TensorSpace Package

Joshua Maglione

Universität Bielefeld
jmaglione@math.uni-bielefeld.de

James B. Wilson

Colorado State University
james.wilson@colostate.edu

Version 2.3
May 11, 2020

Contents

CHAPTER 1

Introduction

This documentation describes Magma operations with multilinear algebra. Our notation follows [**FMW**] closely and can be viewed as companion literature. 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.

We follow the types described in [**FMW**, Section 7] to the degree feasible within the Magma type-system.

Citing TensorSpace. To cite the TensorSpace package, please use the following

Joshua Maglione and James B. Wilson, *TensorSpace*, version 2.3, GitHub, 2020. Contributions from Peter A. Brooksbank, https://github.com/algeboy/TensorSpace.

For AMSRefs:

```
\bib{TensorSpace}{misc}{
   author={Maglione, Joshua},
   author={Wilson, James B.},
   title={TensorSpace},
   publisher={GitHub},
   year={2020},
   edition={version 2.3},
   note={Contributions from Peter A. Brooksbank,
    \texttt{https://github.com/algeboy/TensorSpace}},
}
```

1.1. Overview

We set up some notation used throughout. Define K to be a commutative ring (typically a field), $\mathtt{v}$ to be a nonnegative integer (in some literature this is denoted by the Hebrew letter ו), and a sequence of right K-modules $U_0, \ldots, U_{\mathtt{v}}$. Set $[\![\mathtt{v}]\!] = \{0, \ldots, \mathtt{v}\}$, resp. $[\mathtt{v}] = \{1, \ldots, \mathtt{v}\}$. For a set R and $A \subset R$, $\bar{A} = R - A$, assuming R is fixed in the context.

Abbreviate

$$U_0 \oslash U_1 = \hom_K(U_1, U_0) \qquad\qquad U_0 \oslash \cdots \oslash U_{\mathtt{v}} = (U_0 \oslash \cdots \oslash U_{\mathtt{v}-1}) \oslash U_{\mathtt{v}}.$$

Also write $\oslash_{a\in[\mathtt{v}]} U_a = U_0 \oslash \cdots \oslash U_{\mathtt{v}}$. So $U_0 \oslash U_1$ denotes the space of *linear maps* $U_1 \to U_0$, which we evaluate with the the notation $\langle t|u_1\rangle$ fpr $u_1 \in U_1$. Likewise, $U_0 \oslash U_1 \oslash U_2$ denotes the space of *bilinear maps* $\bar{t} : U_2 \to (U_1 \to U_0)$ evaluated on inputs $u_2 \in U_2$ as

$$\langle t|u_2\rangle \in U_0 \oslash U_1.$$

Often the intent is to then evaluate that result on some $u_1 \in U_1$ for which we may write

$$\langle t|u_2, u_1\rangle := \langle t|u_2\rangle|u_1\rangle.$$

So we may treat $\langle t|$ as function $U_2 \times U_1 \rightarrowtail U_0$ — the symbol $\rightarrowtail$ replaces $\to$ to distinguish *bi*-linear from simply linear. More generally, $\langle t| \in U_0 \oslash \cdots \oslash U_{\mathtt{v}}$ is a $\mathtt{v}$-multilinear map (of valence $\mathtt{v}+1$) and can be evaluated on $u : (a \in [\mathtt{v}]) \to (u_a \in U_a)$ as

$$\langle t|u_{\mathtt{v}}, \ldots, u_1\rangle = \langle t|u_{\mathtt{v}}\rangle \cdots |u_1\rangle.$$

We also write $\langle t| : U_{\mathtt{v}} \times \cdots \times U_1 \rightarrowtail U_0$, where $\rightarrowtail$ denotes *multilinear maps*.

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

$$\langle\cdot| : T \hookrightarrow U_0 \oslash \cdots \oslash U_{\mathtt{v}}.$$

Tensors are elements of T, $(U_{\mathtt{v}}, \ldots, U_0)$ is the *frame*, and $\mathtt{v}+1$ is the valence. In Dirac styled $\langle t|$ "bra t" and $|u\rangle$ "ket u" notation, we evaluate $\langle t| \in U_0 \oslash \cdots \oslash U_{\mathtt{v}}$ on inputs $|u\rangle = |u_{\mathtt{v}}, \ldots, u_1\rangle \in U_{\mathtt{v}} \times \cdots \times U_1$ denote $\langle t|u\rangle$ or $\langle t|u_{\mathtt{v}}, \ldots, u_1\rangle$. When regarding $\bar{t}$ as a function on $U_{\mathtt{v}} \times \cdots \times U_1$ into U_0 we use $\rightarrowtail$ for the arrow emphasize $\bar{t} : U_{\mathtt{v}} \times \cdots \times U_1 \rightarrowtail U_0$ is *multilinear*.

Every tensor in Magma is treated as an element of a tensor space. By default a universal tensor space:

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

In Magma, the tensor space is what determines the associated multilinear function of a given tensor T. Evaluation of T mimics a map $U_{\mathtt{v}} \times \cdots \times U_1 \to U_0$, for instance

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

Special attention is given to bilinear maps $* : U_2 \times U_1 \rightarrowtail U_0$ including the ability to use infix notation $\mathtt{u}_2 * \mathtt{u}_1$. 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 $K^d \to K^d$, another assigns the same matrix to a bilinear form $K^d \times K^d \rightarrowtail K$. Functors are provided to change tensor categories and to define standard categories.

1.2. Printing and Version

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

```
SetVerbose(MonStgElt, RngIntElt) : ->
```

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

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

Example 1.1. `VerbosePrinting`

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

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

```
Sprint(t) : TenSpcElt -> MonStgElt
Sprint(T) : TenSpc -> MonStgElt
Sprint(C) : TenCat -> MonStgElt
```

Returns strings that can be executed in Magma to construct the given object. If t is a black-box tensor where structure constants cannot be computed, then the returned string is equivalent to `print t`, which cannot be executed to construct t.

Example 1.2. `PrintToString`

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

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

Now we will print to string each of the objects we created.

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

`TensorSpaceVersion() : -> MonStgElt`

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

Example 1.3. `Version`

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

```
> TensorSpaceVersion();
2.2
```

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 $[U_{\mathtt{v}}, \ldots, U_0]$ of K-modules called the *frame*.
- A function $U_{\mathtt{v}} \times \cdots \times U_1 \rightarrowtail U_0$ that is K-linear in each U_i.

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

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

The left hand module is 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 may be specified together with 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

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. Note that t is a tensor over vector spaces—essentially forgetting all other structure. The last entry of `S` is assumed to be the codomain of the multilinear map. The user-defined function F should take as input a tuple of elements of the domain and return an element of the codomain. If no tensor category is provided, the homotopism category is used.

Example 2.1. `BBTensorsFrame`

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

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

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

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

We will provide a tensor category for the dot product tensor, so that the returned tensor is not in the default homotopism category. We will use instead the $\{2, 1\}$-adjoint category. While the returned tensor prints out the same as above, it does indeed live in a different 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 $\times : \mathbb{R}^3 \times \mathbb{R}^3 \rightarrowtail \mathbb{R}^3$ and verify that $\mathbf{i} \times \mathbf{j} = \mathbf{k}$. However, to do this test, we will input integer sequences (specifically `[RngIntElt]`), and we will still be able to evaluate.

```
> K := RealField(5);
> V := VectorSpace(K, 3);
> CP := function(x)
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. Note that t is a tensor over vector spaces—essentially forgetting all other structure. The user-defined function F should take as input a tuple of elements of D and return an element of C. If no tensor category is provided, then the 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 $\langle\,\rangle : \mathbb{M}_{2\times3}(K) \times \mathbb{M}_{2\times3}(K) \times \mathbb{M}_{2\times3}(K) \rightarrowtail \mathbb{M}_{2\times3}(K)$, given by $\langle A, B, C\rangle = AB^tC$.

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

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

```
> A := U![1,0,0,0,0,0];
> A;
[  1   0   0]
[  0   0   0]
>
```

```
> <A,A,A> @ t;  // A is a generalized idempotent
(  1   0   0   0   0   0)
```

We can experiment to see if this triple product is left associative. To do this, we will construct five random matrices $\{X_1, \ldots, X_5\} \subset \mathbb{M}_{2\times 3}(K)$, 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 $\langle\!\langle\cdot\rangle\!\rangle : \prod_{k=1}^{5} \mathbb{M}_{2\times 3}(K) \rightarrowtail \mathbb{M}_{2\times 3}(K)$ where

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

Therefore, if $\mathrm{im}(\langle\!\langle\cdot\rangle\!\rangle) = 0$, then $\langle\cdot\rangle$ 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, e.g. in the homotopism category, $t^{j_0}_{j_{\mathtt{v}}\cdots j_1} \in K$. Here t is framed by free K-modules $[U_{\mathtt{v}}, \ldots, U_0]$ with each U_i having an ordered bases $\mathcal{B}_i = [e_{i1}, \ldots, e_{id_i}]$. The interpretation of structure constants is that the associated multilinear function $[x_{\mathtt{v}}, \ldots, x_1]$ from $U_{\mathtt{v}} \times \cdots \times U_1$ into U_0 is determined on bases as follows:

$$[e_{\mathtt{v}\, j_{\mathtt{v}}}, \ldots, e_{1j_1}] = \sum_{k=1}^{d_0} t^{j_0}_{j_{\mathtt{v}}\cdots j_1} e_{0k}.$$

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

$$f(j_{\mathtt{v}}, \ldots, j_0) = 1 + \sum_{k=0}^{\mathtt{v}} (j_k - 1) \prod_{\ell=0}^{k-1} d_\ell.$$

So $S[f(j_{\mathtt{v}}, \ldots, j_0)] = t^{j_0}_{j_{\mathtt{v}}\cdots j_1}$ 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 $T^{b_q\cdots b_1}_{a_p\cdots a_1}$ should be input as $T_{a_{p+q}\cdots a_{1+q} b_q \cdots b_1}$ and the tensor category changed to mark $\{q, \ldots, 1\}$ as contravariant. Intrinsics are provided to facilitate this approach. See Chapter 4 for more details on tensor categories.

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

Given dimensions $D = [d_{\mathtt{v}}, \ldots, d_0]$, returns the tensor in $R^{d_{\mathtt{v}}\cdots d_0}$ identified by structure constant sequence S. If R is not provided, then the parent ring of the first element of S is used. The ring R must be commutative and unital. 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: GF(64).

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

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

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

`StructureConstants(t) : TenSpcElt -> SeqEnum`
`Eltseq(t) : TenSpcElt -> SeqEnum`

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

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

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

Example 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 TensorSpace. 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 $\mathbb{M}_{20}(\mathbb{F}_3)$, namely $* : \mathfrak{sl}_{20}(\mathbb{F}_3) \times \mathbb{M}_4(\mathbb{F}_3) \rightarrowtail \mathbb{M}_{20}(\mathbb{F}_3)$.

```
> 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 $U_2 \times U_1 \rightarrowtail U_0$ is to format the data as lists of matrices $[M_1, \ldots, M_d]$. This can be considered as a left (resp. right) representation $U_2 \to \hom_K(U_1, U_0)$, (resp. $U_1 \to \hom_K(U_2, U_0)$). Or it can be treated as *systems of bilinear forms* $[M_1, \ldots, M_d]$ where the matrices are the Gram matrices of bilinear forms $\phi_i : U_2 \times U_1 \rightarrowtail K$. Here the associated bilinear map $U_2 \times U_1 \rightarrowtail U_0$ is specified by

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

```
Tensor(M, a, b) : Mtrx, RngIntElt, RngIntElt -> TenSpcElt
Tensor(M, a, b, Cat) : Mtrx, RngIntElt, RngIntElt, TenCat -> TenSpcElt
Tensor(M, a, b) : [Mtrx], RngIntElt, RngIntElt -> TenSpcElt
Tensor(M, a, b, Cat) : [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 a and b which must be positions in $\{2, 1, 0\}$. Optionally a tensor category Cat 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 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, a, b) : TenSpcElt, RngIntElt, RngIntElt -> SeqEnum`
`SystemOfForms(t) : TenSpcElt -> SeqEnum`

For a tensor t with frame $[K^{d_{\mathrm{v}}}, \ldots, K^{d_0}]$, returns a list $[M_1, \ldots, M_d]$, $d = (d_{\mathrm{v}} \cdots d_0)/d_a d_b$, of $(d_a \times d_b)$-matrices in K representing the tensor as an element of $\hom_K(K^{d_a} \otimes_K K^{d_b}, K^d)$. For `SystemOfForms`, t must have valence 3 and the implied values are $a = 2$ and $b = 1$.

Example 2.8. `TrilinearAsMats`

We construct the associator of $\mathfrak{sl}_2(\mathbb{Q})$ where $\langle\,\rangle : \prod_{k=1}^{3} \mathfrak{sl}_2 \rightarrowtail \mathfrak{sl}_2$ 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 and presents the data as a sequence of matrices.

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

Calling `AsMatrices(t, 3, 1)` returns a sequence of nine matrices, but we only show the first four. As explained in the documentation above, this sequence of matrices can be interpreted as the system of bilinear forms for the 3-tensor $\circ_{31} : V_3 \times V_1 \rightarrowtail \hom_K(V_2, V_0)$ given by

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

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

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

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

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

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 $* : A \times A \rightarrowtail A$ is given by multiplication in A. Like with the previous sections on tensor constructions, all tensors will be constructed over vector spaces. The user can still input elements from the original algebra, but map(s) will also be returned. Furthermore, each tensor is assigned a category relevant to its origin, see Chapter 4 for more details on tensor categories.

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

Returns the bilinear tensor given by the product in algebra 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, $\Omega = \mathrm{End}(U_2)$ instead of $\Omega = \mathrm{End}(U_2) \times \mathrm{End}(U_1) \times \mathrm{End}(U_0)$.

```
> 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, $\mathrm{Rad}(D) = K^2$.

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

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

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

Example 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 $\mathbb{M}_4(\mathbb{Q})$.

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

With this tensor, we will compute the dimension of the centralizer of the diagonal matrix M with diagonal entries $(1, 1, -1, -1)$ in $\mathbb{M}_4(\mathbb{Q})$. 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> @ t);
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 $* : \mathbb{M}_4(\mathbb{Q}) \times \mathbb{M}_4(\mathbb{Q}) \rightarrowtail \mathbb{M}_4(\mathbb{Q})$ where $A * B = \frac{1}{2}(AB + BA)$.

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

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

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

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

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

```
> IsSymmetric(s);
true
> JordanID := func< x, y | (x*s*y)*s*(x*s*x) - x*s*(y*s*(x*s*x)) >;
> forall{ <x,y> : x in Basis(A), y in Basis(A) | \
>     JordanIdentity(x, y) eq Codomain(s)!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 O!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, a, b) : Grp, RngIntElt, RngIntElt, RngIntElt -> TenSpcElt, List
pCentralTensor(G, a, b) : Grp, RngIntElt, RngIntElt -> TenSpcElt, List
pCentralTensor(G, a, b) : GrpPC, 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_a/η_{a+1} with η_b/η_{b+1} into η_{a+b}/η_{a+b+1}. If $a = b$ 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, a, and b are not given, it is assumed G is a p-group and $a = b = 1$.

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 $G = \mathrm{SL}(3, 125)$; however, we will lose the fact that there is a field $\mathbb{F}_{125}$. The tensor we will get back is $[,] : K^6 \times K^6 \rightarrowtail K^3$, 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 $\cdot : \mathbb{F}_{125}^2 \times \mathbb{F}_{125}^2 \rightarrowtail \mathbb{F}_{125}$. 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)
```

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

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

$$\mathbb{M}_{s_1 \times s_2}(K) \times \cdots \times \mathbb{M}_{s_{\mathrm{v}-1} \times s_{\mathrm{v}}}(K) \rightarrowtail \mathbb{M}_{s_1 \times s_{\mathrm{v}}}(K),$$

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

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

Returns the polarization of the homogeneous multivariate polynomial (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 $f(x, y) = x^2 y$. 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 $K^2 \times K^2 \times K^2 \rightarrowtail K$. The polarization of f is given by $P(x_1, x_2, y_1, y_2, z_1, z_2) = 2(x_1 y_1 z_2 + x_1 y_2 z_1 + x_2 y_1 z_1)$.

