Algebras

An *algebra* is a *vector space* equipped with a *multiplication*.

So an algebra in GAP is a vector space consisting of elements that can be multiplied. In particular all the functionality for vector spaces (computation of a basis of a subspace, coefficients of a vector with respect to a basis etc.) is immediately available for algebras.

Before we can compute with an algebra A we have to present it in a form that is understandable by a computer. In GAP there are three main ways of doing that:

- by matrices,
- by a multiplication table,
- by a finite presentation.

Matrix algebras

Currently in GAP a matrix algebra is either associative or Lie. In the first case the multiplication is the ordinary matrix multiplication $\cdot$. In the second case the multiplication is the commutator $(a, b) \mapsto a \cdot b - b \cdot a$. In this case the matrices are wrapped up as `LieObjects`.

```
gap> m:= [ [ 1, 2 ], [ 3, 4 ] ];;
gap> ml:= LieObject( m );
LieObject( [ [ 1, 2 ], [ 3, 4 ] ] )
gap> ml*ml;
LieObject( [ [ 0, 0 ], [ 0, 0 ] ] )
```

A matrix algebra is given by a set of matrices that generate it as a subalgebra of the full matrix algebra.

```
gap> m1:= [ [ 1, 2 ], [ 0, 4 ] ];;
gap> m2:= [ [ 3, 1 ], [ 0, 1 ] ];;
gap> A:= Algebra( Rationals, [ m1, m2 ] );
<algebra over Rationals, with 2 generators>
gap> Dimension( A );
3
gap> L:= LieAlgebra( Rationals, [ m1, m2 ] );
<Lie algebra over Rationals, with 2 generators>
gap> Dimension( L );
3
```

Structure constants

Let A be a finite-dimensional algebra over a field F. Let $\{x_1, \dots, x_n\}$ be a basis of A. Then there are c_{ij}^k in F such that

$$x_i x_j = \sum_{k=1}^{n} c_{ij}^k x_k.$$

Conversely, it is not difficult to show that the constants c_{ij}^k determine the multiplication of A completely. Therefore the c_{ij}^k are called *structure constants*. The array (c_{ij}^k) is called a *multiplication table*.

In GAP it is possible to define an algebra by a multiplication table. However, in most practical cases many of the c_{ij}^k are zero, and we do not want to store all those zeros.

GAP uses the following approach. For fixed i, j two lists are stored. The first list contains the indices k such that $c_{ij}^k \neq 0$. The second list contains the corresponding c_{ij}^k. So

$$\mathtt{T}[\mathtt{i}][\mathtt{j}] = [[\mathtt{2}, \mathtt{7}], [\mathtt{1/2}, -\mathtt{1}]]$$

means that $x_i x_j = (1/2)x_2 - x_7$.

For creating an empty SC table we have the function

`EmptySCTable( dim, zero, [string] ).`

Where

- `dim` is the dimension of the algebra,
- `zero` is the zero of the field over which the algebra is defined,
- `string` (optional) is `"symmetric"` or `"antisymmetric"` (this can be used if the product is known to be symmetric or antisymmetric).

For filling an empty SC table there is the function

`SetEntrySCTable( T, i, j, [c1, i1, c2, i2, ...] ).`

This sets the value of `T[i][j]` to `[ [ i1, i2, ...], [ c1, c2, ...] ]`. If the product is known to be (anti)symmetric then also the value of `T[j][i]` is set.

```
gap> T:= EmptySCTable( 3, 0, "antisymmetric" );;
gap> SetEntrySCTable( T, 1, 2, [ 1/2, 3 ] );;
gap> TestJacobi( T );
true
gap> A:= AlgebraByStructureConstants( Rationals, T );
<algebra of dimension 3 over Rationals>
gap> IsLieAlgebra( A );
true
gap> A;
<Lie algebra of dimension 3 over Rationals>
gap> L:= LieAlgebraByStructureConstants( Rationals, T );
<Lie algebra of dimension 3 over Rationals>
```

Finitely presented algebras

The third way of presenting an algebra is by generators and relations. One constructs a free algebra $F(X)$ generated by a finite set X. Then one specifies a finite set $R \subset F(X)$. The algebra A described by this data is the "most general" algebra generated by X and satisfying the relations in R. (In the sense that every algebra generated by X and satisfying the relations in R is a homomorphic image of A).

