LieRing

Computing with finitely presented Lie rings

Version 2.4.2

10 February 2022

Serena Cicalò

Willem Adriaan de Graaf

Serena Cicalò

Email: cicalo@science.unitn.it

Address: Serena Cicalò
Dipartimento di Matematica e Informatica
Via Ospedale 72
Italy

Willem Adriaan de Graaf

Email: degraaf@science.unitn.it

Homepage: http://www.science.unitn.it/~degraaf

Abstract

This package provides functions for constructing and working with Lie rings. There are functions for dealing with finitely-presented Lie rings, and for performing the Lazard correspondence. The package also contains a small database of finitely-generated Lie rings satisfying an Engel condition.

Copyright

Contents

Chapter 1

Introduction

1.1 Preliminaries

A Lie ring L is a $\mathbb{Z}$-module equipped with a multiplication, denoted by a bracket $[\,,\,]$ with

- $[x,x] = 0$ for all x in L,

- $[x,[y,z]] + [z,[x,y]] + [y,[z,x]] = 0$ for all x,y,z in L.

Contrary to Lie algebras (which are defined over a field), Lie rings may have torsion elements, i.e., elements $x \neq 0$ such that $mx = 0$ for some $m \in \mathbb{Z}$.

We say that a Lie ring is finite-dimensional if it is finitely-generated as abelian group. All functions of this package deal with finite-dimensional Lie rings.

Here is an example of a Lie ring L of order 5^6. As abelian group L is generated by x_1, x_2, x_3, x_4, x_5. We have $5x_i = 0$ for $i = 1, \ldots, 4$, and $25x_5 = 0$. Furthermore,

$$[x_1, x_4] = 4x_2 + 5x_5, \ [x_3, x_4] = 4x_1, \ [x_3, x_5] = 4x_2, \ [x_4, x_5] = 4x_3.$$

One of the main functions of this package constructs a Lie ring given by a multiplication table (as above) from a finite presentation. The Lie ring above can be obtained as follows.

```
──────────────────────────── Example ────────────────────────────
gap> L:= FreeLieRing( Integers, ["a","b"] );
<Free algebra over Integers generators: a, b >
gap> a:= L.1; b:= L.2;
a
b
gap> S:= [ 5*a-(b*a)*a-((b*a)*b)*b,5*b];
[ (5)*a+(-1)*(a,(a,b))+(b,(b,(a,b))), (5)*b ]
gap> K:= FpLieRing( L, S : maxdeg:= 4 );
<Lie ring with 5 generators>
gap> v:=BasisVectors( Basis(K) );
[ v_1, v_2, v_3, v_4, v_5 ]
gap> v[1]*v[4];
4*v_2+5*v_5
gap> Torsion( Basis(K) );
[ 5, 5, 5, 5, 25 ]
```

1.2 The free Lie ring

Let X be a set of letters, which we denote by $x_1, \ldots, x_n$. Then the free magma $M(X)$ on X is defined to be the set of all bracketed expressions in the elements of X. More precisely, we have that X is a subset of $M(X)$ and if $a, b \in M(X)$, then also $(a, b) \in M(X)$. The free magma has a natural binary operation m with $m(a, b) = (a, b)$.

The elements of the free magma have a degree which is defined as $\deg(a, b) = \deg(a) + \deg(b)$. The degree of the elements of X can be set to be any positive integer. (Usually this is 1, but it is possible to use different degrees for the elements of X.)

Let R be a ring; then the free algebra $A_R(X)$ on X over R is the R-span of $M(X)$. The product on $A_R(X)$ is obtained by bilinearly extending the map m.

The elements of $M(X)$ are called monomials of $A_R(X)$. We use the following ordering on them. The elements of X are ordered arbitrarily. Then $(a, b) < (c, d)$ if $\deg(a, b) < \deg(c, d)$. If these two numbers are equal, then $(a, b) < (c, d)$ if $a < c$, and in case $a = c$, if $b < d$. Using this ordering we can speak of leading monomial, and leading coefficient of an element of $A_R(X)$. Using these notions one can develop a Groebner basis theory for ideals in $A_R(X)$ (see [CdG07] and [CdG09]).

