FPLSA

Finitely Presented Lie Algebras

1.2.6

3 January 2023

Vladimir Gerdt

Vladimir Kornyak

Vladimir Gerdt
Email: gerdt@jinr.ru
Homepage: http://compalg.jinr.ru/CAGroup/Gerdt/

Vladimir Kornyak
Email: vkornyak@gmail.com
Homepage: http://compalg.jinr.ru/CAGroup/Kornyak

Contents

Chapter 1

The FPLSA Package

This chapter describes the GAP package FPLSA , an interface to the `fplsa` program by V. Gerdt and V. Kornyak (version 4) for the computation with finitely presented Lie superalgebras.

At present GAP uses only the facility to compute a structure constants table of a finite dimensional Lie algebra over the rationals that is given by a finite presentation.

The package uses an external binary, probably it will only work on UNIX platforms.

1.1 Main Functions

A finitely presented Lie algebra is a quotient of a free Lie algebra by an ideal generated by a finite number of elements. In GAP a free Lie algebra can be created by the command `FreeLieAlgebra`; we refer to the GAP Reference Manual for more details. A finitely presented Lie algebra *K* can be constructed by *K* `:=` *L*`/`*rels*, where *L* is a free Lie algebra and *rels* a list of elements of *L* that constitute the relations that hold in *K*. Given a finitely presented Lie algebra we want to calculate a basis and a multiplication table of it. The interface to the FPLSA package comes with two related functions for doing that.

1.1.1 SCAlgebraInfoOfFpLieAlgebra

▷ `SCAlgebraInfoOfFpLieAlgebra(`*L*`, `*rels*`, `*limit*`, `*words*`, `*rels*`)` (function)

Let *L* be a free Lie algebra over the rationals, *rels* a list of elements in *L*, *limit* a positive integer and *words* and *rels* two booleans.

If the algebra *L* `/` *rels* is finite-dimensional and if a basis of this algebra can be constructed using elements in *L* that involve only words of length at most *limit* then `SCAlgebraInfoOfFpLieAlgebra` returns a record whose component `sc` contains an algebra that is isomorphic with *L* `/` *rels*. Otherwise `fail` is returned.

The function calls the `fplsa` standalone.

If *words* is `true` then the component `words` of the result record contains a list of elements in *L* that correspond to the basis elements.

If *rels* is `true` then the component `rels` of the result record contains a list of reduced relators in *L* that describes how algebra generators of *L* are expressed in terms of the basis elements.

Example

```
gap> LoadPackage( "fplsa" );
true
```

```
gap> L:= FreeLieAlgebra( Rationals, "x", "y" );
<Lie algebra over Rationals, with 2 generators>
gap> g:= GeneratorsOfAlgebra( L );; x:= g[1];; y:= g[2];;
gap> rels:= [ x*(x*y) - x*y, y*(y*(x*y)) ];
[ (-1)*(x*y)+(1)*(x*(x*y)), (1)*(((x*y)*y)*y) ]
gap> SCAlgebraInfoOfFpLieAlgebra( L, rels, 100, true, true );
rec( sc := <Lie algebra of dimension 4 over Rationals>,
  words := [ (1)*x, (1)*y, (1)*(y*x), (1)*((y*x)*y) ],
  rels := [ (1)*((y*x)*x)+(1)*(y*x), (1)*(((y*x)*y)*y), (1)*(((y*x)*y)*(y*x))
     ] )
```

1.1.2 IsomorphicSCAlgebra

▷ `IsomorphicSCAlgebra(`*`K[, bound]`*`)` (function)

computes a Lie algebra given by a multiplication table isomorphic to the finitely presented Lie algebra *K*. If the optional parameter *bound* is specified the computation will be carried out using monomials of degree at most *bound*. If *bound* is not specified, then it will initially be set to 10000. If this does not suffice to calculate a multiplication table of the algebra, then the bound will be increased until a multiplication table is found. If the computation was successful then a structure constants algebra will be returned isomorphic to *K*. Otherwise `fail` will be returned.

