codomain, and the $\{1\}$-local ideal must contain $\langle A_1, A_3 \rangle$ in the codomain.

```
> s := Ideal(t, [A.1, A.2, A.3]);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Vector space of degree 5, dimension 1 over Rational Field
Generators:
(1 0 0 0 0)
Echelonized basis:
(1 0 0 0 0)
U1 : Vector space of degree 5, dimension 1 over Rational Field
Generators:
(0 1 0 0 0)
Echelonized basis:
(0 1 0 0 0)
U0 : Vector space of degree 5, dimension 3 over Rational Field
Generators:
(1 0 0 0 0)
(0 1 0 0 0)
(0 0 1 0 0)
Echelonized basis:
(1 0 0 0 0)
(0 1 0 0 0)
(0 0 1 0 0)
```

Finally, we verify that the subtensor containing $\langle A_1 \rangle \times \langle A_2 \rangle \rightarrowtail \langle A_2, A_3 \rangle$ is not an ideal.

```
> r := Subtensor(t, [A.1, A.2], [A.2, A.3]);
> IsIdeal(t, r);
false
```

LocalQuotient(T, S, I : parameters): TenSpcElt, TenSpcElt, {RngIntElt} -> TenSpcElt, Hmtp

```
Check: BoolElt                    Default: true
```

Returns the local quotient of T by S at $I \subseteq [\nu] - 0$. If you know S is a local ideal of T at I, set `Check` to `false` to skip the verification. A homotopism is also returned, mapping from T to T/S.

Quotient(T, S : parameters): TenSpcElt, TenSpcElt -> TenSpcElt, Hmtp

```
Check: BoolElt                    Default: true
```

T / S: TenSpcElt, TenSpcElt -> TenSpcElt, Hmtp

Returns the quotient of T by S. If you know S is an ideal of T, set `Check` to `false` to skip the verification. A homotopism is also returned, mapping from T to T/S.

Example: Fully Nondegenerate Quotient Tensor (ex-0f6850)

We will demonstrate one of the most common uses for quotienting tensors: constructing the associated fully nondegenerate tensor. We first construct a tensor with a nontrivial radical given by matrix multiplication: $\mathrm{Mat}_{3\times 2}(\mathbb{Q}) \times \mathbb{Q}^3 \rightarrowtail \mathbb{Q}^3$, where we take a projection of $\mathbb{Q}^3$ onto $\mathbb{Q}^2$ in the 1 coordinate.

```
> K := Rationals();
> F := [*KMatrixSpace(K, 3, 2), VectorSpace(K, 3), VectorSpace(K, 3)*];
> mult := function(x)
>   return Transpose(x[1]*Matrix(2, 1, Eltseq(x[2])[2..3]));
> end function;
> t := Tensor(F, mult);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 6 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 3 over Rational Field
> s := Subtensor(t, [*[F[1].1, F[1].4], F[2], F[3]*]);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Vector space of degree 6, dimension 2 over Rational Field
Generators:
(1 0 0 0 0 0)
(0 0 0 1 0 0)
Echelonized basis:
(1 0 0 0 0 0)
(0 0 0 1 0 0)
U1 : Full Vector space of degree 3 over Rational Field
Generators:
(1 0 0)
(0 1 0)
(0 0 1)
U0 : Full Vector space of degree 3 over Rational Field
Generators:
(1 0 0)
(0 1 0)
(0 0 1)
> IsFullyNondegenerate(s);
false
```

Now we will construct the ideal r that evaluates to 0 and the largest subspace of $\mathbb{Q}^3$ not contained in the image.

```
> r := Ideal(t, [*F[1]!0, F[2].1, F[3].3*]);
> r;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Vector space of degree 6, dimension 0 over Rational Field
Generators:

U1 : Vector space of degree 3, dimension 1 over Rational Field
Generators:
(1 0 0)
Echelonized basis:
(1 0 0)
U0 : Vector space of degree 3, dimension 1 over Rational Field
Generators:
(0 0 1)
Echelonized basis:
(0 0 1)
> IsIdeal(t, r);
true
```

Finally, we quotient s by r to obtain a fully nondegenerate tensor.

```
> q := s/r;
> q;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 2 over Rational Field
U1 : Full Vector space of degree 2 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
> IsFullyNondegenerate(q);
true
```

Categorical Operations on Tensor Spaces

We have categorical notions for tensor spaces as well, and these are inherited from the module structure on tensor spaces.

SubConstructor(T, L): TenSpc, Any -> TenSpc, Map

sub< T | L >: TenSpc, Any -> TenSpc, Map

Returns the subtensor space of T generated by the tensors in the sequence L.

IsSubtensorSpace(T, S): TenSpc, TenSpc -> BoolElt

Decides if the tensor space S is a subtensor space of T.

Example: Subspace of Symmetric Forms (ex-40cd43)

We will construct the subspace S of symmetric forms from the tensor space T with frame $\mathbb{Q}^2 \times \mathbb{Q}^2 \rightarrowtail \mathbb{Q}$.

```
> K := Rationals();
> T := KTensorSpace(K, [2,2,1]);
> T;
Tensor space of dimension 4 over Rational Field with valence 3
U2 : Full Vector space of degree 2 over Rational Field
U1 : Full Vector space of degree 2 over Rational Field
U0 : Full Vector space of degree 1 over Rational Field
> S := sub< T | T.1, T.2+T.3, T.4 >;
> S;
Tensor space of dimension 3 over Rational Field with valence 3
U2 : Full Vector space of degree 2 over Rational Field
U1 : Full Vector space of degree 2 over Rational Field
U0 : Full Vector space of degree 1 over Rational Field
> IsSymmetric(S);
true
```

Now we will construct the subspace A of alternating forms from T.

```
> A := sub< T | T.2-T.3 >;
> A;
Tensor space of dimension 1 over Rational Field with valence 3
U2 : Full Vector space of degree 2 over Rational Field
U1 : Full Vector space of degree 2 over Rational Field
U0 : Full Vector space of degree 1 over Rational Field
> IsAlternating(A);
true
```

Now we verify that A is not a subtensor space of S, and thus, we have constructed a direct decomposition of T into its symmetric and alternating space.

```
> IsSubtensorSpace(S, A);
false
```

`QuoConstructor(T, X): TenSpc, Any -> TenSpc, Map`

`quo< T | X >: TenSpc, Any -> TenSpc, Map`

`T / S: TenSpc, TenSpc -> TenSpc, Map`

Returns the quotient tensor space of T by S.

Example: Quotients of Tensor Spaces (ex-87c8de)

We pick up with the same tensor spaces as the previous example: T has frame $\mathbb{Q}^2 \times \mathbb{Q}^2 \rightarrowtail \mathbb{Q}$, S is the symmetric subspace, and A is the alternating subspace.

```
> K := Rationals();
> T := KTensorSpace(K, [2,2,1]);
> T;
Tensor space of dimension 4 over Rational Field with valence 3
U2 : Full Vector space of degree 2 over Rational Field
U1 : Full Vector space of degree 2 over Rational Field
U0 : Full Vector space of degree 1 over Rational Field
> S := sub< T | T.1, T.2+T.3, T.4 >;
> A := sub< T | T.2-T.3 >;
```

Now we construct the quotient of T by A. The result is not a symmetric tensor space. Note that Q_2 is equivalent to a symmetric tensor modulo A, but this choice is arbitrary.

```
> Q := T/A;
> Q;
Tensor space of dimension 3 over Rational Field with valence 3
U2 : Full Vector space of degree 2 over Rational Field
U1 : Full Vector space of degree 2 over Rational Field
U0 : Full Vector space of degree 1 over Rational Field
> SystemOfForms(Q.1);
[
    [1 0]
    [0 0]
]
> SystemOfForms(Q.2);
[
    [0 0]
    [1 0]
]
> SystemOfForms(Q.3);
[
    [0 0]
    [0 1]
]
```

63.4.4 Homotopisms

Magma provides functions for homotopisms, i.e. morphisms of tensors. Homotopisms are also equipped with a tensor category. Because homotopisms can contain multiple maps between modules, it is not really clear what is meant by the domain or codomain of a given homotopism. In this context, the *domain* of a homotopism H will refer to the tensor, $t : U_\nu \times \cdots \times U_1 \rightarrowtail U_0$, where an arrow equal to 1 at coordinate a will mean that the corresponding map at coordinate a has domain equal to U_a. And in the same vain, the tensor $s : V_\nu \times \cdots \times V_1 \rightarrowtail V_0$ is the *codomain* of H if an arrow equal to 1 at coordinate a implies that the corresponding map at coordinate a has codomain equal to V_a.

Constructions of Homotopisms

`Homotopism(T, S, M : parameters): TenSpcElt, TenSpcElt, List -> Hmtp`

`Homotopism(T, S, M : parameters): TenSpcElt, TenSpcElt, SeqEnum -> Hmtp`

`Check: BoolElt` Default: `true`

`Homotopism(T, S, M, C : parameters): TenSpcElt, TenSpcElt, List, TenCat -> Hmtp`

`Homotopism(T, S, M, C : parameters): TenSpcElt, TenSpcElt, SeqEnum, TenCat -> Hmtp`

`Check: BoolElt` Default: `true`

Returns the homotopism from T to S given by the list of maps M and the category C. The default tensor category is the same as tensor categories for T and S. If the maps M will produce a homotopism, then set `Check` to `false` to skip the verification.

`Homotopism(M, C): List, TenCat -> Hmtp`

`Homotopism(M, C): SeqEnum, TenCat -> Hmtp`

Returns the homotopism given by the maps in M with tensor category C.

