Introduction to the Lie Algebra Package

by Yuly Billig (billig@math.carleton.ca) and Mátyás Mazzag (matyas.mazzag@gmail.com)

Description

The LieAlg package can be used to perform calculations in infinite dimensional graded Lie algebras, their enveloping algebras, vertex algebras and highest weight modules.

Functions available are:

generators, using, isgenerator, wt, lsimplify, genhallmon, delta, store
unienv, asimplify, simple, roots, dotproduct, affine,
KacMoody, triangular, directsum, ideal, quotalgebra, basis
hwrep, submodule, quotrep, rep simplify, singularvector, characteristic
&*, &@, &^, &<, &<=, &>, &>=

To use the LieAlg library download the files **LieAlg.m** (contains the interface) and **LieAlg_Hidden.m** (contains supporting operations used by the library). If needed, these files may be generated from the sourse file by opening **LieAlg_source.mws** worksheet with MAPLE and executing it (go to Edit -> Execute -> Worksheet). In your own worksheet set the Maple environment variable **libname** so that it includes the directory where the files are located. Activate **LieAlg.m** and **LieAlg_Hidden.m** using **read** function and then call **with(LieAlg)** command to load all the functions in the library.

To activate help on LieAlg functions, place **LieAlg.help** file into **/lib** directory (the path to this directory can be found by typing **libname;** when you start MAPLE). For older versions of MAPLE, place into **/lib** the file **LieAlg.hdb** instead (but not both!).

Initialization

```
> restart;
> #libname := "C:\\Users\\Yuly\\Documents\\MAPLE\\LIEALG", libname;
  libname := "/home/yuly/MAPLE/LIEALG", libname;
```

Add a path to directory with LieAlg files to libname. The first format is for Windows, second is for Linux

$$libname := \text{"/home/yuly/MAPLE/LIEALG", "/opt/maple2016/lib"} \qquad \textbf{(2.1)}$$

Load LieAlg package

```
> read("/home/yuly/MAPLE/LIEALG/LieAlg.m");
  read("/home/yuly/MAPLE/LIEALG/LieAlg_Hidden.m");
> with(LieAlg) :
```

Free Lie algebra

An example of a computation in a free Lie algebra.
Define the set of generators for algebra L and allow the generators() function to assign default weights to them. The variables used as the generators should not be assigned values elsewhere in the code.

```
> generators(F,[x,y,z]);
```

$$F = [x, y, z], [[1, 0, 0], [0, 1, 0], [0, 0, 1]] \tag{3.1}$$

Simplify the expression in L.

```
> lsimplify(y&*x);
```

$$-x \&^* y \tag{3.2}$$

A more comlex simplification.

```
> a:=x&*y; b:=y&*z; lsimplify(a&*b);
```

$$a := x \&^* y$$
$$b := y \&^* z$$
$$((x \&^* y) \&^* y) \&^* z - ((x \&^* y) \&^* z) \&^* y \tag{3.3}$$

Calculate the weight of the resulting expression.

```
> wt(%);
```

$$[1, 2, 1] \tag{3.4}$$

Generate Hall-Shirshov monomials in F of the given weight. In order to construct this basis, we use an ordering on the set of generators.

```
> genhallmon([1,2,1],F);
```

$$[((x \&^* z) \&^* y) \&^* y, ((x \&^* y) \&^* z) \&^* y, ((x \&^* y) \&^* y) \&^* z] \tag{3.5}$$

Free Lie algebra with another grading

Define another set of generators for free Lie algebra of rank 2. We also supply the generators() function with our own weights for the
generators.

```
> generators(FF,[x,y,z],[[1],[1],[1]]);
```

$$FF = [x, y, z], [[1], [1], [1]] \tag{4.1}$$

```
> lsimplify(z&*x);
```

$$-x \&^* z \tag{4.2}$$

With the definition of an algebra the default algebra changes to the newly defined algebra. The default algebra is used

when no algebra is specified for Lie algebra computations. Without any arguments using() displays the algebra currently in
use along with its generators. With using(algname) we can change the default agebra to the desired.

```
> using();
```

$$FF = [x, y, z] \tag{4.3}$$

Calculate the weight of the expression using the user defined weights.

```
> wt(((x&*y)&*z)&*x);
```

$$[4] \tag{4.4}$$

Generating Hall Monomials in FF with weight [4].

```
> genhallmon([4],FF);
```

$$\begin{gathered}[((x \&^* y) \&^* y) \&^* y, ((x \&^* y) \&^* y) \&^* z, ((x \&^* y) \&^* z) \&^* y, \\ ((x \&^* y) \&^* z) \&^* z, ((x \&^* z) \&^* y) \&^* y, ((x \&^* z) \&^* y) \&^* z, \\ ((x \&^* z) \&^* z) \&^* y, ((x \&^* z) \&^* z) \&^* z, ((y \&^* z) \&^* z) \&^* z, \\ (x \&^* (x \&^* y)) \&^* y, (x \&^* (x \&^* y)) \&^* z, (x \&^* (x \&^* z)) \&^* y, \\ (x \&^* (x \&^* z)) \&^* z, (y \&^* (y \&^* z)) \&^* z, (x \&^* y) \&^* (x \&^* z), \\ x \&^* (x \&^* (x \&^* y)), x \&^* (x \&^* (x \&^* z)), y \&^* (y \&^* (y \&^* z))]\end{gathered} \tag{4.5}$$

sl(2)

An example showing how to set up the sl(2) algebra in the LieAlg pacakge.

```
> generators(sl2,[e,h,f],[[1],[0],[-1]]);
```

$$sl2 = [e, h, f], [[1], [0], [-1]] \tag{5.1}$$

Use the store() command to store the simplification rules for the algebra.

```
> store([h&*e=2*e,h&*f=-2*f,e&*f=h]);
```

$$\begin{gathered} h \&^* e = 2\, e \\ h \&^* f = -2\, f \\ e \&^* f = h \end{gathered} \tag{5.2}$$