Let J be the ideal of $A_R(X)$ generated by all elements

- (a, a),

- $(a, b) + (b, a)$,

- $(a, (b, c)) + (c, (a, b)) + (b, (c, a))$,

for $a, b, c \in M(X)$. Set $L_R(X) = A_R(X)/J$, which is called the free Lie ring over R generated by X.

The free Lie ring is one of the central objects of this package. It can be defined over the integers, or over a field. The free Lie rings that can be constructed using this package rewrite their elements using anticommutativity. The Jacobi identity is not used for rewriting; this is because that would lead to expression swell, and sometimes tedious rewriting of elements to a form in which that can no longer be recognised. So, strictly speaking, we work with the free anticommutative algebra.

1.3 The Lazard correspondence

Using the Baker-Campbell-Hausdorff (or BCH) formula one can define an associative multiplication on a nilpotent Lie ring of order p^n and nilpotency class $< p$. This makes the Lie ring into a p-group of the same order and nilpotency class. The BCH-formula also has inverses, which can be used to define an addition and a Lie bracket on a p-group of class $< p$. These make the group into a Lie ring of the same order and nilpotency class.

These two operations are mutually inverse, and so define an equivalence of the categories of p-groups of class $< p$ and nilpotent Lie rings of the same order and nilpotency class. This equivalence is known as the *Lazard correspondence* (see [Khu98]). This package has functions for performing this correspondence, i.e., to make a p-group into a Lie ring and vice versa. For the algorithms used we refer to [CdGVL11].

Chapter 2

The functions

2.1 The free Lie ring

2.1.1 FreeLieRing

▷ FreeLieRing(*R, names*)   (method)
▷ FreeLieRing(*R, names, deg*)   (method)
▷ FreeLieRing(*R, k*)   (method)
▷ FreeLieRing(*R, k, deg*)   (method)

    Here `R` is a ring, which has to be either the integers, or a field. `names` is a list of strings, which will be the names of the generators. This function returns the free Lie ring over `R`, with generators named as in `names` . If `L` denotes the output, then `L.i` will be the i-th generator. If a third argument `deg` is given then this must be a list of positive integers. Then each generator will have a degree equal to the corresponding element of the list `deg`.

    Monomials in the free Lie ring of the form (a,b) with $a > b$ are automatically rewritten as $-(b,a)$. Monomials of the form (a,a) are rewritten as zero. There is no other rewriting done. Therefore, the object returned by this function is strictly speaking not the same as the free Lie ring, it rather is the free anticommutative algebra.

    Monomials in the free Lie ring are printed as bracketed expressions. In a printed element the monomials appear in increasing order; in particular the last monomial is the leading monomial.

    If instead of the list `names` a positive integer `k` is given, then the free Lie ring on that number of generators is returned. Again we can give each generator a degree different from 1 by adding a third argument `deg`.

```
───────────────────────────── Example ─────────────────────────────
  gap> L:= FreeLieRing( Integers, ["a","b"] );
  <Free algebra over Integers generators: a, b >
  gap> a:= L.1; b:= L.2;
  a
  b
  gap> (a*b)*b+2*a*b;
  (2)*(a,b)+(-1)*(b,(a,b))
```

2.1.2 Degree

▷ Degree(*f*) (operation)

Here *f* is an element of a free Lie ring. Its degree is returned.

```
_____ Example _____
gap> L:= FreeLieRing( Integers, ["a","b"] );;
gap> a:= L.1;; b:= L.2;;
gap> f:=(a*b)*b+2*a*b;
(2)*(a,b)+(-1)*(b,(a,b))
gap> Degree(f);
3
```

2.2 Creating Lie rings

The package can deal with finite-dimensional Lie rings given by a multiplication table (which follow the format for multiplication tables in the GAP library), and a list of moduli. This list has to have the same length as the number of basis elements of the Lie ring. If the i-th element of this list is *m* then the additive order of the i-th basis edlement if *m*. If $m = 0$ then the additive order is infinite.

