SLA

Computing with simple Lie algebras

Version 1.5.3

15 November 2019

Willem Adriaan de Graaf

Willem Adriaan de Graaf
Email: degraaf@science.unitn.it
Homepage: http://www.science.unitn.it/~degraaf

Abstract

This package provides functions for computing with various aspects of the theory of simple Lie algebras in characteristic zero.

Copyright

Contents

1 Introduction **4**

2 Auxiliary Functions **5**
2.1 Root Systems 5
2.2 Weyl groups 6
2.3 Lie Algebras and Their Modules 8

3 Nilpotent Orbits **11**
3.1 The functions 12

4 Finite Order Automorphisms and θ-Groups **16**
4.1 The functions 16

5 Semisimple Subalgebras of Semisimple Lie Algebras **22**
5.1 Branching 22
5.2 Constructing Semisimple Subalgebras 23

References **30**

Index **31**

Chapter 1

Introduction

This package is a collection of functions that I wrote for various research projects (e.g., [Gra08], [dG11], [GE09], [Gra11], [dGVY12]). The reason to collect them in a package is to avoid them getting lost. Secondly, I believe that the functions may be of wider interest.

Apart from this one, this manual has four chapters. The second describes various functions that did not fit in any of the other chapters. They vary from short utility functions to functions implementing rather complex algorithms. The remaining three chapters are all devoted to a particular area.

The third chapter contains (descriptions of) functions for computing with the classification of the nilpotent orbits in simple Lie algebras. There are functions for creating the orbits and for computing representatives. We refer to [CM93] for an overview of the theory of nilpotent orbits in simple Lie algebras.

The fourth chapter is dedicated to finite order automorphisms of the simple Lie algebras and the corresponding θ-groups. The finite order automorphisms have been classified by Kac, up to conjugacy in the automorphism group. For the background on this we refer to [Hel78]. The classification is described in terms of so-called Kac diagrams. The package contains a function for creating all automorphisms of a given simple Lie algebra, of a given finite order.

The eigenspaces of an automorphism of finite order of a simple Lie algebra form a grading of that Lie algebra. Moreover, the 0-component is a reductive subalgebra, acting on the 1-component. The 0-component corresponds to a reductive reductive group, also acting on the 1-component. This group (with its action) is called a θ-group. It was introduced and studied in the 70-s by Vinberg ([Vin75] , [Vin76], [Vin79]) The package has a function for listing the nilpotent orbits of this group.

The fifth chapter has functions for working with semisimple subalgebras of semisimple Lie algebras. The package contains a database of semisimple subalgebras of the simple subalgebras of ranks up to 8. Moreover, there are functions for computing the semisimple subalgebras of semisimple Lie algebras on the fly. Finally, there are some functions for computing branching rules.

We remark that the package needs the package QuaGroup.

Chapter 2

Auxiliary Functions

This chapter contains the description of some functions that do not fit in any other chapter.

2.1 Root Systems

2.1.1 ExtendedCartanMatrix

▷ `ExtendedCartanMatrix(R)` (operation)

Here *R* is a root system. This function returns the extended Cartan matrix of *R*. That is the Cartan matrix correponding to the lowest root (coming first), and the simple roots of *R*.

The output is a record with components *ECM* (the extended Cartan matrix) and *labels* (the labels of the corresponding Dynkin diagram; they are the integer coefficients of a linear dependency of the roots corresponding to the nodes).

Example

```
gap> R:= RootSystem("F",4);;
gap> ExtendedCartanMatrix(R);
rec( ECM := [ [ 2, -1, 0, 0, 0 ], [ -1, 2, -1, 0, 0 ], [ 0, -1, 2, -2, 0 ],
      [ 0, 0, -1, 2, -1 ], [ 0, 0, 0, -1, 2 ] ], labels := [ 1, 2, 3, 4, 2 ] )
```

2.1.2 CartanType

▷ `CartanType(C)` (operation)

Here *C* is a Cartan matrix (i.e., an integer matrix with 2-s on the diagonal, non-positive entries otherwise, and there exists a diagonal integer matrix *D* such that *CD* is a positive definite symmetric matrix). This function returns a record with two components: *types*, a list containing the types of the simple components of the corresponding root system, and *enumeration*, a standard enumeration of the vertices of the Dynkin diagram of *C*. So this can be used to construct isomorphisms of root systems.

Example

```
gap> C:= [[2,0,-3,0],[0,2,0,-1],[-1,0,2,0],[0,-1,0,2]];
[ [ 2, 0, -3, 0 ], [ 0, 2, 0, -1 ], [ -1, 0, 2, 0 ], [ 0, -1, 0, 2 ] ]
gap> CartanType(C);
rec( enumeration := [ [ 3, 1 ], [ 2, 4 ] ],
  types := [ [ "G", 2 ], [ "A", 2 ] ] )
```

2.2 Weyl groups

2.2.1 WeylTransversal

▷ `WeylTransversal(R, inds)` (operation)
▷ `WeylTransversal(R, roots)` (operation)

Here `R` is a root system, and `inds` a list of indices of *positive* roots of `R` that form a set of simple roots of a root subsystem of `R` (the system does not check this). Here an index of a positive root is its position in the list `PositiveRootsNF( R )`.

This function returns a list of shortest representatives of the right cosets of the corresponding Weyl subgroup of the Weyl group of `R`. The elements of the Weyl group are given as reduced expressions.

In the second form `rts` is a list of roots of `R`, that form a set of simple roots of a root subsystem of `R` (again, this is not checked). In this form the roots so not have to be positive. They have to be represented with respect to the basis of simple roots, i.e., they are elements of `PositiveRootsNF(R)` or of `NegativeRootsNF(R)`.

Example

```
gap> R:= RootSystem("A",3);;
gap> WeylTransversal( R, [2,6] );
[ [  ], [ 1 ], [ 3 ], [ 1, 2 ], [ 1, 3 ], [ 3, 2 ] ]
gap> R:= RootSystem("E",8);;
gap> p:= PositiveRootsNF(R);;
gap> a:= WeylTransversal( R, [p[1],p[3],p[4],p[5],p[6],p[7],p[8],-p[120]] );;
gap> Length(a);
1920
```

2.2.2 SizeOfWeylGroup

▷ `SizeOfWeylGroup(R)` (operation)
▷ `SizeOfWeylGroup(type)` (operation)
▷ `SizeOfWeylGroup(X, n)` (operation)

In the first from `R` is a root system. In the second form `type` is a list of lists describing the type of a root system. For example: `[["A",3],["B",5],["G",2]]`. In the third form `X` is a letter (i.e., a string) and `n` a positive integer, so that `Xn` is the type of a root system. In all cases the number of elements of the Weyl group is returned.

Example

```
gap> R:= RootSystem( SimpleLieAlgebra("E",6,Rationals) );;
gap> SizeOfWeylGroup(R);
51840
gap> SizeOfWeylGroup( [["E",6]] );
51840
gap> SizeOfWeylGroup( "E", 6 );
51840
```

2.2.3 WeylGroupAsPermGroup

▷ `WeylGroupAsPermGroup(R)` (operation)

Here `R` is a root system. This function returns a permutation group whose set of elements is in bijection with the set of elements of the Weyl group of `R`. More precisely, this bijection works as follows.

Let n be the number of positive roots of `R`. We list the positive roots in the order in which they apppear in `PositiveRootsNF( R )`. To this list we append the negative roots, listed in the same order. Thus the list of roots is $\{\alpha_1, \ldots, \alpha_{2n}\}$, where $\alpha_{i+n} = -\alpha_i$ for $1 \leq i \leq n$. Then a reflection s_α corresponding to the root α corresponds to the permutation π_α, where $s_\alpha(\alpha_i) = \alpha_{i^{\pi_\alpha}}$. Note, however, that s_α acts from the *left*, whereas π_α acts from the *right*. Let W denote the Weyl group of `R` and let G be the group generated by the permutations π_α for α in the fixed set of simple roots of `R`. Then mapping s_α to π_α extends to an *anti-isomorphism* $W \to G$. The reason for doing it like this is that in the vast majority of literature on Lie theory the Weyl group acts from the left, but in GAP permutation groups act from the right. When applying the group that is output by this function this difference has to be kept in mind. For example, the orbit of a root β under W equals the set of images of β under the representatives of the left cosets of the stabilizer of β. But when we use the group G we have to consider the right cosets for this.

