Computer Science Journal of Moldova, vol.**4**, no.3(12), 1996

An Implementation in C of an Algorithm for Construction of Finitely Presented Lie Superalgebras

V.Gerdt V.Kornyak

Abstract

The purpose of this paper is to describe a C program FPLSA for investigating finitely presented Lie algebras and superalgebras. The program takes as input data a finite set of generators and relations for these elements. The relations have the form of Lie polynomials with coefficients being either integers or polynomials over integers in a given finite set of parameters. The program is based on an algorithm of constructing complete set of relations called also standard basis or Gröbner basis of ideal of free Lie (super)algebra generated by the input set of relations. The output data of the program are the Gröbner basis, the explicit form of basis elements for the quotient algebra of the free (super)algebra with respect to given ideal and also the table of their commutators.

1 Introduction

A *Lie algebra* L is an algebra over a commutative ring K with a unit. We shall consider here only zero characteristic rings. By definition, the non-commutative and non-associative multiplication in a Lie algebra called the *Lie product* and denoted by the commutator [,], satisfies the following axioms

$$[u,v] = -[v,u], \qquad skew-symmetry, \quad (1)$$

$$[u,[v,w]] + [v,[w,u]] + [w,[u,v]] = 0, \quad Jacobi\ identity. \quad (2)$$

©1996 by V.Gerdt, V.Kornyak

for all $u, v, w \in L$.

A *Lie superalgebra* is a $\mathbf{Z}_2$-graded algebra $L = L_{\bar{0}} \oplus L_{\bar{1}}$ with product $[\ ,\ ]$, i.e. if $u \in L_\alpha,\ v \in L_\beta,\ \alpha, \beta \in \mathbf{Z}_2 = \{\bar{0}, \bar{1}\}$, then $[u, v] \in L_{\alpha+\beta}$. The elements of $L_{\bar{0}}$ and $L_{\bar{1}}$ are called *even* and *odd*, respectively. The Lie product now satisfies the modified axioms

$$[u, v] = -(-1)^{\alpha\beta}[v, u], \tag{3}$$

$$[u, [v, w]] = [[u, v], w] + (-1)^{\alpha\beta}[v, [u, w]], \tag{4}$$

$$u \in L_\alpha,\ v \in L_\beta.$$

If every element of a Lie algebra can be formed by a linear combination of Lie products of a finite number of elements called *generators*, then such algebra is called *finitely generated*. If generators satisfy a finite number of relations having the form of polynomials in the algebra, then algebra called *finitely defined*. A finitely generated and finitely defined algebra is called *finitely presented*.

Any finite-dimensional algebra is, obviously, a finitely presented one. Besides, there is also a wide class of infinite-dimensional algebras which admit a finite presentation. Among such algebras there are, for example,

1. Kac and Kac-Moody (super)algebras [1] with their generalization known as Borcherds algebras [2].

2. Lie (super)algebras of string theories: Virasoro, Neveu-Schwarz and Ramond algebras [3].

3. Any simple finite-dimensional Lie algebra can be generated by two elements only with the number and structure of relations independent on the rank of the algebra. This allows one to define such objects as Lie algebras of matrices of a complex size $\mathbf{sl}(\lambda)$, $\mathbf{o}(\lambda)$ and $\mathbf{sp}(\lambda)$, where λ may be any complex number or even ∞ [4]. In a similar way, one can define some Lie superalgebras of supermatrices of a complex size [5].

2 Basic definitions

The set $X = \{x_1, x_2, \ldots, x_k\}$ of *generators* is a set of Lie (super)algebra elements such that any other element may be constructed by their Lie product, addition and multiplication by elements in K (scalars).

A *basis* $B(X)$ of a Lie (super)algebra is a minimal set of elements such that any other element is their linear combination over the ring K.

A *Lie monomial* $m(X)$ is an element of L constructed by Lie products of the generators x_i. A Lie polynomial $P(X)$ is a linear combination of Lie monomials.

A set of *defining relations* R is a set of Lie polynomial equalities of the form
$P(X) = 0$.

A Lie (super)algebra L is called *finitely presented* if both sets X and R are finite.

A finitely presented Lie (super)algebra L_F with an empty set of defining relations is called a *free Lie (super)algebra*.

Any finitely presented Lie (super)algebra can be considered as the quotient algebra of L_F by the two-sided ideal generated by relations R. Thus, it makes sense to deal with only those Lie monomials which constitute a basis of the free Lie (super)algebra, i.e., a set of Lie monomials which are not expressible in terms of others by means of (1-4).