2.2.1 IsLieRing

▷ IsLieRing (filter)

This is the category of finite-dimensional Lie rings.

2.2.2 LieRingByStructureConstants

▷ LieRingByStructureConstants(*tor*, *T*) (operation)

Here *T* is a multiplication table, and *tor* is a list of moduli. This function returns the corresponding Lie ring. In the example below we create the Lie ring with basis elements x, y, z, with $[x, y] = z$, $3x = 6y = 3z = 0$.

The multiplication table has to be created using the GAP functions for constructing multiplication tables of Lie algebras. In particular, we refer to the GAP reference manual for descriptions of the functions EmptySCTable (**Reference: EmptySCTable**) SetEntrySCTable (**Reference: SetEntrySCTable**)

```
_____ Example _____
gap> T:= EmptySCTable( 3, 0, "antisymmetric" );;
gap> SetEntrySCTable( T, 1, 2, [1,3] );
gap> LieRingByStructureConstants( [3,6,3], T );
<Lie ring with 3 generators>
```

2.2.3 FpLieRing

▷ FpLieRing(*L*, *R*) (function)

Here `L` is a free Lie ring defined over the integers, and `R` is a set of elements of `L`. This function returns the Lie ring given by structure constants, that is isomorphic to `L` modulo the ideal generated by `R`.

It is possible to set the option `maxdeg` to a positive value `d`. Then a nilpotent quotient is computed, i.e., all elements of `L` of degree strictly greater than `d` will be treated as relations.

The algebra that is output by this function has an attribute, `CanonicalProjection`, which is a function mapping elements of the free Lie ring `L` to their projections in the output algebra.

The algorithm behind this function has been described in [CdG07] and [CdG09].

```
———————————————— Example ————————————————
 gap> L:= FreeLieRing( Integers, ["x","y"], [1,2] );
 <Free algebra over Integers generators: x, y >
 gap> x:= L.1;; y:= L.2;;
 gap> R:= [((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)*(y,(x,y)),
   (-3)*(x,(x,(x,(x,(x,y)))))+(-20)*(y,(x,(x,(x,y)))) ]
 gap> K:= FpLieRing( L, R : maxdeg:= 15 );
 <Lie ring with 75 generators>
 gap> f:=CanonicalProjection(K);
 function( elm ) ... end
 gap> f(R[1]);
 0
 gap> f(x);
 v_1
```

2.2.4 FpLieAlgebra

▷ FpLieAlgebra(`L, R`)   (function)

This is similar to `FpLieRing`, with the difference that the free Lie ring `L` must be defined over a field. Then the algorithms become a lot faster (in most cases). The result however is a Lie algebra, and not a Lie ring.

2.3 Working with Lie rings

2.3.1 Basis

▷ Basis(`L`)   (operation)

Here `L` a Lie ring. Its basis is returned.

We note that in LieRing Lie rings have one basis that is computed by the system; one should not try to set a basis.

```
———————————————— Example ————————————————
 gap> T:= EmptySCTable( 3, 0, "antisymmetric" );;
 gap> SetEntrySCTable( T, 1, 2, [1,3] );
 gap> K:= LieRingByStructureConstants( [3,6,3], T );
 <Lie ring with 3 generators>
 gap> Basis(K);
 Basis( <Lie ring with 3 generators>, [ v_1, v_2, v_3 ] )
```

```
gap> BasisVectors( Basis(K) );
[ v_1, v_2, v_3 ]
```

2.3.2 StructureConstantsTable

▷ StructureConstantsTable(*B*)   (operation)