Example

```
gap> R:= RootSystem("E",6);
<root system of type E6>
gap> G:= WeylGroupAsPermGroup( R );
<permutation group with 6 generators>
gap> Size(G);
51840
```

2.2.4 ApplyWeylPermToWeight

▷ `ApplyWeylPermToWeight(R, p, w)` (operation)

Here `R` is a root system, `p` is an element of the group returned by `WeylGroupAsPermGroup` (2.2.3) with input `R`. Here `w` is a weight, that the list of coefficients of a weight when written as a linear combination of fundamental weights. This function returns the result of acting with the element of the Weyl group corresponding to `p` on `w`.

Example

```
gap> R:= RootSystem("D",4);;
gap> G:= WeylGroupAsPermGroup(R);
<permutation group with 4 generators>
gap> wt:= ApplyWeylPermToWeight( R, Random(G), [1,1,1,1] );;
gap> ConjugateDominantWeight( WeylGroup(R), wt );
[ 1, 1, 1, 1 ]
```

2.2.5 WeylWordAsPerm

▷ `WeylWordAsPerm(R, u)` (operation)

Here `R` is a root system, and `u` is an element of the Weyl group given as a (not necessarily reduced) word, that is, `u` is given by a list of indices between 1 and the rank of `R`. This function returns the permutation corresponding to `u`, that is, the image of `u` under the anti-iromorphism discussed in `WeylGroupAsPermGroup` (2.2.3).

Example

```
gap> R:= RootSystem("D",4);;
gap> WeylWordAsPerm( R, [1,2,1,3,4,2,3,4,1] );
(1,23,12,17)(2,10,14,22)(3,19,16,6)(4,18,15,7)(5,13,11,24)(8,21)(9,20)
```

2.2.6 PermAsWeylWord

▷ PermAsWeylWord(*R, p*) (operation)

This is the inverse operation to the one discussed in WeylWordAsPerm (2.2.5). That is, the element of the Weyl group (this time given as reduced expression) corresponding to the permutation *p* is returned.

In the next example we compute generators of the stabilizer of a subset of a root system. The generators are given as reduced words.

Example

```
gap> R:= RootSystem("D",4);;
gap> rts:= [1,3,4,12,13,15,16,24];;
gap> G:= WeylGroupAsPermGroup(R);;
gap> S:= Stabilizer( G, rts, OnSets );
<permutation group of size 64 with 6 generators>
gap> Size(S);
64
gap> List( GeneratorsOfGroup(S), g -> PermAsWeylWord( R, g ) );
[ [ 3 ], [ 3, 4 ], [ 2, 1, 3, 2, 4, 2, 1, 3, 2, 4 ], [ 2, 1, 3, 2 ],
  [ 2, 1, 4, 2 ], [ 1, 3 ] ]
```

2.3 Lie Algebras and Their Modules

2.3.1 IsomorphismOfSemisimpleLieAlgebras

▷ IsomorphismOfSemisimpleLieAlgebras(*L1, L2*) (operation)

Here *L1* and *L2* are two semisimple Lie algebras that are known to be isomorphic (i.e., they have the same type). This function returns an isomorphism.

2.3.2 AdmissibleLattice

▷ AdmissibleLattice(*V*) (operation)

Here *V* is a *simple* module over a semisimple Lie algebra. This function returns a basis of *V* that spans an admissible lattice in *V*. This means that for a root vector x of the acting Lie algebra the matrix $exp(mx)$ is integral, where mx denotes the matrix of x relative to the admissible lattice.

Example

```
gap> L:= SimpleLieAlgebra("G",2,Rationals);;
gap> V:= HighestWeightModule( L, [2,0] );
<27-dimensional left-module over <Lie algebra of dimension 14 over Rationals>>
gap> B:=AdmissibleLattice(V);;
gap> x:= L.1;
v.1
```

```
gap> mx:= MatrixOfAction( B, x );;
gap> IsZero(mx^4); IsZero(mx^5);
false
true
gap> exp:=Sum( List( [0..4], i -> mx^i/Factorial(i) ) );;
gap> ForAll( Flat(exp), IsInt );
true
```

2.3.3 DirectSumDecomposition

▷ `DirectSumDecomposition(V)` (operation)

Here V is a module over a semisimple Lie algebra; this function computes a list of sub-modules such that V is their direct sum.

Example

```
gap> L:= SimpleLieAlgebra("G",2,Rationals);;
gap> V:= HighestWeightModule( L, [1,0] );;
gap> W:= TensorProductOfAlgebraModules( V, V );
<49-dimensional left-module over <Lie algebra of dimension 14 over Rationals>>
gap> DirectSumDecomposition( W );
[ <left-module over <Lie algebra of dimension 14 over Rationals>>,
  <left-module over <Lie algebra of dimension 14 over Rationals>>,
  <left-module over <Lie algebra of dimension 14 over Rationals>>,
  <left-module over <Lie algebra of dimension 14 over Rationals>> ]
gap> List( last, Dimension );
[ 27, 7, 14, 1 ]
```

2.3.4 CharacteristicsOfStrata

▷ `CharacteristicsOfStrata(L, hw)` (operation)

Here L is a semisimple Lie algebra over a field of characteristic 0. Secondly, hw is a dominant weight, represented as a list of non-negative integers (where the ordering of the fundamantal weights is given by the Cartan matrix of the root system of L). Let G denote the semisimple algebraic group acting on the irreducible representation with highest weight hw. Hesselink ([Hes79]) defined a stratification of the nullcone relative to the action of G. Popov and Vinberg ([VP89]) have described this stratification in terms of characteristics, which are elements of a Cartan subalgebra of L. To each characteristic there corresponds a stratum. This function is an implementation of an algorithm due to Popov ([Pop03]), for computing the characteristics of the strata. It returns a list of two lists. The first list contains the characteristics. The second list contains the dimensions of the corresponding strata. If the highest weight hw defines the adjoint representation, then the characteristics of the strata are exactly the characteristics of the nilpotent orbits in L. This means the following: let h be a characteristic, then there are e, f in L such that the triple h, e, f satisfies the commutation relations of $\mathfrak{sl}_2$, and the elements e thus obtained are the representatives of the nilpotent G-orbits in L.

Example

```
gap> L:= SimpleLieAlgebra("G",2,Rationals);;
gap> L:= SimpleLieAlgebra("G",2,Rationals);;
gap> CharacteristicsOfStrata( L, [0,1] );
```

```
[ [ v.13+(2)*v.14, (2)*v.13+(3)*v.14, (2)*v.13+(4)*v.14, (6)*v.13+(10)*v.14 ],
  [ 6, 8, 10, 12 ] ]
```

Chapter 3

Nilpotent Orbits

This chapter contains functions for dealing with the nilpotent orbits of a semisimple Lie algebra K under its adjoint group G. We refer to the book by Collingwood and McGovern, [CM93] (and the references therein) for an account of the theory of nilpotent orbits. A nilpotent orbit has two important attributes: the weighted Dynkin diagram, and an sl_2-triple. The weighted Dynkin diagram is represented by a list of integers in {0,1,2} of length equal to the rank of the Lie algebra. The i-th position in this list correponds to the i-th node of the Dynkin diagram of the root system. The Dynkin diagram of the root system is described by the Cartan matrix of the root system. Now in GAP this Cartan matrix can be somewhat different from the more usual forms. This holds most particularly for type F4, where the enumeration of the simple roots is rather different from the one usually found. So when using the functions in this chapter one should keep this in mind.

Every nilpotent orbit has an sl_2-triple, that is a triple (y,h,x) of elements of the simple Lie algebra with $[x,y] = h$, $[h,x] = 2x$, $[h,y] = -2y$. The nilpotent orbit corresponding to this is the orbit of the element x under the action of the adjoint group.

Let P be a parabolic subalgebra of K (i.e., generated by the Cartan subalgebra of K, all positive root vectors, along with the negative simple root vectors corresponding to a given subset of the basis of simple roots), L the corresponding Levi subalgebra (i.e., the reductive part of P), and N the nilradical of P. Let O_L be a nilpotent orbit in L. There exists a unique nilpotent orbit O_K in K such that the intersection of O_K and $O_L + N$ is dense in the latter. In this situation O_K is said to be *induced* from O_L. Nilpotent orbits in K which are not induced are said to be *rigid*.

Now consider the variety of all G-orbits in K of a given dimension d. The irreducible components of this variety are called the *sheets* of K. Every sheet has a unique nilpotent orbit. Moreover this nilpotent orbit is induced from an orbit O_L, and O_L is rigid in L. So the sheets are parametrised by pairs (L, O_L), where L is a Levi subalgebra, and O_L a rigid nilpotent orbit in it. This data can conveniently be given by a *sheet diagram*: this is the Dynkin diagram of K, were the nodes that do *not* correspond to simple roots of L have label 2. So, leaving out the nodes with label 2, one obtains the Dynkin diagram of L. The remaining labels in the sheet diagram then correspond to the weighted Dynkin diagram of the nilpotent orbit O_L. Since this orbit is rigid, its weighted Dynki diagram has labels 0 or 1. From that it follows that one can recover L and O_L from the sheet diagram. The *rank* of a sheet is defined as the dimension of the centre of L, obviously that is equal to the number of 2's in the sheet diagram.