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

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 $\mathbb{M}_3(\mathbb{Q})$ 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]
```

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]
```

Shuffle(t, g) : TenSpcElt, GrpPermElt -> TenSpcElt
Shuffle(t, g) : TenSpcElt, SeqEnum -> TenSpcElt

For a tensor t in $\hom(U_{\mathtt{v}}, \dots, \hom(U_1, U_0) \cdots)$, generates a representation of t in

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

In order to be defined, $g \in \mathrm{Sym}(\{0, \dots, \mathtt{v}\})$. 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, $g \in \mathrm{Sym}(\{1, \dots, \mathtt{v}\})$. Sequences $[a_1, \dots, a_{\mathtt{v}+1}]$ will be interpreted as

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

Example 2.18. ShuffleToTranspose

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

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

Example 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)
```

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

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

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 frame $U_{\mathrm{v}} \times \cdots \times U_1 \rightarrowtail U_0$, creates the associated multilinear form $U_{\mathrm{v}} \times \cdots \times U_1 \times U_0^* \rightarrowtail K$. The valence is increased by 1.

`Compress(t) : TenSpcElt -> TenSpcElt`
`Compress(~t) : TenSpcElt ->`

Returns the compression of the tensor. 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)
> s := AssociatedForm(t);
> s;
Tensor of valence 4, U3 x U2 x U1 >-> U0
U3 : Full Vector space of degree 21 over GF(5)
U2 : Full Vector space of degree 21 over GF(5)
U1 : Full Vector space of degree 21 over GF(5)
U0 : Full Vector space of degree 1 over GF(5)
> <L.2, L.11> @ t;
(0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
> <L.2, L.11, L.2> @ s;
(4)
> <L.2, L.11, L> @ s;
Full Vector space of degree 1 over GF(5)
Generators:
(4)
```

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

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

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 $* : \mathbb{M}_{2\times 3}(\mathbb{Q}) \times \mathbb{Q}^3 \rightarrowtail \mathbb{Q}^2$ given by multiplication. Nearly all of this information is displayed when printing a tensor.

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

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

    Full Vector space of degree 3 over Rational Field,

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

`TensorCategory(t) : TenSpcElt -> TenCat`

Returns the underlying tensor category of 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, $* : \mathbb{Q}^2 \times \mathbb{M}_{2\times 2}(\mathbb{Q}) \rightarrowtail \mathbb{Q}^2$.

```
> K := Rationals();
> U := KMatrixSpace(K, 2, 2);
> V := VectorSpace(K, 2);
> mult := func< x | x[1]*x[2] >;
> t := Tensor([* V, U, V *], mult);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 2 over Rational Field
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
```

Because this tensor comes from a module, we want the tensor category to reflect this. Currently, the tensor category is the default homotopism category. We will keep everything about the category the same, except we will fuse coordinates 2 and 0. These changes could easily go unnoticed if no operators are constructed.

```
> TensorCategory(t);
Tensor category of valence 3 (->,->,->) ({ 1 },{ 2 },{ 0 })
> Cat := TensorCategory([1, 1, 1], {{2,0},{1}});
> Cat;
Tensor category of valence 3 (->,->,->) ({ 1 },{ 0, 2 })
> ChangeTensorCategory(~t, Cat);
> TensorCategory(t);
Tensor category of valence 3 (->,->,->) ({ 1 },{ 0, 2 })
> IsCovariant(t);
true
```

`NondegenerateTensor(t) : TenSpcElt -> TenSpcElt, Hmtp`

Returns the nondegenerate tensor associated to t along with a homotopism from the given tensor to the returned nondegenerate tensor.

`IsNondegenerate(t) : TenSpcElt -> BoolElt`

Decides whether t is a nondegenerate tensor.

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 $U_{\widehat{2}}^{\perp}$ and $U_{\widehat{1}}^{\perp}$ are nontrivial. We will construct the associated nondegenerate tensor s, which is given by $U_2/U_{\widehat{2}}^{\perp} \times U_1/U_{\widehat{1}}^{\perp} \rightarrowtail U_0$.

```
> IsNondegenerate(t);
false
> s, H := NondegenerateTensor(t);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 9 over GF(541)
U1 : Full Vector space of degree 3 over GF(541)
U0 : Full Vector space of degree 5 over GF(541)
> H;
Maps from U2 x U1 >-> U0 to V2 x V1 >-> V0.
U2 -> V2: Mapping from: Full Vector space of degree 10 over GF(541) to
Full Vector space of degree 9 over GF(541)
U1 -> V1: Mapping from: Full Vector space of degree 5 over GF(541) to
Full Vector space of degree 3 over GF(541)
U0 -> V0: Mapping from: Full Vector space of degree 5 over GF(541) to
Full Vector space of degree 5 over GF(541)
```

`Image(t) : TenSpcElt -> ModTupRng`

Returns the image of the tensor along with a map to the vector space.

`FullyNondegenerateTensor(t) : TenSpcElt -> TenSpcElt, Hmtp`

Returns the fully nondegenerate tensor 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 tensor.

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 form the previous example is not fully nondegenerate as it is not degenerate, but we check that the image is not isomorphic to the codomain.

```
> IsFullyNondegenerate(t);
false
> Image(t);
Vector space of degree 5, dimension 3 over GF(541)
Generators:
(  1   0   0   0   0)
(  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: $U_2/U_{\hat{2}}^{\perp} \times U_1/U_{\hat{1}}^{\perp} \rightarrowtail U_2 * U_1$. Notice that the morphism between the original tensor and the fully nondegenerate tensor is a cohomotopism.

```
> s, H := FullyNondegenerateTensor(t);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 9 over GF(541)
U1 : Full Vector space of degree 3 over GF(541)
U0 : Vector space of degree 5, dimension 3 over GF(541)
Generators:
(  1   0   0   0   0)
(  0   1   0   0   0)
(  0   0   1   0   0)
Echelonized basis:
(  1   0   0   0   0)
(  0   1   0   0   0)
(  0   0   1   0   0)
> H;
Maps from U2 x U1 >-> U0 to V2 x V1 >-> V0.
U2 -> V2: Mapping from: Full Vector space of degree 10 over GF(541) to
Full Vector space of degree 9 over GF(541)
U1 -> V1: Mapping from: Full Vector space of degree 5 over GF(541) to
Full Vector space of degree 3 over GF(541)
U0 <- V0: Mapping from: Full Vector space of degree 5 over GF(541) to
Full Vector space of degree 5 over GF(541)
Composition of Mapping from: Full Vector space of degree 5 over GF(541)
to Full Vector space of degree 5 over GF(541) and
Mapping from: Vector space of degree 5, dimension 3 over GF(541) to Full
Vector space of degree 5 over GF(541)
```

`IsAlternating(t) : TenSpcElt -> BoolElt`

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 anti-symmetric.

```
> IsAlternating(t);
false
> IsAntisymmetric(t);
false
```

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

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

Evaluates the tensor t at $x \in U_{\mathtt{v}} \times \cdots \times U_1$. The entries can be elements from the vector space U_i or sequences that Magma can naturally coerce into the vector space U_i. In some circumstances, tensors come from algebraic objects (e.g. algebras), and in these cases the entries of x can be contained in the original algebraic object as well.

Example 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.

`t eq s : TenSpcElt, TenSpcElt -> BoolElt`

Decides if the tensors t and s 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 $\phi : A \to V$ is a vector space isomorphism. If we compose t with ϕ^{-1}, the returned tensor is *still* over vector spaces. The codomain is *not* A; this is because all tensors are over vector spaces. In fact, the returned tensor is exactly the same as t.

```
> phi := maps[1];
> t * (phi^-1);
Tensor of valence 4, U3 x U2 x U1 >-> U0
U3 : Full Vector space of degree 9 over GF(3)
U2 : Full Vector space of degree 9 over GF(3)
U1 : Full Vector space of degree 9 over GF(3)
```

```
U0 : Full Vector space of degree 9 over GF(3)
> t * (phi^-1) eq t;
true
```

Let $\mathcal{E} \subset A$ be an orthogonal frame—a set of primitive, orthogonal, idempotents. We will compose t with the linear transformation

$$a \mapsto \sum_{e \in \mathcal{E}} eae.$$

Because the codomain of t is a vector space, we will pre-compose this map by ϕ^{-1}.

```
> E := [A.1, A.2^-1*A.1*A.2, A.2^-2*A.1*A.2^2];
> E;
[
    [1 0 0]
    [0 0 0]
    [0 0 0],

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

    [0 0 0]
    [0 0 0]
    [0 0 1]
]
> f := map< A -> A | x :-> &+[ E[i]*x*E[i] : i in [1..3] ] >;
> f;
Mapping from: AlgMat: A to AlgMat: A given by a rule [no inverse]
> s := t*(phi^-1*f);
> s;
Tensor of valence 4, U3 x U2 x U1 >-> U0
U3 : Full Vector space of degree 9 over GF(3)
U2 : Full Vector space of degree 9 over GF(3)
U1 : Full Vector space of degree 9 over GF(3)
U0 : Full Vector space of degree 9 over GF(3)
> s eq t;
false
```

We can also wrap f as a 2-tensor and compose it with t.

```
> g := Tensor([A, A], func< x | x[1]@f >);
> g;
Tensor of valence 2, U1 >-> U0
U1 : Full Vector space of degree 9 over GF(3)
U0 : Full Vector space of degree 9 over GF(3)
> t * g eq s;
true
```

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 $x * t * y$ notation for a 3-tensor t, and a product $x * y$ notation for the evaluation of bilinear tensors. For the latter, We do this by creating special types `BmpU[Elt]`, `BmpV[Elt]`, and `BmpW[Elt]` for the frame of a bilinear tensor. For bilinear maps, we refer to the modules in the frame as $U \times V \rightarrowtail W$.

```
x * t : Any, TenSpcElt -> Any
t * y : Any, TenSpcElt -> Any
```

Given a bilinear tensor t framed by $[U, V, W]$, $x * t$ returns the action on the right as a linear map $L : V \to W$ given by $vL = x * v$ if x is an element of U. If x is a subspace of U, then this returns a subspace of the tensor space T with frame $V \rightarrowtail W$. For the left action use $t * y$ instead. If t is valence 1, then the image of either x or y is returned. Therefore, the possible outputs are a tensor space `TenSpc`, a tensor `TenSpcElt`, or a vector `ModTupFld`.

Related to this intrinsic is the following: using tensor spaces with the infix notation.

```
x * T : Any, TenSpc -> Any
T * y : Any, TenSpc -> Any
```

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

Example 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 $M = \left(\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right)$, we will use the infix notation to construct the 2-tensor $* : A \rightarrowtail A$, 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 T. Because that tensor space came from t, which originally came from algebraic objects, we can use the returned tensor space to evaluate M. An arbitrary tensor space, however, would not evaluate M unless it is an appropriate vector.

```
> T := VectorSpace(Rationals(), 4)*t;
> T;
Tensor space of dimension 4 over Rational Field with valence 2
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 4 over Rational Field
> T*M;
Vector space of degree 4, dimension 2 over Rational Field
Generators:
(0 1 0 0)
(0 0 0 1)
Echelonized basis:
(0 1 0 0)
(0 0 0 1)
```

The next style of notation we support is the product notation, $x * y$. In order to use this style, the user needs to coerce both x and y into the `LeftDomain` and `RightDomain` respectively.

```
x * y : BmpUElt, BmpVElt -> Any
x * y : BmpU, BmpV -> Any
x * y : BmpUElt, BmpV -> Any
x * y : BmpU, BmpVElt -> Any
```

If x and y are associated to the bilinear map t, these operations return `<x,y> @ t`.

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

Returns the left domain, U, of t framed by $[U, V, W]$, setup for use with infix notation.

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

Returns the right domain, V, of t framed by $[U, V, W]$, setup for use with infix notation.

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

Decides if x can be coerced into U or V, 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 $[,] : U \times V \rightarrowtail W$ is the tensor given by commutation where $U = V = G/\eta_2$ and $W = \eta_2/\eta_3$, where η_i denotes the ith term of the exponent-p central series of G.

```
> G := SmallGroup(512, 10^6);
> t := pCentralTensor(G);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 5 over GF(2)
U1 : Full Vector space of degree 5 over GF(2)
U0 : Full Vector space of degree 4 over GF(2)
> U := LeftDomain(t);
> V := RightDomain(t);
> U;
Bimap space U: Full Vector space of degree 5 over GF(2)
```

```
> V;
Bimap space V: Full Vector space of degree 5 over GF(2)
```

Like with the other styles of notation (infix and tuple), users can evaluate elements from the original algebraic object, vectors from the vector spaces, and even sequences that Magma can easily coerce into vector spaces. To use the product notation, coerce elements into the `LeftDomain` and `RightDomain`.

```
> x := U!(G.1*G.2*G.4);
> y := V![1,0,0,0,0];
> x;
Bimap element of U: (1 1 0 1 0)
> y;
Bimap element of V: (1 0 0 0 0)
> x*y;
(1 0 0 1)
```

We can take this further and evaluate subspaces of U or V.

```
> H := sub< G | G.2,G.4 >;
> U!H * V!G.1;
Vector space of degree 4, dimension 2 over GF(2)
Generators:
(0 0 0 1)
(1 0 0 0)
Echelonized basis:
(1 0 0 0)
(0 0 0 1)
> U!H * V;
Vector space of degree 4, dimension 3 over GF(2)
Generators:
(1 0 0 0)
(0 0 1 0)
(0 0 0 1)
(1 0 0 0)
(1 0 0 0)
Echelonized basis:
(1 0 0 0)
(0 0 1 0)
(0 0 0 1)
```

```
Parent(x) : BmpUElt -> BmpU
Parent(x) : BmpVElt -> BmpV
```

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 t framed by free modules $[U_{\mathtt{v}}, \ldots, U_0]$ with $d_i = \dim U_i$, a `grid` is a sequence $[G_{\mathtt{v}}, \ldots, G_0]$ of subsets $G_i \subseteq \{1, \ldots, d_i\} \cup \{-d_i, \ldots, -1\}$. If an entry $g \in G_i$ is negative, it will be taken to mean $d_i + g + 1$, so -1 would be equivalent to d_i. The slice is the list of entries in the structure constants of the tensor indexed by $G_{\mathtt{v}} \times \cdots \times G_0$. `Slice` returns the structure constants whereas `InducedTensor` produces a tensor with these structure constants.

Example 2.32. `TensorSlicing`

We will construct a tensor $* : \mathbb{Q}^4 \times \mathbb{Q}^3 \rightarrowtail \mathbb{Q}^2$ with a structure constants sequence equal to $[1, \ldots, 24]$. If every $G_i = \{1, \ldots, d_i\}$, then the result is the same as `Eltseq`.

```
> U := VectorSpace(Rationals(),4);
> V := VectorSpace(Rationals(),3);
> W := VectorSpace(Rationals(),2);
> T := TensorSpace([U, V, W]);
> t := T![1..24];
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
```

```
> Slice(t, [{1..4},{1..3},{1..2}]) eq Eltseq(t);
true
```

Now we will slice with the following grid $[\{1,\ldots,4\},\{2\},\{1\}]$. Compare this with the product $U * v_2$.

```
> [ U.i*t*V.2 : i in [1..4]];
[
    (3 4),
    ( 9 10),
    (15 16),
    (21 22)
]
> Slice(t, [{1..4},{2},{1}]);
[ 3, 9, 15, 21 ]
```

If, instead, we slice W at its last basis vector, we get the following.

```
> Slice(t, [{1..4},{2},{-1}]);
[ 4, 10, 16, 22 ]
```

Notice that if we use -1 and 2 in the last set of the grid we get the same output we got above.

```
> Slice(t, [{1..4},{2},{-1,2}]);
[ 4, 10, 16, 22 ]
```

Now we will compare `Slice` and `InducedTensor`. `InducedTensor` is basically a `Tensor` wrapping the `Slice` function.

```
> s := InducedTensor(t, [{1..4}, {2}, {1,2}]);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 1 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
> s2 := Tensor([4, 1, 2], Slice(t, [{1..4}, {2}, {1,2}]));
> s2;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 1 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
>
> s eq s2;
true
> Eltseq(s);
[ 3, 4, 9, 10, 15, 16, 21, 22 ]
```

`SliceAsMatrices(t, grid, a, b) : TenSpcElt, [SetEnum], RngIntElt, RngIntElt -> SeqEnum`

Returns a sequence of matrices whose output is equivalent to composing `InducedTensor` and `AsMatrices`. This intrinsic will be slightly faster than actually composing those two functions together as a tensor is not constructed with `SliceAsMatrices`.

Example 2.33. `SliceAsMatrices`

We will create the same tensor as the previous example.

```
> t := Tensor(Rationals(), [4,3,2], [1..24]);
> t;
Tensor of valence 3, U2 x U1 >-> U0
```

```
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
> AsMatrices(t, 1, 0);
[
    [1 2]
    [3 4]
    [5 6],

    [ 7  8]
    [ 9 10]
    [11 12],

    [13 14]
    [15 16]
    [17 18],

    [19 20]
    [21 22]
    [23 24]
]
```

Now we will slice up this sequence of matrices. We will remove the second row and the second and third matrix from the sequence above.