`IsHomotopism(T, s, H): TenSpcElt, TenSpcElt, Hmtp -> BoolElt`

`IsHomotopism(T, s, M): TenSpcElt, TenSpcElt, List -> BoolElt`

`IsHomotopism(T, s, M): TenSpcElt, TenSpcElt, SeqEnum -> BoolElt`

`IsHomotopism(T, s, M, C): TenSpcElt, TenSpcElt, List, TenCat -> BoolElt`

`IsHomotopism(T, s, M, C): TenSpcElt, TenSpcElt, SeqEnum, TenCat -> BoolElt`

Decides if the list of maps M induces a homotopism from T to S in the tensor category C. The default tensor category is the homotopism category. If it does induce a homotopism, it is also returned.

Example: Constructing Homotopisms (ex-6fd856)

We will construct two symmetric tensors $t, s : \mathbf{F}_3^3 \times \mathbf{F}_3^3 \rightarrowtail \mathbf{F}_3^3$ and apply permutations to the bases.

```
> T := KTensorSpace(GF(3), [3,3,3]);
> t := T.1+T.14+T.27;
> SystemOfForms(t);
[
    [1 0 0]
    [0 0 0]
    [0 0 0],

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

    [0 0 0]
    [0 0 0]
    [0 0 1]
]
> s := (T.4+T.10)+(T.8+T.20)+(T.18+T.24);
> SystemOfForms(s);
[
    [0 1 0]
    [1 0 0]
    [0 0 0],

    [0 0 1]
    [0 0 0]
```

(continues on next page)

(continued from previous page)

```
    [1 0 0],

    [0 0 0]
    [0 0 1]
    [0 1 0]
]
```

Now we construct a homotopism from t to t given by apply a permutation matrix in every coordinate.

```
> P := PermutationMatrix(GF(3), [2,1,3]);
> H := Homotopism(t, t, [*P, P, P*]);
> H;
Maps from U2 x U1 >-> U0 to V2 x V1 >-> V0.
U2 -> V2:
[0 1 0]
[1 0 0]
[0 0 1]
U1 -> V1:
[0 1 0]
[1 0 0]
[0 0 1]
U0 -> V0:
[0 1 0]
[1 0 0]
[0 0 1]
```

Note that this permutation matrix does not induce a homotopism of s.

```
> IsHomotopism(s, s, [*P, P, P*]);
false
```

Example: Homotopisms from Mixed Maps (ex-d02f12)

Homotopisms can take mixed categories of maps. To reuse the above example, we can encode the permutation as a `Map` and construct homotopisms from these types.

```
> V := VectorSpace(GF(3), 3);
> T := TensorSpace([V, V, V]);
> t := T.1+T.14+T.27;
> P := PermutationMatrix(GF(3), [2,1,3]);
> f := hom< V -> V | [<V.1, V.2>, <V.2, V.1>, <V.3, V.3>] >;
> H := Homotopism(t, t, [*f, f, f*]);
> H;
Maps from U2 x U1 >-> U0 to V2 x V1 >-> V0.
U2 -> V2: Mapping from: Full Vector space of degree 3 over GF(3) to Full
Vector space of degree 3 over GF(3)
U1 -> V1: Mapping from: Full Vector space of degree 3 over GF(3) to Full
Vector space of degree 3 over GF(3)
U0 -> V0: Mapping from: Full Vector space of degree 3 over GF(3) to Full
Vector space of degree 3 over GF(3)
```

Furthermore, we can input lists with types `Mtrx` and `Map` included.

```
> H2 := Homotopism(t, t, [*P, f, P*]);
> H2;
Maps from U2 x U1 >-> U0 to V2 x V1 >-> V0.
```

(continues on next page)

(continued from previous page)

```
U2 -> V2:
[0 1 0]
[1 0 0]
[0 0 1]
U1 -> V1: Mapping from: Full Vector space of degree 3 over GF(3) to Full
Vector space of degree 3 over GF(3)
U0 -> V0:
[0 1 0]
[1 0 0]
[0 0 1]
```

Basic Operations with Homotopisms

We provide some operations for homotopisms.

`H1 * H2: Hmtp, Hmtp -> Hmtp`
: Returns the composition of the homotopisms H_1 and H_2.

`H . a: Hmtp, RngIntElt -> Map`
: Returns the map on the ath coordinate.

Example: Operations with Homotopisms (ex-c4051c)

We construct a nondegenerate alternating form t on $V = \mathbb{Q}^6$. The group of isometries are isomorphic to $\mathrm{Sp}(6, \mathbb{Q})$, the group generated by all transvections. We construct a transvection L and a corresponding matrix.

```
> V := VectorSpace(Rationals(), 6);
> T := KTensorSpace(Rationals(), [6, 6, 1]);
> t := T.2-T.7+T.16-T.21+T.30-T.35;
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 6 over Rational Field
U1 : Full Vector space of degree 6 over Rational Field
U0 : Full Vector space of degree 1 over Rational Field
>
> u := V.2+2*V.3-V.5;
> L := map< V -> V | x :-> x + (x*t*u)[1]*u >;
> L;
Mapping from: ModTupFld: V to ModTupFld: V given by a rule [no inverse]
> M := Matrix(6, 6, [V.i @ L : i in [1..6]]);
> M;
[ 1  1  2  0 -1  0]
[ 0  1  0  0  0  0]
[ 0  0  1  0  0  0]
[ 0 -2 -4  1  2  0]
[ 0  0  0  0  1  0]
[ 0  1  2  0 -1  1]
```

We construct a homotopism from the transvection.

```
> H := Homotopism(t, t, [*L, L, IdentityMatrix(Rationals(), 1)*]);
> H;
Maps from U2 x U1 >-> U0 to V2 x V1 >-> V0.
U2 -> V2: Mapping from: Full Vector space of degree 6 over Rational Field to
Full Vector space of degree 6 over Rational Field given by a rule [no inverse]
U1 -> V1: Mapping from: Full Vector space of degree 6 over Rational Field to
Full Vector space of degree 6 over Rational Field given by a rule [no inverse]
```

(continues on next page)

(continued from previous page)

```
U0 -> V0:
[1]
```

Since H is an isometry of t, H^2 is also an isometry of t. We verify that the 2-coordinate map of H^2 is exactly M^2.

```
> H2 := H*H;
> H2;
Maps from U2 x U1 >-> U0 to V2 x V1 >-> V0.
U2 -> V2: Mapping from: Full Vector space of degree 6 over Rational Field to
Full Vector space of degree 6 over Rational Field
Composition of Mapping from: Full Vector space of degree 6 over Rational Field
to Full Vector space of degree 6 over Rational Field given by a rule [no
inverse] and
Mapping from: Full Vector space of degree 6 over Rational Field to Full Vector
space of degree 6 over Rational Field given by a rule [no inverse]
U1 -> V1: Mapping from: Full Vector space of degree 6 over Rational Field to
Full Vector space of degree 6 over Rational Field
Composition of Mapping from: Full Vector space of degree 6 over Rational Field
to Full Vector space of degree 6 over Rational Field given by a rule [no
inverse] and
Mapping from: Full Vector space of degree 6 over Rational Field to Full Vector
space of degree 6 over Rational Field given by a rule [no inverse]
U0 -> V0:
[1]
> M2 := Matrix(6, 6, [V.i @ H2.2 : i in [1..6]]);
> M2;
[ 1  2  4  0 -2  0]
[ 0  1  0  0  0  0]
[ 0  0  1  0  0  0]
[ 0 -4 -8  1  4  0]
[ 0  0  0  0  1  0]
[ 0  2  4  0 -2  1]
> M^2 eq M2;
true
```

Precompose(T, f, a): TenSpcElt, Map, RngIntElt -> TenSpcElt

Precompose(T, M, a): TenSpcElt, Mtrx, RngIntElt -> TenSpcElt

If $a > 0$, then the tensor returned is the tensor that has been pre-composed by the map f or matrix M.

T @ H: TenSpcElt, Hmtp -> TenSpcElt

If H is a cohomotopism (a homotopism in the cohomotopism category), then either the domain or codomain of H is returned, depending on the orientation of the arrows of H.

Basic Properties of Homotopisms

Domain(H): Hmtp -> TenSpcElt

Returns the domain tensor of H.

Codomain(H): Hmtp -> TenSpcElt

Returns the codomain tensor of H.

Maps(H): Hmtp -> List

Returns the list of maps for the various modules in the domain and codomain tensors.

TensorCategory(H): Hmtp -> TenCat

Returns the tensor category of H.

`ChangeTensorCategory(H, C): Hmtp, TenCat -> Hmtp`

`ChangeTensorCategory(~H, C): Hmtp, TenCat`

Changes the tensor category of H to the given category.

`Valence(H): Hmtp -> RngIntElt`

Returns the valence of the underlying tensor category of the homotopism H.

`Kernel(H): Hmtp -> TenSpcElt`

Returns the kernel of H as an ideal of its domain tensor.

`Image(H): Hmtp -> TenSpcElt`

Returns the image of H as a submap of the codomain tensor.