It is known that a suitable basis of a free Lie (super)algebra can be formed by *regular* (ordinary Lie algebra) and *s-regular* (Lie superalgebra) monomials [15, 16].

Monomials are called *regular* if they are either generators or commutators of the form $[u, v]$ or $[w, [u, v]]$, where u, v, w are regular and $u < v$, $w \geq u$ with respect to some linear ordering of Lie monomials. Depending on the ordering chosen, one obtains a particular basis of a free Lie algebra. Among the whole variety of bases the most often used ones were introduced by Hall and Shirshov (see [15]).

Without getting into details we note that Shirshov and Hall orderings are analogous, in some sense, to the pure lexicographical and graded lexicographical orderings for associative words.

In our program we use Hall [15] monomial ordering.

To get a full set of s-regular monomials, we have only to add Lie squares of odd regular monomials.

For shortness the term "regular" will be used below for both regular and s-regular monomials.

A set R of relations generating an ideal I of a free Lie algebra is called *complete* if

(i). all the monomials in R are rewritten in terms of regular ones;

(ii). for each $v \in I$ also expressed in terms of regular monomials there exists a relation $r \in R$ such that the leading monomial of r is a submonomial of the leading monomial of v.

The complete set of relations R is called *minimal* if there is no $R' \subset R$ such that R' is also complete.

The complete set of relations is often called *canonical* or *Gröbner basis*. For Lie algebras the canonical bases were introduced in [6], and for Lie superalgebras in [7] (see also [12, 15, 16]).

Hereafter under *reduction* of a Lie polynomial modulo set of relations R we assume its rewriting in terms of regular monomials with substitutions of their submonomials in accordance with the relations.

3 Outline of the algorithm

For construction of finitely presented Lie algebras several algorithms were elaborated and implemented in Reduce [8, 9, 10]. In [11] the algorithm of paper [8] was extended to Lie superalgebras.

These algorithms are based on the sequential construction of a subset of Lie monomials of a given length (or weight) together with the relevant consequences of the initial relations. Then all the Jacobi identities are verified to select those Lie monomials which are linearly independent modulo both Jacobi identities and the extended set of relations. The selected monomials form the spanning set of the corresponding homogeneous component of the Lie algebra under construction. If the relations are homogeneous, then all the elements of the set are basis

ones. Otherwise, the set could be contracted by further computations with monomials and relations of larger length.

In the present program we use another algorithm [13] which is based, on separated construction of basis elements and derivation of the relation consequence which ends up with the complete relation set.

The complete set of relations is constructed step by step performing a Lie multiplication of every relation by generators and then reducing the product modulo other relations. If the reduced relation is nonzero it is added to the relation set. This computational procedure is much like to the involutive one for computation of Gröbner bases in commutative algebra [14].

In general terms, to rewrite a given set of Lie polynomials to the minimal Gröbner basis one should compute all possible consequences of these polynomials and remove all dependencies among them. The problem is to do that in the most efficient way. There were elaborated a number of optimizing criteria to avoid unnecessary reductions in computation of associative [12] and, especially, commutative [17, 18] Gröbner bases. Unfortunately, analogous criteria have not been formulated yet for the non-associative case in such a way as to be applied in practice.

Nevertheless, we use some optimizing methods to decrease the volume of computation. The most important of them are the following:

1. To produce new relations, that is, the consequences of a given relation *we multiply it by the generators only.* Consider the Jacobi identity (4) for the relation r

$$[[u, v], r] = [u, [v, r]] - (-1)^{\alpha\beta}[v, [u, r]], \tag{5}$$

   where α and β are the parities of u and v, respectively. Here there are three alternatives:

   (a) Both left- and right-hand sides of (5) are reduced to zero. In this case a new relation is not produced.

   (b) Both sides of (5) are reduced to nonzero expressions. In this case the new relation is obtained from r by successive

multiplications by u and v. By applying the formula (5) recursively the process of generating new relations is reduced to successive multiplication by generators.

(c) The right-hand side of (5) is reduced to zero while the left-hand side is not. In this case the corresponding consequence cannot be derived by successive multiplications by generators.

In our algorithm the latter case is treated separately once the subset of the complete set of relations has been generated in accordance with alternative (b).

2. *There is no need to multiply a relation by a generator which forms a regular monomial with the leading monomial* of the relation since all such consequences are automatically reduced to zero. Let the relation have the form $u + a = 0$ with leading monomial u, so that a contains the other terms. Multiplying the relation by a generator x we obtain $[x, u] + [x, a] = 0$. If $[x, u]$ is a regular monomial we must replace u by $-a$. It leads to the identity $-[x, a] + [x, a] \equiv 0$ and, hence, does not produce a new relation.