```
> SliceAsMatrices(t, [{1,-1}, {1,-1}, {1,2}], 1, 0);
[
    [1 2]
    [5 6],

    [19 20]
    [23 24]
]
```

`Foliation(t, a) : TenSpcElt, RngIntElt -> Mtrx`

For a tensor t with frame $U_{\mathrm{v}} \times \cdots \times U_1 \rightarrowtail U_0$, return the matrix representing the linear map $U_a \to \hom(\bigotimes_{b \neq a} U_b, U_0)$ using the bases of each U_b. If $a = 0$, then the returned matrix is given by the representation $U_0^* \to \hom(\bigotimes U_b, K)$.

Example 2.34. `ExfoliateFoliation`

We will, again, construct the same tensor, $* : \mathbb{Q}^4 \times \mathbb{Q}^3 \rightarrowtail \mathbb{Q}^2$, from the previous two examples whose structure constants sequence is $[1, \ldots, 24]$.

```
> K := Rationals();
> Forms := [Matrix(K, 3, 2, [6*i+1..6*(i+1)]) : i in [0..3]];
> t := Tensor(Forms, 1, 0);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
```

We will use `Foliation` to compute the 2-radical of t. (This is essentially what `Radical` does in Magma.) That is, we will compute the subspace $U_{\hat{2}}^{\perp} \leq \mathbb{Q}^4$ such that $U_{\hat{2}}^{\perp} * \mathbb{Q}^3 = 0$. This computation can be regarded as a nullspace computation.

```
> F2 := Foliation(t, 2);
> F2;
[ 1  2  3  4  5  6]
[ 7  8  9 10 11 12]
[13 14 15 16 17 18]
[19 20 21 22 23 24]
> R := Nullspace(F2);
> R;
Vector space of degree 4, dimension 2 over Rational Field
Echelonized basis:
( 1  0 -3  2)
( 0  1 -2  1)
```

We claim this is the 2-radical of t. Our claim is verified if `R*t` is a 0-dimensional subspace of the tensor space T with frame $\mathbb{Q}^3 \rightarrowtail \mathbb{Q}^2$. In other words, `R*t` is the trivial 2-tensor.

```
> R*t;
Tensor space of dimension 0 over Rational Field with valence 2
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
> Dimension(R*t);
0
```

`AsTensorSpace(t, a) : TenSpcElt, RngIntElt -> TenSpc, Mtrx`

Returns the associated tensor space of t at $a > 0$ along with a matrix given by the foliation of t at a. The returned tensor space is framed by $U_{\mathtt{v}} \times \cdots \times U_{a+1} \times U_{a-1} \times \cdots \times U_1 \rightarrowtail U_0$ and is generated by the tensors t_u for each u in the basis of U_a. For $a = 0$, use `AsCotensorSpace`.

`AsCotensorSpace(t) : TenSpcElt -> 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 $U_{\mathtt{v}} \times \cdots \times U_1 \rightarrowtail K$ and is generated by the tensors t_f for each f in the basis of U_0^*. In the case that t is a bilinear map, this is equivalent to the cotensor space generated by the `SystemOfForms`.

Example 2.35. `TensorsToSpaces`

We begin by creating a 4-tensor and constructing the associated tensor space at the third coordinate. `AsCotensorSpace` works similarly but when $a = 0$.

```
> t := Tensor(Rationals(), [5,4,3,2], [1..120]);
> t;
Tensor of valence 4, U3 x U2 x U1 >-> U0
U3 : Full Vector space of degree 5 over Rational Field
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
> T := AsTensorSpace(t, 3);
> T;
Tensor space of dimension 2 over Rational Field with valence 3
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
```

The dimension of T cannot be larger than 5 because that is the dimension of U_3. However, T is 2-dimensional. Slicing t as a sequence of matrices will illuminate why T is 2-dimensional.

```
> F := [SliceAsMatrices(t, [{k},{1..4},{1..3},{1,2}], 2, 1) : \
>     k in [1..5]];
```

Here, F is a sequence of a system of forms for t, one for each basis vector in U_3. If T were 5-dimensional, the systems of forms in F would be linearly independent. However, it is not, and evidently, three of the five systems of forms are linear combinations two systems of forms. We will determine the linear combinations. The first two systems of forms are independent.

```
> F[1];
[
    [ 1  3  5]
    [ 7  9 11]
    [13 15 17]
    [19 21 23],

    [ 2  4  6]
    [ 8 10 12]
    [14 16 18]
    [20 22 24]
]
> F[2];
[
    [25 27 29]
    [31 33 35]
    [37 39 41]
    [43 45 47],

    [26 28 30]
    [32 34 36]
    [38 40 42]
    [44 46 48]
]
```

We see that `F[3]` is `2*F[2]-F[1]`, and we fill in the rest of the linear combinations.

```
> F[3];
[
    [49 51 53]
    [55 57 59]
    [61 63 65]
    [67 69 71],

    [50 52 54]
    [56 58 60]
    [62 64 66]
    [68 70 72]
]
> Tensor(F[3], 2, 1) eq 2*Tensor(F[2], 2, 1) - Tensor(F[1], 2, 1);
true
> Tensor(F[4], 2, 1) eq 3*Tensor(F[2], 2, 1) - 2*Tensor(F[1], 2, 1);
true
> Tensor(F[5], 2, 1) eq 4*Tensor(F[2], 2, 1) - 3*Tensor(F[1], 2, 1);
true
```

So, indeed, T is the tensor space generated by the tensors in `F`. Note that the dimension of the 3-radical of t is 3.

```
> SystemOfForms(T.1) eq F[1];
true
> SystemOfForms(T.2) eq F[2];
true
> Radical(t, 3);
Vector space of degree 5, dimension 3 over Rational Field
Echelonized basis:
( 1  0  0 -4  3)
( 0  1  0 -3  2)
( 0  0  1 -2  1)
Mapping from: Full Vector space of degree 5 over Rational Field to Full
Vector space of degree 5 over Rational Field given by a rule
```

`AsTensor(T) : TenSpc -> TenSpcElt`

Returns a tensor corresponding to the given tensor space T. If the given tensor space is contravariant, then the returned tensor has the frame $U_{\mathtt{v}} \times \cdots \times U_1 \rightarrowtail T$, where T is thought of as a free K-module. If the given tensor space is covariant, then the returned tensor has the frame $T \times U_{\mathtt{v}} \times \cdots \times U_1 \rightarrowtail U_0$. Note that `AsTensor` is "inverse" to `AsCotensorSpace` and `AsTensorSpace` when $a = \mathtt{v}$.

Example 2.36. SpacesToTensors

We will construct the same tensor as the previous example and turn it into a tensor space at the third coordinate.

```
> t := Tensor(Rationals(), [5,4,3,2], [1..120]);
> t;
Tensor of valence 4, U3 x U2 x U1 >-> U0
U3 : Full Vector space of degree 5 over Rational Field
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
> T := AsTensorSpace(t, 3);
> T;
Tensor space of dimension 2 over Rational Field with valence 3
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
```

Now we are going to "recover" t by creating a tensor from T. However, it is not equal to t because the 3-radical of the new tensor is trivial.

```
> s := AsTensor(T);
> s;
Tensor of valence 4, U3 x U2 x U1 >-> U0
U3 : Full Vector space of degree 2 over Rational Field
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
> Radical(s, 3);
Vector space of degree 2, dimension 0 over Rational Field
```

We can even see the same sequences of matrices in s.

```
> AsMatrices(s, 2, 1);
[
    [ 1  3  5]
    [ 7  9 11]
    [13 15 17]
    [19 21 23],

    [ 2  4  6]
    [ 8 10 12]
    [14 16 18]
    [20 22 24],

    [25 27 29]
    [31 33 35]
    [37 39 41]
    [43 45 47],

    [26 28 30]
    [32 34 36]
    [38 40 42]
    [44 46 48]
]
```

2.3. Invariants of tensors

In this subsection, we detail functions to construct invariants of tensors. To access the projections or the objects acting on a specific factor U_i, the following function(s) should be used.

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

Returns the projection from the given object to the induced sub-object on the ath coordinate and the induced sub-object of the associated tensor.

Example 2.37. Inducing

To demonstrate how to Induce, we construct the 2-dimensional symplectic form on $K = \mathrm{GF}(3)$.

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

This tensor has a nontrivial derivation algebra, isomorphic to $K^2 \rtimes \mathfrak{sl}_2(3)$. However, $\mathrm{Der}(t)$ is represented in $\mathrm{End}(U_2) \times \mathrm{End}(U_1) \times \mathrm{End}(U_0)$. We will induce the action on the 1st coordinate.

```
> D := DerivationAlgebra(t);
> D.1;
[2 2 0 0 0]
[1 0 0 0 0]
[0 0 0 2 0]
```

```
[0 0 1 1 0]
[0 0 0 0 0]
> D.2;
[2 0 0 0 0]
[0 1 0 0 0]
[0 0 0 0 0]
[0 0 0 2 0]
[0 0 0 0 1]
> pi, D1 := Induce(D, 1);
> D1;
Matrix Lie Algebra of degree 2 over Finite field of size 3
> pi;
Mapping from: AlgMatLie: D to AlgMatLie: D1 given by a rule [no inverse]
```

Now we can see that have the action of D on U_1.

```
> D1.1;
[0 2]
[1 1]
> D1.2;
[0 0]
[0 2]
```

We include a partner intrinsic to `Include` and that is the following procedure.

`DerivedFrom(~X, t, C, RC) : Any, TenSpcElt, {RngIntElt}, {RngIntElt}`
`Fused : BoolElt : true`

Includes the following tensor data in the matrix object X: the tensor t, the relevant coordinates which are in the set $C \subseteq \{0, \dots, \mathtt{v}\}$, and the coordinates $RC \subseteq C$ for which the object is represented on. The reason for the subset C is so that we know the coordinates which can be induced on, see `Induce`. Currently, this only works for objects of type `AlgMat`, `AlgMatLie`, `GrpMat`, and `ModMatFld`. It is assumed that X is block diagonal, whose blocks starting from the top left go in decreasing order in the coordinates—in the same way the coordinates of the frame decrease from left to right for a tensor.

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

`Radical(t, a) : TenSpcElt, RngIntElt -> ModTupRng`

Returns the a-radical of t as a subspace of U_a. This is the subspace

$$U_{\bar{a}}^{\perp} = \{u_a \in U_a : \forall\, |u_{\bar{a}}\rangle\,,\ \langle t|u\rangle = 0\}\,.$$

`Radical(t) : TenSpcElt -> Tup`

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

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

Returns the coradical of t and a surjection from the codomain to the coradical. This is the quotient $U_0/\langle t|U_{\mathtt{v}}, \dots, U_1\rangle$.

Example 2.38. `Radicals`

We will construct the tensor for multiplication in $\mathfrak{gl}_3(\mathbb{Q})$, or equivalently, the commutator tensor of $\mathbb{M}_3(\mathbb{Q})$. Because $\mathfrak{gl}_3(\mathbb{Q})$ contains the center of $\mathbb{M}_3(\mathbb{Q})$, there will be a 2- and 1-radical.

```
> K := Rationals();
> A := MatrixAlgebra(K, 3);
> t, phi := CommutatorTensor(A);
> t;
```

```
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 9 over Rational Field
U1 : Full Vector space of degree 9 over Rational Field
U0 : Full Vector space of degree 9 over Rational Field
>
> R2 := Radical(t, 2);
> R2.1 @@ phi;
[1 0 0]
[0 1 0]
[0 0 1]
> Radical(t);
<
    Vector space of degree 9, dimension 1 over Rational Field
    Echelonized basis:
    (1 0 0 0 1 0 0 0 1),

    Vector space of degree 9, dimension 1 over Rational Field
    Echelonized basis:
    (1 0 0 0 1 0 0 0 1)
>
```

Similarly, the image of t will be 8-dimensional in $\mathbb{Q}^9$, so the coradical is $\mathbb{Q}$.

```
> Image(t);
Vector space of degree 9, dimension 8 over Rational Field
Generators:
( 1  0  0  0  0  0  0  0 -1)
( 0  1  0  0  0  0  0  0  0)
( 0  0  1  0  0  0  0  0  0)
( 0  0  0  1  0  0  0  0  0)
( 0  0  0  0  1  0  0  0 -1)
( 0  0  0  0  0  1  0  0  0)
( 0  0  0  0  0  0  1  0  0)
( 0  0  0  0  0  0  0  1  0)
Echelonized basis:
( 1  0  0  0  0  0  0  0 -1)
( 0  1  0  0  0  0  0  0  0)
( 0  0  1  0  0  0  0  0  0)
( 0  0  0  1  0  0  0  0  0)
( 0  0  0  0  1  0  0  0 -1)
( 0  0  0  0  0  1  0  0  0)
( 0  0  0  0  0  0  1  0  0)
( 0  0  0  0  0  0  0  1  0)
> Coradical(t);
Full Vector space of degree 1 over Rational Field
Mapping from: Full Vector space of degree 9 over Rational Field to Full
Vector space of degree 1 over Rational Field
```

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

`Discriminant(t) : TenSpcElt -> RngMPolElt`

Returns the discriminant of the bilinear map.

Example 2.39. `DiscriminatingOctonions`

We will compute the discriminant of the tensor $\cdot : \mathbb{O}\times\mathbb{O} \rightarrowtail \mathbb{O}$. The discriminant of this tensor is homogeneous of degree 8 with 330 terms.

```
> A := OctonionAlgebra(GF(7), -1, -1, -1);
> t := Tensor(A);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 8 over GF(7)
U1 : Full Vector space of degree 8 over GF(7)
U0 : Full Vector space of degree 8 over GF(7)
> R<a,b,c,d,e,f,g,h> := PolynomialRing(GF(7), 8);
> disc := R!Discriminant(t);
> Degree(disc);
8
> IsHomogeneous(disc);
true
> #Terms(disc);
330
```

However, if we factor the discriminant, then we can see that `disc` $= \left(x_1^2 + \cdots x_8^2\right)^4$.

```
> Factorization(disc);
[
    <a^2 + b^2 + c^2 + d^2 + e^2 + f^2 + g^2 + h^2, 4>
]
```

`Pfaffian(t) : TenSpcElt -> RngMPolElt`

Returns the Pfaffian of the antisymmetric bilinear map.

Example 2.40. `Genus2Pfaff`

In this demonstration, we pull from [**BMW**]. A p-group G has genus 2, if the image of the exponent-p central tensor of G is 2-dimensional over the centroid. One of the algorithms in [**BMW**] to decide isomorphism of such groups depends on the Pfaffian of the tensor.

First, we create two 3-groups with genus 2. The first group we create as a quotient of the Sylow 3-subgroup of $\mathrm{GL}(3, \mathrm{GF}(3^5))$.

```
> P := ClassicalSylow(GL(3, 3^5), 3);
> P := PCPresentation(UnipotentMatrixGroup(P));
> Z := Center(P);
> N := sub< Z | [Random(Z) : i in [1..3]] >;
> G := P/N;
```

The second group we create will be a quotient of the Sylow 3-subgroup of the of $\mathrm{GL}(3, \mathrm{GF}(9)) \times \mathrm{GL}(3, \mathrm{GF}(27))$.

```
> A := ClassicalSylow(GL(3, 9), 3);
> B := ClassicalSylow(GL(3, 27), 3);
> A := PCPresentation(UnipotentMatrixGroup(A));
> B := PCPresentation(UnipotentMatrixGroup(B));
> Q, inc := DirectProduct(A, B);
> ZA := Center(A);
> ZB := Center(B);
> gens := [(ZA.i@inc[1])*(ZB.i@inc[2])^-1 : i in [1..2]] \
>     cat [ZB.3@inc[2]];
> M := sub< Q | gens >;
```

```
> H := Q/M;
```

Now we will construct the exponent-p central tensors of G and H. From the way we have created the groups, $G \cong H$ if, and only if, their tensors are pseudo-isometric.

```
> t := pCentralTensor(G);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 10 over GF(3)
U1 : Full Vector space of degree 10 over GF(3)
U0 : Full Vector space of degree 2 over GF(3)
>
> s := pCentralTensor(H);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 10 over GF(3)
U1 : Full Vector space of degree 10 over GF(3)
U0 : Full Vector space of degree 2 over GF(3)
```

An implication of the main algorithm in [BMW] is that if the splitting behavior of the Pfaffians are different, then the groups are not isomorphic. Therefore, because the Pfaffian of G has a different splitting behavior from the Pfaffian of H, we conclude that $G \ncong H$.