Example: Properties of Homotopisms `(ex-5a1efc)`

We demonstrate how to access properties of a homotopism. We construct tensors $t : \mathbb{Q}^4 \times \mathbb{Q}^4 \rightarrowtail \mathbb{Q}$ and $s : \mathbb{Q}^6 \times \mathbb{Q}^6 \rightarrowtail \mathbb{Q}$ given by the dot product.

```
> t := Tensor(IdentityMatrix(Rationals(), 4), 2, 1);
> s := Tensor(IdentityMatrix(Rationals(), 6), 2, 1);
> Z := ZeroMatrix(Rationals(), 4, 6);
> M := InsertBlock(Z, IdentityMatrix(Rationals(), 4), 1, 1);
> H := Homotopism(t, s, [*M, M, IdentityMatrix(Rationals(), 1)*]);
> H;
Maps from U2 x U1 >-> U0 to V2 x V1 >-> V0.
U2 -> V2:
[1 0 0 0 0 0]
[0 1 0 0 0 0]
[0 0 1 0 0 0]
[0 0 0 1 0 0]
U1 -> V1:
[1 0 0 0 0 0]
[0 1 0 0 0 0]
[0 0 1 0 0 0]
[0 0 0 1 0 0]
U0 -> V0:
[1]
```

Like with maps, we can obtain standard properties of homotopisms.

```
> Domain(H);
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 1 over Rational Field
> Codomain(H);
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 6 over Rational Field
U1 : Full Vector space of degree 6 over Rational Field
U0 : Full Vector space of degree 1 over Rational Field
> Maps(H);
[*
    [1 0 0 0 0 0]
    [0 1 0 0 0 0]
    [0 0 1 0 0 0]
    [0 0 0 1 0 0],

    [1 0 0 0 0 0]
    [0 1 0 0 0 0]
```

(continues on next page)

(continued from previous page)

```
    [0 0 1 0 0 0]
    [0 0 0 1 0 0],

    [1]
*]
> TensorCategory(H);
Tensor category of valence 3 (->,->,->) ({ 1 },{ 2 },{ 0 })
```

When the image and kernel can be computed for the each of the maps in the homotopism, then the image and kernel can be computed for the homotopism.

```
> Im := Image(H);
> Im;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Vector space of degree 6, dimension 4 over Rational Field
Echelonized basis:
(1 0 0 0 0 0)
(0 1 0 0 0 0)
(0 0 1 0 0 0)
(0 0 0 1 0 0)
U1 : Vector space of degree 6, dimension 4 over Rational Field
Echelonized basis:
(1 0 0 0 0 0)
(0 1 0 0 0 0)
(0 0 1 0 0 0)
(0 0 0 1 0 0)
U0 : Full Vector space of degree 1 over Rational Field
> Ker := Kernel(H);
> Ker;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Vector space of degree 4, dimension 0 over Rational Field
U1 : Vector space of degree 4, dimension 0 over Rational Field
U0 : Vector space of degree 1, dimension 0 over Rational Field
```

Shuffle(H, g): Hmtp, GrpPermElt -> Hmtp

Shuffle(H, g): Hmtp, [RngIntElt] -> Hmtp

Just like the shuffle for tensors, this returns the shuffle of the homotopism H. This is a functor from one tensor category to another and changes the order of the maps to

$$\{H_{\nu^g}, \ldots, H_{1^g}, H_{0^g}\}.$$

In order to be defined, $g \in \mathrm{Sym}(\{0, \ldots, \nu\})$. If $0^g \neq 0$, then both the image and pre-image of 0 under g will be replaced by their K-dual space. For cotensors, $g \in \mathrm{Sym}(\{1, \ldots, \nu\})$. Sequences $[a_1, \ldots, a_{\nu+1}]$ will be interpreted as a permutation in one-line notation.

63.5 Linear Invariants of Tensors

63.5.1 Invariants for Bilinear Tensors

The following intrinsics are specialized for tensors of valence 3, but equivalent intrinsics for general tensors are presented in the proceeding subsection.

AdjointAlgebra(B): TenSpcElt -> AlgMat

Returns the adjoint $*$-algebra of the given Hermitian bilinear map B, represented on $\mathrm{End}(\mathrm{U}_2)$. This is using algorithms from StarAlge, see the `AdjointAlgebra` intrinsic in [BW]. If the current version of StarAlge is not attached, the default Magma version will be used instead.

Example: Adjoint Algebra of a p-Group Tensor (ex-55dbd9)

Given the context of [BW12b], we construct a tensor from a p-group and compute its adjoint algebra.

```
> G := SmallGroup(3^7, 7000);
> t := pCentralTensor(G);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over GF(3)
U1 : Full Vector space of degree 4 over GF(3)
U0 : Full Vector space of degree 3 over GF(3)
```

Unlike other intrinsics that compute invariants of tensors, `AdjointAlgebra` exploits the fact that t is Hermitian so that the adjoint algebra is faithfully represented on $\mathrm{End}(\mathrm{U}_2) = \mathrm{End}(\mathrm{U}_1)$.

```
> A := AdjointAlgebra(t);
> A;
Matrix Algebra of degree 4 and dimension 4 with 4 generators over GF(3)
> A.1;
[1 0 0 0]
[0 0 0 0]
[0 0 0 0]
[0 0 0 1]
```

Because A is constructed from algorithms for $*$-algebras, we can apply other algorithms from that package specifically dealing with the involution on A.

```
> RecognizeStarAlgebra(A);
true
> SimpleParameters(A);
[ <"symplectic", 2, 3> ]
> Star(A);
Mapping from: AlgMat: A to AlgMat: A given by a rule [no inverse]
```

`LeftNucleus(B : parameters): TenSpcElt -> AlgMat`

```
op: BoolElt                    Default: false
```

Returns the left nucleus of the bilinear map B as a subalgebra of $\mathrm{End}_{\mathrm{K}}(\mathrm{U}_2) \times \mathrm{End}_{\mathrm{K}}(\mathrm{U}_0)$. In previous versions of TensorSpace (and eMAGma), the left nucleus was returned as a subalgebra of $\mathrm{End}(\mathrm{U}_2)^\circ \times \mathrm{End}(\mathrm{U}_0)^\circ$. To enable this, set the optional argument `op` to `true`.

`MidNucleus(B): TenSpcElt -> AlgMat`

Returns the mid nucleus of the bilinear map B as a subalgebra of $\mathrm{End}_{\mathrm{K}}(\mathrm{U}_2) \times \mathrm{End}_{\mathrm{K}}(\mathrm{U}_1)$.

`RightNucleus(B): TenSpcElt -> AlgMat`

Returns the right nucleus of the bilinear map B as a subalgebra of $\mathrm{End}_{\mathrm{K}}(\mathrm{U}_1) \times \mathrm{End}_{\mathrm{K}}(\mathrm{U}_0)$.

Example: Nuclei Embed in the Derivation Algebra (ex-75db37)

We will verify a theorem from [FMW19] and [Wil17]: all the nuclei of a tensor embed into the derivation algebra. We construct the tensor given by $(3 \times 4 \times 5)$-matrix multiplication.

```
> K := Rationals();
> A := KMatrixSpace(K, 3, 4);
> B := KMatrixSpace(K, 4, 5);
> C := KMatrixSpace(K, 3, 5);
> F := func< x | x[1]*x[2] >;
> t := Tensor([A, B, C], F);
```

(continues on next page)

(continued from previous page)

```
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 12 over Rational Field
U1 : Full Vector space of degree 20 over Rational Field
U0 : Full Vector space of degree 15 over Rational Field
```

Because matrix multiplication is associative, the left, middle, and right nuclei contain $\mathrm{Mat}_3(\mathbb{Q})$, $\mathrm{Mat}_4(\mathbb{Q})$, and $\mathrm{Mat}_5(\mathbb{Q})$ respectively. In fact, the following computation shows that this is equality.

```
> L := LeftNucleus(t : op := true);
> M := MidNucleus(t);
> R := RightNucleus(t);
> Dimension(L), Dimension(M), Dimension(R);
9 16 25
> D := DerivationAlgebra(t);
> Dimension(D);
49
```

Now we will embed these nuclei into the derivation algebra of t.

```
> Omega := KMatrixSpace(K, 47, 47);
> Z1 := ZeroMatrix(K, 20, 20);
> L_L2, L2 := Induce(L, 2);
> L_L0, L0 := Induce(L, 0);
> embedL := map< L -> Omega | x :->
>     DiagonalJoin(<Transpose(x @ L_L2), Z1, Transpose(x @ L_L0)>) >;
>
> Z0 := ZeroMatrix(K, 15, 15);
> M_M2, M2 := Induce(M, 2);
> M_M1, M1 := Induce(M, 1);
> embedM := map< M -> Omega | x :->
>     DiagonalJoin(<x @ M_M2, -Transpose(x @ M_M1), Z0>) >;
>
> Z2 := ZeroMatrix(K, 12, 12);
> R_R1, R1 := Induce(R, 1);
> R_R0, R0 := Induce(R, 0);
> embedR := map< R -> Omega | x :->
>     DiagonalJoin(<Z2, x @ R_R1, x @ R_R0>) >;
>
> Random(Basis(L)) @ embedL in D;
true
> Random(Basis(M)) @ embedM in D;
true
> Random(Basis(R)) @ embedR in D;
true
```

63.5.2 Invariants of General Multilinear Maps

The following functions can be used for general tensors.

Centroid(T): TenSpcElt -> AlgMat

Centroid(T, A): TenSpcElt, {RngIntElt} -> AlgMat

Returns the A-centroid of the tensor as a subalgebra of

$$\prod_{a \in A} \mathrm{End}(\mathrm{U_a}),$$

where $A \subseteq [\nu]$. If no A is given, it is assumed that $A = [\nu]$. If t is contained in a category where coordinates a and b (also contained in A) are fused together, then the corresponding operators on those coordinates will be equal.