Simplify an sl(2) expression.

```
> lsimplify((e&*f)&*(e+f));
```

$$2\, e - 2\, f \tag{5.3}$$

2-Dimensional solvable algebra

```
> generators(sol,[d,n]); store([d&*n=n]);
```

$$\begin{gathered} sol = [d, n], [[1, 0], [0, 1]] \\ d \&^* n = n \end{gathered} \tag{6.1}$$

```
> lsimplify((((d&*n)&*d)&*d)&*d,sol);
```

$$-n \tag{6.2}$$

```
> lsimplify((((d&*n)&*d)&*d)&*n);
```

$$0 \tag{6.3}$$

Direct Sums

Form a direct sum of sl(2) with a 2-dimensional solvable algebra

```
> directsum(newalg,[sl2,sol]);
```

$$newalg = [sl2, sol] \quad (7.1)$$

```
> lsimplify((e+d)&*(f+n));
```

$$h + n \quad (7.2)$$

```
> using();
```

$$newalg = [sl2, sol] \quad (7.3)$$

```
> wt(e); wt(h); wt(f);
```

$$[1, 0]$$
$$[0, 0]$$
$$[-1, 0] \quad (7.4)$$

```
> wt(d); wt(n);
```

$$[1, 0]$$
$$[0, 1] \quad (7.5)$$

Virasoro algebra

When setting up a Lie algebra, we may use basis vectors parametrized by index variables. Below we use this method to define the Virasoro algebra, using parameters to give the Lie brackets between basis elements, as well as their weights. Please note that the variables used as indices should not be assigned any values elsewhere in the code.

```
> generators(Vir,[e[alpha],z],[[alpha],[0]]); store([e[alpha]&*e
  [beta]=(beta-alpha)*e[alpha+beta]+delta(alpha,-beta)*((alpha^3-
  alpha)/12)*z, e[alpha]&*z=0]);
```

$$Vir = [e_{\alpha}, z], [[\alpha], [0]]$$

$$e_{\alpha} \,\&^*\, e_{\beta} = (\beta - \alpha)\, e_{\alpha+\beta} + \delta(\alpha, -\beta) \left(\frac{1}{12}\alpha^3 - \frac{1}{12}\alpha\right) z$$

$$e_{\alpha} \,\&^*\, z = 0 \quad (8.1)$$

Simplifying an expression in Vir.

```
> lsimplify(e[3]&*e[-3]);
```

$$-6\, e_0 + 2\, z \quad (8.2)$$

Simple finite-dimensional algebras

Defining simple algebra F4 of type F[4].

```
> simple(F4, F[4]);
```

$$F4 = [e_{k1,\, k2,\, k3,\, k4}, h_{i1}], [[k1, k2, k3, k4], [0, 0, 0, 0]] \quad (9.1)$$

The matrix generated for F. The general naming convention is that the matrix is called A_*algname*.

```
> eval(A_F4);
```

$$\begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -2 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} \quad (9.2)$$

Simplification of expressions in F.

```
> lsimplify(e[1,0,0,0]&*e[0,1,0,0]);
  lsimplify(e[1,1,1,1]&*e[1,1,0,0]);
  lsimplify(e[2,2,1,1]&*e[1,0,0,0]);
  lsimplify(e[-1,-2,-1,-1]&*e[1,2,1,1]);
```

$$e_{1,\,1,\,0,\,0}$$
$$-2\,e_{2,\,2,\,1,\,1}$$
$$0$$
$$-h_1 - 2\,h_2 - 2\,h_3 - 2\,h_4 \quad (9.3)$$

A list of positive roots (as linear combinations of simple roots) may be obtained using function *roots*.

```
> roots(F[4]);
```

[[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1], [1, 1, 0, 0], [0, 1, 1, 0], [0, 2, 1, 0], [0, 0, 1, 1], [1, 1, 1, 0], [0, 1, 1, 1], [1, 2, 1, 0], [2, 2, 1, 0], [0, 2, 1, 1], [1, 1, 1, 1], [1, 2, 1, 1], [2, 2, 1, 1], [0, 2, 2, 1], [1, 2, 2, 1], [2, 2, 2, 1], [1, 3, 2, 1], [2, 3, 2, 1], [2, 4, 2, 1], [2, 4, 3, 1], [2, 4, 3, 2]] **(9.4)**

An invariant bilinear form may be evaluated using function *dotproduct*

```
> dotproduct(e[2,4,3,2], e[-2,-4,-3,-2]);
  dotproduct(e[1,0,0,0], e[0,-1,0,0]);
  dotproduct(h[2],h[3]);
```

$$1$$
$$0$$
$$-2 \quad (9.5)$$

Quotient algebras of a free Lie algebra

To build quotient algebras first define an algebra (P in our case) with its generators.

```
> generators(P,[x,y]);
```

$$P = [x,\, y],\ [[1,\, 0],\, [0,\, 1]] \quad (10.1)$$

Then we set up an ideal (Q in this example) by specifying its generators. Generators of an ideal must be homogeneous with respect to the grading being used.

```
> ideal(Q,[x&*(x&*(x&*y)),((x&*y)&*y)&*y]);
```

$$Q = [x \,\&^*\, (x \,\&^*\, (x \,\&^*\, y)),\ ((x \,\&^*\, y) \,\&^*\, y) \,\&^*\, y] \quad (10.2)$$

Define the quotient algebra S of a free Lie algebra P by its ideal Q.

```
> quotalg(S,Q);
```

$$S = [P, Q] \tag{10.3}$$

Simplifying an expression in S and generating basis for the quotient algebra.

```
> lsimplify(x&*(y&*(x&*(y&*x))),S);
> basis([3,2],S);
> basis([4,2],S);
```

$$(x \&^* (x \&^* y)) \&^* (x \&^* y)$$
$$[(x \&^* (x \&^* y)) \&^* (x \&^* y)]$$
$$[\ ] \tag{10.4}$$