```
> R<x,y> := PolynomialRing(GF(3), 2);
> f := R!Pfaffian(t);
> g := R!Pfaffian(s);
> f;
x^5 + x^3*y^2 + 2*x^2*y^3 + 2*x*y^4 + y^5
> g;
x^5 + 2*x^4*y + x^3*y^2 + 2*x^2*y^3 + x*y^4 + y^5
> Factorization(f), Factorization(g);
[
    <x^5 + x^3*y^2 + 2*x^2*y^3 + 2*x*y^4 + y^5, 1>
]
[
    <x^2 + x*y + 2*y^2, 1>,
    <x^3 + x^2*y + x*y^2 + 2*y^3, 1>
]
```

2.4. Exporting tensors

In the previous sections, we used groups, rings, and algebras to define tensors, but tensors can be used define algebraic structures as well. In this section, we describe ways to build groups and algebras from tensors. Currently, all intrinsics in this section only support 3-tensors.

`HeisenbergAlgebra(t) : TenSpcElt -> AlgGen`

Returns the Heisenberg algebra A induced by the bilinear tensor $t : U \times V \rightarrowtail W$. The algebra A depends on the tensor category of t. If the tensor category forces equality between U, V, and W, then $A \cong U$ as vector spaces and is a nonassociative algebra (i.e. not necessarily associative). If the tensor category forces equality between U and V, then $A \cong U \oplus W$ as vector spaces and is a nilpotent algebra where $C(A) \geq W$ and $A^2 \leq W$. Otherwise, $A \cong U \oplus V \oplus W$ as vector spaces, and using $*$ for t,

$$(u, v, w) \cdot (u', v', w') = (0, 0, u * v').$$

Example 2.41. `CraftingAlgebras`

We will demonstrate some of the nuances of `HeisenbergAlgebra` and how it interacts with the tensor category of the given tensor. We will use the same tensor throughout but changing the categories. The frame of the tensor is $\mathbb{Q}^3 \times \mathbb{Q}^3 \rightarrowtail \mathbb{Q}^3$.

```
> t := Tensor(Rationals(), [3, 3, 3], [i : i in [1..27]]);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 3 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 3 over Rational Field
> SystemOfForms(t);
[
    [ 1  4  7]
    [10 13 16]
    [19 22 25],

    [ 2  5  8]
    [11 14 17]
    [20 23 26],

    [ 3  6  9]
    [12 15 18]
    [21 24 27]
]
> Radical(t);
<
    Vector space of degree 3, dimension 1 over Rational Field
    Echelonized basis:
    ( 1 -2  1),

    Vector space of degree 3, dimension 1 over Rational Field
    Echelonized basis:
    ( 1 -2  1)
>
> Image(t);
Vector space of degree 3, dimension 2 over Rational Field
Generators:
( 1  0 -1)
( 0  1  2)
Echelonized basis:
( 1  0 -1)
( 0  1  2)
```

The default tensor category of t above is the homotopism category where none of the modules are fused together. Therefore, if we construct the Heisenberg algebra of t, the resulting algebra will be 9-dimensional. Looking at the system of forms, the first 6 matrices will be zero, and the last 3 matrices will contain the above system of forms as a 3×3 block, starting at the $(1, 4)$ entry.

```
> A := HeisenbergAlgebra(t);
> A;
Algebra of dimension 9 with base ring Rational Field
> Center(A);
Algebra of dimension 5 with base ring Rational Field
> SystemOfForms(Tensor(A))[7..9];
[
    [ 0  0  0  1  4  7  0  0  0]
```

```
    [ 0  0  0 10 13 16  0  0  0]
    [ 0  0  0 19 22 25  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0],

    [ 0  0  0  2  5  8  0  0  0]
    [ 0  0  0 11 14 17  0  0  0]
    [ 0  0  0 20 23 26  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0],

    [ 0  0  0  3  6  9  0  0  0]
    [ 0  0  0 12 15 18  0  0  0]
    [ 0  0  0 21 24 27  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0  0]
]
```

Now we will fuse U and V together with a new category. The resulting algebra will then be 6-dimensional. We will also look at the system of forms of A to see the structure constants of A.

```
> NilCat := TensorCategory([1, 1, 1], {{2,1},{0}});
> NilCat;
Tensor category of valence 3 (->,->,->) ({ 0 },{ 1, 2 })
> ChangeTensorCategory(~t, NilCat);
> A := HeisenbergAlgebra(t);
> A;
Algebra of dimension 6 with base ring Rational Field
> Center(A);
Algebra of dimension 4 with base ring Rational Field
> A^2;
Algebra of dimension 2 with base ring Rational Field
> SystemOfForms(Tensor(A))[4..6];
[
    [ 1  4  7  0  0  0]
    [10 13 16  0  0  0]
    [19 22 25  0  0  0]
    [ 0  0  0  0  0  0]
    [ 0  0  0  0  0  0]
    [ 0  0  0  0  0  0],

    [ 2  5  8  0  0  0]
    [11 14 17  0  0  0]
    [20 23 26  0  0  0]
```

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

    [ 3  6  9  0  0  0]
    [12 15 18  0  0  0]
    [21 24 27  0  0  0]
    [ 0  0  0  0  0  0]
    [ 0  0  0  0  0  0]
    [ 0  0  0  0  0  0]
]
```

Finally, we will put a tensor category on t where U, V and W are all fused together. THe Heisenberg algebra from this tensor is 3-dimensional, and the product is exactly the tensor t.

```
> AlgCat := TensorCategory([1, 1, 1], {{2,1,0}});
> AlgCat;
Tensor category of valence 3 (->,->,->) ({ 0, 1, 2 })
> ChangeTensorCategory(~t, AlgCat);
> A := HeisenbergAlgebra(t);
> A;
Algebra of dimension 3 with base ring Rational Field
> SystemOfForms(Tensor(A));
[
    [ 1  4  7]
    [10 13 16]
    [19 22 25],

    [ 2  5  8]
    [11 14 17]
    [20 23 26],

    [ 3  6  9]
    [12 15 18]
    [21 24 27]
]
```

HeisenbergLieAlgebra(t) : TenSpcElt -> AlgLie

Returns the Heisenberg Lie algebra L with Lie bracket given by the 3-tensor $t : U \times V \rightarrowtail W$. If the tensor category of t forces equality only between U and V and t is alternating, then the Heisenberg Lie algebra will have structure constants equal to t. In this case, $L \cong U \oplus W$ as vector spaces. Otherwise, $L \cong U \oplus V \oplus W$ as vector spaces, and, using $*$ in place for t, the product in L is given by

$$(u, v, w) * (u', v', w') = (0, 0, uv' - u'v).$$

Example 2.42. CraftingLieAlgberas

We will construct an alternating tensor with frame $\mathbb{Q}^3 \times \mathbb{Q}^3 \rightarrowtail \mathbb{Q}^2$.

```
> t := Tensor(Rationals(), [3, 3, 2], &cat[[1,-1,0] : i in [1..6]]);
> t := AlternatingTensor(t);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 3 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
```

```
U0 : Full Vector space of degree 2 over Rational Field
> SystemOfForms(t);
[
    [ 0 -1 -2]
    [ 1  0 -1]
    [ 2  1  0],

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

First, we will just construct the Heisenberg Lie algebra from t as is. Because the tensor category of t does not fuse together U, V, and W, the Lie algebra will be 8-dimensional.

```
> L := HeisenbergLieAlgebra(t);
> L;
Lie Algebra of dimension 8 with base ring Rational Field
> SystemOfForms(Tensor(L))[7..8];
[
    [ 0  0  0  0 -1 -2  0  0]
    [ 0  0  0  1  0 -1  0  0]
    [ 0  0  0  2  1  0  0  0]
    [ 0 -1 -2  0  0  0  0  0]
    [ 1  0 -1  0  0  0  0  0]
    [ 2  1  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0]
    [ 0  0  0  0  0  0  0  0],

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

Now if we fuse U and V together, the resulting Lie algebra will be 5-dimensional, and the Lie bracket will essentially be equal to t.

```
> NilCat := TensorCategory([1, 1, 1], {{2,1},{0}});
> NilCat;
Tensor category of valence 3 (->,->,->) ({ 0 },{ 1, 2 })
> ChangeTensorCategory(~t, NilCat);
> L := HeisenbergLieAlgebra(t);
> L;
Lie Algebra of dimension 5 with base ring Rational Field
> SystemOfForms(Tensor(L))[4..5];
[
    [ 0 -1 -2  0  0]
    [ 1  0 -1  0  0]
    [ 2  1  0  0  0]
    [ 0  0  0  0  0]
    [ 0  0  0  0  0],
```

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

HeisenbergGroup(t) : TenSpcElt -> GrpMat
HeisenbergGroupPC(t) : TenSpcElt -> GrpPC

Returns the class 2, exponent p, Heisenberg p-group with commutator given by the bilinear tensor $t : U \times V \rightarrowtail W$ over a finite field. If t is alternating and the tensor category of t forces equality between U and V, then the group returned is an extension of V by W, so $|G| = |V| \cdot |W|$. Otherwise, the group returned is an extension of $U \oplus V$ by W, so $|G| = |U| \cdot |V| \cdot |W|$. If t is not full, then G is an extension of $U \oplus V \oplus W/\operatorname{im}(t)$ (or $V \oplus W/\operatorname{im}(t)$) by $\operatorname{im}(t)$.

Example 2.43. `CraftingPGroups`

Here we demonstrate how to export a group from a tensor. We will start with a tensor that is alternating and compare the different outputs based on the category of the tensor.

```
> t := KTensorSpace(GF(5), [5, 5, 4])!0;
> for i in [1..4] do
for>   Assign(~t, [i, i+1, i], 1);   // 1s above diag
for>   Assign(~t, [i+1, i, i], -1);  // 4s below diag
for> end for;
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 5 over GF(5)
U1 : Full Vector space of degree 5 over GF(5)
U0 : Full Vector space of degree 4 over GF(5)
> IsAlternating(t);
true
```

In this example, we will focus on the more straight-forward case, when t is full. The next example demonstrates a tensor that is not full.

```
> SystemOfForms(t);
[
    [0 1 0 0 0]
    [4 0 0 0 0]
    [0 0 0 0 0]
    [0 0 0 0 0]
    [0 0 0 0 0],

    [0 0 0 0 0]
    [0 0 1 0 0]
    [0 4 0 0 0]
    [0 0 0 0 0]
    [0 0 0 0 0],

    [0 0 0 0 0]
    [0 0 0 0 0]
    [0 0 0 1 0]
    [0 0 4 0 0]
    [0 0 0 0 0],
```

```
    [0 0 0 0 0]
    [0 0 0 0 0]
    [0 0 0 0 0]
    [0 0 0 0 1]
    [0 0 0 4 0]
]
> IsFullyNondegenerate(t);
true
```

Even though t is alternating, the category does not fuse together U and V. Therefore, the Heisenberg group H is an extension of $U \oplus V$ by W, and hence, $|H| = 5^{5+5+4}$.

```
> H := HeisenbergGroup(t);
> LMGOrder(LMGCenter(H)) eq 5^4;
true
> s := pCentralTensor(H, 5, 1, 1);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 10 over GF(5)
U1 : Full Vector space of degree 10 over GF(5)
U0 : Full Vector space of degree 4 over GF(5)
```

Now we will fuse the 2 and 1 coordinate, so that $|H| = 5^{5+4}$.

```
> PgrpCat := TensorCategory([1, 1, 1], {{2,1},{0}});
> PgrpCat;
Tensor category of valence 3 (->,->,->) ({ 0 },{ 1, 2 })
> ChangeTensorCategory(~t, PgrpCat);
> H := HeisenbergGroup(t);
> LMGOrder(LMGCenter(H)) eq 5^4;
true
> s := pCentralTensor(H, 5, 1, 1);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 5 over GF(5)
U1 : Full Vector space of degree 5 over GF(5)
U0 : Full Vector space of degree 4 over GF(5)
```

Note that t and s are not the same tensor. However, they are pseudo-isometric (isotopic in the category with $U_2 = U_1$). This effect is exaggerated with `HeisenbergGroupPC` because of how PC-generators are organized.

```
> SystemOfForms(s);
[
    [0 0 0 0 0]
    [0 0 0 0 1]
    [0 0 0 0 0]
    [0 0 0 0 0]
    [0 4 0 0 0],

    [0 0 0 0 0]
    [0 0 0 0 0]
    [0 0 0 0 4]
    [0 0 0 0 0]
    [0 0 1 0 0],

    [0 0 0 0 0]
    [0 0 0 0 0]
```

```
    [0 0 0 1 0]
    [0 0 4 0 0]
    [0 0 0 0 0],

    [0 0 0 4 0]
    [0 0 0 0 0]
    [0 0 0 0 0]
    [1 0 0 0 0]
    [0 0 0 0 0]
]
```

Example 2.44. PGroupsHalfFull

We will start with an alternating tensor that is not full with the frame $\mathbb{F}_3^4 \times \mathbb{F}_3^4 \rightarrowtail \mathbb{F}_3^3$.

```
> t := Tensor(GF(3), [4, 4, 3], &cat[[1,0,0,1] : i in [1..12]]);
> t := AlternatingTensor(t);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over GF(3)
U1 : Full Vector space of degree 4 over GF(3)
U0 : Full Vector space of degree 3 over GF(3)
> Radical(t);
<
    Vector space of degree 4, dimension 1 over GF(3)
    Echelonized basis:
    (1 2 1 2),

    Vector space of degree 4, dimension 1 over GF(3)
    Echelonized basis:
    (1 2 1 2)
>
> Image(t);
Vector space of degree 3, dimension 2 over GF(3)
Generators:
(1 0 2)
(0 1 0)
Echelonized basis:
(1 0 2)
(0 1 0)
> SystemOfForms(t);
[
    [0 0 2 2]
    [0 0 2 2]
    [1 1 0 0]
    [1 1 0 0],

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

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

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

Because the tensor category of t does not force equality between U and V, the group H created from t will have order 3^{11} and be 8-generated. Furthermore, the tensor has a 2-dimensional image, so the tensor from the exponent-p central series of H will have frame $\mathbb{F}_3^{4+4+1} \times \mathbb{F}_3^{4+4+1} \rightarrowtail \mathbb{F}_3^2$. We display the system of forms of the exponent-p central tensor of H to show how different it looks compared to the original.

```
> G := HeisenbergGroup(t);
> LMGOrder(G) eq 3^11;
true
> s := pCentralTensor(G);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 9 over GF(3)
U1 : Full Vector space of degree 9 over GF(3)
U0 : Full Vector space of degree 2 over GF(3)
> SystemOfForms(s);
[
    [0 0 0 0 0 0 1 0 0]
    [0 0 0 0 0 0 0 2 2]
    [0 0 0 0 0 0 1 0 0]
    [0 0 0 0 0 0 2 1 1]
    [0 0 0 0 0 0 0 0 0]
    [0 0 0 0 0 0 0 0 0]
    [2 0 2 1 0 0 0 0 0]
    [0 1 0 2 0 0 0 0 0]
    [0 1 0 2 0 0 0 0 0],

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

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 $\mathrm{End}_K(A)$.

Example 2.45. `CenterCentroids`

We will construct a representation of $\mathfrak{sl}_2(9)$ in $\mathbb{M}_4(\mathbb{F}_3)$. First we construct $\mathfrak{gl}_2(9)$ from $\mathbb{M}_4(\mathbb{F}_3)$.

```
> M := MatrixAlgebra(GF(3), 4);
> f := ConwayPolynomial(3, 2);
> C := CompanionMatrix(f);
> I := IdentityMatrix(GF(3), 2);
> A := sub< M | [InsertBlock(M!0, X, i, j) : \
>     X in [I, C], i in [1, 3], j in [1, 3]] >;
> T := CommutatorTensor(A);
> T;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 8 over GF(3)
U1 : Full Vector space of degree 8 over GF(3)
U0 : Full Vector space of degree 8 over GF(3)
> gl2 := HeisenbergAlgebra(T);
> gl2;
Algebra of dimension 8 with base ring GF(3)
```

Our Lie algebra is not simple as it has a nontrivial center, so we will obtain $\mathfrak{sl}_2$ by factoring out the center. Note that our algebras are over the prime field $\mathbb{F}_3$, so the center is 2-dimensional (over $\mathbb{F}_3$). Notice that $\mathfrak{sl}_2(9)$ has a trivial center but has a 2-dimensional centroid.

```
> sl2 := gl2/Center(gl2);
> sl2;
Algebra of dimension 6 with base ring GF(3)
> Center(sl2);
Algebra of dimension 0 with base ring GF(3)
> Centroid(sl2);
Matrix Algebra of degree 6 with 2 generators over GF(3)
```

`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 $\mathrm{End}_K(A)$.