Example: Centroid of a Multiplication Tensor (ex-71158e)

The centroid C of a tensor t is the largest ring for which t is C-linear, see [FMW19, Theorem D]. To demonstrate this, we will construct the tensor given by multiplication of the splitting field of $f(x) = x^4 - x^2 - 2$ over $\mathbb{Q}$. However, this field won't explicitly be given with the tensor data.

```
> A := MatrixAlgebra(Rationals(), 4);
> R<x> := PolynomialRing(Rationals());
> F := sub< A | A!1, CompanionMatrix(x^4-x^2-2) >;
> F;
Matrix Algebra of degree 4 with 2 generators over Rational Field
> t := Tensor(F);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 4 over Rational Field
```

The centroid is the field $\mathbb{Q}(\sqrt{2}, i)$.

```
> C := Centroid(t);
> C;
Matrix Algebra of degree 12 with 4 generators over Rational Field
> sub< C | C.1 > eq C;
true
> forall{ c : c in Generators(C) | IsInvertible(c) };
true
> IsCommutative(C);
true
> MinimalPolynomial(C.1);
x^4 + 2*x^2 - 8
> Factorization(MinimalPolynomial(C.1));
[
    <x^2 - 2, 1>,
    <x^2 + 4, 1>
]
```

DerivationAlgebra(T): TenSpcElt -> AlgMatLie

DerivationAlgebra(T, A): TenSpcElt, {RngIntElt} -> AlgMatLie

Returns the A-derivation Lie algebra of the tensor as a Lie subalgebra of

$$\prod_{a \in A} \mathrm{End}(\mathrm{U}_a),$$

where $A \subseteq [\nu]$. If no A is given, it is assumed that $A = [\nu]$. If t is contained in a category where coordinates a and b (also contained in A) are fused together, then the corresponding operators on those coordinates will be equal.

Nucleus(T, a, b): TenSpcElt, RngIntElt, RngIntElt -> AlgMat

Nucleus(T, A): TenSpcElt, SetEnum -> AlgMat

Returns the A-nucleus, for $A = \{a, b\}$ $(a \neq b)$, of the tensor as a subalgebra of $\mathrm{End}(\mathrm{U}_i) \times \mathrm{End}(\mathrm{U}_j)$, where $i = \max(a, b)$ and $j = \min(a, b)$. If $j > 0$, then replace $\mathrm{End}(\mathrm{U}_j)$ with $\mathrm{End}(\mathrm{U}_j)^\circ$. If T is contained in a category where coordinates a and b are fused together, then the corresponding operators on those coordinates will not be forced to be equal.

Example: Restricting the Derivation Algebra (ex-84e946)

In a previous example, we embedded the nuclei of a tensor into the derivation algebra. For a tensor $t : U_\nu \times \cdots \times U_1 \rightarrowtail U_0$, the derivation algebra is represented in

$$\Omega = \prod_{a \in [\nu]} \mathrm{GL}(\mathrm{U_a}).$$

We will restrict the derivation algebra to

$$\prod_{c \notin \{a,b\}} \mathrm{GL}(\mathrm{U_c})$$

for distinct $a, b \in [\nu]$. From [FMW19, Lemma 4.11], the kernel of this restriction is equal to $\mathrm{Nuc}_{\{a,b\}}(t)^-$. We will just verify that the dimensions match.

We will construct a tensor given by matrix multiplication:

$$\mathrm{Mat}_{3\times 4}(\mathbf{F}_2) \times \mathrm{Mat}_{4\times 2}(\mathbf{F}_2) \times \mathrm{Mat}_{2\times 2}(\mathbf{F}_2) \rightarrowtail \mathrm{Mat}_{3\times 2}(\mathbf{F}_2).$$

```
> A := KMatrixSpace(GF(2), 3, 4);
> B := KMatrixSpace(GF(2), 4, 2);
> C := KMatrixSpace(GF(2), 2, 2);
> D := KMatrixSpace(GF(2), 3, 2);
> trip := func< x | x[1]*x[2]*x[3] >;
> t := Tensor([A, B, C, D], trip);
> t;
Tensor of valence 4, U3 x U2 x U1 >-> U0
U3 : Full Vector space of degree 12 over GF(2)
U2 : Full Vector space of degree 8 over GF(2)
U1 : Full Vector space of degree 4 over GF(2)
U0 : Full Vector space of degree 6 over GF(2)
```

Now we will compute the derivation algebra of `t`. We choose $a = 3$ and $b = 2$, so the $\{3, 2\}$-nucleus is $\mathrm{Mat}_{4\times 4}(\mathbf{F}_2)$.

```
> D := DerivationAlgebra(t);
> Dimension(D);
32
> N32 := Nucleus(t, 3, 2);
> N32;
Matrix Algebra of degree 20 with 16 generators over GF(2)
```

To construct the restriction of Der(t) into $\mathrm{GL}(\mathrm{U}_1) \times \mathrm{GL}(\mathrm{U}_0)$ we will use the `Induce` function.

```
> Omega_10 := KMatrixSpace(GF(2), 10, 10);
> D_vs := sub< KMatrixSpace(GF(2), 30, 30) | Basis(D) >;
> pi1, D1 := Induce(D, 1);
> pi0, D0 := Induce(D, 0);
> res := hom< D_vs -> Omega_10 |
>     [<x, DiagonalJoin(x @ pi1, x @ pi0)> : x in Basis(D)] >;
> res;
Mapping from: ModMatFld: D_vs to ModMatFld: Omega_10
> Kernel(res);
KMatrixSpace of 30 by 30 matrices and dimension 16 over GF(2)
```

`SelfAdjointAlgebra(t, a, b): TenSpcElt, RngIntElt, RngIntElt -> ModMatFld`

Returns the self-adjoint elements of the ab-nucleus of t as a subspace of $\mathrm{End}(\mathrm{U_a})$, with $a, b \in [\nu]$ and $a \neq b$. It is not required that t be in a tensor category with coordinates a and b fused. Unlike other invariants associated to tensors, the self-adjoint algebra is not currently stored with the tensor.

`TensorOverCentroid(T): TenSpcElt -> TenSpcElt, Hmtp`

If the given tensor T is framed by K-vector spaces, then the returned tensor is framed by E-vector spaces where E is the residue field of the centroid. The returned homotopism is an isotopism of the K-tensors. This only works if the centroid of T is a finite commutative local ring. We employ the algorithms developed by Brooksbank and Wilson [BW15] to efficiently determine if a matrix algebra is cyclic.

Example: Recovering a Field with the Centroid (ex-93b5e7)

In the context of groups, centroids can be used to recover an underlying field of a matrix group, even if the given group is not input as such. Here we will construct the exponent-p central tensor of the Sylow 2-subgroup of $\mathrm{GL}(3, \mathrm{GF}(2^{10}))$. We will not print the `GrpPC` version of this group as the number of relations is very large.

```
> U := ClassicalSylow(GL(3, 2^10), 2);
> U.3;
[       1    $.1^2        0]
[       0        1        0]
[       0        0        1]
> G := PCPresentation(UnipotentMatrixGroup(U));
> #G eq 2^30;
true
> t := pCentralTensor(G);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 20 over GF(2)
U1 : Full Vector space of degree 20 over GF(2)
U0 : Full Vector space of degree 10 over GF(2)
```

Even though our tensor right now is $\mathbf{F}_2^{20} \times \mathbf{F}_2^{20} \rightarrowtail \mathbf{F}_2^{10}$, we know it is the 2-dimensional alternating form over $\mathrm{GF}(2^{10})$. We will construct the centroid, and then rewrite our tensor over the centroid to get the tensor we expect.

```
> C := Centroid(t);
> C;
Matrix Algebra of degree 50 and dimension 10 with 1 generator over GF(2)
> IsCyclic(C) and IsSimple(C);
true
> s := TensorOverCentroid(t);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 2 over GF(2^10)
U1 : Full Vector space of degree 2 over GF(2^10)
U0 : Full Vector space of degree 1 over GF(2^10)
```

63.6 Some Extended Examples

In this section some further examples are presented to demonstrate some uses of this package.

63.6.1 Distinguishing Groups

Example: Distinguishing Payne's Elation Groups (ex-d908f2)

We can use these functions to build groups from bilinear maps and distinguish seemingly indistinguishable groups. In 2004, S. E. Payne asked if two elation groups were isomorphic but suspected they were not [Pay06].