Kac-Moody algebras

Defining Kac-Moody algebra G. KacMoody() sets up three algebras with names *algname*_plus, *algname*_zero and
*algname*_minus that have positive, zero and negative weights respectively.

```
> C:=matrix([[2,-3],[-3,2]]);
> KacMoody(G,C);
```

$$C := \begin{bmatrix} 2 & -3 \\ -3 & 2 \end{bmatrix}$$
$$G = [G\_minus, G\_zero, G\_plus] \tag{11.1}$$

```
> lsimplify(e[1]&*f[1]);
```

$$h_1 \tag{11.2}$$

```
> lsimplify((e[2]&*e[1])&*e[1], G);
```

$$(e_2 \&^* e_1) \&^* e_1 \tag{11.3}$$

```
> lsimplify((((f[2]&*f[1])&*f[1])&*f[1])&*f[1], G);
```

$$0 \tag{11.4}$$

Simplifying expressions in G and generating basis for weight [4,4] in the positive algebra G_plus.

```
> lsimplify(((((e[1]&*e[2])&*e[1])&*e[2])&*f[1]),G);
> lsimplify(e[1]&*(e[1]&*(e[1]&*(e[1]&*e[2]))),G);
> basis([4,4],G_plus);
```

$$4\,(e_2 \&^* (e_2 \&^* e_1))$$
$$0$$
$$\begin{aligned}
&[(((e_2 \&^* (e_2 \&^* (e_2 \&^* e_1))) \&^* (e_2 \&^* e_1)) \&^* e_1) \&^* e_1,\\
&\quad (((e_2 \&^* (e_2 \&^* e_1)) \&^* (e_2 \&^* e_1)) \&^* (e_2 \&^* e_1)) \&^* e_1,\\
&\quad ((e_2 \&^* (e_2 \&^* e_1)) \&^* ((e_2 \&^* (e_2 \&^* e_1)) \&^* e_1)) \&^* e_1,\\
&\quad (((e_2 \&^* (e_2 \&^* (e_2 \&^* e_1))) \&^* e_1) \&^* e_1) \&^* (e_2 \&^* e_1),\\
&\quad (((e_2 \&^* (e_2 \&^* e_1)) \&^* (e_2 \&^* e_1)) \&^* e_1) \&^* (e_2 \&^* e_1),\\
&\quad (((e_2 \&^* (e_2 \&^* e_1)) \&^* e_1) \&^* (e_2 \&^* e_1)) \&^* (e_2 \&^* e_1)]
\end{aligned} \tag{11.5}$$

Lie algebras with triangular decomposition

A tringular decomposition of a Lie algebra is a decomposition in 3 subalgebras: positive subalgebra, negative subalgebra and Cartan subalgebra. Examples of such algebras include Kac-Moody algebras, Virasoro Lie algebra and other Kac-Moody like algebras. The function below is a tool for building algebras with triangular decompositions. To do this, one should first define the three subalgebras, and then glue them together, specifying Lie brackets between the generators from distinct subalgebras. The bracket between a generator of Cartan and a generator of a positive (resp. negative) subalgebra should be in positive (resp. negative) subalgebra, and brackets between pairs of generators, one from positive and one from negative subalgebra, should be in the Cartan.

Elements of the Cartan subalgra should be assigned zero weights. For a pair of generators from opposite subalgebras, if their weights do not add up to zero, the Lie bracket is automatically assumed to be zero. All other relations between pairs of generators from different subalgebras should be specified. The list of such relations should be either passed as the third argument to the function, if not, the function opens a dialog window to input relations between the generators.

Below we build K2 Lie algebra, introduced by V.Kac [V.Kac, *Simple irreducible graded Lie algebras of finite growth.* (Russian) Izv. Akad. Nauk SSSR Ser. Mat. 32, 1968, 1323–1367]. This Lie algebra has rank 2 free Lie algebras as positive and negative subalgebras, glued together with a 2-dimensional Cartan subalgebra. It is a Kac-Moody like algebra, but without Serre's relations, and it corresponds to Cartan-like matrix [[2, 2], [2,2]].

```
> generators(Kpos,[E[1],E[2]],[[1,0],[0,1]]);
```
$$Kpos = [E_1, E_2], [[1, 0], [0, 1]] \quad (12.1)$$

```
> generators(Kneg,[F[1],F[2]],[[-1,0],[0,-1]]);
```
$$Kneg = [F_1, F_2], [[-1, 0], [0, -1]] \quad (12.2)$$

```
> generators(Cartan, [H[1],H[2]],[[0,0],[0,0]]); store([H[1]&*H
  [2]=0]);
```
$$Cartan = [H_1, H_2], [[0, 0], [0, 0]]$$
$$H_1 \&^* H_2 = 0 \quad (12.3)$$

```
> # triangular(K2, [Cartan, Kneg, Kpos]); # This will use a
  dialog to enter the relations
```

```
> triangular(K2, [Cartan, Kneg, Kpos], [H[1]&*E[1]=2*E[1],H[1]&*E
  [2]=2*E[2],H[2]&*E[1]=2*E[1],H[2]&*E[2]=2*E[2],H[1]&*F[1]=-2*F
  [1], H[1]&*F[2]=-2*F[2], H[2]&*F[1]=-2*F[1], H[2]&*F[2]=-2*F
  [2], E[1]&*F[1]=H[1], E[2]&*F[2]=H[2]]);
```
$$K2 = [Kneg, Cartan, Kpos] \quad (12.4)$$

```
> lsimplify(E[1]&*F[2]);
```
$$0 \quad (12.5)$$

```
> lsimplify(((E[1]&*E[2])&*E[1])&*(F[1]&*F[2]));
```
$$12\, E_1 \quad (12.6)$$