Example 2.46. `DerivationAlg`

We will compute the derivation algebra of the (rational) octonions $\mathbb{O}$ and also the 27 dimension exceptional Jordan algebra $\mathfrak{H}_3(\mathbb{O})$. Because the intrinsics use exact linear algebra, we do not use the more familiar field $\mathbb{C}$ in this context. First we consider $\mathbb{O}$. We verify that $\mathrm{Der}(\mathbb{O}) \cong G_2$.

```
> A := OctonionAlgebra(Rationals(), -1, -1, -1);
> A;
Algebra of dimension 8 with base ring Rational Field
> D := DerivationAlgebra(A);
> D;
Matrix Lie Algebra of degree 8 over Rational Field
> SemisimpleType(D);
G2
```

Now we will just briefly perform a sanity check and verify that `D` acts as it should.

```
> a := Random(Basis(A));
> b := Random(Basis(A));
> del := Random(Basis(D));
> (a*b)*del eq (a*del)*b + a*(b*del);
true
```

Finally, we construct $\mathfrak{H}_3(\mathbb{O})$ the 3×3 Hermitian matrices, and we verify that $\mathrm{Der}(\mathfrak{H}_3(\mathbb{O}) \cong F_4$.

```
> J := ExceptionalJordanCSA(A);
> J;
Algebra of dimension 27 with base ring Rational Field
> D_J := DerivationAlgebra(J);
> Dimension(D_J);
52
> SemisimpleType(D_J);
F4
```

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 also assigned to its tensors. Note that in [FMW, Section 7] tensors do not explicitly require a parent tensor space type. This deviation is in accordance with Magma's design patterns for vectors and vector spaces, matrices and matrix spaces. Consequently, the pattern here is more verbose and all primitive data types representing tensors are afforded as tensor spaces of specific types, e.g. `KTensorSpace` for tensor spaces for fields K, `RTensorSpace` for commutative ring R and free modules, and `TensorSpace` for black-box tensor spaces. Sparse and dense representations are indeed handled by system level types to remain efficient.

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 $S = [d_{\mathrm{v}}, \ldots, d_0]$, returns the universal tensor space $K^{d_0} \oslash \cdots \oslash K^{d_{\mathrm{v}}}$ with covariant tensor category given by C. The default category is the homotopism category.

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

For a commutative ring R and sequence $S = [d_{\mathrm{v}}, \cdots, d_0]$, returns the universal tensor space $R^{d_0} \oslash \cdots \oslash R^{d_{\mathrm{v}}}$ with covariant tensor category given by C. The default category is the homotopism category.

Example 3.1. `UniversalKTenSpc`

We demonstrate how to construct universal tensor spaces from a field $K = \mathbb{Q}$ and a sequence of nonnegative integers $[6, 5, 4, 3]$. The resulting tensor space is isomorphic to the space of multilinear maps with frame

$$\mathbb{Q}^6 \times \mathbb{Q}^5 \times \mathbb{Q}^4 \rightarrowtail \mathbb{Q}^3.$$

```
> K := Rationals();
> S := [6, 5, 4, 3];
> T := KTensorSpace(K, S);
> T;
Tensor space of dimension 360 over Rational Field with valence 4
U3 : Full Vector space of degree 6 over Rational Field
U2 : Full Vector space of degree 5 over Rational Field
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 3 over Rational Field
```

Because tensor spaces act like modules, we can construct tensors in the same way we construct vectors.

```
> t := T![Random([-1, 0, 1]) : i in [1..Dimension(T)]];
> t;
Tensor of valence 4, U3 x U2 x U1 >-> U0
```

```
U3 : Full Vector space of degree 6 over Rational Field
U2 : Full Vector space of degree 5 over Rational Field
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 3 over Rational Field
> Parent(t);
Tensor space of dimension 360 over Rational Field with valence 4
U3 : Full Vector space of degree 6 over Rational Field
U2 : Full Vector space of degree 5 over Rational Field
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 3 over Rational Field
```

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

Given a sequence $S = [U_{\mathrm{v}}, \ldots, U_0]$ of R-modules returns a universal tensor space equivalent to $U_0 \oslash \cdots \oslash U_{\mathrm{v}}$ with covariant tensor category given by C and a list of maps into the vector spaces in the frame. The default category is the homotopism category.

Example 3.2. `UniversalTenSpc`

We will construct a similar tensor space as the previous example. However, the objects will be different, according to Magma.

```
> R := Integers();
> S := [* RMatrixSpace(R, 2, 3), RSpace(R, 5), RMatrixSpace(R, 2, 2), \
>     RSpace(R, 3) *];
> T := TensorSpace(S);
> T;
Tensor space of dimension 360 over Integer Ring with valence 4
U3 : Full RSpace of degree 6 over Integer Ring
U2 : Full RSpace of degree 5 over Integer Ring
U1 : Full RSpace of degree 4 over Integer Ring
U0 : Full RSpace of degree 3 over Integer Ring
```

Even though the frame of T does not include matrix spaces, like with tensors, it can still evaluate matrices. For example, we evaluate T at $(0, 0, 0)$, which is the trivial subspace in $\mathbb{Z}^3$.

```
> x := < X!0 : X in S[1..3] >;
> x;
<
    [0 0 0]
    [0 0 0],

    (0 0 0 0 0),

    [0 0]
    [0 0]
>
> x @ T;
RSpace of degree 3, dimension 0 over Integer Ring
Generators:
```

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

Returns the signatured (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 $[V, \ldots,_p V, V^*, \ldots,_q V^*, K]$ and the tensor category with arrows $[1, \ldots,_p 1, -1, \ldots,_q -1, 0]$ and duplicates $\{\{p+q, \ldots, 1+q\}, \{q, \ldots, 1\}, \{0\}\}$. The valence of the returned tensor space will be $p + q + 1$.

Example 3.3. `SignaturedTenSpc`

Here we simply demonstrate the nuances of the signatured tensor space constructor. We set $V = \mathbb{F}_5^4$, $p = 3$, and $q = 2$, and the tensor space we will construct will have frame

$$V \times V \times V \times V^* \times V^* \rightarrowtail \mathbb{F}_5.$$

Notice this is equivalent to the frame $V \times V \times V \rightarrowtail V \times V$.

```
> K := GF(5);
> T := TensorSpace(K, 4, 3, 2);
> T;
Tensor space of dimension 1024 over GF(5) with valence 6
U5 : Full Vector space of degree 4 over GF(5)
U4 : Full Vector space of degree 4 over GF(5)
U3 : Full Vector space of degree 4 over GF(5)
U2 : Full Vector space of degree 4 over GF(5)
U1 : Full Vector space of degree 4 over GF(5)
U0 : Full Vector space of degree 1 over GF(5)
> S := KTensorSpace(K, [4, 4, 4, 4, 4, 1]);
> S;
Tensor space of dimension 1024 over GF(5) with valence 6
U5 : Full Vector space of degree 4 over GF(5)
U4 : Full Vector space of degree 4 over GF(5)
U3 : Full Vector space of degree 4 over GF(5)
U2 : Full Vector space of degree 4 over GF(5)
U1 : Full Vector space of degree 4 over GF(5)
U0 : Full Vector space of degree 1 over GF(5)
```

However, the subtleties of this construction lies in the tensor category. On the surface, the spaces T and S look the same, but probing the category reveals their differences.

```
> TensorCategory(S); // default category
Tensor category of valence 6 (->,->,->,->,->,->) ({ 1 },{ 2 },{ 0 },{ 3 },{ 4
},{ 5 })
> TensorCategory(T);
Tensor category of valence 6 (<-,<-,<-,->,->,==) ({ 0 },{ 3 .. 5 },{ 1 .. 2 })
> S eq T;
false
```

3.1.2. Universal cotensor spaces.

Currently, we only consider cotensor spaces over fields.

```
KCotensorSpace(K, S) : Fld, [RngIntElt] -> TenSpc
KCotensorSpace(K, S, C) : Fld, [RngIntElt], TenCat -> TenSpc
```

For a field K and sequence $S = [d_{\mathbf{v}}, \ldots, d_1]$ returns the universal cotensor space $\bigotimes_{a \in \bar{0}} K^{d_a}$ with the given contravariant tensor category C. The default category is the homotopism category. Notice the sequence S indices do *not* include 0.

```
CotensorSpace(S) : SeqEnum -> TenSpc, List
CotensorSpace(S) : List -> TenSpc, List
```

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

Given a sequence $S = [U_{\mathrm{v}}, \dots, U_1]$ of K-vector spaces returns the universal tensor space equivalent to $\bigotimes_{a \in \bar{0}} U_a$ with contravariant tensor category given by C and a list of maps into the vector spaces of the frame. The default category is the homotopism category. Notice the sequence S indices do *not* include 0.

Example 3.4. `UniversalCoTenSpc`

Constructing cotensor spaces is nearly the same as constructing tensor spaces. When providing a sequence of modules or a sequence of dimensions, we do not include the entry for U_0. It is automatically set to $U_0 = K$. Furthermore, the black-box construction forgets all other structure of the vector spaces and only builds a frame out of vector spaces. The maps are returned as a `List` in case they are needed.

```
> K := Rationals();
> S := [* KSpace(K, 3), MatrixAlgebra(K, 3), KMatrixSpace(K, 2, 3) *];
> S;
[*
    Full Vector space of degree 3 over Rational Field,
    Full Matrix Algebra of degree 3 over Rational Field,
    Full KMatrixSpace of 2 by 3 matrices over Rational Field
*]
> T := CotensorSpace(S);
> T;
Cotensor space of dimension 162 over Rational Field with valence 4
U3 : Full Vector space of degree 3 over Rational Field
U2 : Full Vector space of degree 9 over Rational Field
U1 : Full Vector space of degree 6 over Rational Field
```

Like with tensor spaces, cotensors are elements of cotensor spaces, and for many purposes in Magma, cotensor and tensor spaces have similar functionality.

```
> t := T.3;
> t;
Cotensor of valence 4, U3 x U2 x U1 >-> K
U3 : Full Vector space of degree 3 over Rational Field
U2 : Full Vector space of degree 9 over Rational Field
U1 : Full Vector space of degree 6 over Rational Field
> Eltseq(t);
[ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0 ]
```

3.1.3. Some standard constructions.

We include some subspaces generated by well-known tensors.

`AlternatingSpace(T) : TenSpc -> TenSpc, Map`

Returns the sub(co-)tensor space generated by all the alternating (co-)tensors contained in the given (co-)tensor space and an embedding into T.

`AntisymmetricSpace(T) : TenSpc -> TenSpc, Map`

Returns the sub(co-)tensor space generated by all the antisymmetric (co-)tensors contained in the given (co-)tensor space and an embedding into T.

`SymmetricSpace(T) : TenSpc -> TenSpc, Map`

Returns the sub(co-)tensor space generated by all the symmetric (co-)tensors contained in the given (co-)tensor space and an embedding into T.

Example 3.5. StandardTenSubspcs

We will construct some standard tensor subspaces of the universal tensor space with frame $\mathbb{F}_3^4 \times \mathbb{F}_3^4 \rightarrowtail \mathbb{F}_3^4$.

```
> K := GF(3);
> T := KTensorSpace(K, [4, 4, 4]);
> T;
Tensor space of dimension 64 over GF(5) with valence 3
U2 : Full Vector space of degree 4 over GF(5)
U1 : Full Vector space of degree 4 over GF(5)
U0 : Full Vector space of degree 4 over GF(5)
```

First, we will construct the subspace of T containing all alternating bilinear maps. Since $\binom{4}{2} = 6$, we expect the subspace to be 24-dimensional in T.

```
> Alt := AlternatingSpace(T);
> Alt;
Tensor space of dimension 24 over GF(3) with valence 3
U2 : Full Vector space of degree 4 over GF(3)
U1 : Full Vector space of degree 4 over GF(3)
U0 : Full Vector space of degree 4 over GF(3)
> t := Random(Alt);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over GF(3)
U1 : Full Vector space of degree 4 over GF(3)
U0 : Full Vector space of degree 4 over GF(3)
> IsAlternating(t);
true
```

Now we will construct the symmetric subspace of the alternating subspace, which is trivial.

```
> S := SymmetricSpace(Alt);
> S;
Tensor space of dimension 0 over GF(3) with valence 3
U2 : Full Vector space of degree 4 over GF(3)
U1 : Full Vector space of degree 4 over GF(3)
U0 : Full Vector space of degree 4 over GF(3)
```

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

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

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

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

Example 3.6. StandardCoTenSubspcs

We set $V = \mathbb{F}_5^6$ and we construct the cotensor space of the symmetric square of V, $V \wedge V$. The frame is $V \times V \rightarrowtail \mathbb{F}_5$, so the cotensor space is $\binom{6}{2}$-dimensional.

```
> V := VectorSpace(GF(5), 6);
> T := ExteriorCotensorSpace(V, 2);
> T;
Cotensor space of dimension 15 over GF(5) with valence 3
U2 : Full Vector space of degree 6 over GF(5)
U1 : Full Vector space of degree 6 over GF(5)
```

The cotensor space T is generated by all alternating tensors $\langle t| : V \times V \rightarrowtail \mathbb{F}_5$. We will demonstrate by constructing a random cotensor from T.

```
> t := Random(T);
> t;
Cotensor of valence 3, U2 x U1 >-> K
U2 : Full Vector space of degree 6 over GF(5)
U1 : Full Vector space of degree 6 over GF(5)
> SystemOfForms(t);
[
    [0 2 2 2 4 1]
    [3 0 4 1 0 0]
    [3 1 0 3 1 1]
    [3 4 2 0 2 0]
    [1 0 4 3 0 2]
    [4 0 4 0 3 0]
]
> IsAlternating(t);
true
```

Notice this construction is not $V \times V \rightarrowtail V \wedge V$. However, if we want this tensor, we can use the intrinsic `AsTensor`.

```
> s := AsTensor(T);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 6 over GF(5)
U1 : Full Vector space of degree 6 over GF(5)
U0 : Full Vector space of degree 15 over GF(5)
> IsAlternating(s);
true
```

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 T : TenSpcElt, TenSpc -> BoolElt
```

Decides if t is contained in the tensor space T.

```
IsCoercible(T, t) : TenSpc, TenSpcElt -> BoolElt
T ! t : TenSpc, TenSpcElt -> TenSpcElt
```

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

```
IsCoercible(T, S) : TenSpc, [RngElt] -> BoolElt
T ! S : TenSpc, SeqEnum -> TenSpcElt
```

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

```
IsCoercible(T, n) : TenSpc, RngIntElt -> BoolElt
T ! n : TenSpc, RngIntElt -> 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.

Example 3.7. `Coercion`

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

```
> T := KTensorSpace( GF(2), [2,3,2] );
> T;
Tensor space of dimension 12 over GF(2) with valence 3
U2 : Full Vector space of degree 2 over GF(2)
U1 : Full Vector space of degree 3 over GF(2)
U0 : Full Vector space of degree 2 over GF(2)
>
> S := [ 1 : i in [1..12] ];
> t := T!S;
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 2 over GF(2)
U1 : Full Vector space of degree 3 over GF(2)
U0 : Full Vector space of degree 2 over GF(2)
>
> t eq Tensor(GF(2), [2,3,2], S);
true
>
> T!0 in T;
true
> SystemOfForms(T!0);
[
    [0 0 0]
    [0 0 0],

    [0 0 0]
    [0 0 0]
]
```

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

Decides if the tensor spaces S and T are equal.

`S subset T : TenSpc, TenSpc -> BoolElt`

Decides if S is a subset of the tensor space T.

`IsCoercible(T, S) : TenSpc, TenSpc -> BoolElt`
`T ! S : TenSpc, TenSpc -> TenSpc`

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

Example 3.8. `TenSpcContainment`

We construct the universal tensor space T with the frame $\mathbb{F}_2^4 \times \mathbb{F}_2^4 \rightarrowtail \mathbb{F}_2^4$. We also create a tensor space T_2 with frame $\mathbb{F}_2^4 \times \mathbb{F}_2 \times \mathbb{F}_2^4 \times \mathbb{F}_2 \rightarrowtail \mathbb{F}_2^4$.

```
> T := KTensorSpace(GF(2), [4,4,4]);
> T;
Tensor space of dimension 64 over GF(2) with valence 3
U2 : Full Vector space of degree 4 over GF(2)
U1 : Full Vector space of degree 4 over GF(2)
U0 : Full Vector space of degree 4 over GF(2)
>
> T2 := KTensorSpace(GF(2), [4,1,4,1,4]);
```

```
> T2;
Tensor space of dimension 64 over GF(2) with valence 5
U4 : Full Vector space of degree 4 over GF(2)
U3 : Full Vector space of degree 1 over GF(2)
U2 : Full Vector space of degree 4 over GF(2)
U1 : Full Vector space of degree 1 over GF(2)
U0 : Full Vector space of degree 4 over GF(2)
>
> S := sub< T | T.2, T.4, T.8 >;
> S;
Tensor space of dimension 3 over GF(2) with valence 3
U2 : Full Vector space of degree 4 over GF(2)
U1 : Full Vector space of degree 4 over GF(2)
U0 : Full Vector space of degree 4 over GF(2)
```

We verify that S is a subset of T and not T_2. Since the tensors of T naturally embed into T_2, the subspace S can be coerced into T_2, which we label as S_2. We then verify that S_2 is a subset of T_2 and not T.

```
> S subset T2;
false
> S2 := T2!S;
> S2 subset T2;
true
> S2 subset T;
false
```

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.