The first group, G_f, is the elation group of the generalized quadrangle $H(3, q^2)$, the Hermitian geometry. This group is defined as a Heisenberg group whose bilinear map is the usual dot product.

```
> p := 3;
> e := 4;
> q := p^e; // q = 3^e >= 27
> F := [KSpace(GF(q),2), KSpace(GF(q),2), KSpace(GF(q),1)];
>
> DotProd := function(x)
>   return KSpace(GF(q),1)!(x[1]*Matrix(2,1,x[2]));
> end function;
>
> DoubleForm := function(T)
>   F := SystemOfForms(T)[1];
>   K := BaseRing(F);
>   n := Nrows(F);
>   m := Ncols(F);
>   MS := KMatrixSpace(K,n,m);
>   Z := MS!0;
>   M1 := HorizontalJoin(Z,-Transpose(F));
>   M2 := HorizontalJoin(F,Z);
>   D := VerticalJoin( M1, M2 );
>   return Tensor( D, 2, 1 );
> end function;
>
> f := DoubleForm( Tensor( F, DotProd ) );
> f;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over GF(3^4)
U1 : Full Vector space of degree 4 over GF(3^4)
U0 : Full Vector space of degree 1 over GF(3^4)
>
> IsAlternating(f);
true
> Gf := HeisenbergGroupPC(f);
```

Now we define Payne's second group, $G_{\bar{f}}$, which is the elation group of the Roman quadrangle with parameters (q^2, q). In this example, $\bar{f}$ is a biadditive map, but is bilinear over the prime field $\mathbb{F}_3$. Therefore, we construct a vector space isomorphism from $\mathbb{F}_3^e$ to $\mathbb{F}_{3^e}$ and the bilinear commutator map, induced by $\bar{f}$. Hence, $G_{\bar{f}}$ is the Heisenberg group of this bilinear commutator map.

```
> n := PrimitiveElement(GF(q)); // nonsquare
> MS := KMatrixSpace(GF(q),2,2);
> A := MS![-1,0,0,n];
> B := MS![0,1,1,0];
> C := MS![0,0,0,n^-1];
> F1 := Frame(f);
> F2 := [KSpace(GF(p),4*e), KSpace(GF(p),4*e), KSpace(GF(p),e)];
>
> // take 1/3^r root
> Root := function(v,r)
>   k := Eltseq(v)[1];
>   K := Parent(k);
>   if k eq K!0 then return k; end if;
>   R<x> := PolynomialRing(K);
>   f := Factorization(x^(3^r)-k)[1][1];
>   return K!(x-f);
> end function;
>
> // bi-additive map defining elation grp
```

(continues on next page)

(continued from previous page)

```
> RomanGQ := function(x)
>   u := Matrix(1,2,x[1]);
>   v := Matrix(2,1,x[2]);
>   M := [A,B,C];
>   f := &+[Root(u*M[i]*v,i-1) : i in [1..3]];
>   return KSpace(GF(q),1)![f];
> end function;
>
> // vector space isomorphisms
> phi := map< F2[1] -> F1[1] |
>   x :-> F1[1]![ GF(q)![ s : s in Eltseq(x)[i+1..e+i] ] :
>     i in [0,e,2*e,3*e] ] >;
> gamma := map< F1[3] -> F2[3] |
>   x :-> F2[3]!&cat[ Eltseq(s) : s in Eltseq(x) ] >;
>
> // bilinear commutator from RomanGQ
> RomanGQComm := function(x)
>   x1 := Eltseq(x[1]@phi)[1..2];
>   x2 := Eltseq(x[1]@phi)[3..4];
>   y1 := Eltseq(x[2]@phi)[1..2];
>   y2 := Eltseq(x[2]@phi)[3..4];
>   comm := RomanGQ( <x2,y1> ) - RomanGQ( <y2,x1> );
>   return comm @ gamma;
> end function;
>
> f_bar := Tensor( F2, RomanGQComm );
> f_bar;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 16 over GF(3)
U1 : Full Vector space of degree 16 over GF(3)
U0 : Full Vector space of degree 4 over GF(3)
>
> IsAlternating(f_bar);
true
> Gfb := HeisenbergGroupPC(f_bar);
```

The groups G_f and $G_{\bar{f}}$ have order 3^{20} and are class 2, exponent 3, and minimally generated by 16 elements. In other words, the groups G_f and $G_{\bar{f}}$ are central extensions of $\mathbb{Z}_3^{16}$ by $\mathbb{Z}_3^4$ and have exponent 3. Using standard heuristics, these groups are indistinguishable. However, the invariants associated to their exponent-p central tensor are vastly different, and thus, they determine that these groups are nonisomorphic. We show that the centroids of the tensors are not isomorphic.

```
> Tf := pCentralTensor(Gf,1,1);
> Tf;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 32 over GF(3)
U1 : Full Vector space of degree 32 over GF(3)
U0 : Full Vector space of degree 4 over GF(3)
>
> Tfb := pCentralTensor(Gfb,1,1);
> Tfb;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 32 over GF(3)
U1 : Full Vector space of degree 32 over GF(3)
U0 : Full Vector space of degree 4 over GF(3)
>
```

(continues on next page)

(continued from previous page)

```
> Cf := Centroid(Tf);
> Cfb := Centroid(Tfb);
> Dimension(Cf) eq Dimension(Cfb);
false
```

63.6.2 Simplifying Automorphism Group Computations

Example: Automorphisms via the Adjoint Algebra (ex-c87b8b)

We demonstrate how to simplify the automorphism group computation as discussed in [BW14]. We construct a class 2, exponent p, p-group G which is a quotient of a maximal unipotent subgroup of $\mathrm{GL}(3, 317^4)$.

```
> p := 317;
> e := 4;
> H := ClassicalSylow( GL(3,p^e), p );
> U := UnipotentMatrixGroup(H);
> P := PCPresentation(U);
> Z := Center(P);
>
> N := sub< P | >;
> while #N lt p^2 do
>   N := sub< P | Random(Z), N >;
> end while;
>
> G := P/N;
> G;
GrpPC : G of order 1024690293163428677944144 9 = 317^10
PC-Relations:
    G.5^G.1 = G.5 * G.9^62 * G.10^133,
    G.5^G.2 = G.5 * G.9^312 * G.10^295,
    G.5^G.3 = G.5 * G.9^316,
    G.5^G.4 = G.5 * G.10^316,
    G.6^G.1 = G.6 * G.9^312 * G.10^295,
    G.6^G.2 = G.6 * G.9^316,
    G.6^G.3 = G.6 * G.10^316,
    G.6^G.4 = G.6 * G.9^138 * G.10^163,
    G.7^G.1 = G.7 * G.9^316,
    G.7^G.2 = G.7 * G.10^316,
    G.7^G.3 = G.7 * G.9^138 * G.10^163,
    G.7^G.4 = G.7 * G.9^188 * G.10^50,
    G.8^G.1 = G.8 * G.10^316,
    G.8^G.2 = G.8 * G.9^138 * G.10^163,
    G.8^G.3 = G.8 * G.9^188 * G.10^50,
    G.8^G.4 = G.8 * G.9^125 * G.10^151
```

We construct the exponent-p central tensor of G and compute its adjoint $*$-algebra A.

```
> T := pCentralTensor(G,1,1);
> T;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 8 over GF(317)
U1 : Full Vector space of degree 8 over GF(317)
U0 : Full Vector space of degree 2 over GF(317)
>
> A := AdjointAlgebra(T);
```

(continues on next page)

(continued from previous page)

```
> Dimension(A);
16
> star := Star(A);
```

If $V = G/\Phi(G)$ is the Frattini quotient of G, then our goal is to get the cotensor space $V \wedge_A V$. Note that $\dim V \wedge V = 28$, so standard methods will compute a stabilizer of $\mathrm{GL}(8, 317)$ inside $V \wedge V$. We will decrease the size of the ambient space resulting in an easier stabilizer computation.

```
> V := Domain(T)[1];
> E := ExteriorCotensorSpace(V,2);
> E;
Cotensor space of dimension 28 over GF(317) with valence 1
U2 : Full Vector space of degree 8 over GF(317)
U1 : Full Vector space of degree 8 over GF(317)
```

Now we create a sub cotensor space S generated by all $(e_i X) \wedge e_j - e_i \wedge (e_j X)$ for $X \in A$, and then quotient $V \wedge V$ by S. The result is a 4 dimensional space.

```
> L := [];
> for E_gen in Generators(E) do
>   F := SystemOfForms(E_gen)[1];
>   for X in Basis(A) do
>     L cat:= [E!Eltseq(X*F - F*Transpose(X@star))];
>   end for;
> end for;
>
> S := SubtensorSpace(E,L);
> S;
Cotensor space of dimension 24 over GF(317) with valence 1
U2 : Full Vector space of degree 8 over GF(317)
U1 : Full Vector space of degree 8 over GF(317)
>
> Q := E/S;
> Q;
Cotensor space of dimension 4 over GF(317) with valence 1
U2 : Full Vector space of degree 8 over GF(317)
U1 : Full Vector space of degree 8 over GF(317)
```

63.7 Bibliography

- [Alb42] A. A. Albert. Non-associative algebras. I. Fundamental concepts and isotopy. *Ann. of Math. (2)*, 43:685–707, 1942. URL: https://doi.org/10.2307/1968960, doi:10.2307/1968960.
- [BMW17] Peter A. Brooksbank, Joshua Maglione, and James B. Wilson. A fast isomorphism test for groups whose Lie algebra has genus 2. *J. Algebra*, 473:545–590, 2017. URL: https://doi.org/10.1016/j.jalgebra.2016.12.007, doi:10.1016/j.jalgebra.2016.12.007.
- [BW] Peter A. Brooksbank and James B. Wilson. StarAlge. https://github.com/thetensorspace/StarAlge.
- [BW12b] Peter A. Brooksbank and James B. Wilson. Computing isometry groups of Hermitian maps. *Trans. Amer. Math. Soc.*, 364(4):1975–1996, 2012. URL: https://doi.org/10.1090/S0002-9947-2011-05388-2, doi:10.1090/S0002-9947-2011-05388-2.
- [BW14] Peter A. Brooksbank and James B. Wilson. Groups acting on tensor products. *J. Pure Appl. Algebra*, 218(3):405–416, 2014. URL: https://doi.org/10.1016/j.jpaa.2013.06.011, doi:10.1016/j.jpaa.2013.06.011.