3. *All computations*, starting with processing the input relations, *are executed modulo identities* (1-4) and modulo *the relations treated up to that moment.* This allows us to minimize resimplification of the calculated structures and to keep the system of Lie monomials and relations as compact as possible at all times in the computation.

The input data for the algorithm are:

- The ordered set of generators $X = \{x_1, x_2, \ldots\}$ with prescribed parities $\alpha_i \in \mathbf{Z}_2$ and positive integer weights w_i (= 1 by default).
- The set of scalar parameters $P = \{p_1, p_2, \ldots\}$ if they are present in the relations.
- The set of defining relations $R = \{r_1, r_2, \ldots\}$, where r_i are Lie polynomials with coefficients from the commutative ring $\mathbf{Z}[p_1, p_2, \ldots]$ of scalar polynomials.

- The limiting number of relations to be generated.

The output data for the algorithm are:

- The interreduced set of consequences of the input relations $\tilde{R} = \{\tilde{r}_1, \ldots\}$.
- The list $E = \{e_1, e_2, \ldots\}$ of Lie algebra elements linearly independent modulo $\tilde{R}$.
- The commutator table $[e_i, e_j] = c_{ij}^k e_k$.
- The table of scalar polynomials in p_i which have been treated as nonzero during computation. Particular values of p_i providing vanishing of these polynomials may cause branching of computation and, hence, changes of the algebra structure.
- The dimensions of homogeneous components.

There are three steps in the algorithm:

1. *Generation of the relation set $\tilde{R} = \{\tilde{r}_1, \tilde{r}_2, \ldots\}$ of the consequences of the initial set R.* This step executes the subsequent multiplying of relations by generators adding nonzero results to the set of relations and substituting these new relations into the other ones. The process terminates if either all newly arising relations are reduced to zero or the number of relations goes up to the limit fixed at input. The second case means that either the algebra is infinite-dimensional or the input limiting number of the relations is too small.

2. *Completion of the set $E = \{e_1, e_2, \ldots\}$.* Some elements e_i are obtained at Step 1 as Lie (sub)monomials of $\tilde{r}_i$. However, generally, the set E produced must be completed by those regular commutators of already existing elements which do not occur in E. In doing so one must verify if new elements are indeed independent. It may happen that there exists a Jacobi identity containing the new element as a term and such that this identity is reduced to

a new relation missing in the output of Step 1. If so one should add the new relation to $\tilde{R}$ and go back to Step 1. Besides, in the case of a Lie superalgebra the Lie squares of the odd elements are also to be added. This step guarantees the termination of the algorithm in the case of finite-dimensional algebra because *all* possible regular monomials and *all* relations involving them are constructed.

3. *Construction of the commutator table.* At this step the commutators of the elements obtained at Step 2 are computed directly. The commutators produced are reduced modulo the relation set $\tilde{R}$.

If the above algorithm terminates due to the input limiting number of relations we can not distinguish two alternatives: either the algebra is finite-dimensional but the limiting number is too small or the algebra is infinite-dimensional. In the last case the truncated output makes sense only if all $\tilde{r}_i \in \tilde{R}$ are homogeneous. In this case we obtain an initial part of the whole Gröbner basis, and the set E forms a subbasis of the Lie (super)algebra under construction.

Otherwise, the algorithm termination means we have a reduced and, hence, minimal finite Gröbner basis. Generally, it does not mean that the algebra is necessarily finite-dimensional. However, if at the last iteration of Step 2 no new elements e_i were obtained, then we are done with a finitely-dimensional algebra. In the case of a finite Gröbner basis generating an infinite-dimensional algebra only those additional elements are to be included in set E which are regular Lie pairs of elements e_i obtained at Step 2.

4 Description of the program

The algorithm has been implemented in the C language. We used Borland C++ version 4.5 and GCC version 2.6.3 compilers under MS-DOS 6.22. We hope, however, that the C text can be compiled by other compilers because it contains only the standard C statements.

The source code has a total length of almost 8000 lines and contains about 150 C functions realizing: top level algorithms, Lie (super)algebra operations, manipulation with scalar polynomials, multiprecision integer arithmetic, substitutions, list processing, input and output handling, etc.

The code is contained in the file FPLSA.C. Executable file, message file FPLSA.MSG and input files must live in one directory. The program produces result both on the screen and into the file FPLSA.SES in the same directory.