Affine Lie algebras

Even though affine Lie algebras belong to the class of Kac_moody algebras, setting them up with Serre's relations is computationally inefficient. It is much more effective to use their realizations as central extensions of loop Lie algebras. Below we show how to use *affine* function to set up an untwisted affine Lie algebra of type G(2) . This construction is very similar to the construction of finite-dimensional simple Lie algebras, except that each basis element has an extra (last) index that indicates the power of the loop parameter, and there is a central element K.

```
> affine(affG2, G[2]);
```

$$affG2 = [e_{r1, r2, k}, h_{i1, k}, K], [[r1, r2, k], [0, 0, k], [0, 0, 0]] \quad \textbf{(13.1)}$$

```
> lsimplify(e[1,1,2]&*e[-2,-1,1]);
```

$$4\, e_{-1, 0, 3} \quad \textbf{(13.2)}$$

```
> lsimplify(e[3,2,7]&*e[-3,-2,-7]);
```

$$72\, h_{2, 0} + 36\, h_{1, 0} + 252\, K \quad \textbf{(13.3)}$$

It is possible to customize the names of the generators of affine algebra, and more importantly their weights, by using additional arguments for this function. Here is an example of the same affine G[2] algebra with different generator names:

```
> affine(AFFG2, G[2], [E,H,K]);
```

$$AFFG2 = [E_{r1, r2, k}, H_{i1, k}, K], [[r1, r2, k], [0, 0, k], [0, 0, 0]] \quad \textbf{(13.4)}$$

```
> lsimplify(E[1,1,2]&*E[2,1,3]);
```

$$-E_{3, 2, 5} \quad \textbf{(13.5)}$$

Here is the same affine G[2] algebra, but with a Z-grading, which is more natural in the context of vertex algebras:

```
> affine(affG2Z, G[2], [[k],[k],[0]]);
```

$$affG2Z = [e_{r1, r2, k}, h_{i1, k}, K], [[k], [k], [0]] \quad \textbf{(13.6)}$$

```
> wt(e[3,2,7]);
```

$$[7] \quad \textbf{(13.7)}$$

Another important grading on affine algebras is the Kac-Moody grading, for which all components are either non-negative, or all non-positive:

```
> affine(affG2KM, G[2], [[r1+3*k, r2+2*k, k], [3*k, 2*k, k], [0,
  0,0]]);
```

$$affG2KM = [e_{r1, r2, k}, h_{i1, k}, K], [[r1 + 3\, k, r2 + 2\, k, k], [3\, k, 2\, k, k], [0, 0, 0]] \quad \textbf{(13.8)}$$

Here is the alpha_0 root vector:

```
> wt(e[-3, -2, 1]);
```

$$[0, 0, 1] \quad \textbf{(13.9)}$$

Universal enveloping algebras

Given a Lie algebra L, we can construct its universal enveloping algebra U_L. All computations in the universal enveloping algebra are carried out in the Poincar - Birkhoff-Witt basis, associated with the total order &< on the basis elements of the Lie algebra. We use operation &@ to denote associative product. Associative powers of elements are denoted by &^; for example, *x&^3* is a short form for *x&@x&@x*.To bring an element of the universal enveoping algebra to its canonical PBW form, use function *asimplify*.

```
> unienv(sl2);
```

(14.1)

$$U\_sl2 = [f, h, e], [[-1], [0], [1]] \tag{14.1}$$

```
> asimplify(e&@h&@f&@e);
```

$$-2\,h \mathbin{\&@} e - 4\,f \mathbin{\&@} (e \mathbin{\&\wedge} 2) + (h \mathbin{\&\wedge} 2) \mathbin{\&@} e + (f \mathbin{\&@} h) \mathbin{\&@} (e \mathbin{\&\wedge} 2) \tag{14.2}$$

```
> simple(sl3, A[2]);
```

$$sl3 = [e_{k1,\, k2}, h_{i1}], [[k1, k2], [0, 0]] \tag{14.3}$$

```
> unienv(sl3);
```

$$U\_sl3 = [e_{k1,\, k2}, h_{i1}], [[k1, k2], [0, 0]] \tag{14.4}$$

Casimir element

```
> Casimir:=e[1,0]&@e[-1,0]+e[0,1]&@e[0,-1]-e[1,1]&@e[-1,-1]+e[-1,
  0]&@e[1,0]+e[0,-1]&@e[0,1]-e[-1,-1]&@e[1,1]+(2/3)*h[1]&@h[1]+
  (1/3)*h[1]&@h[2]+(1/3)*h[2]&@h[1]+(2/3)*h[2]&@h[2];
```

$$Casimir := (e_{1,\,0}) \mathbin{\&@} (e_{-1,\,0}) + (e_{0,\,1}) \mathbin{\&@} (e_{0,\,-1}) - (e_{1,\,1}) \mathbin{\&@} (e_{-1,\,-1}) + (e_{-1,\,0}) \mathbin{\&@} (e_{1,\,0}) + (e_{0,\,-1}) \mathbin{\&@} (e_{0,\,1}) - (e_{-1,\,-1}) \mathbin{\&@} (e_{1,\,1}) + \frac{2}{3}(h_1) \mathbin{\&@} (h_1) + \frac{1}{3}(h_1) \mathbin{\&@} (h_2) + \frac{1}{3}(h_2) \mathbin{\&@} (h_1) + \frac{2}{3}(h_2) \mathbin{\&@} (h_2) \tag{14.5}$$