- [BW15] Peter A. Brooksbank and James B. Wilson. The module isomorphism problem reconsidered. *J. Algebra*, 421:541–559, 2015. URL: https://doi.org/10.1016/j.jalgebra.2014.09.004, doi:10.1016/j.jalgebra.2014.09.004.
- [FMW19] Uriya First, Joshua Maglione, and James B. Wilson. A spectral theory for transverse tensor operators. 2019. arXiv:1911.02518.
- [Lan12] J. M. Landsberg. *Tensors: geometry and applications*. Volume 128 of Graduate Studies in Mathematics. American Mathematical Society, Providence, RI, 2012. ISBN 978-0-8218-6907-9.
- [Lee13] John M. Lee. *Introduction to smooth manifolds*. Volume 218 of Graduate Texts in Mathematics. Springer, New York, second edition, 2013. ISBN 978-1-4419-9981-8.
- [Pay06] Stanley E. Payne. Finite groups that admit Kantor families. In *Finite geometries, groups, and computation*, pages 191–202. Walter de Gruyter, Berlin, 2006.
- [Wey50a] Herman Weyl. *The theory of groups and quantum mechanics*. Dover, New York, 1950.
- [Wil13a] James B. Wilson. Division, adjoints, and dualities of bilinear maps. *Comm. Algebra*, 41(11):3989–4008, 2013. URL: https://doi.org/10.1080/00927872.2012.660668, doi:10.1080/00927872.2012.660668.
- [Wil17] James B. Wilson. On automorphisms of groups, rings, and algebras. *Comm. Algebra*, 45(4):1452–1478, 2017. URL: https://doi.org/10.1080/00927872.2016.1175617, doi:10.1080/00927872.2016.1175617.

References

[Alb42] A. A. Albert. Non-associative algebras. I. Fundamental concepts and isotopy. *Ann. of Math. (2)*, 43:685–707, 1942. URL: https://doi.org/10.2307/1968960, doi:10.2307/1968960.

[BMW17] Peter A. Brooksbank, Joshua Maglione, and James B. Wilson. A fast isomorphism test for groups whose Lie algebra has genus 2. *J. Algebra*, 473:545–590, 2017. URL: https://doi.org/10.1016/j.jalgebra.2016.12.007, doi:10.1016/j.jalgebra.2016.12.007.

[BW] Peter A. Brooksbank and James B. Wilson. StarAlge. https://github.com/thetensorspace/StarAlge.

[BW12b] Peter A. Brooksbank and James B. Wilson. Computing isometry groups of Hermitian maps. *Trans. Amer. Math. Soc.*, 364(4):1975–1996, 2012. URL: https://doi.org/10.1090/S0002-9947-2011-05388-2, doi:10.1090/S0002-9947-2011-05388-2.

[BW14] Peter A. Brooksbank and James B. Wilson. Groups acting on tensor products. *J. Pure Appl. Algebra*, 218(3):405–416, 2014. URL: https://doi.org/10.1016/j.jpaa.2013.06.011, doi:10.1016/j.jpaa.2013.06.011.

[BW15] Peter A. Brooksbank and James B. Wilson. The module isomorphism problem reconsidered. *J. Algebra*, 421:541–559, 2015. URL: https://doi.org/10.1016/j.jalgebra.2014.09.004, doi:10.1016/j.jalgebra.2014.09.004.

[FMW19] Uriya First, Joshua Maglione, and James B. Wilson. A spectral theory for transverse tensor operators. 2019. arXiv:1911.02518.

[Lan12] J. M. Landsberg. *Tensors: geometry and applications*. Volume 128 of Graduate Studies in Mathematics. American Mathematical Society, Providence, RI, 2012. ISBN 978-0-8218-6907-9.

[Lee13] John M. Lee. *Introduction to smooth manifolds*. Volume 218 of Graduate Texts in Mathematics. Springer, New York, second edition, 2013. ISBN 978-1-4419-9981-8.

[Pay06] Stanley E. Payne. Finite groups that admit Kantor families. In *Finite geometries, groups, and computation*, pages 191–202. Walter de Gruyter, Berlin, 2006.

[Wey50a] Herman Weyl. *The theory of groups and quantum mechanics*. Dover, New York, 1950.

[Wil13a] James B. Wilson. Division, adjoints, and dualities of bilinear maps. *Comm. Algebra*, 41(11):3989–4008, 2013. URL: https://doi.org/10.1080/00927872.2012.660668, doi:10.1080/00927872.2012.660668.

[Wil17] James B. Wilson. On automorphisms of groups, rings, and algebras. *Comm. Algebra*, 45(4):1452–1478, 2017. URL: https://doi.org/10.1080/00927872.2016.1175617, doi:10.1080/00927872.2016.1175617.

Index

Symbols

(u, v) (*literal*), 8
T (*operation*), 105
- u (*operation*), 7
-T (*operation*), 65

A

a * b (*operation*), 17
a * g (*operation*), 40
a ^ -1 (*operation*), 17
a(u) (*literal*), 17
AbsoluteBasis(M) (*function*), 33
Adjoint Algebra of a p-Group Tensor (*example*), 126
AdjointAlgebra(B) (*function*), 126
AdjointCategory(v, s, t) (*function*), 111
Algebra(C) (*function*), 36
Alternating Tensor of a Lie Algebra (*example*), 62
AlternatingSpace(T) (*function*), 101
AlternatingTensor(T) (*function*), 62
AnticommutatorTensor(A) (*function*), 59
AntisymmetricSpace(T) (*function*), 101
AntisymmetricTensor(T) (*function*), 62
Arithmetic on Module Elements (*example*), 7
Arrows(C) (*function*), 112
AsCotensorSpace(T) (*function*), 82
AsMatrices(T, s, t) (*function*), 56
Assign(~T, ind, k) (*procedure*), 53
Assign(T, ind, k) (*function*), 53
Associated Form of a Lie Algebra Tensor (*example*), 66
AssociatedForm(T) (*function*), 66
Associator Tensor of the Octonions (*example*), 60
AssociatorTensor(A) (*function*), 60
AsTensor(S) (*function*), 84
AsTensorSpace(T, i) (*function*), 82
Automorphisms via the Adjoint Algebra (*example*), 134

B

BaseField(M) (*function*), 5
BaseField(T) (*function*), 67, 108
BaseRing(M) (*function*), 5, 23
BaseRing(T) (*function*), 67, 108
Bases and Other Module Functions (*example*), 28
Basic Properties of a Tensor (*example*), 67
Basic Tensor Category Construction (*example*), 110
Basis(M) (*function*), 9, 27
Basis(P) (*function*), 33
Basis(T) (*function*), 105
Black-Box Cross Product (*example*), 49
Black-Box Dot Product Tensor (*example*), 49
Black-Box Triple Product (*example*), 50
BoundaryMap(C, n) (*function*), 36
BoundaryMaps(C) (*function*), 36

C

c * u (*operation*), 32
C : Magma (*literal*), 47
C1 eq C2 (*operation*), 111
Categories of Tensor Spaces (*example*), 108
Category of a Right Module Tensor (*example*), 68
Centroid of a Multiplication Tensor (*example*), 129
Centroid(T) (*function*), 128
Centroid(T, A) (*function*), 128
Chain Maps over a Basic Algebra (*example*), 41
ChainMap(Q, C, D, n) (*function*), 40
ChangeRing(M, S) (*function*), 5
ChangeRing(M, S, f) (*function*), 5
ChangeTensorCategory(~H, C) (*procedure*), 125
ChangeTensorCategory(~T, C) (*procedure*), 68, 108
ChangeTensorCategory(H, C) (*function*), 125
ChangeTensorCategory(T, C) (*function*), 68, 108
ChangeUniverse(~x, R) (*function*), 5
CharacteristicVector(M, S) (*function*), 6
Codomain(a) (*function*), 17
Codomain(H) (*function*), 124
Codomain(S) (*function*), 17
Codomain(T) (*function*), 67
CoefficientField(M) (*function*), 5
CoefficientIdeals(P) (*function*), 33
CoefficientRing(M) (*function*), 5, 23
Coercion into a Tensor Space (*example*), 103
Coercion into Quotient Modules (*example*), 31
CohomotopismCategory(v) (*function*), 111
Cokernel(a) (*function*), 17
Cokernel(f) (*function*), 40
Commutator Tensor of a Matrix Algebra (*example*), 59
CommutatorTensor(A) (*function*), 59
Complex(f, d) (*function*), 35

Complex(L, d) (*function*), 35
Complexes over a Quiver Algebra (*example*), 38
Composing Tensors with Linear Maps (*example*), 74
Compress(~T) (*procedure*), 66
Compress(T) (*function*), 66
ConnectingHomomorphism(f, g, n) (*function*), 43
Constructing Homotopisms (*example*), 121
Containment of Tensor Spaces (*example*), 104
Contents(M) (*function*), 24
Coordinates(M, u) (*function*), 9
Coradical(T) (*function*), 86
CotensorCategory(A, P) (*function*), 110
CotensorSpace(S) (*function*), 100
CotensorSpace(S, C) (*function*), 100
Creating Modules over Dedekind Domains (*example*), 20

D

Degree(f) (*function*), 40
Degree(M) (*function*), 23
Degrees(C) (*function*), 36
Depth(v) (*function*), 8
DerivationAlgebra(T) (*function*), 129
DerivationAlgebra(T, A) (*function*), 129
DerivedFrom(~X, t, C, RC : parameters) (*procedure*), 86
Determinant(M) (*function*), 24
Dimension(M) (*function*), 24
Dimension(pm) (*function*), 33
Dimension(T) (*function*), 105
DimensionOfHomology(C, n) (*function*), 36
DimensionsOfHomology(C) (*function*), 36
DimensionsOfTerms(C) (*function*), 36
DirectSum(C, D) (*function*), 37
DirectSum(M, N) (*function*), 6
DirectSum(M1, M2) (*function*), 25
DirectSum(Q) (*function*), 6
DirectSum(S) (*function*), 25
Discriminant of the Octonions (*example*), 87
Discriminant(B) (*function*), 87
Distinguishing Payne's Elation Groups (*example*), 131
Domain(a) (*function*), 17
Domain(H) (*function*), 124
Domain(S) (*function*), 17
Domain(T) (*function*), 67
Domains of a Bilinear Map (*example*), 78
Dual(C) (*function*), 35
Dual(C, n) (*function*), 35
Dual(M) (*function*), 27