Our *C* code can be easily modified for working over finite fields and generalized to handle color Lie superalgebras with finite grading groups [15]. More important development could be adding of additional means for analysis of the structure of obtained algebras, because it is not clear what to do with thousands of output relations.

4.1 Run of the program

The program starts with the request

```
Enter name of existing or new input file ->
```

The user must enter the name of the file containing input data or the name of the file to be created. The default extension for the input file name is .IN and such an extension can be omitted. If there is no file in the current directory with the typed name, then the program asks

```
File doesn't exist. Create new file? (y/n) ->
```

If the user enters **n** (or **N**) the program is cancelled. Otherwise the program asks

```
Enter generator symbols ->
```

The user should type the names of generators separated by blanks (or commas) and then push "Enter". The odd generators of Lie superalgebra must be prefixed by the sign "−". The user may enter the names with subscripts printed at output, e.g., `X_1, Z_a`. Then the program asks

```
Enter scalar parameters ->
```

If the task includes parameters the user must enter their names just in the same way as it done above for generators. Otherwise he or she must press the key "Enter". After the request

```
Enter defining relations ->
```

the user must enter the left hand sides of relations separated by pressing of "Enter" if there are relations in the task and push "Enter" twice, otherwise. All the Lie monomials of input relations must be in the standard form with square brackets. The syntax of the input expressions assumes blank (not asterisk) for multiplication and the sign "^" for raising to power. Note that the program is working over a coefficient ring, rather then over a field. Thus, the division sign should be excluded from the input relations.

At this moment the program reproduces the input data and reads the input system of relations. Note that after reading an input relation the program automatically reduces every its Lie monomial into the regular form and performs interreductions of the relations have been input.

If the set of input relations is not empty and has not been reduced to the empty set in the process of interreductions the program asks

```
Enter limiting number for relations ->
```

The user must enter integer number exceeding the initial number of relations. This limiting number is needed to prevent infinite generation of the consequences of initial relations in the case of infinite Gröbner basis.

The next question of the program is

```
Right-normed output for Lie monomials? (y/n) ->
```

If the user chooses **y** the Lie monomials take the *right-normed* form in the output. It means, for instance, that Lie monomials $[x,[x,[y,x]]$, $[[x,y],[x,[x,y]]]$ are to be presented in the output as x^2yx and $(xy)x^2y$, respectively. This form is often used for the purpose of compactness in

representing of non-associative monomials, especially for those of high degrees. Otherwise the standard form with explicit square brackets is used in the output expressions.

Reply **y** to the following question of the program

```
Standard grading assumes unit weight for every generator.
Do you want to use a different grading? (y/n) ->
```

causes, e. g., the dialogue

```
Enter non-zero positive integer weights for generators:
Weight for X -> 2
Weight for Y -> 1
Weight for Z -> 3
```

One should be noted that prescribing non-trivial weights to generators in such a way to gain homogeneity of the input relations may allow one to handle infinite-dimensional finitely presented algebras.

If the user types **n** the program prescribes unit weights to all generators.

In the case of interreduction of all the input relations to zero the program gives the message

```
The algebra is free.
```

and asks

```
Enter limiting weight for basis ->
```

The user must enter the maximum weight of basis elements of free algebra which should be printed out. These basis elements and their commutator table are printed and the program ends up its work as soon as the basis elements of the limiting weight are constructed.

If the algebra is not free the program generates the consequences of the relations performing their interreduction. If in this process a newly arising relation has parametric coefficients at the leading Lie monomial, then the computation can be branched depending on whether that coefficient is equal to zero or not. If the program has been compiled with defined constant `PUT_BRANCHING` then the produced relation is printed with foregoing message

```
*** Possible parameter branching in relation:
```

provided also with additional clarifications

```
*** Parametric factor of relation:
```

in the presence of common parametric factor for all the terms in the relation, and

```
*** Parametric coefficient at leading term of relation:
```

in the case without such common parametric factors. In the latter case vanishing the leading coefficient leads to change of the leading monomial while in the former case the whole relation is cancelled.

This feature allows to choose most interesting cases of branching. The program memorizes all such parametric coefficients in the table which is printed out after the end of the generation and reduction process. This table is useful for the systematic classification of the structure of algebra for different values of parameters.

Finally the program prints out

- the reduced system of relations expressed in terms of the initial generators;
- basis elements in the form $E_i = ..., O_i = ...$ for even and odd elements, respectively;
- non-zero commutators of basis elements and dimensions of the homogeneous, with respect to the input grading, components of the algebra.