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

Returns the ith basis tensor of the tensor space T.

```
NumberOfGenerators(T) : TenSpc -> RngIntElt
Ngens(T) : TenSpc -> RngIntElt
```

Returns the number of generators of the tensor space T.

```
Dimension(T) : TenSpc -> RngIntElt
```

Returns the dimension of the tensor space T as a free K-module.

```
# T : TenSpc -> RngIntElt
```

Returns the size of the tensor space, provided T is finite.

Example 3.9. `BasicModule`

We demonstrate the basic module functions of tensor spaces. We construct the universal tensor space T with frame $\mathbb{F}_8^3 \times \mathbb{F}_8^5 \rightarrowtail \mathbb{F}_8^7$.

```
> K := GF(8);
> T := KTensorSpace(K, [3,5,7]);
> T;
Tensor space of dimension 105 over GF(2^3) with valence 3
U2 : Full Vector space of degree 3 over GF(2^3)
U1 : Full Vector space of degree 5 over GF(2^3)
U0 : Full Vector space of degree 7 over GF(2^3)
```

Now we obtain module properties of T.

```
> Dimension(T);
105
> #Basis(T);
105
> T.100 in Basis(T);
true
> T.100;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 3 over GF(2^3)
U1 : Full Vector space of degree 5 over GF(2^3)
U0 : Full Vector space of degree 7 over GF(2^3)
> #T eq 8^(3*5*7);
true
```

`Random(T) : TenSpc -> TenSpcElt`

Provided the base ring has a random algorithm in Magma, it returns a random element of the tensor space T.

`RandomTensor(R, S) : Rng, [RngIntElt] -> TenSpcElt`
`RandomTensor(R, S, C) : Rng, [RngIntElt], TenCat -> TenSpcElt`
`RandomCotensor(K, S) : Fld, [RngIntElt] -> TenSpcElt`

Provided R has a random algorithm in Magma, it returns a random (co)tensor from the universal (co)tensor space $\oslash_{s\in S} R^s$, with category C. The default category is the homotopism category.

Example 3.10. `RandomTensors`

We create some random tensors, which works similarly to `RandomMatrix` in Magma. There are two main ways to obtain a random tensor: from a given tensor space or from a ring and a dimension sequence (the universal tensor space). First we will construct the alternating tensor space of the universal tensor space with frame $\mathbb{F}_3^4 \times \mathbb{F}_3^4 \rightarrowtail \mathbb{F}_3^2$.

```
> T := KTensorSpace(GF(3), [4,4,2]);
> T;
Tensor space of dimension 32 over GF(3) with valence 3
U2 : Full Vector space of degree 4 over GF(3)
U1 : Full Vector space of degree 4 over GF(3)
U0 : Full Vector space of degree 2 over GF(3)
> S := AlternatingSpace(T);
> S;
Tensor space of dimension 12 over GF(3) with valence 3
U2 : Full Vector space of degree 4 over GF(3)
U1 : Full Vector space of degree 4 over GF(3)
U0 : Full Vector space of degree 2 over GF(3)
```

Now we construct a random alternating tensor from S.

```
> t := Random(S);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over GF(3)
U1 : Full Vector space of degree 4 over GF(3)
U0 : Full Vector space of degree 2 over GF(3)
> SystemOfForms(t);
[
```

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

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

RandomAlternatingTensor(R, d, n, c) : Rng, RngIntElt, RngIntElt, RngIntElt -> TenSpcElt
RandomAlternatingTensor(R, S) : Rng, [RngIntElt] -> TenSpcElt

Returns a random alternating tensor from the universal tensor space $\prod_{k=1}^{n} R^d \rightarrowtail R^c$. If S is given instead, then we assume $S = [d, \ldots, d, c]$. The returned tensor has the homotopism category but fuses every module in the domain. An error is raised if R does not have a random algorithm in Magma.

RandomAntisymmetricTensor(R, d, n, c) : Rng, RngIntElt, RngIntElt, RngIntElt -> TenSpcElt
RandomAntisymmetricTensor(R, S) : Rng, [RngIntElt] -> TenSpcElt

Returns a random antisymmetric tensor from the universal tensor space $\prod_{k=1}^{n} R^d \rightarrowtail R^c$. If S is given instead, then we assume $S = [d, \ldots, d, c]$. The returned tensor has the homotopism category but fuses every module in the domain. An error is raised if R does not have a random algorithm in Magma.

RandomSymmetricTensor(R, d, n, c) : Rng, RngIntElt, RngIntElt, RngIntElt -> TenSpcElt
RandomSymmetricTensor(R, S) : Rng, [RngIntElt] -> TenSpcElt

Returns a random symmetric tensor from the universal tensor space $\prod_{k=1}^{n} R^d \rightarrowtail R^c$. If S is given instead, then we assume $S = [d, \ldots, d, c]$. The returned tensor has the homotopism category but fuses every module in the domain. An error is raised if R does not have a random algorithm in Magma.

Example 3.11. RandomSymTen

We just demonstrate how to use the RandomSymmetricTensor. The other two intrinsics are the same. This is almost equivalent to constructing a random tensor whose category fuses all the modules in the domain and then applying SymmetricTensor. In characteristic 2, all entries on the diagonal will be 0, but with RandomSymmetricTensor this is not the case.

```
> t := RandomSymmetricTensor(GF(2), 4, 2, 3);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over GF(2)
U1 : Full Vector space of degree 4 over GF(2)
U0 : Full Vector space of degree 3 over GF(2)
>
> TensorCategory(t);
Tensor category of valence 3 (->,->,->) ({ 0 },{ 1 .. 2 })
>
> SystemOfForms(t);
[
    [1 0 0 1]
    [0 1 1 1]
    [0 1 1 0]
    [1 1 0 0],

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

```
    [0 0 0 0]
    [0 1 0 1],

    [0 0 0 0]
    [0 0 1 0]
    [0 1 0 0]
    [0 0 0 1]
]
> IsSymmetric(t);
true
```

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.

Example 3.12. `TenSpcProperties`

As with tensors, we can obtain the basic tensor properties from a tensor space.

```
> T := KTensorSpace(Rationals(), [7,5,3,2]);
> T;
Tensor space of dimension 210 over Rational Field with valence 4
U3 : Full Vector space of degree 7 over Rational Field
U2 : Full Vector space of degree 5 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
> Valence(T);
4
> Frame(T);
[*
    Full Vector space of degree 7 over Rational Field,

    Full Vector space of degree 5 over Rational Field,

    Full Vector space of degree 3 over Rational Field,

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

`TensorCategory(T) : TenSpc -> TenCat`

Returns the underlying tensor category of the tensor space.

`IsCovariant(T) : TenSpc -> BoolElt`
`IsContravariant(T) : TenSpc -> BoolElt`

Decides if the underlying tensor category is covariant or contravariant.

`ChangeTensorCategory(T, C) : TenSpc, TenCat -> TenSpc`
`ChangeTensorCategory(~T, C) : TenSpc, TenCat ->`

Returns the tensor category with the given tensor category.

Example 3.13. `TenSpcCategories`

Again, like with tensors, we can obtain categorical information from tensor spaces.

```
> T := KTensorSpace(Rationals(), [4,4,4]);
> T;
Tensor space of dimension 64 over Rational Field with valence 3
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 4 over Rational Field
>
> TensorCategory(T);
Tensor category of valence 3 (->,->,->) ({ 1 },{ 2 },{ 0 })
> C := TensorCategory([1,1,-1], {{0},{1,2}});
> C;
Tensor category of valence 3 (->,->,<-) ({ 0 },{ 1, 2 })
>
> ChangeTensorCategory(~T, C);
> T;
Tensor space of dimension 64 over Rational Field with valence 3
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 4 over Rational Field
> TensorCategory(T);
Tensor category of valence 3 (->,->,<-) ({ 0 },{ 1, 2 })
```

`IsAlternating(T) : TenSpc -> BoolElt`

Decides if every tensor in the tensor space is an alternating tensor.

`IsAntisymmetric(T) : TenSpc -> BoolElt`

Decides if every tensor in the tensor space is an antisymmetric tensor.

`IsSymmetric(T) : TenSpc -> BoolElt`

Decides if every tensor in the tensor space is a symmetric tensor.

`UniversalTensorSpace(T) : TenSpc -> TenSpc`
`UniversalCotensorSpace(T) : TenSpc -> TenSpc`
`Generic(T) : TenSpc -> TenSpc`

Returns the universal (co-)tensor space with the same frame and category as T.

Example 3.14. `UniversalConst`

We construct the subspace of alternating tensors S of the tensor space T with the frame $\mathbb{Q}^6 \times \mathbb{Q}^6 \rightarrowtail \mathbb{Q}^2$. We verify that the space is in fact alternating (i.e. every tensor is alternating).

```
> T := KTensorSpace(Rationals(), [6,6,2]);
> S := AlternatingSpace(T);
> S;
Tensor space of dimension 30 over Rational Field with valence 3
U2 : Full Vector space of degree 6 over Rational Field
U1 : Full Vector space of degree 6 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
> IsAlternating(S);
```

```
true
```

From S, we construct the universal tensor space U, which in this example is equal to T.

```
> U := UniversalTensorSpace(S);
> U;
Tensor space of dimension 72 over Rational Field with valence 3
U2 : Full Vector space of degree 6 over Rational Field
U1 : Full Vector space of degree 6 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
> U eq T;
true
```

CHAPTER 4

Tensor categories

Following [FMW, Section 5.7 & Section 7], 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 homotopism category [A; W2; FMW, Section 5.7]. 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 [FMW, Section 5.7; W2]. In particular given a tensor t framed by $[U_{\mathtt{v}}, \ldots, U_0]$ then a tensor category for t will specify a function $A : [\![\mathtt{v}]\!] \to \{-1, 0, 1\}$ along with a partition $\mathcal{P}$ of $[\![\mathtt{v}]\!]$ such that the following rules apply to the tensors and morphisms in the category.

(1) for each tensor t framed by $[U_{\mathtt{v}}, \ldots, U_0]$, if $X \in \mathcal{P}$, then

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

(2) Given a second tensor s framed by $[V_{\mathtt{v}}, \ldots, V_0]$, a morphism $f : t \to s$ (Magma type `Hmtp`) will be a list $[f_{\mathtt{v}}, \ldots, f_0]$ of homomorphisms as follows:
- (Covariant) if $A(i) = 1$ then $f_i : U_i \to V_i$;
- (Constant) if $A(i) = 0$ then $U_i = V_i$ and $f_i = 1_{U_i}$; or else
- (Contravariant) $A(i) = -1$ and $f_i : U_i \leftarrow V_i$.

So if $A(0) = 1$ then

$$\left\langle \sum_{i \in A^{-1}(1)} u_i f_i + \sum_{j \notin A^{-1}(-1)} v_j \right\rangle_S = \left\langle \sum_{i \in A^{-1}(1)} u_i + \sum_{j \notin A^{-1}(-1)} v_j f_j \right\rangle_T f_0;$$

if $A(0) = 0$ then

$$\left\langle \sum_{i \in A^{-1}(1)} u_i f_i + \sum_{j \notin A^{-1}(-1)} v_j \right\rangle_S = \left\langle \sum_{i \in A^{-1}(1)} u_i + \sum_{j \notin A^{-1}(-1)} v_j f_j \right\rangle_T ;$$

else $A(0) = -1$ and

$$\left\langle \sum_{i \in A^{-1}(1)} u_i f_i + \sum_{j \notin A^{-1}(-1)} v_j \right\rangle_S f_0 = \left\langle \sum_{i \in A^{-1}(1)} u_i + \sum_{j \notin A^{-1}(-1)} v_j f_j \right\rangle_T .$$

Magma manages internally the differences between vectors and covectors and more generally tensors and cotensors. Both types are issued the Magma type `TenSpcElt`. For operations sensitive to the difference, Magma stores a value of co/contra-variance of the tensor as a property of the tensor category. This 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 $A : [\![\mathtt{v}]\!] \to \{-1, 0, 1\}$ and a partition $\mathcal{P}$ of $[\![\mathtt{v}]\!]$. 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. Constructing tensor categories

```
TensorCategory(A, P) : [RngIntElt], {SetEnum} -> TenCat
TensorCategory(A, P) : Map, {SetEnum} -> TenCat
```

Sets up a covariant tensor space category with specified direction of arrows A, and a partition $\mathcal{P}$ indicating variables to be treated as equivalent. The fiber $A^{-1}(1)$ denotes the covariant variables, $A^{-1}(0)$ identifies the constant variables, and $A^{-1}(-1)$ marks the contra-variant variables.

```
CotensorCategory(A, P) : [RngIntElt], {SetEnum} -> TenCat
CotensorCategory(A, P) : Map, {SetEnum} -> TenCat
```

Sets up a contra-variant tensor space category with specified direction of arrows A, and a partition $\mathcal{P}$ indicating variables to be treated as equivalent. The fiber $A^{-1}(1)$ denotes the covariant variables, $A^{-1}(0)$ identifies the constant variables, and $A^{-1}(-1)$ marks the contra-variant variables.

Example 4.1. `BasicCatConst`

We demonstrate the basic tensor category constructor. The only difference between `TensorCategory` and `CotensorCategory` is that the former is covariant and the latter is contravariant.

```
> C := TensorCategory([1,0,-1], {{0},{1},{2}});
> C;
Tensor category of valence 3 (->,==,<-) ({ 1 },{ 2 },{ 0 })
> IsCovariant(C);
true
>
> arrows := map< {1..5} -> {1} | x :-> 1 >;
> C := CotensorCategory(arrows, {{1..5}});
> C;
Cotensor category of valence 6 (->,->,->,->,->,==) ({ 0 },{ 1 .. 5 })
> IsContravariant(C);
true
```

```
HomotopismCategory(v : parameters) : RngIntElt -> TenCat
    Contravariant : BoolElt : false
```

Returns Albert's homotopism category – all modules categories are covariant and no duplicates considered. Set the optional parameter `Contravariant` to `true` to make it a cotensor category.

```
CohomotopismCategory(v) : RngIntElt -> TenCat
```

Returns the cohomotopism category – all domain modules categories are covariant, the codomain is contravariant, and no duplicates considered.

```
AdjointCategory(v, s, t) : RngIntElt, RngIntElt, RngIntElt -> TenCat
LinearCategory(v, s, t) : RngIntElt, RngIntElt, RngIntElt -> TenCat
```

Returns the tensor category where all modules are constant except in position s and t. Both s and t are in $[v]$. Position s is covariant, position t is contravariant.

Example 4.2. `TenCatSpecial`

Now we look at a few special tensor category constructors. The default tensor category is the homotopism category, so we construct the homotopism category using `TensorCategory` and verify they are equivalent.

```
> C := TensorCategory([1,1,1,1], {{i} : i in [0..3]});
> C;
Tensor category of valence 4 (->,->,->,->) ({ 1 },{ 2 },{ 0 },{ 3 })
> HomotopismCategory(4) eq C;
true
```

The other special tensor categories can be constructed using `TensorCategory` as well, but we just construct a few to show their properties.

```
> CohomotopismCategory(3);
Tensor category of valence 3 (->,->,<-) ({ 1 },{ 2 },{ 0 })
>
```

```
> AdjointCategory(5, 4, 1);
Tensor category of valence 5 (<-,==,==,->,==) ({ 1 },{ 0, 2, 3 },{ 4 })
```

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 4.3. `TenCatProperties`

We obtain basic properties of tensor categories.

```
> C := CotensorCategory([1,0,-1,1],{{4,3},{1},{2}});
> C;
Cotensor category of valence 5 (->,==,<-,->,==) ({ 1 },{ 2 },{ 0 },{ 3, 4 })
>
> Valence(C);
5
> Arrows(C);
[ 1, 0, -1, 1 ]
> IsContravariant(C);
true
> RepeatPartition(C);
{
    { 1 },
    { 2 },
    { 3, 4 }
}
```

4.3. Categorical operations

In this section, we define subtensors, local ideals, ideals, and quotients of tensors. For the following definitions fix a tensor $t \in T$, with frame $U_0 \oslash \cdots \oslash U_{\mathrm{v}}$—that is $U_{\mathrm{v}} \times \cdots \times U_1 \rightarrowtail U_0$.

Definition 4.3.1. A tensor $s : V_{\mathrm{v}} \times \cdots \times V_1 \rightarrowtail V_0$ is a *subtensor* of t if for all a, $V_a \leq U_a$.

Definition 4.3.2. For $A \subseteq [\mathrm{v}]$, a tensor $s : V_{\mathrm{v}} \times \cdots \times V_1 \rightarrowtail V_0$ is an *A-local ideal* of t if s is a subtensor of t and for each $a \in A$,

$$\langle s \,|\, V_{\mathrm{v}}, \ldots, V_{a+1}, U_a, V_{a-1}, \ldots, V_1 \rangle \leq V_0.$$

Definition 4.3.3. A tensor s is an *ideal* of t if it is a $\{1, \ldots, \mathrm{v}\}$-local ideal of t.