3.1 The functions

3.1.1 NilpotentOrbit

▷ `NilpotentOrbit(L, wd)` (operation)

Here `L` is a simple Lie algebra and `wd` a weighted Dynkin diagram (i.e., a list containing the weights of the weighted Dynkin diagram, in the same order as the nodes of the Dynkin diagram of the root system of `L`; that order can be deduced from the Cartan matrix of the same root system). The corresponding nilpotent orbit is returned. It is the responsibility of the user to make sure that the weighted Dynkin diagram corresponds to a nilpotent orbit.

Example

```
gap> L:= SimpleLieAlgebra("E",6,Rationals);;
gap> o:= NilpotentOrbit( L, [1,2,0,0,0,1] );
<nilpotent orbit in Lie algebra of type E6>
```

3.1.2 NilpotentOrbits

▷ `NilpotentOrbits(L)` (attribute)

Here `L` is a semisimple Lie algebra. This function returns the list of all nilpotent orbits of `L`. If `L` is simple of classical type, then the nilpotent orbits correpond to partitions (of $n+1$ for type A_n, of $2n+1$ for type B_n, of $2n$ for type C_n and of $2n$ for type D_n, see [CM93]). If `L` is of one of these types then the orbits returned by this function have the attribute `OrbitPartition` set, which returns the corresponding partition.

Example

```
gap> L:= SimpleLieAlgebra("E",6,Rationals);;
gap> orbs:= NilpotentOrbits(L);;
gap> orbs[10];
<nilpotent orbit in Lie algebra of type E6>
gap> Length(orbs);
20
gap> L:= SimpleLieAlgebra("B",4,Rationals);;
gap> orbs:= NilpotentOrbits(L);;
gap> OrbitPartition( orbs[10] );
[ 5, 3, 1 ]
```

3.1.3 WeightedDynkinDiagram

▷ `WeightedDynkinDiagram(o)` (attribute)

Here `o` is a nilpotent orbit; this function returns its weighted Dynkin diagram.

3.1.4 WeightedDynkinDiagram

▷ `WeightedDynkinDiagram(L, x)` (method)

Here `L` is a semisimple Lie algebra, and `x` a nilpotent element. This function returns the weighted Dynkin diagram of the orbit containing `x`.

Example
```
gap> L:= SimpleLieAlgebra("B",3,Rationals);;
gap> WeightedDynkinDiagram( L, L.1+L.9 );
[ 2, 0, 0 ]
gap> L:= SimpleLieAlgebra("E",6,Rationals );;
gap> WeightedDynkinDiagram(L, L.1+L.6+L.20+2*L.32 : table:= true );
[ 0, 0, 0, 1, 0, 0 ]
```

3.1.5 AmbientLieAlgebra

▷ AmbientLieAlgebra(*o*) (attribute)

Here *o* is a nilpotent orbit; this function returns the Lie algebra it lives in.

3.1.6 SemiSimpleType

▷ SemiSimpleType(*o*) (attribute)

Here *o* is a nilpotent orbit; this function returns the type of the Lie algebra it lives in.

3.1.7 SL2Triple

▷ SL2Triple(*o*) (attribute)

Here *o* is a nilpotent orbit; this function returns an sl_2-triple (y,h,x) corresponding to *o*. For the exceptional types the x is as in the paper [Gra08]. For the classical types the x is computed on the fly.

Example
```
gap> L:= SimpleLieAlgebra("E",6,Rationals);;
gap> orbs:= NilpotentOrbits(L);;
gap> SL2Triple( orbs[10] );
[ (4)*v.51+(3)*v.53+(3)*v.56+v.59, (4)*v.73+(6)*v.74+(8)*v.75+(11)*v.76+(
    8)*v.77+(4)*v.78, v.15+v.17+v.20+v.23 ]
```

3.1.8 RandomSL2Triple

▷ RandomSL2Triple(*o*) (operation)

Here *o* is a nilpotent orbit; this function returns a random sl_2-triple (x,h,y) corresponding to *o*. This means that every call (potentially) returns a different sl_2-triple.

Example
```
gap> L:= SimpleLieAlgebra("E",6,Rationals);;
gap> orbs:= NilpotentOrbits(L);;
gap> RandomSL2Triple( orbs[10] );
[ (3)*v.49+(3)*v.50+v.51+(4)*v.59, (4)*v.73+(6)*v.74+(8)*v.75+(11)*v.76+(
    8)*v.77+(4)*v.78, v.13+v.14+v.15+v.23 ]
gap> RandomSL2Triple( orbs[10] );
[ (3)*v.49+(4)*v.54+(3)*v.56+v.57, (4)*v.73+(6)*v.74+(8)*v.75+(11)*v.76+(
    8)*v.77+(4)*v.78, v.13+v.18+v.20+v.21 ]
```

3.1.9 SL2Grading

▷ SL2Grading(*L*, *h*) (operation)

Here *L* is a Lie algebra, and *h* is an element of it, such that there is an sl_2 triple of which it is the Cartan element (the system does not check that). This function returns the grading of *L* in eigenspaces of *h*. A list containing three lists is returned: the first list contains bases of the components with degrees 1,2,3,... the second list has bases of the components with degrees -1,-2,-3,..., the last list contains a basis of the zero component.

Example

```
gap> L:= SimpleLieAlgebra("F",4,Rationals);;
gap> orbs:= NilpotentOrbits(L);;
gap> sl2:= RandomSL2Triple( orbs[6] );
[ (2)*v.37+(2)*v.39+v.41, (3)*v.49+(4)*v.50+(6)*v.51+(8)*v.52, v.13+v.15+v.17
 ]
gap> SL2Grading( L, sl2[2] );
[ [ [ v.3, v.5, v.7, v.8, v.9, v.11 ],
      [ v.10, v.12, v.13, v.14, v.15, v.16, v.17, v.18, v.20 ],
      [ v.19, v.21 ], [ v.22, v.23, v.24 ] ],
  [ [ v.27, v.29, v.31, v.32, v.33, v.35 ],
      [ v.34, v.36, v.37, v.38, v.39, v.40, v.41, v.42, v.44 ],
      [ v.43, v.45 ], [ v.46, v.47, v.48 ] ],
  [ v.1, v.2, v.4, v.6, v.25, v.26, v.28, v.30, v.49, v.50, v.51, v.52 ] ]
```

3.1.10 SL2Triple

▷ SL2Triple(*L*, *x*) (operation)

Here *L* is a simple Lie algebra, and *x* is a nilpotent element of it. A list of three elements is returned, forming an sl_2-triple, the last of which is equal to *x*.

Example

```
gap> L:= SimpleLieAlgebra("F",4,Rationals);;
gap> SL2Triple( L, L.1+L.20 );
[ v.16+v.25, v.49, v.1+v.20 ]
```

3.1.11 InducedNilpotentOrbits

▷ InducedNilpotentOrbits(*L*) (attribute)

Here *L* is a simple Lie algebra. This function returns the list of all induced nilpotent orbits of *L*. An induced orbit is given by a record containing two fields: *sheetdiag*, which is a diagram describing the Levi subalgebra and the rigid nilpotent orbit in it from which the nilpotent orbit is induced, and *norbit*, which is the induced nilpotent orbit in *L*. The sheet diagram is a labeled Dynkin diagram, and the labels are 0, 1 or 2. If we take the Dynkin diagram and erase the nodes which have label 2 then we obtain the Dynkin diagram of the Levi subalgebra. Moreover, the labels 0 and 1 on that diagram give the rigid nilpotent orbit in the Levi subalgebra. From this pair the nilpotent orbit *norbit* is induced. It may happen that the same nilpotent orbit is induced from more pairs consisting of a Levi subalgebra and a rigid nilpotent orbit in it. In that case the same nilpotent orbit appears more than once in the list, each time with a different sheet diagram attached. This function works for the Lie

algebras of exceptional type and for the Lie algebras of type A regardless of the rank. It works for the Lie algebras of the other types up to rank 10.