Note that in the case of infinite-dimensional algebra the end of list of basis elements may contain some wrong elements. These "boundary" violations are caused by incompleteness of reduction of higher degree relations.

In the case of infinite-dimensional algebra with a finite Gröbner basis the program prints separately "free" basis elements generating infinite-dimensional ideal. These elements have the form $e_i = ..., o_i = ...$ in the output.

The program prints also the time and space statistics.

4.2 Control of compiling

We present here the list of constants and macros defined by preprocessor commands with explanation of their meaning.

`#define SUPERALGEBRA` switches on the special instructions for handling superalgebra. Of course, as ordinary Lie algebra is a particular case of superalgebra, and it can also be processed with the statement being on. Nevertheless the computation of ordinary algebra is slightly faster with the statement switched off.

`#define SPACE_STATISTICS` collects information on the space consumed by the generated data structures.

`#define LIMITS` controls the sizes of different working arrays and stacks. If the index variable of one of them exceeds the size, then the program is terminated with the corresponding message. Besides, the actual maximum values of indices for a number of arrays and stacks are printed out.

`#define PUT_BRANCHING` prints out the intermediate relations with parametric coefficients at the leading monomials showing at which ones the branching of computation is possible.

`#define GENERATION_STATISTICS` prints out the total number of generated relations, the number of those reduced to zero and their relative percentage.

`#define TEST_FUNCTION` allows to run separately any C function of the program for purpose of detailed debugging.

`#define DEBUG` serves for tracing input and output of a C function during the computation. Almost any function can be included in or excluded from this tracing by defining the correspondent constants.

`#define CHECK_NODE_BALANCE` shows the loss of memory in the result of garbage accumulation.

`#define D_CHECK_EXACTNESS_OF_DIVISION` controls the divisions of multiprecision integers in the program which must not have remainders. The statement allows to detect possible violations in the process of

debugging.

`#define D_CHECK_COMPLETENESS` controls expressibility of all commutators of basis elements in terms of existing basis elements.

`#define D_PUT_ORDINAL_STACK`, `#define D_PUT_ORDINAL_TABLE`, `#define D_PUT_RELATIONS`, `#define D_PUT_SUB_SUM_STACK`.
These statements allow to switch on or off the printing of large tables in the debugging process.

`#define D_BEGIN()` provides conditions to avoid unnecessary outputs over debugging.

`#define D_EXIT` sets the conditions to terminate computation in the process of debugging.

4.3 Sizes, limitations and constants for input and output

The defined constants below determine sizes of different arrays. Their values must be set depending on the memory available on the user's computer.

`SIZE_NODES` defines size of the pool of free nodes for the list structures.

`SIZE_ORDINAL_TABLE` gives size of the table for Lie monomials.

`SIZE_SUB_SUM_STACK` defines size of the stack for working subexpressions.

`SIZE_ORDINAL_STACK` gives size of the stack for working ordinals of Lie monomials.

`SIZE_MIN_OF_WEIGHT` controls size of the arrays keeping watch on the ends of homogeneous components of the sets of relations and Lie monomials.

`MAX_N_NAMES` gives maximum summary number of the input generator and parameter names.

`SIZE_BIG_N_W` determines size of the working arrays in two-byte limbs for operations with multiprecision numbers.

`SIZE_PARA_MON_W` fixes size of the working arrays for operations with parameter monomials.

`MAX_PARAMETER_DEGREE` gives the maximum admissible degree of a parameter in the whole set of monomials.

`I_SIZE_COEFF_SUM_TABLE`, `D_SIZE_COEFF_SUM_TABLE` define the initial size and increment of the table for non-zero parametric coefficients.

`MAX_SIZE_NAME` gives the maximum length of the input name for generators and parameters.

`SIZE_INPUT_STRING` defines size of the working string for reading the input expressions.

`MAX_FULL_OUTPUT_SIZE, SIZE_OUTPUT_PART` control the output for the relations. The resulting number of reduced relations may be very large, e.g., Serre relations for exceptional Lie algebra E_8 produces more than 23000 relations. The program prints out the full set of reduced relations only if its number does not exceed the constant `MAX_FULL_OUTPUT_SIZE`. Otherwise only the first and last `SIZE_OUTPUT_PART` relations are printed.

`LINE_LENGTH` gives the width of the output page.

`I_SIZE_OUT_LINE, D_SIZE_OUT_LINE` define the initial size and increment of the working string for preparing the current portion of output.

