

eMAGma Package

Joshua Maglione

Colorado State University
`maglione@math.colostate.edu`

James B. Wilson

Colorado State University
`james.wilson@colostate.edu`

Version 1.1.0
June 13, 2017

Copyright 2016, 2017 Joshua Maglione, James B. Wilson

Contents

Chapter 1. Introduction	1
1.1. Overview	1
1.2. Verbose Printing and Version	1
Chapter 2. Tensors	3
2.1. Creating tensors	3
2.1.1. Black-box tensors	3
2.1.2. Tensors with structure constant sequences	7
2.1.3. Bilinear tensors	10
2.1.4. Tensors from algebraic objects	12
2.1.5. New tensors from old	18
2.2. Operations with Tensors	21
2.2.1. Elementary operations	21
2.2.2. General properties	23
2.2.3. As multilinear maps	29
2.2.4. Operations with Bilinear maps	32
2.2.5. Manipulating tensor data	36
2.3. Invariants of tensors	38
2.3.1. Standard invariants	38
2.3.2. Invariants for bilinear tensors	39
2.3.3. Invariants of general multilinear maps	39
2.4. Exporting tensors	42
2.5. Invariants of nonassociative algebras	43
Chapter 3. Tensor spaces	45
3.1. Constructions of tensor and cotensor spaces	45
3.1.1. Universal tensor spaces	45
3.1.2. Universal cotensor spaces	47
3.1.3. Some standard constructions	48
3.1.4. Subspaces as closures	49
3.2. Operations on tensor spaces	51
3.2.1. Membership and comparison with tensor spaces	51
3.2.2. Tensor spaces as modules	52
3.2.3. Properties of tensor spaces	54
Chapter 4. Tensor categories	57
4.1. Creating tensor categories	58
4.2. Operations on tensor categories	59
4.3. Categorical operations	60
4.3.1. Categorical operations on tensors	60
4.3.2. Categorical operations on tensor spaces	62
4.4. Homotopisms	63

4.4.1. Constructions of Homotopisms	63
4.4.2. Basic Operations with Homotopisms	64
Chapter 5. Exceptional tensors	69
5.1. Generics for nonassociative algebras	69
5.1.1. Nonassociative algebras with involutions	69
5.1.2. Operations on power associative algebras	69
5.2. Compositions algebras	70
5.3. Jordan algebras	71
Chapter 6. Some examples	73
6.1. Distinguishing groups	73
6.2. Simplifying automorphism group computations	75
Appendix A. Cyclic algebras and their modules	79
Bibliography	81
Intrinsics	83

CHAPTER 1

Introduction

This chapter discusses Magma operations with multilinear algebra. Wherever possible we follow the conventions in use in physics [W1, Chapter V], differential geometry [L2, Chapter 10], and algebra [L1]. Necessary categorical formalism is drawn largely from [W2]. 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) Exceptional tensors of octonions and Jordan algebras

1.1. Overview

Throughout \mapsto denotes multilinear maps $[\cdot] : U_v \times \cdots \times U_1 \mapsto U_0$ of K -modules $\{U_v, \dots, U_0\}$ known as the *frame* of a multilinear map or tensor. The module U_0 is reserved for the codomain which inturn makes reverse indexing the simplest notation. Every tensor in Magma is treated as an element of a tensor space. By default a universal tensor space:

$$\text{hom}_K(U_v, \dots, \text{hom}_K(U_1, U_0) \cdots).$$

In Magma, the tensor space is what determines the associated multilinear function of a given tensor T . Evaluation of T mimics a map $U_v \times \cdots \times U_1 \rightarrow U_0$, for instance, $\langle \mathbf{u}_v, \dots, \mathbf{u}_1 \rangle @T$;. Special attention is given to bilinear maps $* : U_2 \times U_1 \mapsto U_0$ including the ability to use infix notation $\mathbf{u}_2 * \mathbf{u}_1$. Tensor spaces have type **TenSpc** and behave like modules in that they have subspaces and quotient spaces. Tensors have type **TenSpcElt** and behave similar to Magma matrices.

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

Tensor categories, type **TenCat**, tell Magma how to interpret the contents of a tensor space. For example, one tensor category treats a $(d \times d)$ -matrix F over a field K as a linear map $K^d \rightarrow K^d$, another assigns the same matrix to a bilinear form $K^d \times K^d \mapsto K$. Functors are provided to change tensor categories and to define standard categories.

1.2. Verbose Printing and Version

We have included intrinsics to allow for verbose printing. Currently, there is only one level of printing, so either it is on or off.

`SetVerbose(MonStgElt, RngIntElt) : ->`

SetVerbose is a built in Magma function, but with this package, this intrinsic accepts the string "eMAGma" and an integer in $\{0, 1\}$.

We have included an intrinsic to check which version of eMAGma you have attached in Magma.

Example 1.1. VerbosePrinting

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

```
> SetVerbose("eMAGma", 1);  
>  
> t := RandomTensor(GF(2), [32, 32, 32]);  
> D := DerivationAlgebra(t);  
Setting up linear system: 3072 by 32768  
Solving up linear system: 3072 by 32768
```

`eMAGmaVersion()` : -> MonStgElt

Returns the version number for the eMAGma package attached in Magma.

Example 1.2. Version

We verify that we have attached the current version of the eMAGma package. Even though `eMAGmaVersion` has no arguments, `()` is still required.

```
> eMAGmaVersion();  
1.1.0
```

CHAPTER 2

Tensors

Tensors are required to have the following information.

- A commutative ring K of coefficients.
- A valence v indicating the number of variables to include in its associated multilinear map.
- A list $[U_v, \dots, U_0]$ of K -modules called the *frame*.
- A function $U_v \times \dots \times U_1 \rightarrow U_0$ 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:

$$\text{hom}_K(U_v, \dots, \text{hom}_K(U_1, U_0) \cdots) \cong \text{hom}_K(U_v \otimes_K \cdots \otimes_K U_1, U_0).$$

The left hand module is used primarily as 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. Attributes such as coefficients, valence, and frame apply to the tensor space as well.

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 as well as imposing symmetry conditions. If no tensor category is prescribed then a default tensor category is used based on the method of creation.

Contents

2.1. Creating tensors	3
2.1.1. Black-box tensors	3
2.1.2. Tensors with structure constant sequences	7
2.1.3. Bilinear tensors	10
2.1.4. Tensors from algebraic objects	12
2.1.5. New tensors from old	18
2.2. Operations with Tensors	21
2.2.1. Elementary operations	21
2.2.2. General properties	23
2.2.3. As multilinear maps	29
2.2.4. Operations with Bilinear maps	32
2.2.5. Manipulating tensor data	36
2.3. Invariants of tensors	38
2.3.1. Standard invariants	38
2.3.2. Invariants for bilinear tensors	39
2.3.3. Invariants of general multilinear maps	39
2.4. Exporting tensors	42
2.5. Invariants of nonassociative algebras	43

2.1. Creating tensors

2.1.1. 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. 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, the Albert’s homotopism category is used.

Example 2.1. BBTensorsFrame

We demonstrate the black-box constructions by first constructing the dot product $\cdot : \mathbb{Q}^4 \times \mathbb{Q}^4 \rightarrow \mathbb{Q}$. 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 $U_v \times \cdots \times U_1$. Note that `x[i]` is the i th entry in the tuple and not the i th 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 21-adjoint category. While the returned tensor prints out the same as above, it does indeed live in a universe. The details of tensor categories are discussed in Chapter 4.

```

> 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 2.2. BBCrossProduct

We will construct the cross product $\times : \mathbb{R}^3 \times \mathbb{R}^3 \rightarrow \mathbb{R}^3$ 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)
function>   return V![x[1][2]*x[2][3] - x[1][3]*x[2][2], \
function|return>   x[1][3]*x[2][1] - x[1][1]*x[2][3], \
function|return>   x[1][1]*x[2][2] - x[1][2]*x[2][1] ];
function> 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. 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, the Albert’s homotopism category is used.

Example 2.3. BBTripleProduct

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 $\langle \rangle : \mathbb{M}_{2 \times 3}(K) \times \mathbb{M}_{2 \times 3}(K) \times \mathbb{M}_{2 \times 3}(K) \rightarrow \mathbb{M}_{2 \times 3}(K)$, 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 from 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 $\{X_1, \dots, X_5\} \subset \mathbb{M}_{2 \times 3}(K)$, 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 $\mathbb{M}_{2 \times 3}(K)$.

```
> 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 see that its image is 0. We will create a 6-tensor $\{ \} : \prod_{k=1}^5 \mathbb{M}_{2 \times 3}(K) \mapsto \mathbb{M}_{2 \times 3}(K)$ where

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

Therefore, if $\text{im}(\langle \langle \rangle \rangle) = 0$, 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)
>
> I := Image(LT);
> I;
Vector space of degree 6, dimension 0 over GF(541)
Generators:

>
> Dimension(I);
0

```

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

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

$$[e_{vj_v}, \dots, e_{1j_1}] = \sum_{k=1}^{d_0} T_{j_v \dots j_0} e_{0k}.$$

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

$$f(j_v, \dots, j_0) = 1 + \sum_{s=0}^v (j_s - 1) \prod_{t=0}^{s-1} d_t.$$

So $S[f(j_v, \dots, j_0)] = T_{j_v \dots j_0}$ specifies the structure constants as a sequence.

Notes.

- Magma does not presently support the notion of a sparse sequence of structure constants. A user can provide this functionality by specifying a tensor with a user program rather than structure constants.
- Some routines in Magma require structure constant sequences. If they are not provided, Magma may compute and store a structure constant representation inside the tensor.
- We do not separate structure constant indices that are contravariant. Instead contravariant variables are signaled by tensor categories. So Ricci styled tensors $T_{a_p \dots a_1}^{b_q \dots b_1}$ should be input as $T_{a_p+q \dots a_1+q b_q \dots b_1}$ and the tensor category changed to mark $\{q..1\}$ as contravariant. Intrinsic are provided to facilitate this approach. See Chapter 4 for more details on tensor categories.

```

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

```

Given dimensions $D = [d_v, \dots, d_0]$, returns the tensor in $R^{d_v \cdots d_0}$ identified by structure constant sequence S . If R is not provided then the parent ring of the first element of S is used. R must be a commutative unital ring. The default tensor category Cat is the homotopism category.

Example 2.4. SCTensors

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: $\text{GF}(64)$.

```

> 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 .

Example 2.5. SCFromBBTensors

We will construct the natural Lie module action for \mathfrak{sl}_2 , but we will construct it as a *left* module. To do this, we construct a function that takes elements from $\mathfrak{sl}_2 \times V$ 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 \mathfrak{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 2.6. SCStored

The structure constants are convenient data structure for nearly all the algorithms in eMAGma. 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 $M_{20}(F_3)$, namely $*$: $\mathfrak{sl}_{20}(F_3) \times M_4(F_3) \rightarrow M_{20}(F_3)$.

```

> 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

```

2.1.3. Bilinear tensors. A special case of structure constants for bilinear maps $U_2 \times U_1 \rightarrow U_0$ is to format the data as lists of matrices $[M_1, \dots, M_a]$. This can be considered as a left (resp. right) representation $U_2 \rightarrow \text{hom}_K(U_1, U_0)$, (resp. $U_1 \rightarrow \text{hom}_K(U_2, U_0)$). Or it can be treated as *systems of bilinear forms* $[M_1, \dots, M_a]$ where the matrices are the Gram matrices of bilinear forms $\phi_i : U_2 \times U_1 \rightarrow K$. Here the associated bilinear map $U_2 \times U_1 \rightarrow U_0$ is specified by

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

```

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

```

Returns the bilinear tensor given by the list of matrices. The interpretation of the matrices as structure constants is specified by the coordinates s and t which must be positions in $\{2, 1, 0\}$. Optionally a tensor category C can be assigned.

Example 2.7. SymplecticForm

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  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])) >;
> 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 $[K^{d_v}, \dots, K^{d_0}]$, returns a list $[M_1, \dots, M_d]$, $d = (d_v \cdots d_0)/d_s d_t$, of $(d_s \times d_t)$ -matrices in K representing the tensor as an element of $\text{hom}_K(K^{d_s} \otimes_K K^{d_t}, K^d)$. For `SystemOfForms`, T must have valence 3 and the implied values are $s = 2$ and $t = 1$.