Here *B* is the basis of a Lie ring. Its structure constants table is returned.

```
——————— Example ———————
gap> T:= EmptySCTable( 3, 0, "antisymmetric" );;
gap> SetEntrySCTable( T, 1, 2, [1,3] );
gap> K:= LieRingByStructureConstants( [3,6,3], T );
<Lie ring with 3 generators>
gap> StructureConstantsTable( Basis(K) );
[ [ [ [ ], [ ] ], [ [ 3 ], [ 1 ] ], [ [ ], [ ] ] ],
  [ [ [ 3 ], [ -1 ] ], [ [ ], [ ] ], [ [ ], [ ] ] ],
  [ [ [ ], [ ] ], [ [ ], [ ] ], [ [ ], [ ] ] ], -1, 0 ]
```

2.3.3 Torsion

▷ Torsion(*B*)   (operation)

Here *B* is the basis of a Lie ring. The list of torsion moduli of its basis elements is returned.

```
——————— Example ———————
gap> T:= EmptySCTable( 3, 0, "antisymmetric" );;
gap> SetEntrySCTable( T, 1, 2, [1,3] );
gap> K:= LieRingByStructureConstants( [3,6,3], T );
<Lie ring with 3 generators>
gap> Torsion( Basis(K) );
[ 3, 6, 3 ]
```

2.3.4 Coefficients

▷ Coefficients(*B*, *elm*)   (operation)

Here *B* is the basis of a Lie ring, and *elm* is an element of the same Lie ring. The coefficients of *elm* with respect to *B* are returned.

```
——————— Example ———————
gap> L:= FreeLieRing( Integers, ["x","y"] );; x:= L.1;; y:= L.2;;
gap> rr:=[((y*x)*x)*x-6*(y*x)*y, 3*((((y*x)*x)*x)*x)*x-20*(((y*x)*x)*x)*y ];;
gap> K:= FpLieRing( L, rr : maxdeg:= 6 );;
gap> C:=LieCentre(K);
<Lie ring with 9 generators>
gap> Coefficients( Basis(K), Basis(C)[6] );
[ 5, 5, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
gap> Coefficients( Basis(C), Basis(C)[6] );
[ 0, 0, 0, 0, 0, 1, 0, 0, 0 ]
```

2.3.5 SubLieRing

▷ SubLieRing(*L, gens[, string]*) (operation)

Here `L` is a Lie ring, and `gens` a list of elements of `L`. This function constructs the subring generated by the elements in `gens`. If these elements are known to form a basis of the subalgebra, then as a third argument the string `"basis"` can be added. That makes the execution of the function a lot faster.

This function depends on hermite and Smith normal form computations. Therefore in practice, for bigger inputs, it can be slow.

```
————————————————————— Example —————————————————————
  gap> L:= FreeLieRing( Integers, ["x","y"] );;
  gap> x:= L.1;; y:= L.2;;
  gap> rr:=[((y*x)*x)*x-6*(y*x)*y, 3*(((((y*x)*x)*x)*x)*x-20*(((y*x)*x)*x)*y ];;
  gap> K:= FpLieRing( L, rr : maxdeg:= 8 );
  <Lie ring with 41 generators>
  gap> b:= Basis(K);;
  gap> M:= SubLieRing( K, [ b[30], b[40] ] );
  <Lie ring with 6 generators>
  gap> Torsion(Basis(M));
  [ 3, 6, 6, 12, 360, 0 ]
  gap> Basis(M)[2];
  3*v_2+2*v_3+2*v_10+4*v_12+4*v_13+5*v_14+v_15+3*v_17+3*v_18+6*v_20+10*v_22+6*v_
  24+6*v_25+10*v_26+4*v_27+18*v_28+30*v_29+60*v_30+360*v_31+5040*v_32
```

2.3.6 LieRingIdeal

▷ LieRingIdeal(*L, gens[, string]*) (operation)

This is the same as `SubLieRing` except that the output is an ideal (on the level of data structures that is the same as a Lie subring).

2.3.7 NaturalHomomorphismByIdeal

▷ NaturalHomomorphismByIdeal(*L, I*) (operation)

Here `L` is a Lie ring, and `I` an ideal of `L`. This function constructs the canonical projection of `L` on the quotient of `L` by `I`.