The current implementation has the following limitations due to the internal data structure used.

The maximum number of input generators is 256.
The maximum number of input parameters is 256.
The maximum degree of a single parameter is 255.
The maximum number of Lie monomials is 1 073 741 824.
The maximum weight of Lie monomials is 255.
The maximum number of relations is 4 194 304 if the program is compiled for superalgebras and 8 388 608, otherwise.
The maximum summary number of two-byte limbs for representing a monomial together with multiprecision number

is 2 147 483 647.

The following symbolic constants are used for input and output.

`DOWN_INPUT_SIGN` is used for subscripts in the input names, currently "_".

`BASIS_NAME_EVEN_MAIN`, `BASIS_NAME_ODD_MAIN`, `BASIS_NAME_EVEN_FREE`, `BASIS_NAME_ODD_FREE` are used for output of basis elements which occur in a Gröbner basis and for those generating a free ideal in the infinite-dimensional case with a finite Gröbner basis. Currently they are 'E', 'O', 'e', 'o', respectively.

5 Examples of output

The following session files have been produced on a 66 Mhz MS-DOS based AT/486 computer. We use here the 32bit *GCC* compiler and *GO32* DOS extender.

5.1 Finite-dimensional Lie superalgebra with parameters

The first illustrative example, arises in investigating integrability [20] of $N = 1$ supersymmetric fermionic extension of the Korteveg-de Vries (KdV) equation [19]. The supersymmetric version of the conventional KdV equation

$$u_t = -u_{xxx} + 6\,u\,u_x$$

is given by

$$\left\{ \begin{array}{lcl} u_t &=& -u_{xxx} + 6\,u\,u_x - 3\,\phi\,phi_{xx}, \\ \phi_t &=& -\phi_{xxx} + 3\,u_x\phi + 3\,u\,\phi_x. \end{array} \right. \tag{6}$$

Here $u(x,t)$ and $\phi(x,t)$ are even (bosonic) and odd (fermionic) components of the extended fermionic field $\Phi = \phi + \theta\,u$, respectively, and θ is a Grassman variable.

The prolongation algebra [20] of (6) is generated by the system of seventeen relations in three even generators x_1, x_2 and x_2 and one odd generator y. We deform the original system introducing two parameters p and q to get a parametric coefficient ring in order to illustrate the classification aspects of the problem.