Example 2.8. TrilinearAsMats

We construct the associator of $\mathfrak{sl}_2(\mathbb{Q})$ where $\langle \rangle : \prod_{k=1}^3 \mathfrak{sl}_2 \rightarrow \mathfrak{sl}_2$ 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 asnd 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, -2,
0, 0, 0, -2, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, -1, 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 $\circ_{31} : V_3 \times V_1 \rightarrow \text{hom}_K(V_2, V_0)$ 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]
]

```

2.1.4. Tensors from algebraic objects. A natural and important source of tensors come from algebraic object with a distributive property. One main source is from algebras, where $*$: $A \times A \rightarrow A$ 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 Chapter 4 for more details on tensor categories.

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

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

Returns the bilinear tensor given by the product in A .

Example 2.9. D4LieAlgebra

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, $\Omega = \text{End}(U_2)$ instead of $\Omega = \text{End}(U_2) \times \text{End}(U_1) \times \text{End}(U_0)$.

```
> D := DerivationAlgebra(T);
> 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);
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, $\text{Rad}(D) = K^2$.

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

Example 2.10. WittAlgebra

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 $\mathbb{F}_5[x]/(x^5)$. 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);
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

Returns 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 2.11. CommutatorFromAlgebra

We will construct the commutator tensor from $M_4(\mathbb{Q})$.

```
> A := MatrixAlgebra(Rationals(), 4);
> C := CommutatorTensor(A);
> C;
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(C); // [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 $\mathbb{M}_4(\mathbb{Q})$. 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> @ C);
8
```

Example 2.12. MatrixJordanAlgebra

This time, we will obtain a Jordan product from $\mathbb{M}_4(\mathbb{Q})$. That is, we will construct the bilinear map $*$: $\mathbb{M}_4(\mathbb{Q}) \times \mathbb{M}_4(\mathbb{Q}) \rightarrow \mathbb{M}_4(\mathbb{Q})$ where $A * B = \frac{1}{2}(AB + BA)$.

```
> A := MatrixAlgebra(Rationals(), 4);
> J := AnticommutatorTensor(A);
> J;
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(J)[1];
[2 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 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 1 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 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*J*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 J by $1/2$ to get what we want. Of course this won't affect the proceeding tests though.

```
> T := (1/2)*J;
> A.1*T*A.1;
(1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
```

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

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

AssociatorTensor(A) : Alg -> TenSpcElt, Map

Returns the trilinear associator map $[a, b, c] = (ab)c - a(bc)$ of the algebra A .

Example 2.13. AssociatorFromAlgebra

Do three random octonions associate? 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 0!0;
false
```

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

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

pCentralTensor(G, p, s, t) : Grp, RngIntElt, 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 2.14. TensorPGroup

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 $G = \text{SL}(3, 125)$; however, we will lose the fact that there is a field \mathbb{F}_{125} . The tensor we will get back is $[\cdot] : K^6 \times K^6 \rightarrow K^3$, where $K = \mathbb{F}_5$.

```
> P := ClassicalSylow(SL(3,125),5);
> Q := PCGroup(P); // Loose track 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 \mathbb{F}_{125} , we know that the commutator is really just the alternating form $\cdot : \mathbb{F}_{125}^2 \times \mathbb{F}_{125}^2 \rightarrow \mathbb{F}_{125}$. This information can be extracted from the centroid of T above, and we can rewrite T over the field \mathbb{F}_{125} .

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

```
Polarisation(f) : MPolElt -> TenSpcElt, MPolElt
Polarisation(f) : RngUPolElt -> TenSpcElt, MPolElt
Polarization(f) : MPolElt -> TenSpcElt, MPolElt
Polarization(f) : RngUPolElt -> TenSpcElt, MPolElt
```

Returns the polarization of the homogeneous multivariate polynomial (or univariate polynomial) f as a tensor and as a multivariate polynomial. Polarization does *not* normalize by $1/d!$, where d is the degree of f .

Example 2.15. TensorPolarization

We polarize the polynomial $f(x, y) = x^2y$. Because f is homogeneous of degree 3 with 2 variables, we expect that the polarization will have 6 variables and that the corresponding multilinear form will be $K^2 \times K^2 \times K^2 \rightarrow K$. The polarization of f is given by $P(x_1, x_2, y_1, y_2, z_1, z_2) = 2(x_1y_1z_2 + x_1y_2z_1 + x_2y_1z_1)$.

```
> 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)
```

2.1.5. New tensors from old. We can construct new tensors from old.

AlternatingTensor(T) : TenSpcElt -> TenSpcElt

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

Example 2.16. AlternatingTensor

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 2.17. MakeSymmetric

We will make the tensor coming from the product in $M_3(\mathbb{Q})$ 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]
[0 0 0 0 0 0 0 0 0]
```

We see that 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]
[0 0 0 0 0 0 0 0 0]
```

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

For a tensor T in $\text{hom}(U_1, \dots, \text{hom}(U_1, U_0) \cdots)$, generates a representation of T in

$$\text{hom}(U_{v^g}, \dots, \text{hom}(U_{1^g}, U_{0^g}) \dots).$$

In order to be defined, $g \in \text{Sym}(\{0..v\})$. 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, $g \in \text{Sym}(\{1..v\})$. Sequences $[a_1, \dots, a_{i+1}]$

will be interpreted as

$$\begin{array}{cccc} 0 & 1 & \cdots & 1 \\ \downarrow & \downarrow & & \downarrow \\ a_1 & a_2 & \cdots & a_{1+1}. \end{array}$$

Example 2.18. ShuffleToTranspose

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

```
> 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 2.19. Shuffling

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);
```



```

> 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)

```

2.2. Operations with Tensors

We take two perspectives for operations with tensors. First, tensors determine multilinear maps and so behave as functions. Second, tensors are elements of a tensor space and so behave as elements in a module.

2.2.1. 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

```

Returns the sum, difference, scalar multiple, or additive inverse of the given tensor(s) as module elements of the tensor space. The corresponding multilinear maps are the sum, difference, scalar multiple, or additive inverse of the multilinear maps.

Example 2.20. ModuleOperations

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 multilinear maps $U_1 \times \cdots \times U_1 \rightarrow U_0$, creates the associated multilinear form $U_1 \times \cdots \times U_1 \times U_0^* \rightarrow K$. The valence is increased by 1.

Compress(T) : TenSpcElt -> TenSpcElt

Compress(~T) : TenSpcElt

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

Example 2.21. CompressAssocForm

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 dualized) 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)
> AF := AssociatedForm(T);
> AF;
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)
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> @ AF;
(4)
> <L.2, L.11, L> @ AF;
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 S by (0,2,1).

```

> S := Shuffle(AF, [3,1,2,0]);
> S;
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)
> C := Compress(S);
> C;
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)
> C eq Shuffle(T, [2, 0, 1]);
true

```

2.2.2. General properties. We provide basic intrinsics to get 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 T . The default space is the universal tensor space.

`Domain(T) : TenSpcElt -> List`

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

`Codomain(T) : TenSpcElt -> Any`

Returns the codomain of the tensor.

`Valence(T) : TenSpcElt -> RngIntElt`

Returns the valence of the tensor.

`Frame(T) : TenSpcElt -> List`

Returns the modules in the frame of T ; this is the concatenation of the domain modules and the codomain.

`BaseRing(T) : TenSpcElt -> Rng`
`BaseField(T) : TenSpcElt -> Fld`

Returns the base ring or field of the tensor.

Example 2.22. BasicProps

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

```
> K := Rational();
> 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 T .

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

ChangeTensorCategory(~T, C) : TenSpcElt, TenCat

Returns the tensor with the given category.

IsCovariant(T) : TenSpcElt -> BoolElt

IsContravariant(T) : TenSpcElt -> BoolElt

Decides if the underlying category of T is covariant or contravariant.

Example 2.23. TensorCatProps

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

```
> 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 multilinear map associated to T along with a homotopism from the given tensor to the returned nondegenerate tensor.

IsNondegenerate(T) : TenSpcElt -> BoolElt

Decides whether T is a nondegenerate multilinear map.

Example 2.24. Nondegeneracy

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)
function>   M := Matrix(3, 3, Eltseq(x[1])[2..10]);
function>   v := VectorSpace(K, 3)!(Eltseq(x[2])[1,3,5]);
function>   return Eltseq(v*M) cat [0,0];
function> 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_2^\perp and U_1^\perp are nontrivial. We will construct the associated nondegenerate tensor S , which is given by $U_2/U_2^\perp \times U_1/U_1^\perp \rightarrow U_0$.

```

> 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 along with a map to the vector space.

FullyNondegenerateTensor(T) : TenSpcElt -> TenSpcElt, Hmtp

Returns the fully nondegenerate multilinear map associated to T along with a cohomotopism from the given tensor to the returned tensor.

IsFullyNondegenerate(T) : TenSpcElt -> BoolElt

Decides whether T is a fully nondegenerate multilinear map.

Example 2.25. FullyNondegenerate

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)
function>   M := Matrix(3, 3, Eltseq(x[1])[2..10]);
function>   v := VectorSpace(K, 3)!(Eltseq(x[2])[1,3,5]);

```

```
function> return Eltseq(v*M) cat [0,0];
function> 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 from 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)
( 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: $U_2/U_2^\perp \times U_1/U_1^\perp \rightarrow U_2 * U_1$. 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`

Decides whether T is an alternating tensor.

`IsAntisymmetric(T) : TenSpcElt -> BoolElt`

Decides whether T is an antisymmetric tensor.

`IsSymmetric(T) : TenSpcElt -> BoolElt`

Decides whether T is a symmetric tensor.

Example 2.26. SymmetricPolar

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*$.9 +
    6*$.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

```

2.2.3. As multilinear maps. Regarding tensors as multilinear maps, we allow for composition and evaluation.

`x @ T : Tup, TenSpcElt -> Any`

Evaluates the tensor T at $x \in U_1 \times \cdots \times U_1$. 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 from algebraic objects, and in these cases, the entries can be contained in the original algebraic object as well.

Example 2.27. MultiMapEval

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 $A = \mathbb{M}_3(\mathbb{F}_3)$, the returned tensor is over vector spaces \mathbb{F}_3^9 . However, the tensor T can still evaluate elements from $\mathbb{M}_3(\mathbb{F}_3)$ as well as \mathbb{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 $\mathbb{M}_3(\mathbb{F}_3)$.

```

> 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 $\mathbb{M}_3(\mathbb{F}_3)$ and other entries are contained in \mathbb{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),

      [ 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

Returns the tensor which is the composition of T with the given map f . If a tensor S is used instead of a Map, S must have valence ≤ 1 .

S eq T : TenSpcElt, TenSpcElt -> BoolElt

Decides if the tensors are the same. Two tensors are equivalent if, and only if, they have the same tensor category, base ring, frame, and structure constants.

Example 2.28. TensorComp

We start with the same tensor as the previous example: the tensor given by the Jacobi identity on the algebra $A = \mathbb{M}_3(\mathbb{F}_3)$.

```

> 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 $\phi : A \rightarrow V$ 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 .

```

> F := Tensor([A, A], func< x | x[1]@f >);
> F;
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 * F eq S;
true

```

2.2.4. Operations with Bilinear maps. Tensors of valence 3, also known as bilinear tensors, or as bilinear maps, are commonly described as distributive products. For instance as the product of an algebra, the product of a ring on a module, or an inner product. We support these interpretations in two ways: by permitting an infix $x * B * y$ notation for a 3-tensor B , and a product $x * y$ notation for the evaluation of bilinear tensors. For the latter, We do this by creating special types $\text{BmpU}[\text{Elt}]$, $\text{BmpV}[\text{Elt}]$, and $\text{BmpW}[\text{Elt}]$ for the frame of a bilinear tensor. For bilinear maps, we refer to the modules in the frame as $U \times V \rightrightarrows W$.

```

x * B : Any, TenSpcElt -> Any
B * y : Any, TenSpcElt -> Any

```

Given a bilinear tensor B framed by $[U, V, W]$, $x * B$ returns the action on the right as a linear map $L : V \rightarrow W$ 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 $V \rightrightarrows W$. For the left action use $B * y$ instead. If B 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 * S : Any, TenSpc -> Any
S * y : Any, TenSpc -> Any

```

Given a subspace of bilinear tensors, return the subspace generated by all $x * s$, for $s \in S$. This is either a tensor space or a vector space.

Example 2.29. BimapInfix

We demonstrate the infix notation by constructing the tensor in $A = \mathbb{M}_2(\mathbb{Q})$ 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. For $M = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$, we will use the infix notation to construct the 2-tensor $* : A \rightarrow A$, where $X \mapsto MX$.

```
> 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. 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.

```
> S := VectorSpace(Rationals(), 4)*T;
> S;
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
> S*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 $\langle x, y \rangle @ B$.