We remark that it is *not checked* whether `I` is an ideal or not. if `I` is just a subalgebra, then nothing is guaranteed about the result of this function.

Also this function depends on Smith normal form computations; therefore it can be slow on bigger inputs.

```
————————————————————— Example —————————————————————
  gap> L:= FreeLieRing( Integers, ["x","y"] );;
  gap> x:= L.1;; y:= L.2;;
  gap> rr:=[((y*x)*x)*x-6*(y*x)*y, 3*(((((y*x)*x)*x)*x)*x-20*(((y*x)*x)*x)*y ];;
  gap> K:= FpLieRing( L, rr : maxdeg:= 8 );;
  gap> b:= Basis(K);;
  gap> I:= LieRingIdeal( K, [ b[29] ] );
  <Lie ring with 23 generators>
```

```
gap> f:= NaturalHomomorphismByIdeal( K, I );;
gap> M:= Range(f);
<Lie ring with 27 generators>
gap> Torsion(Basis(M));
[ 2, 2, 2, 2, 2, 2, 2, 2, 2, 6, 6, 6, 12, 12, 12, 120, 720, 10080, 0, 0, 0,
  0, 0, 0, 0, 0, 0 ]
gap> Image( f, b[30] );
v_16+716*v_17
gap> PreImagesRepresentative( f, Basis(M)[10] );
4*v_2+4*v_3+4*v_4+4*v_5+5*v_6+v_7+5*v_8+v_9+5*v_10+v_11+5*v_12+v_13+5*v_14+v_
24+v_25+11*v_26+v_29+10*v_30+100*v_31
```

2.3.8 LieLowerCentralSeries

▷ LieLowerCentralSeries(*L*)   (operation)

Here *L* is a Lie ring. Its lower central series is returned.

This repeatedly constructs ideals of *L*; therefore also this function can be rather slow on bigger inputs.

```
──────────────────── Example ────────────────────
gap> L:= FreeLieRing( Integers, ["x","y"] );; x:= L.1;; y:= L.2;;
gap> rr:=[((y*x)*x)*x-6*(y*x)*y, 3*((((y*x)*x)*x)*x)*x-20*(((y*x)*x)*x)*y ];;
gap> K:= FpLieRing( L, rr : maxdeg:= 7 );;
gap> LieLowerCentralSeries(K);
[ <Lie ring with 26 generators>, <Lie ring with 24 generators>,
  <Lie ring with 23 generators>, <Lie ring with 22 generators>,
  <Lie ring with 21 generators>, <Lie ring with 19 generators>,
  <Lie ring with 16 generators>, <Lie ring with 0 generators> ]
```

2.3.9 LieLowerPCentralSeries

▷ LieLowerPCentralSeries(*L*, *p*)   (operation)

Here *L* is a Lie ring, and *p* is a prime. The lower *p*-central series of *L* is returned. This is the series where the L^{k+1} is generated by $[L, L^k]$ and pL^k. Note that this may not be a finite series, if *L* is not of exponent p^n (as abelian group). The function does not check this; if the series is infinite, then it will loop forever.

This repeatedly constructs ideals of *L*; therefore also this function can be rather slow on bigger inputs.

```
──────────────────── Example ────────────────────
gap> L:= FreeLieRing( Integers, ["x","y"] );; x:= L.1;; y:= L.2;;
gap> rr:=[((y*x)*x)*x-7*(y*x)*y, 7*((((y*x)*x)*x)*x)*x-49*(((y*x)*x)*x)*y,
> 7*x, 49*y ];;
gap> K:= FpLieRing( L, rr : maxdeg:= 5 );;
gap> LieLowerPCentralSeries(K,7);
[ <Lie ring with 11 generators>, <Lie ring with 10 generators>,
  <Lie ring with 8 generators>, <Lie ring with 6 generators>,
  <Lie ring with 4 generators>, <Lie ring with 0 generators> ]
```