Casimir element is central

```
> asimplify(Casimir&@e[1,0]-e[1,0]&@Casimir, U_sl3);
```

$$0 \tag{14.6}$$

Verma modules

Given a Z-graded algebra, we can construct a highest weight Verma module using function *hwrep*. In the example below the arguments to this function are: M - name of the module, sl2 - name of the Lie algebra, v - name of the highest weight vector, [h=5] - list of rules indicating how elements of weight zero act on the highest weight vector. In this instance, we constructed a Verma module with the highest weight 5. The action of the universal enveloping algebra on the module is still denoted by *&@*. To bring an element of the module to its canonical form, use function *repsimplify*.

```
> hwrep(M, sl2, v, [h=5]);
```

$$M = v_5 \tag{15.1}$$

```
> repsimplify(f&@v[5]);
```

$$f \mathbin{\&@} (v_5) \tag{15.2}$$

```
> repsimplify(h&@v[5]);
```

$$5\,v_5 \tag{15.3}$$

```
> repsimplify(e&@v[5]);
```

$$0 \tag{15.4}$$

```
> repsimplify((e&^2)&@(f&^4)&@v[5]);
```

$$72\,(f \mathbin{\&\wedge} 2) \mathbin{\&@} (v_5) \tag{15.5}$$

```
> repsimplify(e&@(f&^6)&@v[5]);
```

$$0 \tag{15.6}$$

Verma modules for algebras with a tringular decomposition

Verma modules may be defined for any Lie algebra with a triangular decomposition, that is satisfying the condition that weights of element either have all non-negative or all non-positive components. This condition ensures that weight spaces in the Verma module are finite-dimensional (of course we assume here that graded components of the Lie algebra are finite-dimensional). Note that the construction of the highest weight module depends not only on Lie algebra, but also on the grading imposed on it. The grading should be chosen in a way that there will be no Lie algebra elements having weights with mixed signs. Below we construct the Verma module for affine Lie algebra of type A[1] with the Kac-Moody grading.

```
> affine(affsl2KM, A[1], [[r1+k,k], [k,k], [0,0]]);
```

$$affsl2KM = [e_{r1,\,k},\ h_{i1,\,k},\ K],\ [[r1+k,\ k],\ [k,\ k],\ [0,\ 0]] \quad (16.1)$$

```
> hwrep(MaffKM, affsl2KM, v, [h[1,0]=0, K=1]);
```

$$MaffKM = v_{0,\,1} \quad (16.2)$$

```
> using();
```

$$U\_affsl2KM = [e_{r1,\,k},\ h_{i1,\,k},\ K] \quad (16.3)$$

```
> repsimplify(e[-1,0]&@v[0,1]);
```

$$(e_{-1,\,0})\ \&@\ (v_{0,\,1}) \quad (16.4)$$

```
> repsimplify(e[1,-1]&@v[0,1]);
```

$$(e_{1,\,-1})\ \&@\ (v_{0,\,1}) \quad (16.5)$$

```
> repsimplify(e[1,0]&@v[0,1]);
```

$$0 \quad (16.6)$$

```
> repsimplify(e[1,0]&@e[-1,0]&@v[0,1]);
```

$$0 \quad (16.7)$$

```
> repsimplify(e[-1,1]&@e[1,-1]&@v[0,1]);
```

$$v_{0,\,1} \quad (16.8)$$

In this example we construct a Verma module for affine A[1] Lie algebra with the Z-grading, where we define the highest weigth with a function, rather than with the list of rules.

```
> affine(affsl2, A[1], [[r1+2*k], [2*k], [0]]);
```

$$affsl2 = [e_{r1,\,k},\ h_{i1,\,k},\ K],\ [[r1+2\,k],\ [2\,k],\ [0]] \quad (16.9)$$

```
> wt(h[1,3]);
```

$$[6] \quad (16.10)$$

```
> wt(e[-1,1]);
```

$$[1] \quad (16.11)$$

```
> unienv(affsl2);
```

$$U\_affsl2 = [e_{r1,\,k},\ h_{i1,\,k},\ K],\ [[r1+2\,k],\ [2\,k],\ [0]] \quad (16.12)$$

```
> mu := proc(x)
    if (x = h[1,0]) then RETURN(0); fi;
    if (x = h[2,0]) then RETURN(0); fi;
    if (x = K) then RETURN(1); fi;
  end proc;
```

$\mu := \textbf{proc}(x)$ (16.13)

  if $x = h[1,0]$ **then** $RETURN(0)$ **end if**;
  if $x = h[2,0]$ **then** $RETURN(0)$ **end if**;

```
    if x = K then RETURN(1) end if
end proc
> hwrep(Maff, affsl2, v, mu);
```
$$Maff = v \quad (16.14)$$
```
> using();
```
$$U\_affsl2 = [e_{r1,k}, h_{i1,k}, K] \quad (16.15)$$
```
> repsimplify(e[-1,0]&@v);
```
$$(e_{-1,0})\ \&@\ v \quad (16.16)$$
```
> repsimplify(h[1,0]&@v);
```
$$0 \quad (16.17)$$
```
> repsimplify(h[1,1]&@v);
```
$$0 \quad (16.18)$$
```
> repsimplify(h[1,-1]&@v);
```
$$(h_{1,-1})\ \&@\ v \quad (16.19)$$
```
> repsimplify(e[1,-1]&@v);
```
$$(e_{1,-1})\ \&@\ v \quad (16.20)$$
```
> wt(e[1,-1]);
```
$$[-1] \quad (16.21)$$
```
> repsimplify(e[1,1]&@e[-1,-1]&@v);
```
$$v \quad (16.22)$$

Verma modules with generic highest weights

We can also use indeterminants in the highest weight. These indeterminants, however, should be declared as consants in order for this to work.

```
> constants:=constants, s, r;
```
$$constants := false, \gamma, \infty, true, Catalan, FAIL, \pi, s, r \quad (17.1)$$
```
> unienv(Vir);
```
$$U\_Vir = [e_\alpha, z], [[\alpha], [0]] \quad (17.2)$$
```
> hwrep(V, Vir, u, [z=s, e[0]=r]);
```
$$V = u_{s,r} \quad (17.3)$$
```
> repsimplify(e[2]&@u[s,r]);
```
$$0 \quad (17.4)$$
```
> repsimplify(e[-1]&@u[s,r]);
```
$$(e_{-1})\ \&@\ (u_{s,r}) \quad (17.5)$$
```
> w := repsimplify((e[-1]&^2)&@e[-2]&@u[s,r]);
```
$$w := 2\,(e_{-4})\ \&@\ (u_{s,r}) - 2\,((e_{-3})\ \&@\ (e_{-1}))\ \&@\ (u_{s,r}) + ((e_{-2})\ \&@\ ((e_{-1})\ \&\hat{}\ 2))\ \&@\ (u_{s,r}) \quad (17.6)$$
```
> repsimplify((e[2]&^2)&@w);
```
$$(-48\,r^2 + 6\,r\,s + 156\,r - 6\,s)\ u_{s,r} \quad (17.7)$$
```
> repsimplify(e[1]&@u[s,r]);
```
$$0 \quad (17.8)$$
```
> repsimplify(e[2]&@e[-2]&@u[s,r]);
```
(17.9)