```
LeftDomain(B) : TenSpcElt -> BmpU
```

Returns the left domain, U_2 , of B framed by $[U_2, U_1, U_0]$, setup for use with infix notation.

```
RightDomain(B) : TenSpcElt -> BmpV
```

Returns the right domain, U_1 , of B framed by $[U_2, U_1, U_0]$. setup for use with infix notation.

```
IsCoercible(S,x) : BmpU, Any -> BoolElt, BmpUElt
IsCoercible(S,x) : BmpV, Any -> BoolElt, BmpVElt
S!x : BmpU, Any -> BmpUElt
S!x : BmpV, Any -> BmpVElt
```

Decides if x can be coerced into S , and if it can, it returns the coerced element.

Example 2.30. BimapProduct

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 $[\cdot] : U \times V \rightarrow W$ is the tensor given by commutation where $U = V = G/\eta_2$ and $W = \eta_2/\eta_3$, where η_i denotes the i th 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`

Returns the parent space of the bilinear map element.

`Parent(X) : BmpU -> TenSpcElt`

`Parent(X) : BmpV -> TenSpcElt`

Returns the original bilinear map where these spaces came from.

`u1 eq u2 : BmpUElt, BmpUElt -> BoolElt`

`v1 eq v2 : BmpUElt, BmpUElt -> BoolElt`

`U1 eq U2 : BmpU, BmpU -> BoolElt`

`V1 eq V2 : BmpV, BmpV -> BoolElt`

Decides if the elements or spaces are equal.

Example 2.31. BimapProduct2

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

```

2.2.5. Manipulating tensor data. The data from a tensor is accessible in multiple ways. For tensors given by structure constants this can be described as the multidimensional analog of choosing a row or column of a matrix. Other operations are generalization of the transpose of a matrix. We do these operations with some care towards efficiency, e.g. it may not physically move the values in a structure constant sequence but instead permute the lookup 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 framed by free modules $[U_v, \dots, U_0]$ with $d_i = \dim U_i$, a grid is a sequence $[G_v, \dots, G_0]$ of subsets $G_i \subseteq \{1..d_i\}$. The slice is the list of entries in the structure constants of the tensor indexed by $G_v \times \dots \times G_0$. `Slice` returns the structure constants whereas `InducedTensor` produces a tensor with these structure constants.

Example 2.32. Ten_Slice

We demonstrate slicing. If every $G_i = \{1..d_i\}$, then the result is the same as `Eltseq`.

```

> U := VectorSpace(Rationals(),4);
> V := VectorSpace(Rationals(),3);
> W := VectorSpace(Rationals(),2);
> TS := TensorSpace([U,V,W]);
> T := TS![ i : i in [1..24] ];
> Slice(T,[{1..4},{1..3},{1..2}]); // structure constants
[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
18, 19, 20, 21, 22, 23, 24 ]

```

We obtain the slice data as structure constants of a tensor.

```

> Slice(T,[{1..4},{2},{1}]);
[ 3, 9, 15, 21 ]
>
> W1 := VectorSpace(Rationals(),1);
> pi := hom< W -> W1 | <W.1,W1.1>, <W.2,W1!0> >; // project
> Eltseq( (T*V.2)*pi );
[ 3, 9, 15, 21 ]

```

However, in this example, the tensors are not the same. When we compress the induced tensor, then they become equal.

```

> T_ind := InducedTensor(T,[{1..4},{2},{1}]);
> T_ind;
Tensor of valence 2, 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 1 over Rational Field
> S := (T*V.2)*pi;
> S;
Tensor of valence 1, U1 -> U0
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 1 over Rational Field
>
> Compress(T_ind) eq S;
true

```

SliceAsMatrices(T , grid , i , j) : TenSpcElt , $\text{SeqEnum}[\text{SetEnum}]$, RngIntElt , $\text{RngIntElt} \rightarrow [\text{Mtrx}]$

Returns a sequence of matrices equivalent to composing **Slice** and **AsMatrices**.

Foliation(T , i) : TenSpcElt , $\text{RngIntElt} \rightarrow \text{Mtrx}$

For a tensor T contained in $\text{hom}(U_v \otimes \cdots \otimes U_1, U_0)$, return the matrix representing the linear map $U_i \rightarrow \text{hom}(\bigotimes_{j \neq i} U_j, U_0)$ using the bases of each U_j . If $i = 0$, then the returned matrix is given by the representation $U_0^* \rightarrow \text{hom}(\bigotimes U_i, K)$.

AsTensorSpace(T , i) : TenSpcElt , $\text{RngIntElt} \rightarrow \text{TenSpc}$, Mtrx

Returns the associated tensor space of T at $i > 0$ along with a matrix given by the foliation of T at i . The returned tensor space is framed by $U_v \times \cdots \times U_{i+1} \times U_{i-1} \times \cdots \times U_1 \rightarrow U_0$ and is generated by the tensors T_u for each u in the basis of U_i .

AsCotensorSpace(T) : $\text{TenSpcElt} \rightarrow \text{TenSpc}$, Mtrx

Returns the associated cotensor space of T along with a matrix given by the foliation of T at 0. The returned cotensor space is framed by $U_v \times \cdots \times U_1 \rightarrow K$ and is generated by the tensors Tf for each f in the basis of U_0^* .

AsTensor(T) : $\text{TenSpc} \rightarrow \text{TenSpcElt}$

Returns a tensor corresponding to the given tensor space. If the given tensor space is contravariant, then the returned tensor has the frame $U_v \times \cdots \times U_1 \rightarrow T$. If the given tensor space is covariant, then the returned tensor has the frame $T \times U_v \times \cdots \times U_1 \rightarrow U_0$. Note that **AsTensor** is “inverse” to **AsCotensorSpace** and **AsTensorSpace** when $i = v$.

Example 2.33. Ten_Foliation

We demonstrate foliation.

```

> T := RandomTensor( GF(7), [2,3,4] );
> Foliation(T,0);
[3 4 3 3 6 4]
[0 6 4 3 3 4]
[0 0 6 2 3 3]
[4 6 4 0 0 3]
>
> Slice(T, [{1,2},{1..3},{3}]); // row 3
[ 0, 0, 6, 2, 3, 3 ]
>
> Slice(T, [{2},{1},{1..4}]); // col 4
[ 3, 3, 2, 0 ]

```

The cotensor space associated to a tensor is the subcotensor space spanned by multilinear forms obtained from T . For a fixed basis of U_0 , we get a basis for the cotensor space by projecting onto various basis vectors of U_0 .

```
> CT := AsCotensorSpace(T);
> CT;
Cotensor space of dimension 4 over GF(7) with valence 1
U2 : Full Vector space of degree 2 over GF(7)
U1 : Full Vector space of degree 3 over GF(7)
>
> S := Random(CT);
> MS := KMatrixSpace(GF(7), 2, 3);
> SystemOfForms(S) subset sub<MS|SystemOfForms(T)>;
true
```

The same applies for the associated tensor space to a tensor. We obtain a basis by projecting onto the basis vectors of U_i .

```
> TS := AsTensorSpace(T, 1);
> TS;
Tensor space of dimension 3 over GF(7) with valence 1
U1 : Full Vector space of degree 2 over GF(7)
U0 : Full Vector space of degree 4 over GF(7)
>
> S := Random(TS);
> MS := KMatrixSpace(GF(7), 2, 4);
> AsMatrices(S, 1, 0) subset sub<MS|AsMatrices(T, 2, 0)>;
true
```

2.3. Invariants of tensors

To access the projections or the objects acting on a specific factor, the following should be used.

```
Induce(X, i) : AlgMat, RngIntElt -> AlgMat, Map
Induce(X, i) : AlgMatLie, RngIntElt -> AlgMatLie, Map
Induce(X, i) : GrpMat, RngIntElt -> GrpMat, Map
```

Returns the induced sub-object associated to the i th factor of the associated tensor and a projection from the given object to the returned sub-object.

2.3.1. Standard invariants. We integrate the invariant theory associated to bilinear and multilinear maps into the realm of tensors.

```
Radical(T, s) : TenSpcElt, RngIntElt -> ModTupRng, Map
```

Returns the s th (categorical) radical as a subspace of U_s along with an isomorphism from U_s to K^{d_s} . If U_s is already a vector space, then the returned map is the identity.

```
Radical(T) : TenSpcElt -> Tup
```

Returns the tuple of all the s -radicals for each $s \in \{1, \dots, v\}$.

```
Coradical(T) : TenSpcElt -> ModTupRng, Map
```

Returns the (categorical) coradical of T and a vector space surjection from the codomain to the coradical.

2.3.2. Invariants for bilinear tensors. The following are used only for tensors of valence 2.

AdjointAlgebra(B) : TenSpcElt -> AlgMat

Returns the adjoint $*$ -algebra of the given bilinear map.

Example 2.34. Ten_Adj_Alg

We get a pair of random cotensors from the exterior square of $V = \mathbb{F}_5^{10}$ and compute its adjoint $*$ -algebra as a tensor.

```
> V := VectorSpace(GF(5),10);
> E := ExteriorCotensorSpace(V,2);
> E;
Cotensor space of dimension 45 over GF(5) with valence 1
U2 : Full Vector space of degree 10 over GF(5)
U1 : Full Vector space of degree 10 over GF(5)
>
> T := Random(E);
> S := Random(E);
> CT := SubTensorSpace(E,[T,S]);
> T2 := AsTensor(CT);
> T2;
Tensor of valence 2, 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 2 over GF(5)
>
> A := AdjointAlgebra(T2);
> RecognizeStarAlgebra(A);
true
> Star(A);
Mapping from: AlgMat: A to AlgMat: A given by a rule [no
inverse]
```

LeftNucleus(B) : TenSpcElt -> AlgMat

Returns the left nucleus of the bilinear map B as a subalgebra of $\text{End}_K(U_2) \times \text{End}_K(U_0)$.

MidNucleus(B) : TenSpcElt -> AlgMat

Returns the mid nucleus of the bilinear map B as a subalgebra of $\text{End}_K(U_2) \times \text{End}_K(U_1)$.

RightNucleus(B) : TenSpcElt -> AlgMat

Returns the right nucleus of the bilinear map B as a subalgebra of $\text{End}_K(U_1) \times \text{End}_K(U_0)$.

2.3.3. Invariants of general multilinear maps. The following functions can be used for general multilinear maps.

Centroid(T) : TenSpcElt -> AlgMat

Returns the centroid of the tensor as a subalgebra of $\prod_{i=0}^v \text{End}_K(U_i)$.

Example 2.35. Ten_Centroid

We will compute the centroid of the tensor given by field multiplication in $\mathbb{F}_{3^{25}}$ as a subalgebra of $M_{25}(\mathbb{F}_3)$.

```

> A := MatrixAlgebra(GF(3),25);
> f := RandomIrreduciblePolynomial(GF(3),25);
> S := sub< A | CompanionMatrix(f) >; // GF(3^25) inside A
> T := Tensor(S);
> T;
Tensor of valence 2, U2 x U1 -> U0
U2 : Matrix Algebra of degree 25 with 1 generator over GF(3)
U1 : Matrix Algebra of degree 25 with 1 generator over GF(3)
U0 : Matrix Algebra of degree 25 with 1 generator over GF(3)
> C := Centroid(T);
> Dimension(C);
25
> Ngens(C);
1

```

DerivationAlgebra(T) : TenSpcElt -> AlgMatLie

Returns the derivation Lie algebra of the tensor as a Lie subalgebra of $\prod_{i=0}^v \text{End}_K(U_i)$.

Nucleus(T, s, t) : TenSpcElt, RngIntElt, RngIntElt -> AlgMat

Returns the st -nucleus ($s \neq t$) of the tensor as a subalgebra of $\text{End}_K(U_i) \times \text{End}_K(U_j)$, where $i = \max(s, t)$ and $j = \min(s, t)$.

Example 2.36. Ten_Der_Nuc

We first construct the derivation of the quaternion algebra over \mathbb{Q} and verify that it is of type A_1 .

```

> H := QuaternionAlgebra(Rationals(),-1,-1);
> T := Tensor(H);
> T;
Tensor of valence 2, U2 x U1 -> U0
U2 : Quaternion Algebra with base ring Rational Field,
defined by i^2 = -1, j^2 = -1
U1 : Quaternion Algebra with base ring Rational Field,
defined by i^2 = -1, j^2 = -1
U0 : Quaternion Algebra with base ring Rational Field,
defined by i^2 = -1, j^2 = -1
> D := DerivationAlgebra(T);
> SemisimpleType(D);
A1

```

Now we verify that the mid nucleus of T is the quaternion algebra.

```

> ChangeTensorCategory(~T, HomotopismCategory(2));
> N := Nucleus(T,2,1);
> Dimension(N);
4
> N.1^2 eq N!-1;
true
> N.2^2 eq N!-1;
true
> N.1*N.2 eq -N.2*N.1;
true

```

If the centroid of a tensor is a commutative local ring, we can rewrite a tensor over its centroid. We employ the algorithms developed by Brooksbank and Wilson [BW2] to efficiently determine if a matrix algebra is cyclic.