Example

```
gap> L:= SimpleLieAlgebra("E",6,Rationals);;
gap> s:= InducedNilpotentOrbits(L);;
gap> s[19];
rec( norbit := <nilpotent orbit in Lie algebra of type E6>,
  sheetdiag := [ 2, 0, 0, 1, 0, 2 ] )
gap> WeightedDynkinDiagram( s[19].norbit );
[ 0, 0, 0, 2, 0, 0 ]
```

3.1.12 RigidNilpotentOrbits

▷ `RigidNilpotentOrbits(`L`)` (attribute)

Here L is a simple Lie algebra. This function returns the list of all rigid nilpotent orbits of L, *except* the zero orbit (which is always rigid).

Example

```
gap> L:= SimpleLieAlgebra("E",6,Rationals);;
gap> RigidNilpotentOrbits(L);
[ <nilpotent orbit in Lie algebra of type E6>,
  <nilpotent orbit in Lie algebra of type E6>,
  <nilpotent orbit in Lie algebra of type E6> ]
gap> List( last, WeightedDynkinDiagram );
[ [ 0, 1, 0, 0, 0, 0 ], [ 0, 0, 0, 1, 0, 0 ], [ 1, 0, 0, 1, 0, 1 ] ]
```

Chapter 4

Finite Order Automorphisms and θ-Groups

This chapter contains functions for creating and working with finite order automorphisms of simple Lie algebras (or, more precisely, representatives of the conjugacy classes of such automorphisms).

NB: such automorphisms are not created for a given Lie algebra, but the Lie algebra is constructed at the same time as the automorphism. This because the base field may need extending (it needs enough roots of unity).

As noted above the functions give representatives of the conjugacy classes, in the automorphism group of the underlying Lie algebra, of finite order automorphisms. Such conjugacy classes are classified in terms of Kac diagrams. Roughly, this works as follows. A finite order automorphism f corresponds to a diagram automorphism of order $d = 1,2,3$. The inner automorphisms correspond to a diagram automorphism of order 1, the outer automorphisms to a diagram automorphism of order 2 or 3. Let L_0, L_1 denote the eigenspaces of the underlying Lie algebra L, with respect to the diagram automorphism, respectively corresponding to the eigenvalues 1 and w (where w is a primitive d-th root of unity). (In case of $d = 1$, we have $L_0 = L$, $L_1 = 0$.) Then L_0 is semisimple and we choose a set of canonical generators of L_0, denoted x_i, y_i, h_i, for $i = 1,\ldots,s$. Moreover, L_1 is an L_0-module. Let x_0 be the lowest weight vector in L_1. (If $d = 1$ then x_0 will be the lowest (negative) root vector.) Let α_i for $i = 0,\ldots,s$ be the roots corresponding to x_i, with respect to the subalgebra spanned by the h_i. Let C be the Cartan matrix of these roots. The rows of C are linearly dependent. The Dynkin diagram of C is labeled with integers a_i with greatest common divisor 1, that form the coefficients of a linear dependency of the rows of C. Furthermore, the x_i generate L and the automorphism f is described by $f(x_i) = v^{s_i} x_i$, where the non-negative integers s_i have greatest common divisor 1, and are such that $m = d\sum a_i s_i$ is the order of f, and where v is a primitive m-th order root of unity. Now the Kac diagram of the automorphism f is the Dynkin diagram of C, labelled with the labels s_i.

4.1 The functions

4.1.1 FiniteOrderInnerAutomorphisms

▷ `FiniteOrderInnerAutomorphisms(type, rank, m)` (operation)

Let L be the simple Lie algebra of type `type` and rank `rank`. The function returns representatives of the conjugacy classes of inner automorphisms of L of order `m`. As noted also in the introduction to

this chapter, this function constructs the Lie algebra as well as the automorphisms (and the Lie algebra is accessible through the source of these automorphisms). The reason for this is that depending on the order of the automorphisms, the base field needs certain roots of unity.

Example

```
gap> f:= FiniteOrderInnerAutomorphisms("E",6,3);
[ [ v.72, v.1, v.2, v.3, v.4, v.5, v.6 ] -> [ (E(3))*v.72, (E(3)^2)*v.1, v.2,
      v.3, v.4, v.5, v.6 ], [ v.72, v.1, v.2, v.3, v.4, v.5, v.6 ] ->
    [ v.72, (E(3))*v.1, (E(3))*v.2, v.3, v.4, v.5, v.6 ],
  [ v.72, v.1, v.2, v.3, v.4, v.5, v.6 ] -> [ (E(3))*v.72, v.1, (E(3))*v.2,
      v.3, v.4, v.5, v.6 ], [ v.72, v.1, v.2, v.3, v.4, v.5, v.6 ] ->
    [ v.72, v.1, v.2, v.3, (E(3))*v.4, v.5, v.6 ],
  [ v.72, v.1, v.2, v.3, v.4, v.5, v.6 ] -> [ (E(3))*v.72, (E(3))*v.1, v.2,
      v.3, v.4, v.5, (E(3))*v.6 ] ]
gap> Source(f[1]);
<Lie algebra of dimension 78 over CF(3)>
```

4.1.2 FiniteOrderOuterAutomorphisms

▷ `FiniteOrderOuterAutomorphisms(`*type*`,` *rank*`,` *m*`,` *d*`)` (operation)

Let L be the simple Lie algebra of type *type* and rank *rank*. The function returns representatives of the conjugacy classes of outer automorphisms of L of order *m*, corresponding to a diagram automorphism of order *d*.

4.1.3 Order

▷ `Order(`*f*`)` (attribute)

Here *f* is a finite order automorphism. This returns its order.

4.1.4 KacDiagram

▷ `KacDiagram(`*f*`)` (attribute)

Here *f* is a finite order automorphism. This returns its Kac diagram. This is a record with three components: *CM*, which is the Cartan matrix of the Dynkin diagram, *labels* the integers with gcd equal to 1 that are the coefficients of a linear dependency of the rows of *CM*, and *weights* that are the integers s_i that define the automorphism.

Example

```
gap> f:= FiniteOrderOuterAutomorphisms( "A", 5, 4, 2 );;
gap> r:= KacDiagram( f[1] );
rec(
  CM := [ [ 2, 0, -1, 0 ], [ 0, 2, -1, 0 ], [ -1, -1, 2, -1 ],
      [ 0, 0, -2, 2 ] ], labels := [ 1, 1, 2, 1 ], weights := [ 1, 1, 0, 0 ] )
gap> r.labels*r.CM;
[ 0, 0, 0, 0 ]
```

4.1.5 Grading

▷ `Grading(f)` (attribute)

Here `f` is a finite order automorphism of order m. This returns a list of length m. The i-th element contains a basis of the eigenspace of `f` with eigenvalue v^i, where v is a primitive m-th root of unity (i.e., `v=E(m)`).

4.1.6 NilpotentOrbitsOfThetaRepresentation

▷ `NilpotentOrbitsOfThetaRepresentation(f)` (operation)
▷ `NilpotentOrbitsOfThetaRepresentation(L, d)` (operation)

Here `f` is an automorphism of a simple Lie algebra L of order m. Then `f` defines a grading on L. Let the homogeneous components of this grading be denoted L_i for $i = 0,...,m-1$. Let G_0 be the group corresponding to L_0 (i.e., the connected subgroup of the adjoint group of L with Lie algebra L_0). This function computes representatives for the nilpotent orbits of G_0 acting on L_1. The output is a list of triples. Each triple is an sl_2-triple (y,h,x), with $h \in L_0$, $x \in L_1$ (the representative of the orbit), and $y \in L_{m-1}$. The element h also lies in the dominant Weyl chamber of a Cartan subalgebra of L_0. Finally we note that all elements lie in `Source( f )`.

It is possible to add an extra optional argument: `method:= "Carrier"`, or `method:= "WeylOrbit"`. Then a method based on finding carrier algebras (respectively, computing orbits under the Weyl group) is chosen. If no optional argument is chosen, then the system will make its own choice. (In the case of outer automorphisms, currently the only available method is the one based on orbits of the Weyl group.) The method based on carrier algebras tends to work better for the higher order automorphisms.

This function prints some information on what it is doing to the info class `InfoSLA`. In order to suppress these messages one can do `SetInfoLevel( InfoSLA, 1 );`.

In the two-argument version, the first argument `L` has to be a semisimple Lie algebra, and the second argument `d` a list of non-negative integers. Then `L` is Z-graded by giving the root space corresponding to the i-th simple root the degree `d[i]`. Apart from this the function works the same in this case as in the one-argument version.