In GAP there are a few types of free algebra:

- *non-associative and non-commutative*
- *associative*
- *associative with one*
- *Lie.*

```
gap> A:= FreeAssociativeAlgebra( Rationals, 2 );
<algebra over Rationals, with 2 generators>
gap> g:= GeneratorsOfAlgebra( A );
[ 1*x.1, 1*x.2 ]
gap> r:= [ g[1]*g[2]*g[2]-g[1], g[1]*g[2]-g[2] ];
[ (-1)*x.1+1*x.1*x.2^2, (-1)*x.2+1*x.1*x.2 ]
gap> K:= A/r;
<algebra over Rationals, with 2 generators>
```

Fp Lie algebras

```
gap> L:= FreeLieAlgebra( Rationals, 2 );
<Lie algebra over Rationals, with 2 generators>
gap> g:= GeneratorsOfAlgebra( L );
[ 1*x1, 1*x2 ]
gap> r:= [ g[1]*( g[1]*g[2] ) - g[1]*g[2], g[2]*( g[2]*( g[2]*g[1] )
) ];
[ (-1)*(x1*x2)+1*(x1*(x1*x2)), -1*(((x1*x2)*x2)*x2) ]
gap> K:= L/r;
<Lie algebra over Rationals, with 2 generators>
gap> Dimension( K );
4
gap> h:= NiceAlgebraMonomorphism( K );
[ [1*x1], [1*x2] ] -> [ v.1, v.2 ]
gap> Range( h );
<Lie algebra of dimension 4 over Rationals>
```

Nilpotent quotients of Lie algebras

A nilpotent quotient of a Lie algebra L is a quotient L/L^c, for some $c > 0$ (i.e., all products of length c are taken as relations). In GAP there is the function

`NilpotentQuotientOfFpLieAlgebra( FpL, cls, weights );`

where

- `FpL` is the finitely presented Lie algebra,
- `cls` is the nilpotency class of the quotient,
- `weights` (optional) are the weights of the generators.

This provides a way of studying finite quotients of inifinite dimensional Lie algebras.

```
gap> L:= FreeLieAlgebra( Rationals, "x", "y" );;
gap> g:= GeneratorsOfAlgebra(L);;
gap> x:= g[1]; y:= g[2];
1*x
1*y
gap> rr:=[((y*x)*x)*x-6*(y*x)*y,
3*((((y*x)*x)*x)*x)*x-20*(((y*x)*x)*x)*y ];
[ (-1)*(x*(x*(x*y)))+6*((x*y)*y),
(-3)*(x*(x*(x*(x*(x*y)))))+(20)*(x*(x*((x*y)*y)))+-20*((x*(x*y))*(x*y))
]
gap> K:= L/rr;
<Lie algebra over Rationals, with 2 generators>
gap> h:= NilpotentQuotientOfFpLieAlgebra( K, 50, [ 1, 2 ] );
[ [1*x], [1*y] ] -> [ v.1, v.2 ]
gap> Range( h );
<Lie algebra of dimension 50 over Rationals>
```

Subalgebras, ideals

`Subalgebra( A, gens )` and `Ideal( A, gens )` construct the subalgebra (respectively ideal) of `A` generated by `gens`. They are algebras in their own right.

```
gap> A:= AlgebraByStructureConstants( Rationals, T );
<algebra of dimension 3 over Rationals>
gap> b:= BasisVectors( Basis( A ) );
[ v.1, v.2, v.3 ]
gap> B:= Subalgebra( A, [ b[1], b[2] ] );
<algebra over Rationals, with 2 generators>
gap> I:= Ideal( A, [ b[3] ] );
<two-sided ideal in <algebra of dimension 3 over Rationals>, (1
generators)>
```

Assuming we have succesfully presented our favourite algebra in GAP, we want to analyze it. The main types of algebras for which there is functionality for doing this implemented are *associative algebras* and *Lie algebras*.