The standard form of Lie monomials is used here in the output.

```
Enter name of existing or new input file -> skdv-m-r.in
Input data:
Generators, Manin-Radul SKDV: x_1 x_2 x_3 -y;
Parameters, genuine p = -2, q = 6: p q;
Relations:

3 [[[y,x_1],x_2],x_1] + [[x_2,[y,x_2]],x_3] - 3 [y,x_1];
[[y,x_2],x_1];
[[x_2,[y,x_2]],[x_2,[y,x_2]]];
[[x_2,[y,x_2]],x_1];
[[x_2,[y,x_2]],x_2];
[y,x_3] + [[[y,x_1],x_1],x_1];
[x_1,[[x_2,[y,x_2]],y]];
[x_1,[y,y]] + p [x_1,x_2];
[x_2,x_3] - [x_1,[x_1,[x_1,x_2]]];
[[[x_2,[y,x_2]],y],x_3] - 3 [x_1,[x_2,[x_1,x_2]]] + q [x_1,x_2];

[[y,x_2],y];
[[x_2,[y,x_2]],[[x_2,[y,x_2]],y]];
[y,[[x_2,[y,x_2]],y]];
[y,[x_2,[x_1,x_2]]];
[x_1,x_3];
[x_2,[[x_2,[y,x_2]],y]];
[x_2,[x_2,[x_1,x_2]]];

Right-normed output for Lie monomials? (y/n) -> n
Standard grading assumes unit weight for every generator.
Do you want to use a different grading? (y/n) -> n
Enter limiting number for relations -> 300

Initial relations:

(1)  [x ,x ] = 0
       1  3
```

```
(2)  [y,[x ,x ]] - [x ,[x ,y]] = 0
          1  2       2   1

(3)  2 [y,[x ,y]] + p [x ,x ] = 0
            1           1  2

(4)  [y,[x ,y]] = 0
          2

(5)  [x ,[x ,[x ,x ]]] - [x ,x ] = 0
       1   1   1  2        2  3

(6)  [x ,[x ,[x ,y]]] + [x ,y] = 0
       1   1   1          3

(7)  [x ,[x ,[x ,x ]]] = 0
       2   2   1  2

(8)  [x ,[x ,[x ,y]]] = 0
       2   2   2

(9)  [y,[x ,[x ,x ]]] = 0
          2   1  2

(10)  3 [[x ,x ],[x ,y]] - [x ,[x ,[x ,y]]] + 3 [x ,[x ,[x ,y]]]
           1  2    1         3   2   2            2   1   1

      -3 [x ,y] = 0
          1

(11)  [[x ,x ],[x ,y]] = 0
         1  2    2

(12)  [y,[y,[x ,[x ,y]]]] = 0
              2   2

(13)  [[x ,y],[x ,[x ,y]]] = 0
         1      2   2

(14)  [[x ,y],[x ,[x ,y]]] = 0
         2      2   2
```

```
(15)  [[x ,y],[x ,[x ,y]]] + [y,[x ,[x ,[x ,y]]]]
         3      2   2             3   2   2
      - 3 [x ,[x ,[x ,x ]]] +q [x ,x ] = 0
            2   1   1  2         1  2

(16)  [[x ,[x ,y]],[x ,[x ,y]]] = 0
         2   2       2   2

(17)  [[x ,[x ,y]],[y,[x ,[x ,y]]]] = 0
         2   2          2   2


Non-zero parametric coefficients:

(1)  p

(2)  q

(3)  p - 1

      2
(4)  q  - 9 q + 18

(5)  q - 3


Reduced relations:

(1)  [x ,x ] = 0
       1  2

(2)  [x ,x ] = 0
       1  3

(3)  [x ,y] = 0
       1

(4)  [x ,x ] = 0
       2  3

(5)  [x ,y] = 0
       3
```

```
(6)  [x ,[x ,y]] = 0
       3   2

(7)  [y,[x ,y]] = 0
          2

(8)  [x ,[x ,[x ,y]]] = 0
       2   2   2

(9)  [x ,[x ,[x ,y]]] = 0
       3   2   2

(10)  [[x ,y],[x ,y]] + [y,[x ,[x ,y]]] = 0
         2      2            2   2

(11)  [y,[y,[x ,[x ,y]]]] = 0
               2   2

(12)  [[x ,y],[x ,[x ,y]]] = 0
         2      2   2

(13)  [[x ,y],[y,[x ,[x ,y]]]] = 0
         2         2   2

(14)  [[x ,[x ,y]],[x ,[x ,y]]] = 0
         2   2       2   2

(15)  [[x ,[x ,y]],[y,[x ,[x ,y]]]] = 0
         2   2          2   2


Basis elements:

(1)  E  = x
      1    1

(2)  E  = x
      2    2

(3)  E  = x
      3    3
```

```
(4)  O  = y
      4

(5)  O  = [x ,y]
      5     2

(6)  E  = [y,y]
      6

(7)  O  = [x ,[x ,y]]
      7     2   2

(8)  E  = [y,[x ,[x ,y]]]
      8        2   2


Non-zero commutators of basis elements:

(1)  [E ,O ] = O
       2  4     5

(2)  [O ,O ] = E
       4  4     6

(3)  [E ,O ] = O
       2  5     7

(4)  [O ,O ] = - E
       5  5       8

(5)  [O ,O ] = E
       4  7     8


Dimensions of homogeneous components:

dim G   = 4
     1

dim G   = 2
     2
```

```
dim G   = 1
     3

dim G   = 1
     4

Time: 0.16 sec
Number of relations:        128  Relation space:       1024 bytes
Number of ordinals:         257  Ordinal space:        3084 bytes
Number of nodes:            629  Node space:           7548 bytes
Total space: 11656 bytes
```

Here E_i and O_i are even and odd basis elements, respectively. Note that much like to the commutative Gröbner basis method [17], the final structure of the reduced relations and even their number depends essentially on the ordering of the generators chosen.

It can easily be seen that for the generic values of parameters p and q we have a eight-dimensional nilpotent Lie superalgebra. The branching of the algebra structure is possible at the values of parameters $p = 0$, $p = 1$, $q = 0$, $q = 3$ and $q = 6$. The computations with particular values $p = 0$ or $q = 3$ or $q = 6$ allow to suggest that the algebra becomes an infinite-dimensional one whereas other singular values of parameters do not change the structure of the algebra. In [20] the algebra for $q = 6$ is proved to be infinite-dimensional and identified with the product of the eight-dimensional nilpotent algebra and the positive subalgebra of the twisted Kac-Moody superalgebra $C^{(2)}(2)$.