$$\left(-4\,r+\frac{1}{2}\,s\right)u_{s,\,r} \tag{17.9}$$

Singular vectors

We can search for singular vectors in a highest weight module. These are homogeneous (with respect to a given Z-grading) vectors that are annihilated by the positive subalgebra. Singular vectors may be used to construct submodules and quotients. For example, for the Virasoro algebra, every submodule in a Verma module is generated by singular vectors [B.L.Feigin, D. B. Fuks, *Verma modules over the Virasoro algebra*, Functional Analysis and its Applications, 17.3 (1983), 241 -242.]. Below we show how to find a singular vector in one of the minimal model modules for the Virasoro algebra. The arguments to *singularvector* function are the weight of the homogeneous component of the Verma module where we search for the singular vector, the list of homogeneous raising operators and (optionally) the name of the module. The function computes vectors in the specified component of the Verma module that are annihilated simultaneously by all raising operators from the list. These raising operators can be arbitrary homogeneous elements of the univesal enveloping algebra with non-negative weights, but typically are chosen to be the generators of the positive subalgebra in the given Lie algebra. For the Virasoro algebra, its positive subalgebra is generated by e[1] and e[2].

```
> using(Vir);
```

$$Vir=\left[e_{\alpha},\,z\right] \tag{18.1}$$

```
> hwrep(MM, Vir, v, [e[0]=-3/8, z=-2]);
```

$$MM=v_{-\frac{3}{8},\,-2} \tag{18.2}$$

```
> singularvector([-4],[e[1],e[2]], MM);
```

$$\left[\frac{3}{2}\left(e_{-4}\right)\,\&@\,\left(v_{-\frac{3}{8},\,-2}\right)+\left(\left(e_{-3}\right)\,\&@\,\left(e_{-1}\right)\right)\,\&@\,\left(v_{-\frac{3}{8},\,-2}\right)+\frac{9}{4}\left(\left(e_{-2}\right)\,\&\wedge\,2\right)\,\&@\,\left(v_{-\frac{3}{8},\,-2}\right)+5\left(\left(e_{-2}\right)\,\&@\,\left(\left(e_{-1}\right)\,\&\wedge\,2\right)\right)\,\&@\,\left(v_{-\frac{3}{8},\,-2}\right)+\left(\left(e_{-1}\right)\,\&\wedge\,4\right)\,\&@\,\left(v_{-\frac{3}{8},\,-2}\right)\right] \tag{18.3}$$

```
> u:=op(1,%);
```

$$u:=\frac{3}{2}\left(e_{-4}\right)\,\&@\,\left(v_{-\frac{3}{8},\,-2}\right)+\left(\left(e_{-3}\right)\,\&@\,\left(e_{-1}\right)\right)\,\&@\,\left(v_{-\frac{3}{8},\,-2}\right)+\frac{9}{4}\left(\left(e_{-2}\right)\,\&\wedge\,2\right)\,\&@\,\left(v_{-\frac{3}{8},\,-2}\right)+5\left(\left(e_{-2}\right)\,\&@\,\left(\left(e_{-1}\right)\,\&\wedge\,2\right)\right)\,\&@\,\left(v_{-\frac{3}{8},\,-2}\right)+\left(\left(e_{-1}\right)\,\&\wedge\,4\right)\,\&@\,\left(v_{-\frac{3}{8},\,-2}\right) \tag{18.4}$$

```
> repsimplify(e[1]&@u);
  repsimplify(e[2]&@u);
  repsimplify(e[3]&@u);
  repsimplify(e[4]&@u);
```

$$0$$

$$0$$
$$0$$
$$0 \qquad (18.5)$$

Submodules and Quotients

We can use singular vectors to generate a submodule in a highest weight module and then take the quotient by this submodule. Let us do this for the basic module for affine A[1] algebra

```
> using(affsl2KM);
```

$$affsl2KM = [e_{r1,\,k},\, h_{i1,\,k},\, K] \qquad (19.1)$$

```
> hwrep(MaffKM, affsl2KM, v, [h[1,0]=0, K=1]);
```

$$MaffKM = v_{0,\,1} \qquad (19.2)$$

```
> submodule(SM, [e[-1,0]&@v[0,1], (e[1,-1]&^2)&@v[0,1]], MaffKM);
```

$$SM = [(e_{-1,\,0})\ \&@\ (v_{0,\,1}),\ ((e_{1,\,-1})\ \&\wedge\ 2)\ \&@\ (v_{0,\,1})] \qquad (19.3)$$

```
> quotrep(Basic, SM);
```

$$Basic = [MaffKM,\ SM] \qquad (19.4)$$

```
> repsimplify(e[1,-1]&@e[1,-1]&@v[0,1], Basic);
```

$$0 \qquad (19.5)$$

```
> basis([-2,-2], MaffKM); nops(%);
```

$$[(h_{1,\,-2})\ \&@\ (v_{0,\,1}),\ ((e_{-1,\,-1})\ \&@\ (e_{1,\,-1}))\ \&@\ (v_{0,\,1}),\ ((h_{1,\,-1})\ \&\wedge\ 2)\ \&@\ (v_{0,\,1}),\ ((e_{-1,\,0})\ \&@\ (e_{1,\,-2}))\ \&@\ (v_{0,\,1}),\ (((e_{-1,\,0})\ \&@\ (h_{1,\,-1}))\ \&@\ (e_{1,\,-1}))\ \&@\ (v_{0,\,1}),\ (((e_{-1,\,0})\ \&\wedge\ 2)\ \&@\ ((e_{1,\,-1})\ \&\wedge\ 2))\ \&@\ (v_{0,\,1})]$$
$$6 \qquad (19.6)$$

```
> basis([-2,-2], Basic); nops(%);
```