Functions for associative algebras

- `Centre`, `Centralizer`,
- `RadicalOfAlgebra` (computes the radical $\mathrm{Rad}(A)$),
- `LeviMalcevDecomposition` (computes a subalgebra $S \subset A$ such that $A = S \oplus \mathrm{Rad}(A)$),
- `CentralIdempotentsOfAlgebra` (computes a list of central primitive and orthogonal idempotents, with sum 1; therefore only works for algebras with one),
- `FullMatrixAlgebra`, `QuaternionAlgebra`.

```
gap> G:= SymmetricGroup( 5 );;
gap> A:= GroupRing( Rationals, G );
<algebra-with-one over Rationals, with 2 generators>
gap> RadicalOfAlgebra( A );
<algebra of dimension 0 over Rationals>
gap> c:= CentralIdempotentsOfAlgebra( A );;
gap> ids:= List( c, x -> Ideal( A, [ x ] ) );;
gap> List( ids, Dimension );
[ 1, 1, 25, 36, 25, 16, 16 ]
```

Functions for Lie algebras

- `LieCentre`, `LieCentralizer`, `LieNormalizer`,
- `LieDerivedSeries`, `LieLowerCentralseries`, `LieUpperCentralSeries`,
- `LieNilradical`, `LieSolvableRadical` (calculate the nilradical and the solvable radical respectively),
- `CartanSubalgebra` (calculates a Cartan subalgebra),
- `DirectSumDecomposition` (calculates the decomposition of a Lie algebra as a direct sum of ideals),
- `LeviMalcevDecomposition` (calculates a semisimple subalgebra K of a Lie algebra L such that L is the direct sum of K and its solvable radical),
- `SemiSimpleType` (calculates the type of a semisimple Lie algebra),
- `RootSystem` (calculates the root system of a (split) semisimple Lie algebra),
- `SimpleLieAlgebra` (calculates standard simple Lie algebras, of types A-G, W, S, H, K).

```
gap> L:= SimpleLieAlgebra( "G", 2, GF( 3 ) );
<Lie algebra of dimension 14 over GF(3)>
gap> M:= SimpleLieAlgebra( "A", 2, GF( 3 ) );
<Lie algebra of dimension 8 over GF(3)>
gap> K1:= KappaPerp( L, L );
<Lie algebra of dimension 7 over GF(3)>
gap> K2:= M/LieCentre( M );
<Lie algebra of dimension 7 over GF(3)>
gap> H1:=CartanSubalgebra( K1 );
<Lie algebra of dimension 1 over GF(3)>
gap> H2:=CartanSubalgebra( K2 );
<Lie algebra of dimension 1 over GF(3)>
gap> h1:= Basis( H1 )[1];
v.13
gap> h2:= Basis( H2 )[1];
v.7
```

```
gap> AdjointMatrix( Basis( K1 ), h1 );
[ [ Z(3)^0, 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3) ],
[ 0*Z(3), Z(3)^0, 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3) ],
[ 0*Z(3), 0*Z(3), Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3) ],
[ 0*Z(3), 0*Z(3), 0*Z(3), Z(3), 0*Z(3), 0*Z(3), 0*Z(3) ],
[ 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), Z(3), 0*Z(3), 0*Z(3) ],
[ 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), Z(3)^0, 0*Z(3) ],
[ 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3) ] ]
gap> AdjointMatrix( Basis(K2), h2 );
[ [ Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3) ],
[ 0*Z(3), Z(3)^0, 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3) ],
[ 0*Z(3), 0*Z(3), Z(3)^0, 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3) ],
[ 0*Z(3), 0*Z(3), 0*Z(3), Z(3), 0*Z(3), 0*Z(3), 0*Z(3) ],
[ 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), Z(3), 0*Z(3), 0*Z(3) ],
[ 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), Z(3)^0, 0*Z(3) ],
[ 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3) ] ]
```

```
gap> b1:=Basis( K1 )[ 1, 2, 6 ];
[ v.1, v.3, v.10 ]
gap> c1:= Basis( K2 )[ 2, 3, 6 ];
[ v.2, v.3, v.6 ]
gap> b1[ 1 ]*b1[ 2 ];
(Z(3))*v.4
gap> b1[ 1 ]*b1[ 3 ];
v.9
gap> b1[ 2 ]*b1[ 3 ];
(Z(3))*v.7
gap> c1[ 1 ]*c1[ 2 ];
(Z(3))*v.4
gap> c1[ 1 ]*c1[ 3 ];
v.1
gap> c1[ 2 ]*c1[ 3 ];
(Z(3))*v.5
```

```
gap> BasisVectors( Basis( K1 ) );
[ v.1, v.3, v.4, v.7, v.9, v.10, v.13 ]
gap> Append( b1, Basis( K1 )[ 3, 5, 4, 7 ] );
gap> Append( c1, Basis( K2 )[ 4, 1, 5, 7 ]);
gap> T1:= StructureConstantsTable( Basis( K1, b1 ) );;
gap> T2:= StructureConstantsTable( Basis( K2, c1 ) );;
gap> T1 = T2;
true
```