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.

Example 2.37. Ten_Over_Cen

We construct a pc presentation of the upper unitriangular matrices over \mathbb{F}_{1024} . This will “forget” the field structure of our group, so that the tensor given by the commutator will be over \mathbb{F}_2 . We will use the centroid to rewrite our tensor as an alternating form over \mathbb{F}_{1024} .

```
> G := ClassicalSylow( GL(3,1024), 2 );
> P := PCPresentation( UnipotentMatrixGroup(G) );
> T := pCentralTensor(P,1,1);
> T;
Tensor of valence 2, 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)
> TC := TensorOverCentroid(T);
> TC;
Tensor of valence 2, 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)
> IsAlternating(TC);
true
```

We include some well-known polynomial invariants for bilinear maps.

Discriminant(B) : TenSpcElt -> RngMPolElt

Returns the discriminant of the bilinear map.

Pfaffian(B) : TenSpcElt -> RngMPolElt

Returns the Pfaffian of the antisymmetric bilinear map.

Example 2.38. Ten_Disc_Pfaff

We construct the discriminant and the Pfaffian of an antisymmetric 2-tensor.

```
> J := Matrix(GF(7), [[0,1],[-1,0]]);
> J;
[0 1]
[6 0]
> M := [ InsertBlock(ZeroMatrix(GF(7),4,4),J,i,i)
        : i in [1..3] ];
> M;
[
  [0 1 0 0]
  [6 0 0 0]
  [0 0 0 0]
```

```

      [0 0 0 0],
      [0 0 0 0]
      [0 0 1 0]
      [0 6 0 0]
      [0 0 0 0],
      [0 0 0 0]
      [0 0 0 0]
      [0 0 0 1]
      [0 0 6 0]
    ]
> T := Tensor(M,2,1);
> T;
Tensor of valence 2, U2 x U1 -> U0
U2 : Full Vector space of degree 4 over GF(7)
U1 : Full Vector space of degree 4 over GF(7)
U0 : Full Vector space of degree 3 over GF(7)
> Discriminant(T);
$.1^2*$.3^2
> Pfaffian(T);
$.1*$.3

```

2.4. Exporting tensors

Tensors can be used define algebraic structures such as groups and algebras.

HeisenbergAlgebra(B) : TenSpcElt -> AlgGen

Returns the Heisenberg algebra A induced by the bilinear tensor B . If $B = \circ : U \times V \rightarrow W$ is a bilinear map of K -vector spaces, and U , V , and W are isomorphic, then A is the algebra over U with the given product. If U and V are isomorphic but not with W , then A is the algebra over $U \oplus W$ with the given product. If U is not isomorphic to V , then it creates a new bilinear map $\bullet : (U \oplus V) \times (U \oplus V) \rightarrow W$, where

$$(u, v) \bullet (u', v') = u \circ v'.$$

HeisenbergLieAlgebra(B) : TenSpcElt -> AlgLie

Returns the Heisenberg Lie algebra with Lie bracket given by the alternating bilinear tensor induced by B .

HeisenbergGroup(B : parameters) : TenSpcElt -> GrpMat

UseAlt : BoolElt : true

HeisenbergGroupPC(B : parameters) : TenSpcElt -> GrpPC

UseAlt : BoolElt : true

Returns the class 2, exponent p , Heisenberg p -group with commutator given by the bilinear tensor $B : U \times V \rightarrow W$ over a finite field. If B is alternating and **UseAlt** is set to **true**, then the group returned is an extension of V by W , so $|G| = |V| \cdot |W|$. On the other hand, if either B is not alternating or **UseAlt** is set to **false**, then the group returned is an extension of $U \oplus V$ by W , so $|G| = |U| \cdot |V| \cdot |W|$. The intrinsic **HeisenbergGroupPC** uses the **pQuotient** functions to convert a finitely presented group into a polycyclic group, so this can only return groups of order $\leq p^{256}$.

Example 2.39. Ten_Heisenberg

From a single 2-tensor we construct an algebra, a Lie algebra, and a p -group.

```
> T := RandomTensor(GF(3), [10, 10, 4]);
> V := Domain(T)[1];
> V.1*T*V.2;
(0 0 1 0)
> A := HeisenbergAlgebra(T);
> A;
Algebra of dimension 14 with base ring GF(3)
> A.1*A.2;
(0 0 0 0 0 0 0 0 0 0 0 0 1 0)
```

Now we create the Lie algebra. The tensor T is not alternating, so it will induce an alternating tensor form which to create the Lie algebra.

```
> L := HeisenbergLieAlgebra(T);
> L.1*L.2;
(0 0 0 0 0 0 0 0 0 2 0 1 0)
> T2 := AlternatingTensor(T);
> V.1*T2*V.2;
(2 0 1 0)
```

Now we create the Heisenberg 3-group G from T . Because this algorithm uses the p -quotient algorithms, the commutator on G will not be identical to the induced alternating tensor T_2 . Instead, it will be pseudo-isometric to the T_2 .

```
> G := HeisenbergGroup(T);
> (G.2, G.1); // Defining word for 1st gen in Frattini
G.11
```

2.5. Invariants of nonassociative algebras

Converting an algebra to a tensor enables Magma to compute standard invariants of any algebra. We note that there are known errors for \mathbb{R} and \mathbb{C} due to the numerical stability of the linear algebra involved in the computations.

```
Center(A) : Alg -> Alg
Centre(A) : Alg -> Alg
```

Returns the center of the algebra A .

```
Centroid(A) : Alg -> AlgMat
```

Returns the centroid of the K -algebra A as a subalgebra of $\text{End}_K(A)$.

```
LeftNucleus(A) : Alg -> AlgMat
RightNucleus(A) : Alg -> AlgMat
MidNucleus(A) : Alg -> AlgMat
```

Returns the nucleus of the algebra A as a subalgebra of the enveloping algebra of right multiplication $\mathcal{R}(A)$.

```
DerivationAlgebra(A) : Alg -> AlgMatLie
```

Returns the derivation algebra of the algebra A as a Lie subalgebra of $\text{End}_K(A)$.

Example 2.40. Alg_Invariants

We demonstrate how to use these functions to get invariants of nonassociative algebras. First, we will obtain the derivation Lie algebra of the Octonions, which are of type G_2 .

```
> A := OctonionAlgebra(GF(7), -1, -1, -1);
> A;
Algebra of dimension 8 with base ring GF(7)
> D := DerivationAlgebra(A);
> D.1;
[0 0 0 0 0 0 0 0]
[0 0 6 0 6 3 2 1]
[0 1 0 3 4 1 1 3]
[0 0 4 0 6 4 2 3]
[0 1 3 1 0 6 2 0]
[0 4 6 3 1 0 6 2]
[0 5 6 5 5 1 0 4]
[0 6 4 4 0 5 3 0]
> Dimension(D);
14
> SemisimpleType(D);
G2
```

Now we will show that the left, mid, and right nuclei are all one dimensional. All of which are generated by R_1 , multiplication by 1_A .

```
> Z := Center(A);
> Z;
Algebra of dimension 1 with base ring GF(7)
>
> L := LeftNucleus(A);
> L;
Matrix Algebra of degree 8 with 1 generator over GF(7)
> L.1;
[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]
>
> L eq MidNucleus(A);
true
> L eq RightNucleus(A);
true
```


CHAPTER 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 a multilinear map. Each tensor space further maintains a tensor category which is assigned to its tensors.

3.1. Constructions of tensor and cotensor spaces

3.1.1. 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 $S = [d_v, \dots, d_0]$, returns the universal tensor space $\text{hom}_K(K^{d_v}, \dots, \text{hom}_K(K^{d_1}, K^{d_0}))$ with covariant tensor category given by C . The default is Albert's homotopism category.

```
RTensorSpace(R, S) : Rng, [RngIntElt] -> TenSpc
RTensorSpace(R, S, C) : Rng, [RngIntElt], TenCat -> TenSpc
```

For a commutative and unital ring R and sequence $S = [d_v, \dots, d_0]$, returns the universal tensor space $\text{hom}_R(R^{d_v}, \dots, \text{hom}_R(R^{d_1}, R^{d_0})) \cong R^{d_v \cdots d_0}$ with covariant tensor category given by C . The default is Albert's homotopism category.

```
TensorSpace(S) : SeqEnum -> TenSpc
TensorSpace(S) : List -> TenSpc
TensorSpace(S, C) : SeqEnum, TenCat -> TenSpc
TensorSpace(S, C) : List, TenCat -> TenSpc
```

Given a sequence $S = [U_v, \dots, U_0]$ of K -modules returns a universal tensor space equivalent to $\text{hom}_K(U_v \otimes_K \dots \otimes_K U_1, U_0)$ with covariant tensor category given by C . The default is Albert's homotopism category.

```
TensorSpace(V, p, q) : ModTupFld, RngIntElt, RngIntElt -> TenSpc
TensorSpace(K, d, p, q) : Fld, RngIntElt, RngIntElt, RngIntElt -> TenSpc
```

Returns the signed (p, q) -tensor space over the vector space $V = K^d$. The first p indices are covariant and the last q indices are contravariant. This is functionally equivalent to creating a universal tensor space from the sequence $[V, \dots, {}_p V, V^*, \dots, {}_q V^*]$ and the tensor category with arrows $[1, \dots, {}_p 1, -1, \dots, {}_q -1]$ and duplicates $\{\{p + q, \dots, 1 + q\}, \{q, \dots, 1\}, \{0\}\}$.

Example 3.1. TenSpc_Const

We demonstrate how to construct universal tensor spaces.

```
> TS := KTensorSpace(Rationals(), [ i : i in [3..7] ]);
> TS;
Tensor space of dimension 2520 over Rational Field with
valence 4
```

```

U4 : Full Vector space of degree 3 over Rational Field
U3 : Full Vector space of degree 4 over Rational Field
U2 : Full Vector space of degree 5 over Rational Field
U1 : Full Vector space of degree 6 over Rational Field
U0 : Full Vector space of degree 7 over Rational Field
>
> TS.1;
Tensor of valence 4, U4 x U3 x U2 x U1 -> U0
U4 : Full Vector space of degree 3 over Rational Field
U3 : Full Vector space of degree 4 over Rational Field
U2 : Full Vector space of degree 5 over Rational Field
U1 : Full Vector space of degree 6 over Rational Field
U0 : Full Vector space of degree 7 over Rational Field

```

Give a tensor space a frame.

```

> R := pAdicRing(3,6);
> Fr := [RSpace(R,5), sub<RSpace(R,3)|[0,1,0],[0,0,1]>, \
>   RSpace(R,2)];
> TS := TensorSpace(Fr);
> TS;
Tensor space of dimension 20 over 3-adic ring mod 3^6 with
valence 2
U2 : Full RSpace of degree 5 over pAdicRing(3, 6)
U1 : RSpace of degree 3, dimension 2 over pAdicRing(3, 6)
Generators:
(0 1 0)
(0 0 1)
Echelonized basis:
(0 1 0)
(0 0 1)
U0 : Full RSpace of degree 2 over pAdicRing(3, 6)
>
> TS.10;
Tensor of valence 2, U2 x U1 -> U0
U2 : Full RSpace of degree 5 over pAdicRing(3, 6)
U1 : RSpace of degree 3, dimension 2 over pAdicRing(3, 6)
Generators:
(0 1 0)
(0 0 1)
Echelonized basis:
(0 1 0)
(0 0 1)
U0 : Full RSpace of degree 2 over pAdicRing(3, 6)

```

With signed tensor spaces, the tensor category is not immediately obvious at print out. Instead, one can glean categorical information using `TensorCategory` on a tensor space.

```

> TS := TensorSpace( VectorSpace(GF(3),3), 2, 4 );
> TS;
Tensor space of dimension 729 over GF(3) with valence 6
U6 : Full Vector space of degree 3 over GF(3)
U5 : Full Vector space of degree 3 over GF(3)
U4 : Full Vector space of degree 3 over GF(3)

```

```

U3 : Full Vector space of degree 3 over GF(3)
U2 : Full Vector space of degree 3 over GF(3)
U1 : Full Vector space of degree 3 over GF(3)
U0 : Full Vector space of degree 1 over GF(3)
>
> TensorCategory(TS);
Tensor category of Valence 6 (<-,<-,->,->,->,->,==)
({ 0 }, { 5 .. 6 }, { 1 .. 4 })

```

We see that the indices 5 and 6 are contravariant and both are linked together as well. Furthermore the covariant indices are 1–4, which are also linked together.