E

Elementary Functions on a Module (*example*), 24
ElementaryDivisors(M, N) (*function*), 27
Elements of a Free Module over Z[x] (*example*), 6
ElementToSequence(a) (*function*), 32
ElementToSequence(u) (*function*), 6
elt< M | a_1, ..., a_n > (*constructor*), 6
Eltseq(a) (*function*), 32
Eltseq(T) (*function*), 53
Eltseq(u) (*function*), 6
EmbeddingSpace(M) (*function*), 24
Endomorphism Ring of a Vector Space (*example*), 13
EndomorphismAlgebra(M) (*function*), 13
EqualizeDegrees(C, D) (*function*), 38
EqualizeDegrees(C, D, n) (*function*), 38
EqualizeDegrees(Q) (*function*), 38
Evaluating the Jacobi Identity Tensor (*example*), 72
ExactExtension(C) (*function*), 37
Exfoliating and Foliating a Tensor (*example*), 81
ExteriorCotensorSpace(V, n) (*function*), 102

F

f * g (*operation*), 40
f + g (*operation*), 40
Foliation(T, i) (*function*), 81
Frame(T) (*function*), 67, 107
Fully Nondegenerate Quotient Tensor (*example*), 117
Fully Nondegenerate Tensors (*example*), 70
FullyNondegenerateTensor(T) (*function*), 70

G

Generators(M) (*function*), 5
Generators(T) (*function*), 105
Generic(M) (*function*), 5
Generic(T) (*function*), 109

H

H . a (*operation*), 123
H1 * H2 (*operation*), 123
HasDefinedModuleMap(C,n) (*function*), 41
Heisenberg Algebras from Tensors (*example*), 89
HeisenbergAlgebra(T) (*function*), 89
HeisenbergGroup(T) (*function*), 94
HeisenbergGroupPC(T) (*function*), 94
HeisenbergLieAlgebra(T) (*function*), 92
HermiteForm(X) (*function*), 33
Hilbert Matrix as an Endomorphism (*example*), 16
Hom(M, N) (*function*), 11, 30
Hom(M, N, "left") (*function*), 12
Hom(M, N, "right") (*function*), 12
hom<M -> N | T> (*constructor*), 29
Homology(C) (*function*), 37
Homology(C, n) (*function*), 37
HomologyOfChainComplex(C) (*function*), 37

Homomorphisms Between Homomorphism Modules (*example*), 12
Homomorphisms Between Modules over Dedekind Domains (*example*), 30
Homotopism(M, C) (*function*), 121
Homotopism(T, S, M : parameters) (*function*), 121
Homotopism(T, S, M, C : parameters) (*function*), 121
HomotopismCategory(v : parameters) (*function*), 111
Homotopisms from Mixed Maps (*example*), 122

I

I * M (*operation*), 25
I * u (*operation*), 25, 32
I * X (*operation*), 34
Ideal(T, D, C) (*function*), 116
Ideal(T, S) (*function*), 115, 116
Ideals of the Tensor of an Algebra (*example*), 116
Image(a) (*function*), 17
Image(f) (*function*), 40
Image(H) (*function*), 125
Image(T) (*function*), 70
Induce(X, i) (*function*), 85
InducedMapOnHomology(f, n) (*function*), 43
InducedTensor(T, grid) (*function*), 79
Inducing Derivations of a Symplectic Form (*example*), 85
Infix Notation for Bilinear Maps (*example*), 76
InnerProduct(u, v) (*function*), 8
IsAlternating(T) (*function*), 71, 109
IsAntisymmetric(T) (*function*), 71, 109
IsBijective(a) (*function*), 18
IsChainMap(f) (*function*), 41
IsChainMap(L, C, D, n) (*function*), 41
IsCoercible(T, S) (*function*), 104
IsCoercible(TS, x) (*function*), 103
IsCoercible(U,x) (*function*), 77
IsCoercible(V,x) (*function*), 77
IsContravariant(C) (*function*), 112
IsContravariant(T) (*function*), 68, 108
IsCovariant(C) (*function*), 112
IsCovariant(T) (*function*), 68, 108
IsDegenerate(T) (*function*), 69
IsExact(C) (*function*), 38
IsExact(C, n) (*function*), 38
IsFree(M) (*function*), 25
IsFullyNondegenerate(T) (*function*), 70
IsHomotopism(T, s, H) (*function*), 121
IsHomotopism(T, s, M) (*function*), 121
IsHomotopism(T, s, M, C) (*function*), 121
IsIdeal(T, S) (*function*), 116
IsInjective(a) (*function*), 18
IsInjective(f) (*function*), 41
IsIsomorphism(f) (*function*), 41
IsLocalIdeal(T, S, I) (*function*), 114
IsNondegenerate(T) (*function*), 69
IsProperChainMap(f) (*function*), 41
IsShortExactSequence(C) (*function*), 38
IsShortExactSequence(f, g) (*function*), 41
IsSubmodule(M, N) (*function*), 30
IsSubtensor(T, S) (*function*), 113
IsSubtensorSpace(T, S) (*function*), 119
IsSurjective(a) (*function*), 18
IsSurjective(f) (*function*), 41
IsSymmetric(T) (*function*), 71, 109
IsZero(a) (*function*), 32
IsZero(f) (*function*), 41
IsZero(u) (*function*), 8
IsZeroComplex(C) (*function*), 38
IsZeroMap(C, n) (*function*), 38
IsZeroTerm(C, n) (*function*), 38

J

Jordan Product of a Matrix Algebra (*example*), 59

K

k * T (*operation*), 65
KCotensorSpace(K, S) (*function*), 100
KCotensorSpace(K, S, C) (*function*), 100
Kernel(a) (*function*), 17
Kernel(f) (*function*), 40
Kernel(H) (*function*), 125
KMatrixSpaceWithBasis(Q) (*function*), 13
KTensorSpace(K, S) (*function*), 97
KTensorSpace(K, S, C) (*function*), 97

L

LeftDomain(B) (*function*), 77
LeftExactExtension(C) (*function*), 37
LeftNucleus(B : parameters) (*function*), 127
LeftZeroExtension(C) (*function*), 38
LeftZeroExtension(C, n) (*function*), 38
Length(pm) (*function*), 33
Lie Algebras from Tensors (*example*), 92
Lie Bracket Tensor of D4(11) (*example*), 57
LinearCategory(v, s, t) (*function*), 111
Local Ideals of a Tensor (*example*), 114
LocalBasis(M, p) (*function*), 27
LocalIdeal(T, D, C, I) (*function*), 114
LocalIdeal(T, S, I) (*function*), 114
LocalQuotient(T, S, I : parameters) (*function*), 117
Long Exact Sequence in Homology (*example*), 43
LongExactSequenceOnHomology(f, g) (*function*), 43

M

M ! 0 (*operation*), 6
M ! Q (*operation*), 6, 16
M ! v (*operation*), 31

M * I (*operation*), 25
M + N (*operation*), 10
M . i (*operation*), 5, 23
M eq N (*operation*), 10, 25
M meet N (*operation*), 10
M ne N (*operation*), 10
M subset N (*operation*), 25
M1 + M2 (*operation*), 25
M1 meet M2 (*operation*), 27
Maps(H) (*function*), 124
Matrix(P) (*function*), 33
MatrixTensor(K, S) (*function*), 62
MidNucleus(B) (*function*), 127
Module Arithmetic and Predicates (*example*), 26
Module Functions on Tensor Spaces (*example*), 105
Module of 6-Tuples over the Integers (*example*), 4
Module of Homomorphisms of Vector Spaces (*example*), 11
Module Operations on Tensors (*example*), 65
Module(I) (*function*), 20
Module(O) (*function*), 19
Module(O, n) (*function*), 19
Module(S) (*function*), 20
Module(X) (*function*), 34
ModuleMap(f, n) (*function*), 40
Moduli(M) (*function*), 5
Morphism(M, N) (*function*), 17, 30

N

N eq M (*operation*), 10
N ne M (*operation*), 10
N notsubset M (*operation*), 10
N subset M (*operation*), 10
Ngens(M) (*function*), 23
Ngens(T) (*function*), 105
Nondegeneracy of a Tensor (*example*), 69
NondegenerateTensor(T) (*function*), 69
Norm(u) (*function*), 8
Normalise(u) (*function*), 7
Normalize(u) (*function*), 7
Nuclei Embed in the Derivation Algebra (*example*), 127
Nucleus(T, A) (*function*), 129
Nucleus(T, a, b) (*function*), 129
NullSpace(a) (*function*), 17
NumberOfGenerators(M) (*function*), 23
NumberOfGenerators(T) (*function*), 105

O

Operations in a Homomorphism Module (*example*), 18
Operations with Homotopisms (*example*), 123
Order(pm) (*function*), 33
OverDimension(M) (*function*), 5
OverDimension(u) (*function*), 5