Example

```
gap> # reset random state to ensure the output of this example match
gap> Reset(GlobalMersenneTwister, 1);;
gap> f:= FiniteOrderInnerAutomorphisms( "D", 5, 3 );;
gap> s:= NilpotentOrbitsOfThetaRepresentation( f[2] : method:= "Carrier" );;
#I  Selected carrier algebra method.
#I  Constructed 123 root bases of possible flat subalgebras, now checking them...
#I  Obtained 30 Cartan elements, weeding out equivalent copies...
gap> Length(s);
10
gap> s[4];
[ v.14+v.15+v.38, (-2)*v.41+(-1)*v.42, v.18+v.34+v.35 ]
gap> L:= SimpleLieAlgebra("E",6,Rationals);;
gap> NilpotentOrbitsOfThetaRepresentation( L, [0,1,0,0,0,0] );
#I  Selected Weyl orbit method.
#I  Constructed a Weyl transversal of 72 elements.
#I  Obtained 5 Cartan elements, weeding out equivalent copies...
```

```
[ [ v.65+v.66+v.67, (2)*v.73+(3)*v.74+(4)*v.75+(6)*v.76+(4)*v.77+(2)*v.78,
      v.29+v.30+v.31 ],
  [ (2)*v.55+(2)*v.66, (2)*v.73+(4)*v.74+(4)*v.75+(6)*v.76+(4)*v.77+(2)*v.78,
      v.19+v.30 ],
  [ v.66+v.70, (2)*v.73+(2)*v.74+(3)*v.75+(4)*v.76+(3)*v.77+(2)*v.78,
      v.30+v.34 ], [ v.71, v.73+v.74+(2)*v.75+(3)*v.76+(2)*v.77+v.78, v.35 ] ]
```

4.1.7 ClosureDiagram

▷ `ClosureDiagram(L, f, s)` (operation)
▷ `ClosureDiagram(L, d, s)` (operation)

Here `f` is an automorphism of a simple Lie algebra L of order m, and `s` a list of sl_2-triples (y,h,x), with $h \in L_0$, $x \in L_1$ (for instance as computed by the previous function), corresponding to nilpotent orbits in L_1.

This function computes the Hasse diagram of the closures of the nilpotent orbits. The output is a record with two components: `diag` (which is a list of 2-tuples; a tuple `[ i, j ]` means that orbit number `i` is contained in the closure of orbit number `j`), and `sl2` (the same list of sl_2-triples, but sorted according to decreasing dimension, i.e., the highest dimensional orbit comes first). The numbering used in the tuples in `diag` corresponds to the order in which the orbits appear in the component `sl2`.

During the execution of the program a message is printed. This message either states that all inclusions have been proved, or lists a number of possible inclusions, for which it could not be proved with absolute certainty that these do not occur. This is due to the randomised nature of the algorithm: if the algorithm finds an inclusion, then this inclusion is certain. However, sometimes a non-inclusion can only be estabished by random methods, which means that it is possible that there is an inclusion without the program finding it. (This however, is very unlikely, and in practice almost never happens.) Now showing that a non-inclusion really is a non-inclusion can be done by computing the ranks of certain matrices with polynomial entries. In principle GAP can do this; however, the system certainly is not very strong at it. Therefore, as optional argument a filename can be given, by `filenm:= "file.m"`. If this argument is present the program prints a Magma script in the file, which can be loaded directly into the computer algebra system Magma. If the output is always true, then all non-inclusions are proved. If there are non non-inclusions to be proved, then the file is not written.

In the second version, the second argument `d` is a list of non-negative integers. Then `L` is Z-graded by giving the root space corresponding to the i-th simple root the degree `d[i]`. Apart from this the function works in the same way.

We note that the adjoint representation can be obtained by giving a `d` that eintirely consists of zeros.

Example

```
gap> f:= FiniteOrderInnerAutomorphisms( "E", 8, 8 );;
gap> h:= f[8];;
gap> sl2:= NilpotentOrbitsOfThetaRepresentation(h);;
#I  Selected carrier algebra method.
#I  Constructed 2782 root bases of possible flat subalgebras, now checking them...
#I  Obtained 58 Cartan elements, weeding out equivalent copies...
gap> Length(sl2);
27
gap> L:= Source(h);;
gap> r:= ClosureDiagram( L, h, sl2 );;
#I  All (non-) inclusions proved!
```

```
gap> r.diag;
[ [ 2, 1 ], [ 3, 1 ], [ 4, 2 ], [ 4, 3 ], [ 5, 1 ], [ 6, 5 ], [ 7, 2 ],
  [ 7, 5 ], [ 8, 4 ], [ 9, 4 ], [ 9, 7 ], [ 10, 6 ], [ 10, 7 ], [ 11, 3 ],
  [ 11, 6 ], [ 12, 7 ], [ 13, 9 ], [ 13, 10 ], [ 13, 11 ], [ 14, 9 ],
  [ 14, 12 ], [ 15, 8 ], [ 15, 9 ], [ 16, 6 ], [ 17, 10 ], [ 17, 12 ],
  [ 17, 16 ], [ 18, 13 ], [ 18, 16 ], [ 19, 13 ], [ 19, 15 ], [ 20, 11 ],
  [ 20, 16 ], [ 21, 14 ], [ 21, 17 ], [ 21, 18 ], [ 22, 14 ], [ 22, 15 ],
  [ 23, 18 ], [ 23, 20 ], [ 24, 18 ], [ 24, 19 ], [ 25, 21 ], [ 25, 22 ],
  [ 25, 24 ], [ 26, 23 ], [ 26, 24 ], [ 27, 21 ], [ 27, 23 ] ]
gap> # Now we do the adjoint representation of the Lie algebra of type F4:
gap> L:= SimpleLieAlgebra("F",4,Rationals);;
gap> o:= NilpotentOrbits(L);;
gap> sl2:= List( o, SL2Triple );;
gap> r:= ClosureDiagram( L, [0,0,0,0], sl2 );;
#I  All (non-) inclusions proved!
gap> r.diag;
[ [ 2, 1 ], [ 3, 2 ], [ 4, 3 ], [ 5, 3 ], [ 6, 4 ], [ 6, 5 ], [ 7, 6 ],
  [ 8, 7 ], [ 9, 7 ], [ 10, 8 ], [ 10, 9 ], [ 11, 8 ], [ 12, 10 ],
  [ 13, 11 ], [ 13, 12 ], [ 14, 13 ], [ 15, 14 ] ]
```

4.1.8 CarrierAlgebra

▷ `CarrierAlgebra(L, f, e)` (operation)

▷ `CarrierAlgebra(L, d, e)` (operation)

Here `f` is an automorphism of a simple Lie algebra L of order m, and `e` a nilpotent element of L_1. This function returns the carrier algebra of `e`. This is a $\mathbb{Z}$-graded semisimple subalgebra K of L, such that `e` lies in K_1. For the precise definition we refer to [Vin79], [Vin75]. The output is given in the form of a record, with three components: `g0`, a basis of K_0, `gp` a list containing bases of K_1, K_2 and so on, and `gn` a list containing bases of K_{-1}, K_{-2} and so on.

In the second version, the second argument `d` is a list of non-negative integers. Then `L` is Z-graded by giving the root space corresponding to the i-th simple root the degree `d[i]`. Apart from this the function works in the same way.

Example

```
gap> f:= FiniteOrderInnerAutomorphisms( "F", 4, 5 );;
gap> h:= f[4];;
gap> sl2:= NilpotentOrbitsOfThetaRepresentation( h );;
#I  Selected Weyl orbit method.
#I  Constructed a Weyl transversal of 144 elements.
#I  Constructed 621 Cartan elements to be checked.
gap> L:= Source(h);
<Lie algebra of dimension 52 over CF(5)>
gap> r:=CarrierAlgebra( L, h, sl2[1][3] );
rec( g0 := [ v.49+(2)*v.50+(2)*v.51+(3)*v.52, v.50+(1/2)*v.51+v.52 ],
  gn := [ [ v.24, v.33 ], [ v.21 ], [ v.15 ] ],
  gp := [ [ v.9, v.48 ], [ v.45 ], [ v.39 ] ] )
gap> K:= Subalgebra( L, Concatenation( r.g0, Flat(r.gp), Flat(r.gn) ) );
<Lie algebra over CF(5), with 10 generators>
gap> SemiSimpleType( K );
"B2"
```

4.1.9 CartanSubspace

▷ `CartanSubspace(f)` (operation)

Here `f` is an automorphism of a simple Lie algebra L of order m. Then `f` defines a grading on L. Let the homogeneous components of this grading be denoted L_i for $i = 0,\ldots,m-1$. Let G_0 be the group corresponding to L_0 (i.e., the connected subgroup of the adjoint group of L with Lie algebra L_0). This function computes a maximal subspace of L_1 consisting of commuting semisimple elements. (Such a subspace is called a *Cartan subspace*.)

Every semisimple orbit of G_0 in L_1 contains an element of a fixed Cartan subspace.