3.1.2. Universal cotensor spaces.

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 $S = [d_v, \dots, d_1]$ returns the universal cotensor space $\text{hom}_K(K^{d_v} \otimes \dots \otimes K^{d_1}, K) \cong K^{d_v \cdots d_1}$ with the given contravariant tensor category C . The default is Albert's homotopism category.

```

CotensorSpace(S) : SeqEnum -> TenSpc
CotensorSpace(S) : List -> TenSpc
CotensorSpace(S, C) : SeqEnum, TenCat -> TenSpc
CotensorSpace(S, C) : List, TenCat -> TenSpc

```

Given a sequence $S = [U_v, \dots, U_1]$ of K -vector spaces returns the universal tensor space equivalent to $\text{hom}_K(U_v \otimes_K \dots \otimes_K U_1, K)$ with contravariant tensor category given by C . The default is Albert's homotopism category.

Example 3.2. CoTenSpc_Const

We construct cotensor spaces in the same way as we do tensor spaces.

```

> CT := KCotensorSpace(GF(2), [ i : i in [5..7] ]);
> CT;
Cotensor space of dimension 210 over GF(2) with valence 2
U3 : Full Vector space of degree 5 over GF(2)
U2 : Full Vector space of degree 6 over GF(2)
U1 : Full Vector space of degree 7 over GF(2)
>
> CT.1;
Cotensor of valence 2, U3 x U2 x U1 -> K
U3 : Full Vector space of degree 5 over GF(2)
U2 : Full Vector space of degree 6 over GF(2)
U1 : Full Vector space of degree 7 over GF(2)
>
>
> Cat := CotensorCategory([1,0,-1],{{1},{2},{3}});
> Fr := [ VectorSpace(GF(8),4) : i in [1..3] ];
> CT := CotensorSpace(Fr, Cat);
> CT;
Cotensor space of dimension 64 over GF(2^3) with valence 2
U3 : Full Vector space of degree 4 over GF(2^3)
U2 : Full Vector space of degree 4 over GF(2^3)

```

```

U1 : Full Vector space of degree 4 over GF(2^3)
>
> TensorCategory(CT);
Cotensor category of valence 2 (->,==,<-) ({ 1 }, { 2 }, { 3 })

```

3.1.3. Some standard constructions. We include some subspaces generated by well-known tensors.

AlternatingSpace(T) : TenSpc -> TenSpc

Returns the sub(co-)tensor space generated by all the alternating (co-)tensors contained in the given (co-)tensor space.

AntisymmetricSpace(T) : TenSpc -> TenSpc

Returns the sub(co-)tensor space generated by all the antisymmetric (co-)tensors contained in the given (co-)tensor space.

SymmetricSpace(T) : TenSpc -> TenSpc

Returns the sub(co-)tensor space generated by all the symmetric (co-)tensors contained in the given (co-)tensor space.

ExteriorCotensorSpace(V, n) : ModTupFld, RngIntElt -> TenSpc

Returns the cotensor space given by the n th exterior power of the vector space V .

SymmetricCotensorSpace(V, n) : ModTupFld, RngIntElt -> TenSpc

Returns the cotensor space given by the n th symmetric power of the vector space V .

Example 3.3. TenSpc_Const2

We construct the symmetric subtensor space of the universal tensor space.

```

> TS := KTensorSpace(GF(3), [3,3,3,2]);
> TS;
Tensor space of dimension 54 over GF(3) with valence 3
U3 : Full Vector space of degree 3 over GF(3)
U2 : Full Vector space of degree 3 over GF(3)
U1 : Full Vector space of degree 3 over GF(3)
U0 : Full Vector space of degree 2 over GF(3)
>
> SS := SymmetricSpace(TS);
> AsMatrices(Random(SS), 3, 2);
[
  [1 1 1]
  [1 0 0]
  [1 0 0],
  [2 2 2]
  [2 0 1]
  [2 1 0],
  [1 0 0]
  [0 1 2]
  [0 2 2],
  [2 0 1]

```

```

[0 1 1]
[1 1 2],

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

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

```

For $V = \mathbb{F}_{25}^6$, we construct the fourth exterior power of V , $\Lambda^4(V)$ as a sub cotensor space of dimension $\binom{6}{4} = 15$.

```

> V := VectorSpace(GF(25),6);
> E := ExteriorCotensorSpace(V,4);
> E;
Cotensor space of dimension 15 over GF(5^2) with valence 3
U4 : Full Vector space of degree 6 over GF(5^2)
U3 : Full Vector space of degree 6 over GF(5^2)
U2 : Full Vector space of degree 6 over GF(5^2)
U1 : Full Vector space of degree 6 over GF(5^2)
>
> T := Random(E);
> IsAlternating(T);
true

```

3.1.4. Subspaces as closures.

```

DerivationClosure(TS, O) : TenSpc, AlgMat -> TenSpc
DerivationClosure(TS, O) : TenSpc, ModMatFld -> TenSpc
DerivationClosure(TS, O) : TenSpc, AlgMatLie -> TenSpc
DerivationClosure(TS, O) : TenSpc, [Mtrx] -> TenSpc
DerivationClosure(TS, O) : TenSpc, [AlgMatLie] -> TenSpc

```

Returns the derivation closure of the given tensor space TS , with frame $U_2 \times U_1 \rightarrow U_0$, with operators $O \subseteq \text{End}(U_2) \times \text{End}(U_1) \times \text{End}(U_0)$. Currently, this only works for tensor spaces of valence 2. This is the subspace whose tensors' derivation algebra contains O .

```
DerivationClosure(TS, T) : TenSpc, TenSpcElt -> TenSpc
```

Returns the derivation closure of the given tensor space TS , with frame $U_2 \times U_1 \rightarrow U_0$, with operators $O \subseteq \text{End}(U_2) \times \text{End}(U_1) \times \text{End}(U_0)$. Currently, this only works for tensor spaces of valence 2. This is the subspace whose tensors' derivation algebra contains the derivation algebra of T .

```

NucleusClosure(TS, O, s, t) : TenSpc, AlgMat, RngIntElt, RngIntElt -> TenSpc
NucleusClosure(TS, O, s, t) : TenSpc, ModMatFld, RngIntElt, RngIntElt -> TenSpc
NucleusClosure(TS, O, s, t) : TenSpc, [Mtrx], RngIntElt, RngIntElt -> TenSpc

```

Returns the nucleus closure of the tensor space TS , with frame $U_2 \times U_1 \rightarrow U_0$, with operators $O \subseteq \text{End}(U_s) \times \text{End}(U_t)$. Currently, this only works for tensor spaces of valence 2. This returns the subspace whose tensors' st -nucleus contains O .

```
NucleusClosure(TS, T, s, t) : TenSpc, TenSpcElt, RngIntElt, RngIntElt -> TenSpc
```

Returns the nucleus closure of the tensor space TS , with frame $U_2 \times U_1 \mapsto U_0$, with operators $O \subseteq \text{End}(U_s) \times \text{End}(U_t)$. Currently, this only works for tensor spaces of valence 2. This returns the subspace whose tensors' *st*-nucleus contains the *st*-nucleus of T .

Example 3.4. Der_Closure

We illustrate the fact that the (hyper-)matrix multiplication is unique. The derivation closure of the tensor in the universal tensor space is 1-dimensional.

```
> Fr := [ KMatrixSpace(GF(3),2,3),
>         KMatrixSpace(GF(3),3,2),KMatrixSpace(GF(3),2,2) ];
> F := func< x | x[1]*x[2] >;
> T := Tensor(Fr,F);
> T;
Tensor of valence 2, U2 x U1 -> U0
U2 : Full KMatrixSpace of 2 by 3 matrices over GF(3)
U1 : Full KMatrixSpace of 3 by 2 matrices over GF(3)
U0 : Full KMatrixSpace of 2 by 2 matrices over GF(3)
>
> TS := Parent(T);
> TS;
Tensor space of dimension 144 over GF(3) with valence 2
U2 : Full Vector space of degree 6 over GF(3)
U1 : Full Vector space of degree 6 over GF(3)
U0 : Full Vector space of degree 4 over GF(3)
>
> D := DerivationClosure(TS,T);
> D;
Tensor space of dimension 1 over GF(3) with valence 2
U2 : Full Vector space of degree 6 over GF(3)
U1 : Full Vector space of degree 6 over GF(3)
U0 : Full Vector space of degree 4 over GF(3)
```

Example 3.5. Nuc_Closure

We illustrate that the commutator from the Heisenberg group is unique over the correct field.

```
> H := ClassicalSylow( GL(3,125), 5 ); // Heisenberg group
> T := pCentralTensor(H,5,1,1);
> T;
Tensor of valence 2, 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)
```

The centroid of T will be 3-dimensional and is isomorphic to \mathbb{F}_{125} .

```
> C := Centroid(T);
> C;
Matrix Algebra of degree 15 with 3 generators over GF(5)
>
> S := TensorOverCentroid(T);
> S;
Tensor of valence 2, 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)
>
> TS := Parent(S);
> N := NucleusClosure(TS,S,2,1);
> N;
Tensor space of dimension 1 over GF(5^3) with valence 2
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)

```

Compare this closure with the closure of our original tensor over \mathbb{F}_5 .

```

> NT := NucleusClosure(Parent(T),T,2,1);
> NT;
Tensor space of dimension 36 over GF(5) with valence 2
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)

```

3.2. Operations on tensor spaces

3.2.1. 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

Decides if T is contained in the tensor space TS .

IsCoercible(TS, T) : TenSpc, TenSpcElt -> BoolElt, TenSpcElt

TS ! T : TenSpc, TenSpcElt -> BoolElt, TenSpcElt

Decides if the tensor T can be coerced into the tensor space TS . If so, the tensor is returned as an element of TS .

IsCoercible(TS, S) : TenSpc, SeqEnum -> BoolElt, TenSpcElt

TS ! S : TenSpc, SeqEnum -> BoolElt, TenSpcElt

Decides if the sequence S can be coerced into the tensor space TS as a tensor. If so, the corresponding tensor is returned.

IsCoercible(TS, n) : TenSpc, RngIntElt -> BoolElt, TenSpcElt

TS ! n : TenSpc, RngIntElt -> BoolElt, TenSpcElt

This is a shortcut designed to only work when $n = 0$, and thus, return **true** and the zero tensor from the tensor space. Any other integer will yield an error.

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 .

Example 3.6. TenCoerce

We illustrate that using **!** is the same as creating the tensor from scratch.

```

> TS := KTensorSpace( GF(2), [2,3,2] );
> TS;
Tensor space of dimension 12 over GF(2) with valence 2
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 := [ Random(GF(2)) : i in [1..12] ];
> T := TS!S;
> T;
Tensor of valence 2, 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

```

We demonstrate how to coerce into the symmetric cube of $V = \mathbb{Q}^{10}$ and construct a subcotensor space.

```

> V := VectorSpace(Rationals(), 10);
> SS := SymmetricCotensorSpace(V, 3);
> SS;
Cotensor space of dimension 220 over Rational Field with valence 2
U3 : Full Vector space of degree 10 over Rational Field
U2 : Full Vector space of degree 10 over Rational Field
U1 : Full Vector space of degree 10 over Rational Field
>
> CT := SubtensorSpace(SS, SS![1..1000]);
> CT;
Cotensor space of dimension 1 over Rational Field with valence 2
U3 : Full Vector space of degree 10 over Rational Field
U2 : Full Vector space of degree 10 over Rational Field
U1 : Full Vector space of degree 10 over Rational Field
>
> CT subset SS;
true
> CT.1 in SS;
true
> SS.2 in CT;
false

```

3.2.2. 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.i : `TenSpc, RngIntElt -> TenSpcElt`

Returns the i th 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 it is finite.

`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 (co)tensor space $\text{hom}_R(R^{d_v}, \dots, \text{hom}_R(R^{d_1}, R^{d_0}) \dots)$ with category C . The default category is the homotopism category.