2.3.10 LieCentre

▷ LieCentre(*L*) (operation)

Here *L* is a Lie ring. Its centre is returned.

```
────────────────────── Example ──────────────────────
gap> L:= FreeLieRing( Integers, ["x","y"] );; x:= L.1;; y:= L.2;;
gap> rr:=[((y*x)*x)*x-6*(y*x)*y, 3*((((y*x)*x)*x)*x)*x-20*(((y*x)*x)*x)*y ];;
gap> K:= FpLieRing( L, rr : maxdeg:= 7 );;
gap> LieCentre(K);
<Lie ring with 16 generators>
gap> Torsion( Basis(K) );
[ 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 12, 12, 12, 12, 360, 5040, 0, 0, 0, 0,
  0, 0, 0, 0 ]
```

2.3.11 TensorWithField

▷ TensorWithField(*L, F*) (operation)

Here *L* is a Lie ring, and *F* is a field. This function returns the Lie algebra that is obtained by tensoring *L* with *F*.

```
────────────────────── Example ──────────────────────
gap> T:= EmptySCTable( 3, 0, "antisymmetric" );;
gap> SetEntrySCTable( T, 1, 2, [1,3] );
gap> K:= LieRingByStructureConstants( [3,6,3], T );;
gap> TensorWithField( K, GF(3) );
<Lie algebra of dimension 3 over GF(3)>
gap> TensorWithField( K, GF(2) );
<Lie algebra of dimension 1 over GF(2)>
gap> Dimension( TensorWithField( K, GF(5) ) );
0
```

2.4 The Lazard correspondence

By the Lazard correspondence we can put a Lie ring structure on a p-group of class $< p$. Conversely, we can define a group structure on a nilpotent Lie ring of order p^n and class $< p$. The package contains functions for doing this effectively. However, we do not work with a single object having both the structure of a p-group and a Lie ring. Rather we define two objects, a p-group and a Lie ring, along with bijections between the two. Our programs use the BCH-formula and its inverses, that have been stored in a file, truncated at weight 14. This means that currently the package is able to deal with groups and algebras up to class 14. The underlying algorithms have been described in [CdGVL11]

2.4.1 PGroupToLieRing

▷ PGroupToLieRing(*G*) (attribute)

Here *G* is a p-group of class $< p$. This function returns a record with four components: *pgroup* (the group *G*), *liering* (the corresponding Lie ring), *GtoL* (a function mapping elements of the

group to elements of the Lie ring), `LtoG` (a function mapping elements of the Lie ring to elements of the group).

```
──────────────────────────── Example ────────────────────────────
gap> F := FreeGroup(IsSyllableWordsFamily,"a","b","c","d", "e", "f", "g");;
gap> a := F.1;; b := F.2;; c := F.3;; d := F.4;; e := F.5;; f := F.6;; g:=F.7;;
gap> rels := [ a^13, b^13/g, c^13, d^13, e^13, f^13, g^13,
> Comm(b,a)/c, Comm(c,a)/d, Comm(d,a)/e, Comm(e,a)/f, Comm(f,a), Comm(g,a),
> Comm(c,b)/(g^11), Comm(d,b)/g, Comm(e,b)/g, Comm(g,b), Comm(d,c)/(g^12),
> Comm(e,c), Comm(f,c), Comm(g,c), Comm(e,d), Comm(f,d), Comm(g,d), Comm(f,e),
> Comm(g,e), Comm(g,f)];;
gap> G := PcGroupFpGroup( F/rels );
<pc group of size 62748517 with 7 generators>
gap> r:= PGroupToLieRing(G);
rec( GtoL := function( g0 ) ... end, LtoG := function( x0 ) ... end,
  liering := <Lie ring with 6 generators>,
  pgroup := <pc group of size 62748517 with 7 generators> )
gap> f:= r.GtoL; h:= r.LtoG;
function( g0 ) ... end
function( x0 ) ... end
gap> L:= r.liering;
<Lie ring with 6 generators>
gap> b:= Basis(L);
Basis( <Lie ring with 6 generators>, [ v_1, v_2, v_3, v_4, v_5, v_6 ] )
gap> h(b[1]);
a^12*c*d^5*e^3*f^8*g^7
gap> f(h(b[1]));
v_1
```