$$[((h_{1,\,-1})\ \&\wedge\ 2)\ \&@\ (v_{0,\,1}),\ (((e_{-1,\,0})\ \&@\ (h_{1,\,-1}))\ \&@\ (e_{1,\,-1}))\ \&@\ (v_{0,\,1})]$$
$$2 \qquad (19.7)$$

```
> basis([-2,-2], SM); nops(%);
```

$$\Big[(h_{1,\,-2})\ \&@\ (v_{0,\,1}) + (((e_{-1,\,0})\ \&@\ (h_{1,\,-1}))\ \&@\ (e_{1,\,-1}))\ \&@\ (v_{0,\,1}),\ ((e_{-1,\,-1})\ \&@\ (e_{1,\,-1}))\ \&@\ (v_{0,\,1}) - \frac{1}{2}\,((h_{1,\,-1})\ \&\wedge\ 2)\ \&@\ (v_{0,\,1}) - \frac{1}{2}\,(((e_{-1,\,0})\ \&@\ (h_{1,\,-1}))\ \&@\ (e_{1,\,-1}))\ \&@\ (v_{0,\,1}),\ ((e_{-1,\,0})\ \&@\ (e_{1,\,-2}))\ \&@\ (v_{0,\,1}) - (((e_{-1,\,0})\ \&@\ (h_{1,\,-1}))\ \&@\ (e_{1,\,-1}))\ \&@\ (v_{0,\,1}),\ (((e_{-1,\,0})\ \&\wedge\ 2)\ \&@\ ((e_{1,\,-1})\ \&\wedge\ 2))\ \&@\ (v_{0,\,1})\Big]$$
$$4 \qquad (19.8)$$

Implementation of vertex algebras

We can use highest weight representations to implement vertex algebras, computing their fields and n-th products. Since vertex algebras are infinite-dimensional, we set up a cut-off N, and work within the first N+1 graded components of the vertex algebra. Then in each field we keep only the components that change the degree by at most N. For this reason, we will be indexing components of the fields by degrees, rather than by n-th products/powers of z.

Below we consider the universal enveloping vertex algebra for *sl(2),* set up its generating fields, as well as the Sugawara field, and check that Sugawara field satisfies the OPE relations of the Virasoro vertex algebra.

```
> N:=4;
```

$$N := 4 \qquad (20.1)$$

```
> affine(vertsl2, A[1], [[k],[k],[0]]);
```

$$vertsl2 = [e_{r1,\,k},\ h_{i1,\,k},\ K],\ [[k],\ [k],\ [0]] \qquad (20.2)$$

```
> constants:=constants, C;
```

$$constants := false,\ \gamma,\ \infty,\ true,\ Catalan,\ FAIL,\ \pi,\ s,\ r,\ C \qquad (20.3)$$

```
> hwrep(Vaff, vertsl2, v, [e[-1,0]=0, e[1,0]=0, h[1,0]=0, K=C]);
```

$$Vaff = v_{0,\,0,\,0,\,C} \qquad (20.4)$$

```
> u:=op(2,%);
```

$$u := v_{0,\,0,\,0,\,C} \qquad (20.5)$$

Let us set up the generating fields:

```
> Field_e := Array(-N..N): Field_h := Array(-N..N): Field_f :=
  Array(-N..N):
  for i from -N to N do
     Field_e[i] := e[1,i];
     Field_h[i] := h[1,i];
     Field_f[i] := e[-1,i];
  od:
```

Next let us introduce a procedure that computes normally ordered product of two fields. Its arguments are the two fields and the degree (=conformal dimension) of the first factor.

```
> NOP := proc(X, Y, degX)
  global N;
  local i,j,Z;

  Z := Array(-N..N);
  for i from -N to N do
      Z[i]:=0;
  end;

  for i from 1-degX to N do
      for j from -N to N do
           if (abs(i+j) <= N) then
               Z[i+j] := Z[i+j] + Y[j]&@X[i];
           end if;
      end;
  end;

  for i from -N to -degX do
      for j from -N to N do
           if (abs(i+j) <= N) then
```

```
              Z[i+j] := Z[i+j] + X[i]&@Y[j];
           end if;
       end;
   end;

   RETURN(eval(Z));
   end proc:
```

Here is a procedure for differentiation of fields

```
> Der := proc(X, degX)

   global N;
   local i, Z;

   Z:=array(-N..N);
   for i from -N to N do
      Z[i]:=(-i-degX)*X[i];
   end;

   RETURN(Z);
   end proc:
```

Next let us define the Sugawara field. Note that for component-wise arithmetic operations on arrays we should use operations with ~, for example +~

```
> L:=(1/(2*(C+2)))*~(NOP(Field_e, Field_f, 1) +~ NOP(Field_f,
  Field_e, 1) +~ (1/2)*~NOP(Field_h, Field_h, 1)):
```

```
> L[0];
```

$$\frac{1}{2C+4}\Big((e_{-1,0})\ \&@\ (e_{1,0}) + 2\,(e_{-1,-1})\ \&@\ (e_{1,1}) + 2\,(e_{-1,-2})\ \&@\ (e_{1,2}) + 2\,(e_{-1,-3})\ \&@\ (e_{1,3}) + 2\,(e_{-1,-4})\ \&@\ (e_{1,4}) + 2\,(e_{1,-4})\ \&@\ (e_{-1,4}) + 2\,(e_{1,-3})\ \&@\ (e_{-1,3}) + 2\,(e_{1,-2})\ \&@\ (e_{-1,2}) + 2\,(e_{1,-1})\ \&@\ (e_{-1,1}) + (e_{1,0})\ \&@\ (e_{-1,0}) + \frac{1}{2}\,(h_{1,0})\ \&@\ (h_{1,0}) + (h_{1,-1})\ \&@\ (h_{1,1}) + (h_{1,-2})\ \&@\ (h_{1,2}) + (h_{1,-3})\ \&@\ (h_{1,3}) + (h_{1,-4})\ \&@\ (h_{1,4})\Big) \tag{20.6}$$

```
> omega := repsimplify(L[-2]&@u);
```