Example 3.7. TenSpc_Module

We demonstrate the module functions for a tensor space.

```
> TS := KTensorSpace( GF(9), [2,2,2,2] );
> TS;
Tensor space of dimension 16 over GF(3^2) with valence 3
U3 : Full Vector space of degree 2 over GF(3^2)
U2 : Full Vector space of degree 2 over GF(3^2)
U1 : Full Vector space of degree 2 over GF(3^2)
U0 : Full Vector space of degree 2 over GF(3^2)
>
> Ngens(TS);
16
> #TS eq 9^Ngens(TS);
true
>
> Eltseq(TS.2);
[ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
```

We can use `RandomTensor` to get a random tensor or cotensor from scratch.

```
> T := RandomTensor(GF(3), [2,2,2]);
> T;
Tensor of valence 2, 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 2 over GF(3)
>
> Cat := CotensorCategory([1,1,1], {{1,2,3}});
> T := RandomTensor(GF(3), [2,2,2], Cat);
> T;
Cotensor of valence 2, U3 x U2 x U1 -> K
U3 : Full Vector space of degree 2 over GF(3)
U2 : Full Vector space of degree 2 over GF(3)
U1 : Full Vector space of degree 2 over GF(3)
```

3.2.3. 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.

`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.

`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 3.8. TenSpc_Prop

We demonstrate the functions to obtain standard tensor space properties.

```
> TS := KTensorSpace( GF(23), [3,4,5,6] );
> TS;
Tensor space of dimension 360 over GF(23) with valence 3
U3 : Full Vector space of degree 3 over GF(23)
U2 : Full Vector space of degree 4 over GF(23)
U1 : Full Vector space of degree 5 over GF(23)
U0 : Full Vector space of degree 6 over GF(23)
>
> Valence(TS);
3
> Frame(TS);
[*
  Full Vector space of degree 3 over GF(23),
```

```

Full Vector space of degree 4 over GF(23),

Full Vector space of degree 5 over GF(23),

Full Vector space of degree 6 over GF(23)
*]
> TensorCategory(TS);
Tensor category of Valence 3 (->,->,->,->)
({ 1 }, { 2 }, { 0 }, { 3 })
>
> Cat := TensorCategory([1,1,-1,-1],{{0},{1},{2},{3}});
> ChangeTensorCategory(~TS,Cat);
> TensorCategory(TS);
Tensor category of Valence 3 (->,->,<-,<-)
({ 1 }, { 2 }, { 0 }, { 3 })

```

We construct the universal cotensor space of the symmetric cube, $S^3(V)$, of a vector space V .

```

> V := VectorSpace( GF(3), 5 );
> S := SymmetricCotensorSpace(V,3);
> S;
Cotensor space of dimension 30 over GF(3) with valence 2
U3 : Full Vector space of degree 5 over GF(3)
U2 : Full Vector space of degree 5 over GF(3)
U1 : Full Vector space of degree 5 over GF(3)
> UniversalCotensorSpace(S);
Cotensor space of dimension 125 over GF(3) with valence 2
U3 : Full Vector space of degree 5 over GF(3)
U2 : Full Vector space of degree 5 over GF(3)
U1 : Full Vector space of degree 5 over GF(3)
>
> IsSymmetric(S);
true

```


CHAPTER 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. 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 [W2]. In particular given a tensor T framed by $[U_v, \dots, U_0]$ then a tensor category for T will specify a function $A : \{0 \dots v\} \rightarrow \{-1, 0, 1\}$ along with a partition \mathcal{P} of $\{0, \dots, v\}$ such that the following rules apply to the tensors and morphisms in the category.

- (1) for each tensor T framed by $[U_v, \dots, U_0]$, if $X \in \mathcal{P}$, then

$$\forall i, j \in X, \quad U_i = U_j.$$

- (2) Given a second tensor S framed by $[V_v, \dots, V_0]$, a morphism $f : T \rightarrow S$ (Magma type **Hmtp**) will be a list $[f_v, \dots, f_0]$ of homomorphisms as follows:

- (Covariant) if $A(i) = 1$ then $f_i : U_i \rightarrow V_i$;
- (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 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 $A : \{0, \dots, v\} \rightarrow \{-1, 0, 1\}$ and a partition P of $\{0, \dots, v\}$. 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).

4.1. Creating 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 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 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.

`HomotopismCategory(v : parameters) : RngIntElt -> TenCat`

`Contravariant : BoolElt : 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 $\{0, \dots, v\}$. Position s is covariant, position t is contravariant.

Example 4.1. TenCat_Const

We illustrate how to construct tensor categories.

```
> Cat := TensorCategory([1,-1,0],{{0},{1},{2}});
> Cat;
Tensor category of Valence 2 (->,<-,==) ({ 1 },{ 2 },{ 0 })
>
> TS := KTensorSpace(GF(5),[5,3,4],Cat);
> TS;
Tensor space of dimension 60 over GF(5) with valence 2
U2 : Full Vector space of degree 5 over GF(5)
U1 : Full Vector space of degree 3 over GF(5)
U0 : Full Vector space of degree 4 over GF(5)
> TensorCategory(TS);
Tensor category of Valence 2 (->,<-,==) ({ 1 },{ 2 },{ 0 })
>
> IsContravariant(TS);
false
```

All the tensor constructors that allow a `TenCat` input can be used to make cotensors.

```
> Cat := HomotopismCategory(2 : Contravariant := true);
> Cat;
Cotensor category of valence 2 (->,->) ({ 1 },{ 2 })
> T := Tensor(GF(5),[2,2],[1..4],Cat);
> T;
Cotensor of valence 1, U2 x U1 ->-> K
```

```
U2 : Full Vector space of degree 2 over GF(5)
U1 : Full Vector space of degree 2 over GF(5)
```

4.2. Operations on tensor categories

We have basic operations for tensor categories.

`C1 eq C2 : TenCat, TenCat -> BoolElt`

Decides 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 an 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`

Decides if the tensor category is covariant or contravariant.

Example 4.2. TenCat_Prop

We obtain basic properties of tensor categories.

```
> C1 := TensorCategory([1,1,-1,1],{{0,3},{1},{2}});
> C1;
Tensor category of Valence 3 (->,->,<-,->) ({ 1 },{ 2 },
{ 0, 3 })
>
> A := map< {0..3} -> {-1,0,1} | x :-> 1 >;
> C2 := TensorCategory(A,{{0..3}});
> C2;
Tensor category of Valence 3 (->,->,->,->) ({ 0 .. 3 })
>
> C1 eq C2;
false
> RepeatPartition(C2);
{
  { 0 .. 3 }
}
> Valence(C2);
3
> Arrows(C2);
[ 1, 1, 1, 1 ]
```

4.3. Categorical operations

4.3.1. Categorical operations on tensors. We include functions defined for the category of tensors. Most functions are currently defined only for Albert's homotopism category.

`Subtensor(T, S) : TenSpcElt, List -> TenSpcElt`

Returns the smallest submap of T containing S .

`Subtensor(T, D, C) : TenSpcElt, List, 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 .

`LocalIdeal(T, S, I) : TenSpcElt, List, {RngIntElt} -> TenSpcElt`

Returns the local ideal of T at I constaining S .

`LocalIdeal(T, D, C, I) : TenSpcElt, List, Any, {RngIntElt} -> TenSpcElt`

Returns the local ideal of T at I constaining 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 constaining S as a submap.

`IsLocalIdeal(T, S, I) : TenSpcElt, TenSpcElt, {RngIntElt} -> BoolElt`

Decides if S is a local ideal of T at I .

`Ideal(T, S) : TenSpcElt, List -> TenSpcElt`

Returns the ideal of T containing S .

`Ideal(T, D, C) : TenSpcElt, List, 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 .

`LocalQuotient(T, S, I : parameters) : TenSpcElt, TenSpcElt, {RngIntElt} -> TenSpcElt, Hmtp
Check : BoolElt : true`

Returns the local quotient of T by S at I . 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 : 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 4.3. Ten_Cat_Ops

We will construct a quotient of tensors. First, we construct a subtensor from a random tensor.

```
> T := RandomTensor(GF(5), [4,4,2]);
> T := RandomTensor(GF(5), [4,4,2]);
> T;
Tensor of valence 2, U2 x U1 -> U0
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 2 over GF(5)
>
> F := Frame(T);
```



```

> L := [* F[1]![1,1,1,0], F[2]![0,0,0,1], F[3]![0,0] *];
> S := Subtensor(T,L);
> S;
Tensor of valence 2, U2 x U1 >-> U0
U2 : Vector space of degree 4, dimension 1 over GF(5)
Generators:
(1 1 1 0)
Echelonized basis:
(1 1 1 0)
U1 : Vector space of degree 4, dimension 1 over GF(5)
Generators:
(0 0 0 1)
Echelonized basis:
(0 0 0 1)
U0 : Vector space of degree 2, dimension 1 over GF(5)
Generators:
(0 2)
Echelonized basis:
(0 1)
>
> IsSubtensor(T,S);
true

```

Now we construct the ideal of T containing S .

```

> I := Ideal(T,S);
> I;
Tensor of valence 2, U2 x U1 >-> U0
U2 : Vector space of degree 4, dimension 1 over GF(5)
Generators:
(1 1 1 0)
Echelonized basis:
(1 1 1 0)
U1 : Vector space of degree 4, dimension 1 over GF(5)
Generators:
(0 0 0 1)
Echelonized basis:
(0 0 0 1)
U0 : Full Vector space of degree 2 over GF(5)
Generators:
(1 0)
(0 1)
>
> IsIdeal(T,I);
true

```

Finally, we construct the quotient of T by I .

```

> T/I;
Tensor of valence 2, U2 x U1 >-> U0
U2 : Full Vector space of degree 3 over GF(5)
U1 : Full Vector space of degree 3 over GF(5)
U0 : Full Vector space of degree 0 over GF(5)
Maps from U2 x U1 >-> U0 to V2 x V1 >-> V0.

```

```

U2 -> V2: Mapping from: Full Vector space of degree 4 over
GF(5) to Full Vector space of degree 3 over GF(5)
U1 -> V1: Mapping from: Full Vector space of degree 4 over
GF(5) to Full Vector space of degree 3 over GF(5)
U0 -> V0: Mapping from: Full Vector space of degree 2 over
GF(5) to Full Vector space of degree 0 over GF(5)

```

4.3.2. Categorical operations on tensor spaces. We have categorical notions for tensor spaces as well.

```

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 .

```

QuoConstructor(T, S) : TenSpc, TenSpc -> TenSpc, Map
quo< T | S > : TenSpc, TenSpc -> TenSpc, Map
T / S : TenSpc, TenSpc -> TenSpc, Map

```

Returns the quotient tensor space of T by S .

Example 4.4. TenSpc_Cat_Ops

We construct a subtensor space.

```

> T := KTensorSpace(GF(2), [4, 4, 2]);
> T;
Tensor space of dimension 32 over GF(2) with valence 2
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 2 over GF(2)
>
> L := [ T.i : i in [1..Ngens(T)] | IsEven(i) ];
> S := SubtensorSpace(T, L);
> S;
Tensor space of dimension 16 over GF(2) with valence 2
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 2 over GF(2)
> SystemOfForms(Random(S));
[
  [0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]
  [0 0 0 0],
  [1 0 0 0]
  [0 1 0 1]
  [1 0 0 0]
  [0 0 1 0]
]

```

Now we compute the quotient tensor space $Q = T/S$.

```
> Q := T/S;
> Q;
Tensor space of dimension 16 over GF(2) with valence 2
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 2 over GF(2)
> SystemOfForms(Random(Q));
[
  [0 0 0 0]
  [1 1 1 0]
  [1 0 1 1]
  [1 1 0 1],
  [0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]
]
> SystemOfForms(Q![1 : i in [1..32]]);
[
  [1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]
  [1 1 1 1],
  [0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]
]
```

4.4. Homotopisms

Magma provides functions for homotopisms. Homotopisms are also equipped with a tensor category.

4.4.1. Constructions of Homotopisms.

`Homotopism(T, S, M : parameters) : TenSpcElt, TenSpcElt, List -> Hmtp`

`Check : BoolElt`

`Homotopism(T, S, M, C : parameters) : TenSpcElt, TenSpcElt, List, TenCat -> Hmtp`

`Check : BoolElt`

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 .

Example 4.5. Hmtp_Const

We illustrate how to construct tensor categories.