DEFINITION 4.3.4. The *A-local quotient* of a tensor t by an A-local ideal s is the tensor $q : U_{\mathtt{v}}/V_{\mathtt{v}} \times \cdots \times U_1/V_1 \rightarrowtail U_0/V_0$ where for all $|\overline{v}\rangle$,

$$\langle q \,|\, \overline{v}\rangle \equiv \langle q \,|\, v\rangle \mod V_0.$$

DEFINITION 4.3.5. The *quotient* of a tensor t by an ideal s is the $\{1, \ldots, \mathtt{v}\}$-local quotient of t by s.

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

```
Subtensor(t, S) : TenSpcElt, List -> TenSpcElt
Subtensor(t, S) : TenSpcElt, SeqEnum -> TenSpcElt
```

Returns the smallest submap of t containing S.

```
Subtensor(t, D, C) : TenSpcElt, List, Any -> TenSpcElt
Subtensor(t, D, C) : TenSpcElt, SeqEnum, Any -> TenSpcElt
```

Returns the smallest submap of t containing D in the domain and C in the codomain.

```
IsSubtensor(t, s) : TenSpcElt, TenSpcElt -> BoolElt
```

Decides whether s is a subtensor of t.

Example 4.4. Subtensors

We construct the tensor t given by octonion multiplication. The quaternions $H = \mathbb{H}$ are a subalgebra of $A = \mathbb{O}$ generated by the first four basis elements. However, H cannot be coerced into A because of how Magma organizes algebras.

```
> A := OctonionAlgebra(Rationals(), -1, -1, -1);
> t := Tensor(A);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 8 over Rational Field
U1 : Full Vector space of degree 8 over Rational Field
U0 : Full Vector space of degree 8 over Rational Field
> H := sub< A | A.1, A.2, A.3, A.4 >;
> H;
Algebra of dimension 4 with base ring Rational Field
```

There are multiple ways to get the subtensor of multiplication from H. We will create $H \times H \rightarrowtail H$ as a subtensor of $A \times A \rightarrowtail A$.

```
> H_gens := [A.i : i in [1..4]];
> s := Subtensor(t, [*H_gens, H_gens, A!0*]);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Vector space of degree 8, dimension 4 over Rational Field
Generators:
(1 0 0 0 0 0 0 0)
(0 1 0 0 0 0 0 0)
(0 0 1 0 0 0 0 0)
(0 0 0 1 0 0 0 0)
Echelonized basis:
(1 0 0 0 0 0 0 0)
(0 1 0 0 0 0 0 0)
(0 0 1 0 0 0 0 0)
(0 0 0 1 0 0 0 0)
U1 : Vector space of degree 8, dimension 4 over Rational Field
Generators:
(1 0 0 0 0 0 0 0)
(0 1 0 0 0 0 0 0)
(0 0 1 0 0 0 0 0)
```

```
(0 0 0 1 0 0 0 0)
Echelonized basis:
(1 0 0 0 0 0 0 0)
(0 1 0 0 0 0 0 0)
(0 0 1 0 0 0 0 0)
(0 0 0 1 0 0 0 0)
U0 : Vector space of degree 8, dimension 4 over Rational Field
Generators:
(1 0 0 0 0 0 0 0)
(0 1 0 0 0 0 0 0)
(0 0 1 0 0 0 0 0)
(0 0 0 1 0 0 0 0)
Echelonized basis:
(1 0 0 0 0 0 0 0)
(0 1 0 0 0 0 0 0)
(0 0 1 0 0 0 0 0)
(0 0 0 1 0 0 0 0)
```

Note that because H cannot be coerced into A (i.e. `A!H` produces an error), a subtensor of `AlgGen` cannot be done by `Subtensor(t, [H, H, H])`. Now we will construct the tensor straight from H. There is a subtle difference between the subtensor from A and the tensor from H—namely, the frame is different.

```
> s2 := Tensor(H);
> s2;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 4 over Rational Field
> s eq s2;
false
> Eltseq(s) eq Eltseq(s2);
true
```

```
LocalIdeal(t, S, A) : TenSpcElt, List, {RngIntElt} -> TenSpcElt
LocalIdeal(t, S, A) : TenSpcElt, SeqEnum, {RngIntElt} -> TenSpcElt
```

Returns the local ideal of t at A constraining S.

```
LocalIdeal(t, D, C, A) : TenSpcElt, List, Any, {RngIntElt} -> TenSpcElt
LocalIdeal(t, D, C, A) : TenSpcElt, SeqEnum, Any, {RngIntElt} -> TenSpcElt
```

Returns the local ideal of t at A constraining D in the domain and C in the codomain.

```
LocalIdeal(t, s, A) : TenSpcElt, TenSpcElt, {RngIntElt} -> TenSpcElt
```

Returns the local ideal of t at A constraining s as a submap.

```
IsLocalIdeal(t, s, A) : TenSpcElt, TenSpcElt, {RngIntElt} -> BoolElt
```

Decides if s is a local ideal of t at A.

Example 4.5. `LocalIdeals`

We use the same tensor t as the previous example, multiplication in $A = \mathbb{O}$, and we construct the subtensor t_2 given by multiplication in $\mathbb{H}$. We construct a subtensor s of t as the submap containing $\langle A_2\rangle \times \langle A_1, A_4\rangle \rightarrowtail \langle 0\rangle$, which is equal to $\langle A_2\rangle \times \langle A_1, A_4\rangle \rightarrowtail \langle A_2A_1, A_2A_4\rangle$.

```
> A := OctonionAlgebra(Rationals(), -1, -1, -1);
> t := Tensor(A);
> t;
Tensor of valence 3, U2 x U1 >-> U0
```

```
U2 : Full Vector space of degree 8 over Rational Field
U1 : Full Vector space of degree 8 over Rational Field
U0 : Full Vector space of degree 8 over Rational Field
> H_gens := [A.i : i in [1..4]];
> t2 := Subtensor(t, [*H_gens, H_gens, H_gens*]);
> s := Subtensor(t, [* A.2, [A.1, A.4], A!0 *]);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Vector space of degree 8, dimension 1 over Rational Field
Generators:
(0 1 0 0 0 0 0 0)
Echelonized basis:
(0 1 0 0 0 0 0 0)
U1 : Vector space of degree 8, dimension 2 over Rational Field
Generators:
(1 0 0 0 0 0 0 0)
(0 0 0 1 0 0 0 0)
Echelonized basis:
(1 0 0 0 0 0 0 0)
(0 0 0 1 0 0 0 0)
U0 : Vector space of degree 8, dimension 2 over Rational Field
Generators:
(0 1 0 0 0 0 0 0)
(0 0 1 0 0 0 0 0)
Echelonized basis:
(0 1 0 0 0 0 0 0)
(0 0 1 0 0 0 0 0)
```

The quaternions are a subalgebra of A generated by $\{A_1, A_2, A_3, A_4\}$. Therefore, the $\{2\}$-local ideal of s in t must contain A in the codomain. However, the $\{2\}$-local ideal of s in t_2 must only contain $\langle A_1, A_2, A_3, A_4\}$ in the codomain.

```
> s1 := LocalIdeal(t, s, {2});
> Codomain(s1);
Full Vector space of degree 8 over Rational Field
Generators:
(1 0 0 0 0 0 0 0)
(0 1 0 0 0 0 0 0)
(0 0 1 0 0 0 0 0)
(0 0 0 1 0 0 0 0)
(0 0 0 0 1 0 0 0)
(0 0 0 0 0 1 0 0)
(0 0 0 0 0 0 1 0)
(0 0 0 0 0 0 0 1)
> s2 := LocalIdeal(t2, s, {2});
> Codomain(s2);
Vector space of degree 8, dimension 4 over Rational Field
Generators:
(1 0 0 0 0 0 0 0)
(0 1 0 0 0 0 0 0)
(0 0 1 0 0 0 0 0)
(0 0 0 1 0 0 0 0)
Echelonized basis:
(1 0 0 0 0 0 0 0)
(0 1 0 0 0 0 0 0)
(0 0 1 0 0 0 0 0)
```

```
(0 0 0 1 0 0 0 0)
```

`Ideal(t, S) : TenSpcElt, List -> TenSpcElt`
`Ideal(t, S) : TenSpcElt, SeqEnum -> TenSpcElt`

Returns the ideal of t containing S.

`Ideal(t, D, C) : TenSpcElt, List, Any -> TenSpcElt`
`Ideal(t, D, C) : TenSpcElt, SeqEnum, Any -> TenSpcElt`

Returns the ideal of t containing D in the domain and C in the codomain.

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

Returns the ideal of t containing s as a submap.

`IsIdeal(t, s) : TenSpcElt, TenSpcElt -> BoolElt`

Decides if s is an ideal of t.

Example 4.6. `Ideals`

First we will construct the tensor from the $\mathbb{Q}$-algebra, $\mathbb{Q}^5$.

```
> T := KTensorSpace(Rationals(), [5,5,5]);
> A := VectorSpace(Rationals(), 5);
> t := T!0;
> for i in [1..5] do
for>   Assign(~t, [i,i,i], 1);
for> end for;
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 5 over Rational Field
U1 : Full Vector space of degree 5 over Rational Field
U0 : Full Vector space of degree 5 over Rational Field
> SystemOfForms(t);
[
    [1 0 0 0 0]
    [0 0 0 0 0]
    [0 0 0 0 0]
    [0 0 0 0 0]
    [0 0 0 0 0],

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

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

    [0 0 0 0 0]
    [0 0 0 0 0]
    [0 0 0 0 0]
    [0 0 0 1 0]
    [0 0 0 0 0],
```

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

Now we will construct the ideal tensor from the subtensor containing $\langle A_1\rangle \times \langle A_2\rangle \rightarrowtail \langle A_3\rangle$. Note that the $\{2\}$-local ideal must include $\langle A_2, A_3\rangle$ in the codomain, and the $\{1\}$-local ideal must contain $\langle A_1, A_3\rangle$ in the codomain.

```
> s := Ideal(t, [A.1, A.2, A.3]);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Vector space of degree 5, dimension 1 over Rational Field
Generators:
(1 0 0 0 0)
Echelonized basis:
(1 0 0 0 0)
U1 : Vector space of degree 5, dimension 1 over Rational Field
Generators:
(0 1 0 0 0)
Echelonized basis:
(0 1 0 0 0)
U0 : Vector space of degree 5, dimension 3 over Rational Field
Generators:
(1 0 0 0 0)
(0 1 0 0 0)
(0 0 1 0 0)
Echelonized basis:
(1 0 0 0 0)
(0 1 0 0 0)
(0 0 1 0 0)
```

Finally, we verify that the subtensor containing $\langle A_1\rangle \times \langle A_2\rangle \rightarrowtail \langle A_2, A_3\rangle$ is not an ideal.

```
> r := Subtensor(t, [A.1, A.2], [A.2, A.3]);
> IsIdeal(t, r);
false
```

```
LocalQuotient(t, s, A : parameters) : TenSpcElt, TenSpcElt, {RngIntElt} -> TenSpcElt, Hmtp
    Check : BoolElt : true
```

Returns the local quotient of t by s at $A \subseteq [\mathtt{v}]$. If you know s is a local ideal of t at A, 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 4.7. `Quotients`

We will demonstrate one of the most common uses of taking a quotient of tensors: constructing the associated fully nondegenerate tensor. We first construct a tensor with a nontrivial radical given by matrix multiplication: $\mathbb{M}_{3\times 2}(\mathbb{Q}) \times \mathbb{Q}^3 \rightarrowtail \mathbb{Q}^3$, where we take a projection of $\mathbb{Q}^3$ onto $\mathbb{Q}^2$ in the 1 coordinate.

```
> K := Rationals();
> F := [*KMatrixSpace(K, 3, 2), VectorSpace(K, 3), VectorSpace(K, 3)*];
> mult := function(x)
function>   return Transpose(x[1]*Matrix(2, 1, Eltseq(x[2])[2..3]));
function> end function;
> t := Tensor(F, mult);
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 6 over Rational Field
U1 : Full Vector space of degree 3 over Rational Field
U0 : Full Vector space of degree 3 over Rational Field
> s := Subtensor(t, [*[F[1].1, F[1].4], F[2], F[3]*]);
> s;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Vector space of degree 6, dimension 2 over Rational Field
Generators:
(1 0 0 0 0 0)
(0 0 0 1 0 0)
Echelonized basis:
(1 0 0 0 0 0)
(0 0 0 1 0 0)
U1 : Full Vector space of degree 3 over Rational Field
Generators:
(1 0 0)
(0 1 0)
(0 0 1)
U0 : Full Vector space of degree 3 over Rational Field
Generators:
(1 0 0)
(0 1 0)
(0 0 1)
> IsFullyNondegenerate(s);
false
```

Now we will construct the ideal r that evaluates to 0 and the largest subspace of $\mathbb{Q}^3$ not contained in the image.

```
> r := Ideal(t, [*F[1]!0, F[2].1, F[3].3*]);
> r;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Vector space of degree 6, dimension 0 over Rational Field
Generators:

U1 : Vector space of degree 3, dimension 1 over Rational Field
Generators:
(1 0 0)
Echelonized basis:
(1 0 0)
U0 : Vector space of degree 3, dimension 1 over Rational Field
Generators:
(0 0 1)
Echelonized basis:
(0 0 1)
```

```
> IsIdeal(t, r);
true
```

Finally, we quotient s by r to obtain a fully nondegenerate tensor.

```
> q := s/r;
> q;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 2 over Rational Field
U1 : Full Vector space of degree 2 over Rational Field
U0 : Full Vector space of degree 2 over Rational Field
> IsFullyNondegenerate(q);
true
```

4.3.2. Categorical operations on tensor spaces. We have categorical notions for tensor spaces as well, and these are inherited from the module structure on tensor spaces.

`SubConstructor(T, L) : TenSpc, Any -> TenSpc, Map`
`sub< T | L > : TenSpc, Any -> TenSpc, Map`

Returns the subtensor space of T generated by the tensors in the sequence L.

`IsSubtensorSpace(T, S) : TenSpc, TenSpc -> BoolElt`

Decides if the tensor space S is a subtensor space of T.

Example 4.8. SubtensorSpaces

We will construct the subspace S of symmetric forms from the tensor space T with frame $\mathbb{Q}^2 \times \mathbb{Q}^2 \rightarrowtail \mathbb{Q}$.

```
> K := Rationals();
> T := KTensorSpace(K, [2,2,1]);
> T;
Tensor space of dimension 4 over Rational Field with valence 3
U2 : Full Vector space of degree 2 over Rational Field
U1 : Full Vector space of degree 2 over Rational Field
U0 : Full Vector space of degree 1 over Rational Field
> S := sub< T | T.1, T.2+T.3, T.4 >;
> S;
Tensor space of dimension 3 over Rational Field with valence 3
U2 : Full Vector space of degree 2 over Rational Field
U1 : Full Vector space of degree 2 over Rational Field
U0 : Full Vector space of degree 1 over Rational Field
> IsSymmetric(S);
true
```

Now we will construct the subspace A of alternating forms from T.

```
> A := sub< T | T.2-T.3 >;
> A;
Tensor space of dimension 1 over Rational Field with valence 3
U2 : Full Vector space of degree 2 over Rational Field
U1 : Full Vector space of degree 2 over Rational Field
U0 : Full Vector space of degree 1 over Rational Field
> IsAlternating(A);
true
```

Now we verify that A is not a subtensor space of S, and thus, we have constructed a direct decomposition of T into its symmetric and alternating space.

```
> IsSubtensorSpace(S, A);
false
```

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

Returns the quotient tensor space of T by S.