$$\omega := \frac{2\,((e_{1,-1})\ \&@\ (e_{-1,-1}))\ \&@\ (v_{0,0,0,C})}{2C+4} - \frac{(h_{1,-2})\ \&@\ (v_{0,0,0,C})}{2C+4} + \frac{1}{2}\,\frac{((h_{1,-1})\ \&\wedge\ 2)\ \&@\ (v_{0,0,0,C})}{2C+4} \tag{20.7}$$

```
> Domega := repsimplify(L[-3]&@u);
```

$$Domega := \frac{2\,((e_{1,-2})\ \&@\ (e_{-1,-1}))\ \&@\ (v_{0,0,0,C})}{2C+4} + \frac{2\,((e_{-1,-2})\ \&@\ (e_{1,-1}))\ \&@\ (v_{0,0,0,C})}{2C+4} \tag{20.8}$$

$$+\frac{((h_{1,-2})\ \&@\ (h_{1,-1}))\ \&@\ (v_{0,0,0,C})}{2C+4}$$

Let us check the Virasoro OPEs: omega_(0) omega = Domega, omega_(1) omega = 2*omega, omega_(2) omega = 0, omega_(3) omega = K/2*1.

```
> repsimplify(L[-1]&@omega - Domega);
```

$$\left(\frac{4\,(e_{1,-2})\ \&@\ (e_{-1,-1})\ C}{(2C+4)^2}+\frac{4\,(e_{-1,-2})\ \&@\ (e_{1,-1})\ C}{(2C+4)^2}+\frac{2\,(h_{1,-2})\ \&@\ (h_{1,-1})\ C}{(2C+4)^2}+\frac{8\,(e_{1,-2})\ \&@\ (e_{-1,-1})}{(2C+4)^2}+\frac{8\,(e_{-1,-2})\ \&@\ (e_{1,-1})}{(2C+4)^2}+\frac{4\,(h_{1,-2})\ \&@\ (h_{1,-1})}{(2C+4)^2}-\frac{2\,(e_{1,-2})\ \&@\ (e_{-1,-1})}{2C+4}-\frac{2\,(e_{-1,-2})\ \&@\ (e_{1,-1})}{2C+4}-\frac{(h_{1,-2})\ \&@\ (h_{1,-1})}{2C+4}\right)v_{0,0,0,C} \quad (20.9)$$

```
> simplify(%);
```

$$0 \quad (20.10)$$

```
> repsimplify(L[0]&@omega - 2*omega);
```

$$\left(-\frac{4\,(e_{1,-1})\ \&@\ (e_{-1,-1})}{2C+4}+\frac{2\,h_{1,-2}}{2C+4}-\frac{(h_{1,-1})\ \&\wedge\ 2}{2C+4}+\frac{8\,(e_{1,-1})\ \&@\ (e_{-1,-1})\ C}{(2C+4)^2}-\frac{4\,h_{1,-2}\,C}{(2C+4)^2}+\frac{2\,C\,(h_{1,-1})\ \&\wedge\ 2}{(2C+4)^2}+\frac{16\,(e_{1,-1})\ \&@\ (e_{-1,-1})}{(2C+4)^2}-\frac{8\,h_{1,-2}}{(2C+4)^2}+\frac{4\,(h_{1,-1})\ \&\wedge\ 2}{(2C+4)^2}\right)v_{0,0,0,C} \quad (20.11)$$

```
> simplify(%);
```

$$0 \quad (20.12)$$

```
> repsimplify(L[1]&@omega);
```

$$0 \quad (20.13)$$

```
> repsimplify(L[2]&@omega);
```

$$\left(\frac{12\,C}{(2C+4)^2}+\frac{6\,C^2}{(2C+4)^2}\right)v_{0,0,0,C} \quad (20.14)$$

```
> simplify(%);
```

Central charge of the Virasoro is 3C/(C+2).

$$\frac{3\,C\,v_{0,0,0,C}}{2C+4} \quad (20.15)$$

Characteristic p

All of the computations described in this package may be carried out in positive characteristic.

```
> characteristic(5);
```

$$characteristic = 5 \quad (21.1)$$

```
> simple(sl3, A[2]);
```

$$sl3 = [e_{k1,\,k2}, h_{i1}], [[k1, k2], [0, 0]] \tag{21.2}$$

```
> hwrep(Verma, sl3, v, [h[1]=2, h[2]=3]);
```

$$Verma = v_{2,\,3} \tag{21.3}$$

```
> singularvector([-2,-2], [e[1,0],e[0,1]], Verma);
```

This is in agreement with modular representation theory (J.C.Jantzen, *Modular representations of reductive Lie algebras*, Journal of Pure and Applied Algebra, 152 (2000), 133-185).

$$\begin{aligned}&[((e_{-1,\,-1})\ \&\hat{}\ 2)\ \&@\ (v_{2,\,3})\\ &\quad + 4\,(((e_{-1,\,0})\ \&@\ (e_{-1,\,-1}))\ \&@\ (e_{0,\,-1}))\ \&@\ (v_{2,\,3})\\ &\quad + (((e_{-1,\,0})\ \&\hat{}\ 2)\ \&@\ ((e_{0,\,-1})\ \&\hat{}\ 2))\ \&@\ (v_{2,\,3})]\end{aligned} \tag{21.4}$$

```
> characteristic(0);
```

$$characteristic = 0 \tag{21.5}$$

```
> singularvector([-2,-2], [e[1,0],e[0,1]], Verma);
```

$$[\ ] \tag{21.6}$$