```
> TS := KTensorSpace(GF(4), [2, 3, 4]);
> T := Random(TS);
```

```

> S := Random(TS);
> M := [* Random(KMatrixSpace(GF(4),i,i)) : i in [2..4] *];
> H := Homotopism(T,S,M);
> H;
Maps from U2 x U1 -> U0 to V2 x V1 -> V0.
U2 -> V2:
[ 1 0]
[ 0 0]
U1 -> V1:
[ 0 $.1 $.1^2]
[ 1 1 $.1]
[ $.1 $.1 $.1^2]
U0 -> V0:
[ 1 1 $.1 $.1^2]
[ 0 1 $.1^2 $.1]
[ 0 0 $.1 $.1^2]
[ $.1 $.1^2 $.1 0]
>
>
> M[2] := map< Frame(TS)[2] -> Frame(TS)[2] | x :-> x >;
> H2 := Homotopism(T,S,M);
> H2;
Maps from U2 x U1 -> U0 to V2 x V1 -> V0.
U2 -> V2:
[ 1 0]
[ 0 0]
U1 -> V1: Mapping from: Full Vector space of degree 3
over GF(2^2) to Full Vector space of degree 3 over GF(2^2)
given by a rule [no inverse]
U0 -> V0:
[ 1 1 $.1 $.1^2]
[ 0 1 $.1^2 $.1]
[ 0 0 $.1 $.1^2]
[ $.1 $.1^2 $.1 0]

```

4.4.2. 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 .

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.

H.i : Hmtp, RngIntElt -> Map

Returns the map on the i th coordinate.

TensorCategory(H) : Hmtp -> TenCat

Returns the tensor category of H .

```
ChangeTensorCategory(H, C) : Hmtp, TenCat -> Hmtp
ChangeTensorCategory(~H, C) : Hmtp, TenCat -> Hmtp
```

Changes the tensor category of H to the given category.

```
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 4.6. Hmtp_Ops

We perform basic operations with homotopisms.

```
> T := RandomTensor(GF(7), [5, 4, 3]);
> F := Frame(T);
>
> I := [* hom< F[j] -> F[j] | [< F[j].i, F[j].i > : \
>   i in [1..Dimension(F[j])]] > : j in [1..3] *];
> H := Homotopism(T, T, I);
>
> Image(H);
Tensor of valence 2, U2 x U1 -> U0
U2 : Full Vector space of degree 5 over GF(7)
U1 : Full Vector space of degree 4 over GF(7)
U0 : Full Vector space of degree 3 over GF(7)
> Kernel(H);
Tensor of valence 2, U2 x U1 -> U0
U2 : Vector space of degree 5, dimension 0 over GF(7)
U1 : Vector space of degree 4, dimension 0 over GF(7)
U0 : Vector space of degree 3, dimension 0 over GF(7)
```

If the tensor is over vector spaces, then matrices can be used to create a homotopism.

```
> M := [* RandomMatrix(GF(7), i, i) : i in [5, 4, 3] *];
> G := Homotopism(T, T, M);
> G;
Maps from U2 x U1 -> U0 to V2 x V1 -> V0.
U2 -> V2:
[5 3 6 0 4]
[6 0 0 1 1]
[6 4 3 1 5]
[3 4 6 1 4]
[2 4 1 3 2]
U1 -> V1:
[5 5 0 3]
[3 5 1 3]
[3 6 1 5]
[2 3 5 4]
U0 -> V0:
[6 4 1]
[2 6 5]
[1 2 3]
```

Homotopisms can be composed so long as Magma can compose each of the individual maps.

```
> G*G;
```

```

Maps from U2 x U1 >-> U0 to V2 x V1 >-> V0.
U2 -> V2:
[3 6 3 0 5]
[0 5 1 4 2]
[1 5 0 2 1]
[2 4 4 2 2]
[4 2 0 0 5]
U1 -> V1:
[4 3 6 0]
[4 6 0 6]
[4 3 4 3]
[0 4 0 0]
U0 -> V0:
[3 1 1]
[1 5 5]
[6 1 6]

```

We can change the underlying category for the homotopism G to get a different morphism.

```

> Cat := TensorCategory([1,-1,1],{{0},{1},{2}});
> G := Homotopism(T,T,M,Cat);
> G;
Maps from U2 x U1 >-> U0 to V2 x V1 >-> V0.
U2 -> V2:
[5 3 6 0 4]
[6 0 0 1 1]
[6 4 3 1 5]
[3 4 6 1 4]
[2 4 1 3 2]
U1 <- V1:
[5 5 0 3]
[3 5 1 3]
[3 6 1 5]
[2 3 5 4]
U0 -> V0:
[6 4 1]
[2 6 5]
[1 2 3]
>
> Image(G);
Tensor of valence 2, U2 x U1 >-> U0
U2 : Full Vector space of degree 5 over GF(7)
U1 : Vector space of degree 4, dimension 0 over GF(7)
U0 : Vector space of degree 3, dimension 2 over GF(7)
Echelonized basis:
(1 0 4)
(0 1 3)
>
> Kernel(G);
Tensor of valence 2, U2 x U1 >-> U0
U2 : Vector space of degree 5, dimension 0 over GF(7)
U1 : Full Vector space of degree 4 over GF(7)
U0 : Vector space of degree 3, dimension 1 over GF(7)

```

Echelonized basis:
(1 4 0)

CHAPTER 5

Exceptional tensors

Magma provides functionality with common exceptional tensors. Many are used in the construction of nonassociative algebras. A few supporting functions for nonassociative algebras are also provided.

5.1. Generics for nonassociative algebras

5.1.1. Nonassociative algebras with involutions.

`IsStarAlgebra(A) : AlgGen -> BoolElt`

Decides if algebra has an involution, i.e. a *-algebra.

`Star(A) : AlgGen -> Map`

Returns involution of given *-algebra.

Example 5.1. Star_Alg

We demonstrate the functions dealing with involutions of nonassociative algebras.

```
> A := OctonionAlgebra(Rationals(), -1, -1, -1);
> IsStarAlgebra(A);
true
>
> s := Star(A);
> A.1; // A.1 is the mult. id.
(1 0 0 0 0 0 0 0)
> A.1 @ s;
(1 0 0 0 0 0 0 0)
>
> A.2;
(0 1 0 0 0 0 0 0)
> A.2 @ s;
( 0 -1 0 0 0 0 0 0)
```

5.1.2. Operations on power associative algebras. The following operations are defined for nonassociative algebras for which $x * (x * x) = (x * x) * x$.

`GenericMinimalPolynomial(x) : AlgGenElt -> FldElt`

`GenericMinimumPolynomial(x) : AlgGenElt -> FldElt`

The generic minimum polynomial of an element in a power associative algebra.

`GenericNorm(x) : AlgGenElt -> FldElt`

The generic norm of an element in a power associative algebra.

`GenericTrace(x) : AlgGenElt -> FldElt`

The generic trace of an element in a power associative algebra.

`GenericTracelessSubspaceBasis(A) : AlgGen -> Any`

Given a power associative algebra return a basis for the elements of generic trace 0.

Example 5.2. Ten_Generic

The trace $x + \bar{x}$ of a quaternion doubles the rational component, producing degenerate behavior in characteristic 2. The generic trace avoids this.

```
> Q := QuaternionAlgebra(Rationals(), 1,1);
> Trace(Q!1);
2
> GenericTrace(Q!1);
1
> Q := QuaternionAlgebra(GF(2), 1,1);
> Trace(Q!1);
0
> GenericTrace(Q!1);
1
```

The generic minimum polynomial of an element x in power associative algebra need only be a factor of the minimal polynomial of its right regular matrix $yR_x := x * y$.

```
> J := ExceptionalJordanCSA(GF(5));
> p := GenericMinimumPolynomial(J.3+J.12);
> Rx := AsMatrices(Tensor(J), 2,0); // yR_x = y*x.
> q := MinimalPolynomial(Rx[3]+Rx[12]);
> Degree(p);
3
> Degree(q);
6
> q mod p;
0
```

5.2. Compositions algebras

`CompositionAlgebra(K, a) : Fld, [FldElt] -> AlgGen`

`CompositionAlgebra(K, a) : Fld, [RngIntElt] -> AlgGen`

Constructs the composition algebra with specified parameters. The algebra returned has an involution.

The method is modestly intentional choosing Magma's favored representation of the individually classified algebras according to Hurwitz's theorem. In the case of fields the type returned is an algebra with involution, possibly the identity.

`OctonionAlgebra(K, a, b, c) : Fld, FldElt, FldElt, FldElt -> AlgGen`

`OctonionAlgebra(K, a, b, c) : Fld, RngIntElt, RngIntElt, RngIntElt -> AlgGen`

Octonion algebra with involution given by the specified parameters. This builds the Cayley-Dickson algebra over the quaternion algebra $\left(\frac{a,b}{K}\right)$. In particular, Magma's implementation of quaternion algebras is applied.

`SplitOctonionAlgebra(K) : Fld -> AlgGen`

Returns the split octonion algebra over the field F .

Example 5.3. Ten_Triality

The following example demonstrates some of the mechanics by exploring the concept of triality [S, III.8].

The Cartan-Jacobson theorem asserts that for fields of characteristic other than 2 and 3, the derivation algebra of an octonion algebra is of Lie type G_2 .

```
> O := OctonionAlgebra(GF(7), -1, -1, -1);
> L := DerivationAlgebra(O); // Derivations as an algebra.
> SemisimpleType(L);
G2
```

Cartan's triality obtains G_2 from D_4 by relaxing to derivations of the octonions as a generic tensor, rather than as an algebra. This is done computationally by changing the category of the octonion product from an algebra to a tensor.

```
> T := Tensor(O);
> T := ChangeTensorCategory(T, HomotopismCategory(2));
> M := DerivationAlgebra(T); // Derivations as a tensor.
> SemisimpleType(M/SolvableRadical(M));
D4
```

5.3. Jordan algebras

`JordanTripleProduct(J) : AlgGen -> TenSpcElt`

Returns the tensor describing the Jordan triple product.

`JordanSpinAlgebra(F) : TenSpcElt -> AlgGen`

`JordanSpinAlgebra(F) : Any -> AlgGen`

Returns the special Jordan algebra of spin type for given symmetric form.

Example 5.4. Ten_Jordan_Basic

Jordan algebras have suggestive analogues of commutative associative algebras, but experimenting shows serious differences.

```
> F := IdentityMatrix(Rationals(), 2);
> J := JordanSpinAlgebra(F);
> T := Tensor(J);
> R := AsMatrices(T, 2, 0);
> R[1]; // Is J.1 the identity?
[1 0 0]
[0 1 0]
[0 0 1]
> J.2*J.2 eq J.1; // J.2^2=1?
true
> J.2*J.3 eq 0; // Yet J.2 is a zero-divisor.
true
> e := (1/2)*(J.1+J.2);
> e^2 eq e; // An idempotent of J?
true
```

Pierce decompositions in Jordan algebras have the usual 0 and 1 eigenspaces but an additional $1/2$ -eigenspace emerges as well.

```

> Re := (1/2)*(R[1]+R[2]);
> Eigenvalues(Re);
{ <1, 1>, <1/2, 1>, <0, 1> }

```

`ExceptionalJordanCSA(O) : AlgGen -> AlgGen`

`ExceptionalJordanCSA(K) : Fld -> AlgGen`

The exception central simple Jordan algebra over the given octonions. If a field is supplied instead then the split octonion algebra over the field is used.

Example 5.5. Ten_Chevalley_Shafer_F4

In characteristic not 2 or 3, the exceptional central simple Jordan algebra can be used to construct the exceptional Lie algebra of type F_4 .

```

> J := ExceptionalJordanCSA(Rationals());
> T := Tensor(J);
> T := ChangeTensorCategory(T, HomotopismCategory(2));
> D := DerivationAlgebra(T);
> D2 := Induce(D, 2);           // Represent D on U2.
> F4 := D2*D2;                 // Commutator.
> SemisimpleType(F4);
F4
> F4;                          // F4 represented on a 27-dim module.
Matrix Lie Algebra of degree 27 over Rational Field

```

CHAPTER 6

Some examples

We include some examples to demonstrate some of the uses of this package.

6.1. Distinguishing groups

Example 6.1. Payne_Grps

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 [P].