Example

```
gap> LoadPackage( "fplsa" );
true
gap> L:= FreeLieAlgebra( Rationals, "x", "y" );
<Lie algebra over Rationals, with 2 generators>
gap> g:= GeneratorsOfAlgebra( L );; x:= g[1];; y:= g[2];;
gap> rels:= [ x*(x*y) - x*y, y*(y*(x*y)) ];
[ (-1)*(x*y)+(1)*(x*(x*y)), (1)*(((x*y)*y)*y) ]
gap> K:= L/rels;
<Lie algebra over Rationals, with 2 generators>
gap> IsomorphicSCAlgebra( K );
<Lie algebra of dimension 4 over Rationals>
```

1.2 Auxiliary Variables of FPLSA

1.2.1 FPLSA

▷ `FPLSA` (global variable)

is the global record used by the functions in the package. Besides components that describe parameters for the standalone, the following components are used.

`Relation_size`
: parameter that controls the memory usage by the fplsa program,

`Lie_monomial_table_size`
: parameter that controls the memory usage by the fplsa program,

`Node_Lie_term_size`
parameter that controls the memory usage by the fplsa program,

`Node_scalar_factor_size`
parameter that controls the memory usage by the fplsa program,

`Node_scalar_term_size`
parameter that controls the memory usage by the fplsa program,

`progname`
the file name of the executable,

`T` structure constants table of the algebra under consideration,

`words`
list of elements in the free Lie algebra that correspond to the basis elements,

`rels`
list of relators in the free Lie algebra that are used to express redundant algebra generators in terms of the basis.

In order to be able to run the `fplsa` program successfully, it might be necessary to customize the parameters that control the memory the the program uses, to suit the computer the user is working on. In particular if the program exits with an error message of the form: `Error, the process did not succeed`, then it may be necessary to change these parameters.

Example

```
gap> LoadPackage( "fplsa" );
true
gap> L:= FreeLieAlgebra( Rationals, "x", "y" );;
gap> g:= GeneratorsOfAlgebra( L );; x:= g[1];; y:= g[2];;
gap> rels:= [ x*(x*y) - x*y, y*(y*(x*y)) ];;
gap> SCAlgebraInfoOfFpLieAlgebra( L, rels, 100, true, true );;
gap> FPLSA;
rec( Relation_size := 2500000, Lie_monomial_table_size := 1000000,
  Node_Lie_term_size := 2000000, Node_scalar_factor_size := 2000,
  Node_scalar_term_size := 20000, progname := "fplsa4",
  T := [ [ [ [  ], [  ] ], [ [ 3 ], [ -1 ] ], [ [ 3 ], [ 1 ] ],
          [ [ 4 ], [ 1 ] ] ],
      [ [ [ 3 ], [ 1 ] ], [ [  ], [  ] ], [ [ 4 ], [ -1 ] ], [ [  ], [  ] ] ],
      [ [ [ 3 ], [ -1 ] ], [ [ 4 ], [ 1 ] ], [ [  ], [  ] ], [ [  ], [  ] ] ],
      [ [ [ 4 ], [ -1 ] ], [ [  ], [  ] ], [ [  ], [  ] ], [ [  ], [  ] ] ],
      -1, 0 ], words := [ 1, 2, [ 2, 1 ], [ [ 2, 1 ], 2 ] ],
  rels := [ [ [ [ 2, 1 ], 1 ], 1, [ 2, 1 ], 1 ],
      [ [ [ [ 2, 1 ], 2 ], 2 ], 1 ], [ [ [ [ 2, 1 ], 2 ], [ 2, 1 ] ], 1 ] ] )
```

1.3 Installing the FPLSA Package

To install unpack the archive file in a directory in the `pkg` hierarchy of your version of GAP. (This might be the `pkg` directory of the GAP home directory; it is however also possible to keep an additional `pkg` directory in your private directories, see Section (**Reference: Installing a GAP Package**) of the

GAP Reference Manual for details on how to do this.) Go to the newly created `fplsa` directory and call `./configure` *`path`* where *`path`* is the path to the GAP home directory. So for example if you install the package in the main `pkg` directory call

Example

```
./configure ../..
```

This will fetch the architecture type for which GAP has been compiled last and create a `Makefile`. Now simply call

Example

```
make
```

to compile the binary and to install it in the appropriate place.

If you use this installation of GAP on different hardware platforms you will have to compile the binary for each platform separately. This is done by calling `configure` and `make` for the package anew immediately after compiling GAP itself for the respective architecture. If your version of GAP is already compiled (and has last been compiled on the same architecture) you do not need to compile GAP again, it is sufficient to call the `configure` script in the GAP home directory.

Index