2.4.2 LieRingToPGroup

▷ LieRingToPGroup(L)   (attribute)

Here L is a nilpotent Lie ring of class $< p$ and order p^n. This function returns a record with four components: `pgroup` (the p-group corresponding to L), `liering` (the Lie ring L), `GtoL` (a function mapping elements of the group to elements of the Lie ring), `LtoG` (a function mapping elements of the Lie ring to elements of the group).

```
──────────────────────────── Example ────────────────────────────
gap> L:= FreeLieRing( Integers, ["a","b","c"] );;
gap> a:= L.1;; b:= L.2;; c:= L.3;;
gap> rels:= [ (b*a)*b, c*a, c*b-(b*a)*a, 7^2*a, 7*b-((b*a)*a)*a,
> 7*c-((b*a)*a)*a];;
gap> K:= FpLieRing( L, rels );
<Lie ring with 5 generators>
gap> r:= LieRingToPGroup(K);
rec( GtoL := function( g0 ) ... end, LtoG := function( x0 ) ... end,
  liering := <Lie ring with 5 generators>,
  pgroup := <pc group of size 823543 with 7 generators> )
gap> G:= r.pgroup;; f:= r.LtoG;; h:= r.GtoL;;
gap> u:= 5*Basis(K)[2]+9*Basis(K)[5];
5*v_2+9*v_5
gap> f(u);
```

```
    f3^2*f4^2*f5^6*f7^3
gap> h(f(u));
    5*v_2+9*v_5
```

2.5 The database of n-Engel Lie rings

A Lie ring L is said to satisfy the n-Engel condition if for all $x, y \in L$ we have $(adx)^n(y) = 0$. The package LieRing contains a small database of Lie rings that satisfy an n-Engel condition. They have been computed with the algorithms described in [CdG07] and [CdG09].

Currently the database contains the "freeest" (or "largest") n-Engel Lie rings with k generators for $(n,k) = (3,2),(3,3),(3,4),(4,2),(4,3)$.

2.5.1 SmallNEngelLieRing

▷ SmallNEngelLieRing(n, k)   (operation)

This returns the biggest n-Engel Lie ring with k generators, for the values of n,k indicated above. For other values an error is raised.

```
──────────── Example ────────────
gap> L:= SmallNEngelLieRing( 4, 3 );
<Lie ring with 133 generators>
gap> x:= 10*Basis(L)[1]+7*Basis(L)[10]+19*Basis(L)[89];
7*v_10+19*v_89
gap> ForAll( Basis(L), y -> IsZero( x*(x*(x*(x*y))) ) );
true
gap> K:= TensorWithField( L, GF(3) );
<Lie algebra of dimension 83 over GF(3)>
gap> x:= Random(K);;
gap> ForAll( Basis(K), y -> IsZero( x*(x*(x*(x*y))) ) );
true
```

References

[CdG07]   Serena Cicalò and Willem de Graaf. Non-associative Gröbner bases, finitely-presented Lie rings and the Engel condition. In *ISSAC 2007*, pages 100–107. ACM, New York, 2007. 5, 8, 14

[CdG09]   Serena Cicalò and Willem A. de Graaf. Non-associative Gröbner bases, finitely-presented Lie rings and the Engel condition. II. *J. Symbolic Comput.*, 44(7):786–800, 2009. 5, 8, 14

[CdGVL11] Serena Cicalò, Willem A. de Graaf, and Michael Vaughan-Lee. An effective version of the Lazard correspondence. submitted, 2011. 5, 12

[Khu98]   E. I. Khukhro. *p-automorphisms of finite p-groups*, volume 246 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, Cambridge, 1998. 5

Index