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)
function>   return KSpace(GF(q),1)!(x[1]*Matrix(2,1,x[2]));
function> end function;
>
> DoubleForm := function(T)
function>   F := SystemOfForms(T)[1];
function>   K := BaseRing(F);
function>   n := Nrows(F);
function>   m := Ncols(F);
function>   MS := KMatrixSpace(K,n,m);
function>   Z := MS!0;
function>   M1 := HorizontalJoin(Z,-Transpose(F));
function>   M2 := HorizontalJoin(F,Z);
function>   D := VerticalJoin( M1, M2 );
function>   return Tensor( D, 2, 1 );
function> end function;
>
> f := DoubleForm( Tensor( F, DotProd ) );
> f;
Tensor of valence 2, 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 := HeisenbergGroup(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)
function>   k := Eltseq(v)[1];
function>   K := Parent(k);
function>   if k eq K!0 then return k; end if;
function>   R<x> := PolynomialRing(K);
function>   f := Factorization(x^(3^r)-k)[1][1];
function>   return K!(x-f);
function> end function;
>
> // biadditive map defining elation grp
> RomanGQ := function(x)
function>   u := Matrix(1, 2, x[1]);
function>   v := Matrix(2, 1, x[2]);
function>   M := [A, B, C];
function>   f := &+[Root(u*M[i]*v, i-1) : i in [1..3]];
function>   return KSpace(GF(q), 1)!f;
function> 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)
function>   x1 := Eltseq(x[1]@phi)[1..2];
function>   x2 := Eltseq(x[1]@phi)[3..4];
function>   y1 := Eltseq(x[2]@phi)[1..2];
function>   y2 := Eltseq(x[2]@phi)[3..4];
function>   comm := RomanGQ( <x2, y1> ) - RomanGQ( <y2, x1> );
function>   return comm @ gamma;
function> end function;
>
> f_bar := Tensor( F2, RomanGQComm );
> f_bar;
Tensor of valence 2, 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 := HeisenbergGroup(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 2, 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)
>
> Tfb := pCentralTensor(Gfb,1,1);
> Tfb;
Tensor of valence 2, 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)
>
> Cf := Centroid(Tf);
> Cfb := Centroid(Tfb);
> Dimension(Cf) eq Dimension(Cfb);
false

```

6.2. Simplifying automorphism group computations

Example 6.2. Ext_Over_Adj

We demonstrate how to simplify the automorphism group computation as discussed in [BW1]. 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
while>   N := sub< P | Random(Z), N >;

```

```

while> end while;
>
> G := P/N;
> G;
GrpPC : G of order 10246902931634286779441449 = 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 2, 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);
> 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 stablizer of $\text{GL}(8, 317)$ inside $V \wedge V$. We will decrease the size of the ambient space resulting in an easier stablizer 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
for>   F := SystemOfForms(E_gen)[1];
for>   for X in Basis(A) do

```



```

for|for>      L cat:= [E!Eltseq(X*F - F*Transpose(X@star))];
for|for>      end for;
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)

```


APPENDIX A

Cyclic algebras and their modules

Magma supports similarity testing of modules over cyclic associative rings and cyclic groups. Module similarity over general rings and groups is graph isomorphism hard. The algorithms here are based on [BW2].

`IsCyclic(R) : AlgAss -> BoolElt, AlgAssElt`

Decide if the algebra is generated by a single element, and return such a generator.

`IsSimilar(M, N) : ModRng, ModRng -> BoolElt, Map`

Decides if the given modules are similar; requires that one of the given modules have a cyclic coefficient ring.

Example A.1. Star_Alg

In magma modules of a group or algebra are defined by the action of a fixed generating set of the algebra. Therefore isomorphism of modules in Magma assumes the given modules have been specified by the same generating set. This can lead to a stricter interpretation of isomorphism than perhaps intended in some situations. Consider the following example comparing two 1-dimensional vector spaces over the field $GF(9)$.

```
> R := MatrixAlgebra(GF(3),2);
> A := sub<R| [R!1, R![0,1,2,0]]>;
> B := sub<R| [R!1, R![1,1,2,1]]>;
> A eq B;           // Both are a field GF(9).
true
> M := RModule(A); // A 1-dim. GF(9) vector space.
> N := RModule(B); // A 1-dim. GF(9) vector space.
> IsIsomorphic(M,N);
false
> MinimalPolynomial(A.2);
$.1^2 + 1
> MinimalPolynomial(B.2);
$.1^2 + $.1 + 2
```

Isomorphism of the two modules M and N failed because the two vector spaces are specified by different generators of $GF(9)$, as confirmed by the minimum polynomials of the generators. Module similarity allows the comparison of modules specified by different generating sets, so in this example theses to vector spaces can be proven equivalent.

```
> IsSimilar(M,N);
true
[2 0]
[0 2]
```

Similarity can be used to compare modules given by algebras that are conjugate, but perhaps not equal.

```
> p := RandomIrreduciblePolynomial(GF(101), 10);
> q := RandomIrreduciblePolynomial(GF(101), 10);
```

```

> X := CompanionMatrix(p);
> Y := CompanionMatrix(q);
> A := sub<Parent(X)|[X]>;          // Finite field of size 101^10
> B := sub<Parent(Y)|[Y]>;          // Finite field of size 101^10
> M := RModule(A);                  // 1-dim vector space over A.
> N := RModule(B);                  // 1-dim vector space over B.
> IsIsomorphic(M,N);                // Not isomorphic as A and B are distinct.
false
> cyc, f := IsSimilar(M,N);          // But similar as A is isomorphic to B.
> // f conjugates A into B?
> forall { a : a in Generators (A) | f * a * f^-1 in B };
true
> // and f is a semilinear transform M->N ?
> forall{ i : i in [1..Ngens (M)] | forall { j : j in [1..Ngens (A)] | (Vector\
\
> (M.i * A.j) * f) eq (Vector(M.i)*f)*(f^(-1)*A.j*f) } } };
true

```

Similarity is presently available for cyclic algebras. This can be tested and a generator returned.

```

> M := RandomMatrix(GF(9), 100, 100);
> A := sub< Parent(M) | [ M^(Random(50)) : i in [1..10]] >;
> Ngens(A);
> IsCyclic(A);
true
...

```

Bibliography

- [BCP] Wieb Bosma, John Cannon, and Catherine Playoust, *The Magma algebra system. I. The user language*, J. Symbolic Comput. **24** (1997), no. 3-4, 235–265. Computational algebra and number theory (London, 1993). MR1484478
- [BW1] Peter A. Brooksbank and James B. Wilson, *Groups acting on tensor products*, J. Pure Appl. Algebra **218** (2014), no. 3, 405–416. MR3124207
- [BW2] ———, *The module isomorphism problem reconsidered*, J. Algebra **421** (2015), 541–559. MR3272396
- [FMW] Uriya First, Joshua Maglione, and James B. Wilson, *Polynomial identity tensors and their invariants*. in preparation.
- [L1] J. M. Landsberg, *Tensors: geometry and applications*, Graduate Studies in Mathematics, vol. 128, American Mathematical Society, Providence, RI, 2012. MR2865915
- [L2] John M. Lee, *Introduction to smooth manifolds*, 2nd ed., Graduate Texts in Mathematics, vol. 218, Springer, New York, 2013. MR2954043
- [P] Stanley E. Payne, *Finite groups that admit Kantor families*, Finite geometries, groups, and computation, Walter de Gruyter GmbH & Co. KG, Berlin, 2006, pp. 191–202. MR2258010
- [M] A. I. Mal'cev, *Foundations of linear algebra*, Translated from the Russian by Thomas Craig Brown; edited by J. B. Roberts, W. H. Freeman & Co., San Francisco, Calif.-London, 1963. MR0166200
- [S] Richard D. Schafer, *An introduction to nonassociative algebras*, Pure and Applied Mathematics, Vol. 22, Academic Press, New York-London, 1966. MR0210757
- [W1] Herman Weyl, *The theory of groups and quantum mechanics*, Dover, New York, 1950.
- [W2] James B. Wilson, *Division, adjoints, and dualities of bilinear maps*, Comm. Algebra **41** (2013), no. 11, 3989–4008. MR3169502

Intrinsics

- *
 - as module, 21
 - as multilinear map, 30
 - bilinear space (infix), 32
 - bilinear tensor (infix), 32
 - bilinear tensor (product), 34
 - homotopism, 64
- +, 21
- - homotopisms, 64
 - tensor space, 52
- /
 - tensor, 60
 - tensor space, 62
- #, 53
- AdjointAlgebra, 39
- AdjointCategory, 58
- AlternatingSpace, 48
- AlternatingTensor, 18
- AnticommutatorTensor, 14
- AntisymmetricSpace, 48
- AntisymmetricTensor, 18
- Arrows, 59
- AsCotensorSpace, 37
- AsMatrices, 11
- AssociatedForm, 22
- AssociatorTensor, 16
- AsTensor, 37
- AsTensorSpace, 37
- AT, 29
- BANG
 - bilinear, 34
 - sequence, 51
 - tensor, 51
 - zero, 51
- BaseField
 - tensor, 24
 - tensor space, 54
- BaseRing
 - tensor, 24
 - tensor space, 54
- Basis, 52
- Center, 43
- Centre, 43
- Centroid
 - algebra, 43
 - tensor, 39
- ChangeTensorCategory
 - homotopism, 64
 - tensor, 25
 - tensor space, 54
- Codomain
 - homotopism, 64
 - tensor, 23
- CohomotopismCategory, 58
- CommutatorTensor, 14
- CompositionAlgebra, 70
- Compress, 22
- Coradical, 38
- CotensorCategory
 - constructor, 58
- CotensorSpace, 47
- DerivationAlgebra
 - algebra, 43
 - tensor, 40
- DerivationClosure, 49
- Dimension, 53
- Discriminant, 41
- Domain
 - homotopism, 64
 - tensor, 23
- Eltseq, 8
- eMAGmaVersion, 2
- eq
 - bilinear, 35
 - tensor, 30
 - tensor category, 59
 - tensor space, 51
- ExceptionalJordanCSA, 72
- ExteriorCotensorSpace, 48
- Foliation, 37
- Frame
 - tensor, 23
 - tensor space, 54
- FullyNondegenerateTensor, 26
- Generators, 52
- Generic, 54

- GenericMinimalPolynomial, 69
- GenericMinimumPolynomial, 69
- GenericNorm, 69
- GenericTrace, 69
- GenericTracelessSubspaceBasis, 69

- HeisenbergAlgebra, 42
- HeisenbergGroup, 42
- HeisenbergGroupPC, 42
- HeisenbergLieAlgebra, 42
- Homotopism, 63
- HomotopismCategory, 58

- Ideal, 60
- Image
 - homotopism, 65
 - tensor, 26
- in, 51
- Induce, 38
- InducedTensor, 36
- IsAlternating
 - tensor, 28
 - tensor space, 54
- IsAntisymmetric
 - tensor, 28
 - tensor space, 54
- IsCoercible
 - bilinear, 34
 - sequence, 51
 - tensor, 51
 - zero, 51
- IsContravariant
 - tensor, 25
 - tensor category, 59
 - tensor space, 54
- IsCovariant
 - tensor, 25
 - tensor category, 59
 - tensor space, 54
- IsCyclic, 79
- IsFullyNondegenerate, 26
- IsIdeal, 60
- IsLocalIdeal, 60
- IsNondegenerate, 25
- IsSimilar, 79
- IsStarAlgebra, 69
- IsSubtensor, 60
- IsSubtensorSpace, 62
- IsSymmetric
 - tensor, 28
 - tensor space, 54

- JordanSpinAlgebra, 71
- JordanTripleProduct, 71

- KCotensorSpace, 47
- Kernel, 65
- KTensorSpace, 45

- LeftDomain, 34

- LeftNucleus
 - algebra, 43
 - bilinear, 39
- LinearCategory, 58
- LocalIdeal, 60
- LocalQuotient, 60

- Maps, 64
- MidNucleus
 - algebra, 43
 - bilinear, 39

- Ngens, 52
- NondegenerateTensor, 25
- Nucleus, 40
- NucleusClosure, 49
- NumberOfGenerators, 52

- OctonionAlgebra, 70

- Parent
 - bilinear, 35
 - tensor, 23
- pCentralTensor, 16
- Pfaffian, 41
- Polarisation, 17
- Polarization, 17

- quo
 - tensor space, 62
- QuoConstructor
 - tensor space, 62
- Quotient
 - tensor, 60

- Radical, 38
- Random, 53
- RandomCotensor, 53
- RandomTensor, 53
- RepeatPartition, 59
- RightDomain, 34
- RightNucleus
 - algebra, 43
 - bilinear, 39
- RTensorSpace, 45

- SetVerbose, 1
- Shuffle, 19
- Slice, 36
- SliceAsMatrices, 37
- SplitOctonionAlgebra, 70
- Star, 69
- StructureConstants, 8
- sub
 - tensor space, 62
- SubConstructor
 - tensor space, 62
- subset, 51
- Subtensor, 60
- SymmetricCotensorSpace, 48
- SymmetricSpace, 48

SymmetricTensor, 18

SystemOfForms, 11

Tensor

algebra, 12

bilinear, 10

black-box, 3, 5

forms, 10

polynomial ring, 12

structure constants, 7

TensorCategory

constructor, 58

homotopism, 64

tensor, 25

tensor space, 54

TensorOverCentroid, 41

TensorSpace, 45

signed, 45

UniversalCotensorSpace, 54

UniversalTensorSpace, 54

Valence

tensor, 23

tensor category, 59

tensor space, 54