Example

```
gap> f:= FiniteOrderInnerAutomorphisms( "A", 3, 3 );;
gap> c:= CartanSubspace( f[3] );
<vector space of dimension 1 over CF(3)>
gap> BasisVectors( Basis( c ) );
[ v.1+v.5+v.12 ]
```

Chapter 5

Semisimple Subalgebras of Semisimple Lie Algebras

This chapter contains functions for dealing with semisimple subalgebras of semisimple Lie algebras. There are functions for computing branching rules, for computing the regular subalgebras, and for working with the database of semisimple subalgebras of the simple Lie algebras. This last database contains the semisimple subalgebras of the simple Lie algebras of ranks up to 8. The semisimple subalgebras are classified up to linear equivalence. (Two subalgebras are called linearly equivalent if for every representation of the big algebra in the space V the images of the subalgebras are conjugate under $GL(V)$.)

5.1 Branching

5.1.1 ProjectionMatrix

▷ `ProjectionMatrix(L, K)` (operation)

Here `L` and `K` are semisimple Lie algebras with the following properties: `K` is contained in `L`, the Cartan subalgebra of `L`, as returned by `CartanSubalgebra(L)` is split (this is automatic if `L` is created by the built in GAP function) and `K` has a Cartan subalgebra that is a subalgebra of the Cartan subalgebra of `L`. We note that the function checks only the last property. The function returns a matrix `P` such that if `u` is a weight of a `L`-module `V`, then `P*u` is a weight of `V`, when considered as a `K`-module.

Example

```
gap> L:= SimpleLieAlgebra("E",7,Rationals);;
gap> K:= Subalgebra( L, [ L.1,L.3,L.4,L.5,L.6,L.7,L.63,
> L.64,L.66,L.67,L.68,L.69,L.70,L.126] );;
gap> Dimension(K);
63
gap> SemiSimpleType(K);
"A7"
gap> ProjectionMatrix( L, K );
[ [ 2, 2, 3, 4, 3, 2, 1 ], [ 0, 0, -1, 0, 0, 0, 0 ], [ 0, 0, 0, -1, 0, 0, 0 ],
  [ 0, 0, 0, 0, -1, 0, 0 ], [ 0, 0, 0, 0, 0, -1, 0 ],
  [ 0, 0, 0, 0, 0, 0, -1 ], [ -1, -2, -2, -3, -2, -1, 0 ] ]
```

5.1.2 Branching

▷ Branching(*L*, *K*, *hw*) (operation)
▷ Branching(*L*, *K*, *cc*, *hw*) (operation)

Here *L* and *K* are as in the previous function, and *hw* is the highest weight of an irreducible *L*-module. This function computes the splitting of the module when seen as a *K*-module. Returned is a list of two lists: the first list contains the highest weights of the modules involved, the second list contains their multiplicities. In the second form the subalgebra is reductive rather than semisimple. Here *K* is again a semisimple subalgebra, and *cc* is a list of toral elements centralizing *K*. These toral elements must lie in the given Cartan subalgebra of *L*. The reductive subalgebra is the direct sum of *K* and the subalgebra spanned by the elements of *cc*. The output is the same as in the first form. However, the last *t* coordinates of the weights give the eigenvalues of the toral elements in *cc* on the irreducible modules.

Example

```
gap> L:= SimpleLieAlgebra("E",7,Rationals);;
gap> K:= Subalgebra( L, [ L.1,L.3,L.4,L.5,L.6,L.7,L.63,
> L.64,L.66,L.67,L.68,L.69,L.70,L.126] );;
gap> Branching( L, K, [1,0,0,0,0,0,1] );
[ [ [ 1, 1, 0, 0, 0, 0, 1 ], [ 1, 1, 1, 0, 0, 0, 0 ], [ 0, 0, 0, 0, 1, 0, 1 ],
      [ 0, 0, 1, 0, 1, 0, 0 ], [ 1, 0, 0, 1, 0, 0, 0 ],
      [ 0, 1, 0, 0, 0, 1, 0 ], [ 0, 0, 0, 0, 0, 0, 1 ],
      [ 0, 0, 1, 0, 0, 0, 0 ] ], [ 1, 1, 1, 1, 1, 1, 1, 1 ] ]
gap> L:= SimpleLieAlgebra("E",7,Rationals);;
gap> r:= RegularSemisimpleSubalgebras(L);;
gap> K:= Filtered( r, M -> SemiSimpleType(M)="E6" )[1];
<Lie algebra of dimension 78 over Rationals>
gap> cc:= BasisVectors( Basis( LieCentralizer( L, K ) ) );
[ v.127+(3/2)*v.128+(2)*v.129+(3)*v.130+(5/2)*v.131+(2)*v.132+(3/2)*v.133 ]
gap> Branching( L, K, cc, [1,0,0,0,0,0,1] );
[ [ [ 0, 0, 0, 0, 0, 2, -1/2 ], [ 1, 1, 0, 0, 0, 0, -1/2 ],
      [ 0, 0, 0, 0, 1, 0, -1/2 ], [ 1, 0, 0, 0, 0, 0, 5/2 ],
      [ 0, 1, 0, 0, 0, 1, 1/2 ], [ 2, 0, 0, 0, 0, 0, 1/2 ],
      [ 0, 0, 1, 0, 0, 0, 1/2 ], [ 0, 0, 0, 0, 0, 1, -5/2 ],
      [ 1, 0, 0, 0, 0, 1, -3/2 ], [ 1, 0, 0, 0, 0, 1, 3/2 ],
      [ 0, 1, 0, 0, 0, 0, 3/2 ], [ 0, 1, 0, 0, 0, 0, -3/2 ],
      [ 0, 0, 0, 0, 0, 0, 3/2 ], [ 0, 0, 0, 0, 0, 0, -3/2 ],
      [ 0, 0, 0, 0, 0, 1, 1/2 ], [ 1, 0, 0, 0, 0, 0, -1/2 ] ],
  [ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2 ] ]
```

5.2 Constructing Semisimple Subalgebras

5.2.1 RegularSemisimpleSubalgebras

▷ RegularSemisimpleSubalgebras(*L*) (attribute)

Here *L* is a *simple* Lie algebra. This function returns a list of its conjugacy classes of semisimple subalgebras (conjugacy under the adjoint group).

Example

```
gap> L:= SimpleLieAlgebra("E",6,Rationals);;
gap> K:= RegularSemisimpleSubalgebras( L );;
```

```
gap> Length(K);
19
gap> K[5];
<Lie algebra of dimension 45 over Rationals>
gap> SemiSimpleType( K[5] );
"D5"
gap> Branching( L, K[5], [1,0,0,0,0,1] );
[ [ [ 1, 0, 0, 0, 0 ], [ 0, 0, 0, 1, 1 ], [ 1, 0, 0, 1, 0 ],
      [ 1, 0, 0, 0, 1 ], [ 2, 0, 0, 0, 0 ], [ 0, 0, 0, 1, 0 ],
      [ 0, 0, 0, 0, 1 ], [ 0, 1, 0, 0, 0 ], [ 0, 0, 0, 0, 0 ] ],
  [ 2, 1, 1, 1, 1, 1, 1, 1, 1 ] ]
```

5.2.2 SSSTypes

▷ `SSSTypes()` (function)

This returns a list of the types of the semisimple Lie algebras of which the database contains the classification of the semisimple subalgebras, up to linear equivalence. (The three letters S stand for SemiSimple Subalgebras.)

5.2.3 LieAlgebraAndSubalgebras

▷ `LieAlgebraAndSubalgebras(`*`type`*`)` (operation)

Here *`type`* is a string describing the type of a semisimple Lie algebra. A simple type is a capital letter (A, B, C, D, E, F, or G) followed by a positive integer. Example: *`"D5"`*. In general a type is a sequence of simple types separated by spaces. Example: *`"A2 C3 E6"`*. This function is appliccable if each simple type that occurs in *`type`* has rank less than or equal to 8. In that case a record is returned with two components: *`liealg`*, which is a semisimple Lie algebra of type *`type`*, and *`subalgs`* which is the list of its semisimple subalgebras up to linear equivalence. If *`type`* is a simple type then the list will be simply fetched from the database. Otherwise a computation will be triggered, and afterwards the database will be bigger. (One can check this with a call to *`SSSTypes()`*). Also we remark that for non-simple types of not so small rank this computation can be difficult.