P

p1 eq p2 (*operation*), 33
p-Groups from Non-Full Tensors (*example*), 96
p-Groups from Tensors (*example*), 94
Parent(T) (*function*), 67
Parent(u) (*function*), 5
Parent(X) (*function*), 78
Parent(x) (*function*), 78
pCentralTensor(G) (*function*), 61
pCentralTensor(G, p, s, t) (*function*), 61
pCentralTensor(G, s, t) (*function*), 61
Pfaffian(B) (*function*), 88
Pfaffians of Genus 2 Groups (*example*), 88
Polarisation(f) (*function*), 62
Polarization of a Binary Cubic Form (*example*), 62
Polarization(f) (*function*), 62
Precompose(T, f, a) (*function*), 124
Precompose(T, M, a) (*function*), 124
Preprune(C) (*function*), 37
Preprune(C,n) (*function*), 37
Printing Tensors to Strings (*example*), 47
Product Notation for Bilinear Maps (*example*), 77
Properties of Homotopisms (*example*), 125
Properties of Tensor Categories (*example*), 112
Properties of Tensor Spaces (*example*), 108
Prune(C) (*function*), 37
Prune(C,n) (*function*), 37
PseudoBasis(M) (*function*), 27
PseudoGenerators(M) (*function*), 27
PseudoMatrix(I, m) (*function*), 33
PseudoMatrix(M) (*function*), 33
PseudoMatrix(m) (*function*), 33

Q

quo< C | D > (*constructor*), 36
quo< C | L > (*constructor*), 36
quo< C | S > (*constructor*), 36
quo< T | X > (*constructor*), 119
quo<M | L> (*constructor*), 11
quo<M | m1, .., mn> (*constructor*), 22
quo<M | m> (*constructor*), 22
quo<M | S> (*constructor*), 22
QuoConstructor(T, X) (*function*), 119
Quotient(T, S : parameters) (*function*), 117
Quotients of Tensor Spaces (*example*), 120

R

Radical(T) (*function*), 86
Radical(T, s) (*function*), 86
Radicals of the Commutator Tensor (*example*), 86
Random Symmetric Tensors (*example*), 107
Random Tensors (*example*), 105
Random(M) (*function*), 6
Random(T) (*function*), 105

RandomAlternatingTensor(R, d, n, c) (*function*), 106
RandomAlternatingTensor(R, S) (*function*), 106
RandomAntisymmetricTensor(R, d, n, c) (*function*), 106
RandomAntisymmetricTensor(R, S) (*function*), 106
RandomCotensor(K, S) (*function*), 105
RandomSubcomplex(C, Q) (*function*), 36
RandomSymmetricTensor(R, d, n, c) (*function*), 107
RandomSymmetricTensor(R, S) (*function*), 107
RandomTensor(R, S) (*function*), 105
RandomTensor(R, S, C) (*function*), 105
Rank(a) (*function*), 18
Rank(M) (*function*), 9
Recovering a Field with the Centroid (*example*), 131
Recovering a Ring from a p-Group (*example*), 61
Recovering a Tensor from Its Tensor Space (*example*), 84
Reduce(H) (*function*), 14
Reduced Form for Maps Between Hom Modules (*example*), 15
Reduced Form for Maps Between Vector Spaces (*example*), 14
RepeatPartition(C) (*function*), 112
Restricting the Derivation Algebra (*example*), 129
RightDomain(B) (*function*), 77
RightExactExtension(C) (*function*), 37
RightNucleus(B) (*function*), 127
RightZeroExtension(C) (*function*), 38
RightZeroExtension(C, n) (*function*), 38
RMatrixSpace(R, m, n) (*function*), 4, 11
RMatrixSpaceWithBasis(Q) (*function*), 4, 13
RModule(R, n) (*function*), 4
RModuleWithBasis(Q) (*function*), 4
Rotate(~u, k) (*procedure*), 7
Rotate(u, k) (*function*), 7
RSpace(R, n) (*function*), 4
RSpace(R, n, F) (*function*), 4
RSpaceWithBasis(a) (*function*), 4
RSpaceWithBasis(Q) (*function*), 4
RTensorSpace(R, S) (*function*), 97
RTensorSpace(R, S, C) (*function*), 97

S

S + T (*operation*), 65
S : Magma (*literal*), 47
S - T (*operation*), 65
S eq T (*operation*), 74, 104
S subset T (*operation*), 104
SelfAdjointAlgebra(t, a, b) (*function*), 130
Shift(C, n) (*function*), 37
ShiftToDegreeZero(C) (*function*), 37
Shuffle(H, g) (*function*), 126
Shuffle(T, g) (*function*), 63
Shuffling a Random 5-Tensor (*example*), 64
Signatured Tensor Space (*example*), 99
Simplify(M) (*function*), 24
Slice(T, grid) (*function*), 79
SliceAsMatrices(T, grid, a, b) (*function*), 80
Slices as Matrices (*example*), 80
Slicing a Tensor (*example*), 79
Special Tensor Category Constructors (*example*), 111
Splice(C, D) (*function*), 37
Splice(C, D, f) (*function*), 37
Standard Cotensor Subspaces (*example*), 102
Standard Tensor Subspaces (*example*), 101
SteinitzClass(M) (*function*), 28
SteinitzForm(M) (*function*), 28
Stored Structure Constants (*example*), 54
Structure Constants from a Black-Box Tensor (*example*), 53
StructureConstants(T) (*function*), 53
sub< C | L > (*constructor*), 36
sub< C | Q > (*constructor*), 36
sub< T | L > (*constructor*), 119
sub<M | L> (*constructor*), 9
sub<M | m1, .., mn> (*constructor*), 22
sub<M | m> (*constructor*), 22
SubConstructor(T, L) (*function*), 119
Submodule over a Rational Function Field (*example*), 9
Submodules and Quotient Modules (*example*), 22
Subspace of Alternating Tensors (*example*), 109
Subspace of Symmetric Forms (*example*), 119
Subtensor(T, D, C) (*function*), 113
Subtensor(T, S) (*function*), 113
Subtensors of Octonion Multiplication (*example*), 113
Support(u) (*function*), 8
Symmetric Form from a Ternary Cubic (*example*), 71
SymmetricCotensorSpace(V, n) (*function*), 102
SymmetricSpace(T) (*function*), 101
SymmetricTensor(T) (*function*), 62
Symmetrizing a Matrix Algebra Tensor (*example*), 63
Symplectic Bilinear Form (*example*), 55
SystemOfForms(T) (*function*), 56

T

T ! n (*operation*), 103
T ! S (*operation*), 104
T * f (*operation*), 74
t * s (*operation*), 74
T * y (*operation*), 75
T . i (*operation*), 105
T / S (*operation*), 117, 119
T : Magma (*literal*), 47

T @ H (*operation*), 124
T in TS (*operation*), 103
Tensor from Structure Constants (*example*), 52
Tensor of the Witt Algebra (*example*), 58
Tensor Space at a Coordinate (*example*), 82
Tensor(A) (*function*), 57
Tensor(D, C, F) (*function*), 50
Tensor(D, C, F, Cat) (*function*), 50
Tensor(D, S) (*function*), 52
Tensor(D, S, Cat) (*function*), 52
Tensor(M, s, t) (*function*), 55
Tensor(M, s, t, C) (*function*), 55
Tensor(Q) (*function*), 58
Tensor(R, D, S) (*function*), 52
Tensor(R, D, S, Cat) (*function*), 52
Tensor(S, F) (*function*), 48
Tensor(S, F, Cat) (*function*), 48
TensorCategory(A, P) (*function*), 110
TensorCategory(H) (*function*), 124
TensorCategory(T) (*function*), 68, 108
TensorOverCentroid(T) (*function*), 130
TensorProduct(t, s) (*function*), 65
TensorSpace(K, d, p, q) (*function*), 99
TensorSpace(S) (*function*), 98
TensorSpace(S, C) (*function*), 98
TensorSpace(V, p, q) (*function*), 99
Term(C, n) (*function*), 36
Terms(C) (*function*), 36
Transpose(P) (*function*), 33
Transposing a Form by Shuffling (*example*), 64
Trilinear Tensor from Matrices (*example*), 56
TS ! S (*operation*), 103
TS ! T (*operation*), 103

U

U ! x (*operation*), 77
u * a (*operation*), 17
u * c (*operation*), 32
u * I (*operation*), 25, 32
u * x (*operation*), 7
u + v (*operation*), 7
u / c (*operation*), 32
u / x (*operation*), 7
u - v (*operation*), 7
u in M (*operation*), 10
u notin M (*operation*), 10
U1 eq U2 (*operation*), 78
u1 eq u2 (*operation*), 78
Universal Cotensor Spaces (*example*), 100
Universal Tensor Space from Modules (*example*), 98
Universal Tensor Space over a Field (*example*), 98
UniversalCotensorSpace(T) (*function*), 109
UniversalTensorSpace(T) (*function*), 109
u[i] (*indexed*), 7
u[i] := x (*literal*), 7

V

V ! x (*operation*), 77
V1 eq V2 (*operation*), 78
v1 eq v2 (*operation*), 78
Valence(C) (*function*), 112
Valence(H) (*function*), 125
Valence(T) (*function*), 67, 107
Verbose Printing for Tensor Spaces (*example*), 47
VerticalJoin(X, Y) (*function*), 34

W

Weight(u) (*function*), 8

X

X * I (*operation*), 34
x * T (*operation*), 75
x * u (*operation*), 7
x * y (*operation*), 77
x + y (*operation*), 32
x @ T (*operation*), 72
x - y (*operation*), 32
x eq y (*operation*), 32
x in M (*operation*), 25
X meet Y (*operation*), 34

Z

Zero(M) (*function*), 6
ZeroChainMap(C, D) (*function*), 40
ZeroComplex(A, m, n) (*function*), 35
ZeroExtension(C) (*function*), 38