Example 4.9. QuotientTensorSpaces

We pick up with the same tensor spaces as the previous example: T has frame $\mathbb{Q}^2 \times \mathbb{Q}^2 \rightarrowtail \mathbb{Q}$, S is the symmetric subspace, and A is the alternating subspace.

```
> K := Rationals();
> T := KTensorSpace(K, [2,2,1]);
> T;
Tensor space of dimension 4 over Rational Field with valence 3
U2 : Full Vector space of degree 2 over Rational Field
U1 : Full Vector space of degree 2 over Rational Field
U0 : Full Vector space of degree 1 over Rational Field
> S := sub< T | T.1, T.2+T.3, T.4 >;
> A := sub< T | T.2-T.3 >;
```

Now we construct the quotient of T by A. The result is not a symmetric tensor space. Note that Q_2 is equivalent to a symmetric tensor modulo A, but this choice is arbitrary.

```
> Q := T/A;
> Q;
Tensor space of dimension 3 over Rational Field with valence 3
U2 : Full Vector space of degree 2 over Rational Field
U1 : Full Vector space of degree 2 over Rational Field
U0 : Full Vector space of degree 1 over Rational Field
> SystemOfForms(Q.1);
[
    [1 0]
    [0 0]
]
> SystemOfForms(Q.2);
[
    [0 0]
    [1 0]
]
> SystemOfForms(Q.3);
[
    [0 0]
    [0 1]
]
```

4.4. Homotopisms

Magma provides functions for homotopisms, i.e. morphisms of tensors, see definition in [**FMW**, Sections 5.7 & 7]. Homotopisms are also equipped with a tensor category. Because homotopisms can contain multiple maps between modules, it is not really clear what is meant by the domain or codomain of a given

homotopism. In this context, the *domain* of a homotopism H will refer to the tensor, $\langle t| : U_{\mathbf{v}} \times \cdots \times U_1 \rightarrowtail U_0$, where an arrow equal to 1 at coordinate a will mean that the corresponding map at coordinate a has domain equal to U_a. And in the same vain, the tensor $\langle s| : V_{\mathbf{v}} \times \cdots \times V_1 \rightarrowtail V_0$ is the *codomain* of H if an arrow equal to 1 at coordinate a implies that the corresponding map at coordinate a has codomain equal to V_a.

4.4.1. Constructions of Homotopisms.

```
Homotopism(t, s, M : parameters) : TenSpcElt, TenSpcElt, List -> Hmtp
    Check : BoolElt : true
Homotopism(t, s, M : parameters) : TenSpcElt, TenSpcElt, SeqEnum -> Hmtp
    Check : BoolElt : true
Homotopism(t, s, M, C : parameters) : TenSpcElt, TenSpcElt, List, TenCat -> Hmtp
    Check : BoolElt : true
Homotopism(t, s, M, C : parameters) : TenSpcElt, TenSpcElt, SeqEnum, TenCat -> Hmtp
    Check : BoolElt : true
```

Returns the homotopism from t to s given by the list of maps M and the category C. The default tensor category is the same as tensor categories for t and s. If the maps M will produce a homotopism, then set `Check` to `false` to skip the verification.

```
Homotopism(M, C) : List, TenCat -> Hmtp
Homotopism(M, C) : SeqEnum, TenCat -> Hmtp
```

Returns the homotopism given by the maps in M with tensor category C.

```
IsHomotopism(t, s, H) : TenSpcElt, TenSpcElt, Hmtp -> BoolElt
IsHomotopism(t, s, M) : TenSpcElt, TenSpcElt, List -> BoolElt
IsHomotopism(t, s, M) : TenSpcElt, TenSpcElt, SeqEnum -> BoolElt
IsHomotopism(t, s, M, C) : TenSpcElt, TenSpcElt, List, TenCat -> BoolElt
IsHomotopism(t, s, M, C) : TenSpcElt, TenSpcElt, SeqEnum, TenCat -> BoolElt
```

Decides if the list of maps M induces a homotopism from t to s in the tensor category C. The default tensor category is the homotopism category. If it does induce a homotopism, it is also returned.

Example 4.10. `HomotopismConst`

We will construct two symmetric tensors $t, s : \mathbb{F}_3^3 \times \mathbb{F}_3^3 \rightarrowtail \mathbb{F}_3^3$ and apply permutations to the bases.

```
> T := KTensorSpace(GF(3), [3,3,3]);
> t := T.1+T.14+T.27;
> SystemOfForms(t);
[
    [1 0 0]
    [0 0 0]
    [0 0 0],

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

    [0 0 0]
    [0 0 0]
    [0 0 1]
]
> s := (T.4+T.10)+(T.8+T.20)+(T.18+T.24);
> SystemOfForms(s);
[
    [0 1 0]
    [1 0 0]
    [0 0 0],
```

```
    [0 0 1]
    [0 0 0]
    [1 0 0],

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

Now we construct a homotopism from t to t given by apply a permutation matrix in every coordinate.

```
> P := PermutationMatrix(GF(3), [2,1,3]);
> H := Homotopism(t, t, [*P, P, P*]);
> H;
Maps from U2 x U1 >-> U0 to V2 x V1 >-> V0.
U2 -> V2:
[0 1 0]
[1 0 0]
[0 0 1]
U1 -> V1:
[0 1 0]
[1 0 0]
[0 0 1]
U0 -> V0:
[0 1 0]
[1 0 0]
[0 0 1]
```

Note that this permutation matrix does not induce a homotopism of s.

```
> IsHomotopism(s, s, [*P, P, P*]);
false
```

Example 4.11. `MixedHomotopisms`

Homotopisms can take mixed categories of maps. To reuse the above example, we can encode the permutation as a `Map` and construct homotopisms from these types.

```
> V := VectorSpace(GF(3), 3);
> T := TensorSpace([V, V, V]);
> t := T.1+T.14+T.27;
> P := PermutationMatrix(GF(3), [2,1,3]);
> f := hom< V -> V | [<V.1, V.2>, <V.2, V.1>, <V.3, V.3>] >;
> H := Homotopism(t, t, [*f, f, f*]);
> H;
Maps from U2 x U1 >-> U0 to V2 x V1 >-> V0.
U2 -> V2: Mapping from: Full Vector space of degree 3 over GF(3) to Full
Vector space of degree 3 over GF(3)
U1 -> V1: Mapping from: Full Vector space of degree 3 over GF(3) to Full
Vector space of degree 3 over GF(3)
U0 -> V0: Mapping from: Full Vector space of degree 3 over GF(3) to Full
Vector space of degree 3 over GF(3)
```

Furthermore, we can input lists with types `Mtrx` and `Map` included.

```
> H2 := Homotopism(t, t, [*P, f, P*]);
> H2;
```

```
Maps from U2 x U1 >-> U0 to V2 x V1 >-> V0.
U2 -> V2:
[0 1 0]
[1 0 0]
[0 0 1]
U1 -> V1: Mapping from: Full Vector space of degree 3 over GF(3) to Full
Vector space of degree 3 over GF(3)
U0 -> V0:
[0 1 0]
[1 0 0]
[0 0 1]
```

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.

`H.a : Hmtp, RngIntElt -> Map`

Returns the map on the ath coordinate.

Example 4.12. `HomotopismOps`

We construct a nondegenerate alternating form t on $V = \mathbb{Q}^6$. The group of isometries are isomorphic to $\mathrm{Sp}(6, \mathbb{Q})$, the group generated by all transvections. We construct a transvection L and a corresponding matrix.

```
> V := VectorSpace(Rationals(), 6);
> T := KTensorSpace(Rationals(), [6, 6, 1]);
> t := T.2-T.7+T.16-T.21+T.30-T.35;
> t;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 6 over Rational Field
U1 : Full Vector space of degree 6 over Rational Field
U0 : Full Vector space of degree 1 over Rational Field
>
> u := V.2+2*V.3-V.5;
> L := map< V -> V | x :-> x + (x*t*u)[1]*u >;
> L;
Mapping from: ModTupFld: V to ModTupFld: V given by a rule [no inverse]
> M := Matrix(6, 6, [V.i @ L : i in [1..6]]);
> M;
[ 1  1  2  0 -1  0]
[ 0  1  0  0  0  0]
[ 0  0  1  0  0  0]
[ 0 -2 -4  1  2  0]
[ 0  0  0  0  1  0]
[ 0  1  2  0 -1  1]
```

We construct a homotopism from the transvection.

```
> H := Homotopism(t, t, [*L, L, IdentityMatrix(Rationals(), 1)*]);
> H;
Maps from U2 x U1 >-> U0 to V2 x V1 >-> V0.
U2 -> V2: Mapping from: Full Vector space of degree 6 over Rational Field to
Full Vector space of degree 6 over Rational Field given by a rule [no inverse]
U1 -> V1: Mapping from: Full Vector space of degree 6 over Rational Field to
Full Vector space of degree 6 over Rational Field given by a rule [no inverse]
```

```
U0 -> V0:
[1]
```

Since H is an isometry of t, H^2 is also an isometry of t. We verify that the 2-coordinate map of H^2 is exactly M^2.

```
> H2 := H*H;
> H2;
Maps from U2 x U1 >-> U0 to V2 x V1 >-> V0.
U2 -> V2: Mapping from: Full Vector space of degree 6 over Rational Field to
Full Vector space of degree 6 over Rational Field
Composition of Mapping from: Full Vector space of degree 6 over Rational Field
to Full Vector space of degree 6 over Rational Field given by a rule [no
inverse] and
Mapping from: Full Vector space of degree 6 over Rational Field to Full Vector
space of degree 6 over Rational Field given by a rule [no inverse]
U1 -> V1: Mapping from: Full Vector space of degree 6 over Rational Field to
Full Vector space of degree 6 over Rational Field
Composition of Mapping from: Full Vector space of degree 6 over Rational Field
to Full Vector space of degree 6 over Rational Field given by a rule [no
inverse] and
Mapping from: Full Vector space of degree 6 over Rational Field to Full Vector
space of degree 6 over Rational Field given by a rule [no inverse]
U0 -> V0:
[1]
> M2 := Matrix(6, 6, [V.i @ H2.2 : i in [1..6]]);
> M2;
[ 1  2  4  0 -2  0]
[ 0  1  0  0  0  0]
[ 0  0  1  0  0  0]
[ 0 -4 -8  1  4  0]
[ 0  0  0  0  1  0]
[ 0  2  4  0 -2  1]
> M^2 eq M2;
true
```

`Precompose(t, f, a) : TenSpcElt, Map, RngIntElt -> TenSpcElt`
`Precompose(t, M, a) : TenSpcElt, Mtrx, RngIntElt -> TenSpcElt`

If $a > 0$, then the tensor returned is the tensor that has been pre-composed by the map f or matrix M.

`t @ H : TenSpcElt, Hmtp -> TenSpcElt`

If H is a cohomotopism (a homotopism in the cohomotopism category), then either the domain or codomain of H is returned, depending on the orientation of the arrows of H.

4.4.3. Basic Properties of Homotopisms.

`Domain(H) : Hmtp -> TenSpcElt`

Returns the domain tensor of H.

`Codomain(H) : Hmtp -> TenSpcElt`

Returns the codomain tensor of H.

`Maps(H) : Hmtp -> List`

Returns the list of maps for the various modules in the domain and codomain tensors.

`TensorCategory(H) : Hmtp -> TenCat`

Returns the tensor category of H.

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

Changes the tensor category of H to the given category.

`Valence(H) : Hmtp -> RngIntElt`

Returns the valence of the underlying tensor category of the homotopism H.

`Kernel(H) : Hmtp -> TenSpcElt, List`

Returns the kernel of H as an ideal of its domain tensor.

`Image(H) : Hmtp -> TenSpcElt, List`

Returns the image of H as a submap of the codomain tensor.

Example 4.13. `HomotopismProps`

We demonstrate how to access properties of a homotopism. We construct tensors $t : \mathbb{Q}^4 \times \mathbb{Q}^4 \rightarrowtail \mathbb{Q}$ and $s : \mathbb{Q}^6 \times \mathbb{Q}^6 \rightarrowtail \mathbb{Q}$ given by the dot product.

```
> t := Tensor(IdentityMatrix(Rationals(), 4), 2, 1);
> s := Tensor(IdentityMatrix(Rationals(), 6), 2, 1);
> Z := ZeroMatrix(Rationals(), 4, 6);
> M := InsertBlock(Z, IdentityMatrix(Rationals(), 4), 1, 1);
> H := Homotopism(t, s, [*M, M, IdentityMatrix(Rationals(), 1)*]);
> H;
Maps from U2 x U1 >-> U0 to V2 x V1 >-> V0.
U2 -> V2:
[1 0 0 0 0 0]
[0 1 0 0 0 0]
[0 0 1 0 0 0]
[0 0 0 1 0 0]
U1 -> V1:
[1 0 0 0 0 0]
[0 1 0 0 0 0]
[0 0 1 0 0 0]
[0 0 0 1 0 0]
U0 -> V0:
[1]
```

Like with maps, we can obtain standard properties of homotopisms.

```
> Domain(H);
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 4 over Rational Field
U1 : Full Vector space of degree 4 over Rational Field
U0 : Full Vector space of degree 1 over Rational Field
> Codomain(H);
Tensor of valence 3, U2 x U1 >-> U0
U2 : Full Vector space of degree 6 over Rational Field
U1 : Full Vector space of degree 6 over Rational Field
U0 : Full Vector space of degree 1 over Rational Field
> Maps(H);
[*
    [1 0 0 0 0 0]
    [0 1 0 0 0 0]
    [0 0 1 0 0 0]
    [0 0 0 1 0 0],

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

```
    [0 0 0 1 0 0],

    [1]
*]
> TensorCategory(H);
Tensor category of valence 3 (->,->,->) ({ 1 },{ 2 },{ 0 })
```

When the image and kernel can be computed for the each of the maps in the homotopism, then the image and kernel can be computed for the homotopism.

```
> Im := Image(H);
> Im;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Vector space of degree 6, dimension 4 over Rational Field
Echelonized basis:
(1 0 0 0 0 0)
(0 1 0 0 0 0)
(0 0 1 0 0 0)
(0 0 0 1 0 0)
U1 : Vector space of degree 6, dimension 4 over Rational Field
Echelonized basis:
(1 0 0 0 0 0)
(0 1 0 0 0 0)
(0 0 1 0 0 0)
(0 0 0 1 0 0)
U0 : Full Vector space of degree 1 over Rational Field
> Ker := Kernel(H);
> Ker;
Tensor of valence 3, U2 x U1 >-> U0
U2 : Vector space of degree 4, dimension 0 over Rational Field
U1 : Vector space of degree 4, dimension 0 over Rational Field
U0 : Vector space of degree 1, dimension 0 over Rational Field
```

`Shuffle(H, g) : Hmtp, GrpPermElt -> Hmtp`
`Shuffle(H, g) : Hmtp, [RngIntElt] -> Hmtp`

Just like the shuffle for tensors, this returns the shuffle of the homotopism H. This is a functor from one tensor category to another and changes the order of the maps to

$$\{H_{\mathrm{v}^g}, \ldots, H_{1^g}, H_{0^g}\}.$$

In order to be defined, $g \in \mathrm{Sym}(\{0, \ldots, \mathrm{v}\})$. 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, $g \in \mathrm{Sym}(\{1, \ldots, \mathrm{v}\})$. Sequences $[a_1, \ldots, a_{\mathrm{v}+1}]$ will be interpreted as

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

CHAPTER 5

Some examples

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

5.1. Distinguishing groups

Example 5.1. 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)); // non-square
> 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 non-isomorphic. 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
```

5.2. Simplifying automorphism group computations

Example 5.2. `Ext_Over_Adj`

We demonstrate how to to simplify the automorphism group computation as discussed in [**BW2**]. 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 stabilizer of $\mathrm{GL}(8, 317)$ inside $V \wedge V$. We will decrease the size of the ambient space resulting in an easier stabilizer computation.

```
> V := Domain(T)[1];
> E := ExteriorCotensorSpace(V,2);
> E;
Cotensor space of dimension 28 over GF(317) with valence 1
U2 : Full Vector space of degree 8 over GF(317)
U1 : Full Vector space of degree 8 over GF(317)
```

Now we create a sub cotensor space S generated by all $(e_i X) \wedge e_j - e_i \wedge (e_j X)$ for $X \in A$, and then quotient $V \wedge V$ by S. The result is a 4 dimensional space.

```
> L := [];
> for E_gen in Generators(E) do
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

PETER A. BROOKSBANK
BUCKNELL UNIVERSITY

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 [**BW3**].

`IsCyclic(A) : AlgMat -> BoolElt, AlgAssElt`

Decide if the matrix 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. `ModuleSimilarity`

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;
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);
10
> assert IsCyclic(A);
```

Bibliography

[A] A. A. Albert, *Non-associative algebras. I. Fundamental concepts and isotopy*, Ann. of Math. (2) **43** (1942), 685–707. MR0007747

[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

[BMW] Peter A. Brooksbank, Joshua Maglione, and James B. Wilson, *A fast isomorphism test for groups whose Lie algebra has genus 2*, J. Algebra **473** (2017), 545–590. MR3591162

[BW1] Peter A. Brooksbank and James B. Wilson, *Computing isometry groups of Hermitian maps*, Trans. Amer. Math. Soc. **364** (2012), no. 4, 1975–1996. MR2869196

[BW2] ———, *Groups acting on tensor products*, J. Pure Appl. Algebra **218** (2014), no. 3, 405–416. MR3124207

[BW3] ———, *The module isomorphism problem reconsidered*, J. Algebra **421** (2015), 541–559. MR3272396

[BW4] ———, *StarAlge Package*, https://github.com/algeboy/StarAlge.

[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

[W3] ———, *On automorphisms of groups, rings, and algebras*, Comm. Algebra **45** (2017), no. 4, 1452–1478. MR3576669

Intrinsics