5.2.4 InclusionsGraph

▷ `InclusionsGraph(`*`type`*`)` (operation)

Here *`type`* is a string describing the type of a semisimple Lie algebra. This is the same as in the previous function. This function returns a list containing the edges of the inclusion graph of the semisimple subalgebras returned by the previous function. An edge is represented by a list of two integers. If the edge *`[ i, j ]`* occurs, then this means that the subalgebra on position *`j`* in the list is linearly equivalent to a subalgebra of the subalgebra in position *`i`*. Only the maximal subalgebras are considered; so if we have edges *`[ i, j ]`* and *`[ j, k ]`* then there will be no edge *`[ i, k ]`*. (Otherwise this list can become huge.) Edges of the form *`[ 0, j ]`* express that the subalgebra on position *`j`* is a maximal semisimple subalgebra of the Lie algebra of type *`type`*.

Example

```
# Semisimple subalgebras of the Lie algebra of type D4:
gap> s:= LieAlgebraAndSubalgebras( "D4" );;
```

```
gap> L:= s.liealg;
<Lie algebra of dimension 28 over CF(3)>
gap> sub:= s.subalgs;;
gap> Length(sub);
44
gap> g:= InclusionsGraph( "D4" );;
gap> g[1];
[ 12, 1 ]

# Find the maximal semisimple subalgebras:
gap> m:= Filtered( g, x -> x[1]=0 );; i:= List( m, x -> x[2] );
[ 13, 35, 36, 37, 41, 42, 43, 44 ]
gap> List( sub{i}, SemiSimpleType );
[ "A2", "A1 B2", "A1 B2", "A1 B2", "B3", "B3", "B3", "A1 A1 A1 A1" ]

# We see that the subalgebras on positions 35 and 36 are isomorphic;
# however they are not linearly equivalent:
gap> Branching( L, sub[35], [0,0,1,0] );
[ [ [ 1, 0, 1 ] ], [ 1 ] ]
gap> Branching( L, sub[36], [0,0,1,0] );
[ [ [ 0, 1, 0 ], [ 2, 0, 0 ] ], [ 1, 1 ] ]
```

5.2.5 SubalgebrasInclusion

▷ SubalgebrasInclusion(*L, K1, K2*) (operation)

Here *K1*, *K2*, are two subalgebras of the semisimple Lie algebra *L*, constructed using the database. If *K2* contains a subalgebra that is linearly equivalent to *K1* then such a subalgebra is returned. Otherwise the result is *fail*.

Example

```
gap> s:= LieAlgebraAndSubalgebras( "C3" );;
gap> g:= InclusionsGraph( "C3" );
[ [ 10, 1 ], [ 11, 1 ], [ 12, 1 ], [ 8, 2 ], [ 10, 2 ], [ 11, 2 ], [ 11, 3 ],
  [ 13, 3 ], [ 8, 4 ], [ 13, 4 ], [ 9, 5 ], [ 12, 5 ], [ 12, 6 ], [ 13, 6 ],
  [ 0, 7 ], [ 0, 8 ], [ 15, 9 ], [ 9, 10 ], [ 14, 10 ], [ 14, 11 ],
  [ 15, 12 ], [ 0, 13 ], [ 15, 14 ], [ 0, 15 ] ]
gap> # there are the edges [ 14, 10] and [ 10, 2 ]; hence a conjugate of the
gap> # second algebra is contained in the 14-th.
gap> L:= s.liealg;
<Lie algebra of dimension 21 over Rationals>
gap> sub:= s.subalgs;;
gap> K:=SubalgebrasInclusion( L, sub[2], sub[14] );
<Lie algebra of dimension 3 over Rationals>
gap> Basis(K)[1] in sub[14];
true
```

5.2.6 DynkinIndex

▷ DynkinIndex(*K, L*) (operation)

Here *K* is a semisimple subalgebra of the *simple* Lie algebra *L*. This function returns a list of integers, containing the Dynkin indices of the simple components of *K*. If the input Lie algebra *L* is not simple, then still a list of rationals is returned, but they may have no meaning. The Dynkin index is defined as follows. Consider a simple component in *K* and let h be the coroot of the shortest root of *K*. Let k denote the Killing form of *L*, normalised so that the coroot of the shortest root of *L* has squared length 2. Then the Dynkin index is $k(h,h)/2$.

Example

```
gap> s:= LieAlgebraAndSubalgebras( "C7" );;
gap> g:= InclusionsGraph( "C7" );;
gap> m:= Filtered( g, x -> x[1]=0 );; i:= List( m, x -> x[2] );
[ 63, 498, 665, 804, 819, 821, 822 ]
gap> L:= s.liealg;
<Lie algebra of dimension 105 over Rationals>
gap> sub:= s.subalgs;;
gap> List( sub{i}, SemiSimpleType );
[ "A1", "C3", "A1 B3", "A6", "C3 C4", "B2 C5", "A1 C6" ]
gap> DynkinIndex( sub[665], L );
[ 7, 4 ]
```

5.2.7 AreLinearlyEquivalentSubalgebras

▷ `AreLinearlyEquivalentSubalgebras(L, K1, K2)` (operation)

Here *L* is a semisimple Lie algebra, and *K1*, *K2* are subalgebras. It is assumed that the Cartan subalgebras (as returned by *CartanSubalgebra*) of *K1*, *K2* are contained in the Cartan subalgebra of *L* (otherwise *fail* is returned). This function returns *true* if *K1*, *K2* are linearly equivalent, *false* otherwise.

Example

```
# Lets find the subalgebras in the database for C5 that are linearly
# equivalent to regular subalgebras:
gap> s:= LieAlgebraAndSubalgebras("C5");; L:= s.liealg; sub:= s.subalgs;;
<Lie algebra of dimension 55 over Rationals>
gap> reg:= RegularSemisimpleSubalgebras( L );;
gap> posn:= [];;
gap> for K in reg do
> Add(posn,PositionProperty(sub,M -> AreLinearlyEquivalentSubalgebras(L,M,K)));
> od;
gap> posn;
[ 2, 24, 93, 111, 105, 82, 106, 81, 41, 109, 70, 85, 29, 112, 94, 25, 1, 118,
  100, 102, 64, 108, 84, 28, 117, 107, 116, 96, 101, 63, 115, 114, 95, 113 ]
```

5.2.8 MakeDatabaseEntry

▷ `MakeDatabaseEntry(r)` (operation)

▷ `AddToDatabase(d)` (operation)

These are functions that help to save a computed list of subalgebras of a semisimple Lie algebra in a file, and in a new session, read it again. In the first function *r* is a record as produced by `LieAlgebraAndSubalgebras` (5.2.3). It returns a record that can be saved in a file. (It is not advisable

to print it on the screen.) In the second function `d` is a record that is output by `MakeDatabaseEntry`. This function adds this entry to the database.

We give two examples; in the first one we create a new database entry, and save it to a file. In the second example we read it and add it to the database.

Example

```
gap> r:= LieAlgebraAndSubalgebras( "A2 B2" );;
gap> d:= MakeDatabaseEntry( r );;
gap> PrintTo( "A2B2", "d:= ",d,";\n");
```

Example

```
gap> Read("A2B2");
gap> AddToDatabase( d );
gap> SSSTypes();
[ "A1", "A2", "B2", "G2", "A3", "B3", "C3", "A4", "B4", "C4", "D4", "F4",
  "A5", "B5", "C5", "D5", "A6", "B6", "C6", "D6", "E6", "A7", "B7", "C7",
  "D7", "E7", "A8", "B8", "C8", "D8", "E8", "A2 B2" ]
```

5.2.9 ClosedSubsets

▷ `ClosedSubsets(R)` (operation)

Here R is a root system. A subset S of the roots of R is said to be closed if for all $a,b \in S$ we have that $a+b$ lies in S whenever $a+b$ is a root. This function computes the list of the closed subsets of R up to conjugacy by the Weyl group. In other words, each closed subset of R is conjugate under the Weyl group to exactly one element of the output of this function. The output is a list of which each element is a list of roots. A root in such a list, or its negative, lies in the attribute *`PositiveRootsNF( R )`*.

Example

```
gap> R:= RootSystem("F",4);
<root system of type F4>
gap> c:= ClosedSubsets(R);;
gap> Length(c);
4844
gap> c[1005];
[ [ 1, 1, 0, 0 ], [ 0, 0, 1, 1 ], [ 0, 1, 2, 0 ], [ 0, 1, 1, 1 ],
  [ 1, 1, 1, 1 ], [ 1, 2, 2, 0 ], [ 1, 1, 2, 1 ], [ 0, 1, 2, 2 ],
  [ 1, 1, 2, 2 ], [ 1, 2, 3, 1 ], [ 1, 2, 2, 2 ], [ 1, 2, 3, 2 ],
  [ 1, 2, 4, 2 ], [ 1, 3, 4, 2 ], [ 2, 3, 4, 2 ], [ 0, -1, -2, 0 ] ]
```

5.2.10 DecompositionOfClosedSet

▷ `DecompositionOfClosedSet(c)` (operation)

Here `c` is a closed set of roots of some root system. We have that `c` is the disjoint union of its symmetric part (consisting of all roots a in `c` such that $-a$ also lies in `c`) and its special part (consisting of all roots a in `c` such that $-a$ does not lie in `c`). This function returns a list of two entries. The first is the symmetric part of `c`, the second is the special part of `c`.

Example

```
gap> R:= RootSystem("F",4);
<root system of type F4>
gap> c:= ClosedSubsets(R);;
```

```
gap> DecompositionOfClosedSet( c[1005] );
[ [ [ 0, 1, 2, 0 ], [ 0, -1, -2, 0 ] ],
  [ [ 1, 1, 0, 0 ], [ 0, 0, 1, 1 ], [ 0, 1, 1, 1 ], [ 1, 1, 1, 1 ],
      [ 1, 2, 2, 0 ], [ 1, 1, 2, 1 ], [ 0, 1, 2, 2 ], [ 1, 1, 2, 2 ],
      [ 1, 2, 3, 1 ], [ 1, 2, 2, 2 ], [ 1, 2, 3, 2 ], [ 1, 2, 4, 2 ],
      [ 1, 3, 4, 2 ], [ 2, 3, 4, 2 ] ] ]
```

5.2.11 IsSpecialClosedSet

▷ `IsSpecialClosedSet(c)` (operation)

Here `c` is a closed set of roots of some root system. This function returns `true` is `c` is special (that is, for all a in `c` we have that $-a$ does not lie in `c`), otherwise it returns `false`.

Example

```
gap> R:= RootSystem("F",4);
<root system of type F4>
gap> c:= ClosedSubsets(R);;
gap> IsSpecialClosedSet( c[1005] );
false
gap> IsSpecialClosedSet( c[1006] );
true
gap> Length( Filtered( c, IsSpecialClosedSet ) );
3579
```

5.2.12 LieAlgebraOfClosedSet

▷ `LieAlgebraOfClosedSet(L, c)` (operation)

Here `L` is a semisimple Lie algebra and `c` is a closed set of roots of its root system. This function returns the subalgebra of `L` spanned a the Cartan subalgebra of `L` (the one relative to which the root system is taken) along with the root vectors corresponding to the roots in `c`.

Example

```
gap> L:= SimpleLieAlgebra("F",4,Rationals);
<Lie algebra of dimension 52 over Rationals>
gap> R:= RootSystem( L );
<root system of rank 4>
gap> c:= ClosedSubsets(R);;
gap> c[1005];
[ [ 1, 0, 1, 0 ], [ 0, 1, 0, 1 ], [ 1, 0, 1, 1 ], [ 0, 0, 2, 1 ],
  [ 1, 1, 1, 1 ], [ 1, 1, 2, 1 ], [ 2, 0, 2, 1 ], [ 0, 1, 2, 2 ],
  [ 2, 1, 2, 1 ], [ 1, 1, 3, 2 ], [ 2, 1, 2, 2 ], [ 2, 1, 3, 2 ],
  [ 2, 1, 4, 2 ], [ 2, 1, 4, 3 ], [ 2, 2, 4, 3 ], [ 0, 0, -2, -1 ] ]
gap> K:= SubalgebraOfClosedSet( L, c[1005] );
<Lie algebra of dimension 20 over Rationals>
gap> LeviMalcevDecomposition(K);
[ <Lie algebra of dimension 3 over Rationals>,
  <two-sided ideal in <Lie algebra of dimension 20 over Rationals>,
      (dimension 17)> ]
```

References

[CM93] David H. Collingwood and William M. McGovern. *Nilpotent orbits in semisimple Lie algebras*. Van Nostrand Reinhold Mathematics Series. Van Nostrand Reinhold Co., New York, 1993. 4, 11, 12

[dG11] Willem A. de Graaf. Computing representatives of nilpotent orbits of θ-groups. *J. Symbolic Comput.*, 46:438–458, 2011. 4

[dGVY12] W.A. de Graaf, E.B. Vinberg, and O.S. Yakimova. An effective method to compute closure ordering for nilpotent orbits of θ-representations. *J. Algebra*, 371:38–62, 2012. 4

[GE09] Willem A. de Graaf and Alexander G. Elashvili. Induced nilpotent orbits of the simple Lie algebras of exceptional type. *Georgian Mathematical Journal*, 16(2):257–278, 2009. `arXiv:0905.2743v1`[math.RT]. 4

[Gra08] Willem A. de Graaf. Computing with nilpotent orbits in simple Lie algebras of exceptional type. *LMS J. Comput. Math.*, 11:280–297 (electronic), 2008. 4, 13

[Gra11] Willem A. de Graaf. Constructing semisimple subalgebras of semisimple Lie algebras. *J. Algebra*, 325(1):416–430, 2011. 4

[Hel78] Sigurdur Helgason. *Differential geometry, Lie groups, and symmetric spaces*, volume 80 of *Pure and Applied Mathematics*. Academic Press Inc. [Harcourt Brace Jovanovich Publishers], New York, 1978. 4

[Hes79] Wim H. Hesselink. Desingularizations of varieties of nullforms. *Invent. Math.*, 55(2):141–163, 1979. 9

[Pop03] V. L. Popov. The cone of Hilbert null forms. *Tr. Mat. Inst. Steklova*, 241(Teor. Chisel, Algebra i Algebr. Geom.):192–209, 2003. English translation in: *Proc. Steklov Inst. Math.* 241 (2003), no. 1, 177–194. 9

[Vin75] E. B. Vinberg. The classification of nilpotent elements of graded Lie algebras. *Dokl. Akad. Nauk SSSR*, 225(4):745–748, 1975. 4, 20

[Vin76] E. B. Vinberg. The Weyl group of a graded Lie algebra. *Izv. Akad. Nauk SSSR Ser. Mat.*, 40(3):488–526, 709, 1976. English translation: Math. USSR-Izv. 10, 463-495 (1976). 4

[Vin79] E. B. Vinberg. Classification of homogeneous nilpotent elements of a semisimple graded Lie algebra. *Trudy Sem. Vektor. Tenzor. Anal.*, (19):155–177, 1979. English translation: Selecta Math. Sov. 6, 15-35 (1987). 4, 20

[VP89] È. B. Vinberg and V. L. Popov. Invariant theory. In *Algebraic geometry, 4 (Russian)*, Itogi Nauki i Tekhniki, pages 137–314. Akad. Nauk SSSR Vsesoyuz. Inst. Nauchn. i Tekhn. Inform., Moscow, 1989. English translation in: V. L. Popov and È. B. Vinberg, *Invariant Theory*, in: *Algebraic Geometry IV*, Encyclopedia of Mathematical Sciences, Vol. 55, Springer-Verlag, *Proc. Steklov Inst. Math.* 264 (2009), no. 1, 146–158. 9

Index

AddToDatabase, 26
AdmissibleLattice, 8
AmbientLieAlgebra, 13
ApplyWeylPermToWeight, 7
AreLinearlyEquivalentSubalgebras, 26

Branching, 23

CarrierAlgebra, 20
CartanSubspace, 21
CartanType, 5
CharacteristicsOfStrata, 9
ClosedSubsets, 27
ClosureDiagram, 19

DecompositionOfClosedSet, 27
DirectSumDecomposition, 9
DynkinIndex, 25

ExtendedCartanMatrix, 5

FiniteOrderInnerAutomorphisms, 16
FiniteOrderOuterAutomorphisms, 17

Grading, 18

InclusionsGraph, 24
InducedNilpotentOrbits, 14
IsomorphismOfSemisimpleLieAlgebras, 8
IsSpecialClosedSet, 28

KacDiagram, 17

LieAlgebraAndSubalgebras, 24
LieAlgebraOfClosedSet, 28

MakeDatabaseEntry, 26

NilpotentOrbit, 12
NilpotentOrbits, 12
NilpotentOrbitsOfThetaRepresentation, 18

Order, 17

PermAsWeylWord, 8
ProjectionMatrix, 22

RandomSL2Triple, 13
RegularSemisimpleSubalgebras, 23
RigidNilpotentOrbits, 15

SemiSimpleType, 13
SizeOfWeylGroup, 6
SL2Grading, 14
SL2Triple, 13, 14
SSSTypes, 24
SubalgebrasInclusion, 25

WeightedDynkinDiagram, 12
WeylGroupAsPermGroup, 6
WeylTransversal, 6
WeylWordAsPerm, 7