5.2 Infinite-dimensional Lie algebra

The defining relations for this example were proposed in [12]. These relations define infinite-dimensional subalgebra of Lie algebra of vector fields on a line (or on a circle). It is also a subalgebra of the *Virasoro algebra*. We use here the right-normed output for Lie monomials just for a change.

```
Enter name of existing or new input file -> ufnarovs.in
Input data:
Generators, Ufnarovsky's example of Virasoro-like algebra,
natural weights - w(X) = 1, w(Y) = 2: X Y;
Relations:
```

```
[X,[X,[X,Y]]] - 6 [Y,[X,Y]];
2 [Y,[Y,[X,Y]]] - 3 [[X,Y],[X,[X,Y]]];

Right-normed output for Lie monomials? (y/n) -> y
Standard grading assumes unit weight for every generator.
Do you want to use a different grading? (y/n) -> y
Enter non-zero positive integer weights for generators:

Weight for X -> 1
Weight for Y -> 2

Enter limiting number for relations -> 9

Initial relations:

              3
(1)  6 YXY - X Y = 0

            2        3
(2)  3 (XY)X Y - 2 YX Y = 0


Reduced relations:

              3
(1)  6 YXY - X Y = 0

         2     4
(2)  6 YX Y - X Y = 0

          3     5
(3)  10 YX Y - X Y = 0

             2     5
(4)  15 (XY)X Y - X Y = 0

          4     6
(5)  30 YX Y - X Y = 0

             3     6
(6)  15 (XY)X Y - X Y = 0
```

```
      5
(7)  YX Y = 0

            4     7
(8)  30 (XY)X Y - X Y = 0

          2   3     7
(9)  30 (X Y)X Y - X Y = 0

 .
 .
 .

Basis elements:

(1)  E  = X
      1

(2)  E  = Y
      2

(3)  E  = XY
      3

           2
(4)  E  = X Y
      4

           3
(5)  E  = X Y
      5

           4
(6)  E  = X Y
      6

           5
(7)  E  = X Y
      7
```

```
          6
(8)  E  = X Y
      8

          7
(9)  E  = X Y
      9

 .
 .
 .

Non-zero commutators of basis elements:

(1)  [E ,E ] = E
       1  2     3

(2)  [E ,E ] = E
       1  3     4

(3)  [E ,E ] = 1/6 E
       2  3         5

(4)  [E ,E ] = E
       1  4     5

(5)  [E ,E ] = 1/6 E
       2  4         6

(6)  [E ,E ] = 1/15 E
       3  4          7

(7)  [E ,E ] = E
       1  5     6

(8)  [E ,E ] = 1/10 E
       2  5          7

(9)  [E ,E ] = 1/15 E
       3  5          8
```

```
(10)  [E ,E ] = 1/30 E
        4  5          9

(11)  [E ,E ] = E
        1  6     7

(12)  [E ,E ] = 1/30 E
        2  6          8

(13)  [E ,E ] = 1/30 E
        3  6          9

(14)  [E ,E ] = E
        1  7     8

(15)  [E ,E ] = E
        1  8     9

 .
 .
 .

Dimensions of homogeneous components:

dim G   = 1
     1

dim G   = 1
     2

dim G   = 1
     3

dim G   = 1
     4

dim G   = 1
     5

dim G   = 1
     6
```

```
dim G   = 1
     7

dim G   = 1
     8

dim G   = 1
     9

 .
 .
 .

Time: 0.05 sec
Number of relations:          9  Relation space:         72 bytes
Number of ordinals:          21  Ordinal space:         252 bytes
Number of nodes:             36  Node space:            432 bytes
Total space: 756 bytes
```

In the case of an infinite-dimensional algebra the program prints out only those commutators which are expressible in terms of the basis elements computed.

Note that the chosen input limiting number for relations gives the true initial part of the algebra under consideration. Such a well formed output takes place at the limiting numbers 4, 6, 9, 12, 16, 21, 26,..., whereas other values lead to appearing of some wrong higher basis elements.

6 Acknowledgments

We are grateful to D. Leites, A.A.Mikhalev and V. Ufnarovsky for fruitful discussions and useful examples. This work was supported in part by the INTAS project No. 93-0030.

References

[1] Kac, V.G. (1990). *Infinite dimensional Lie algebras.* Cambridge University Press, Cambridge, third edition.

[2] Gebert, R. W. (1994). *Beyond Affine Kac-Moody Algebras in String Theory,* DESY 94-209, Hamburg.

[3] Leites, D. (1984). *Lie superalgebras.* VINITI. Itogi Nauki i Tekhniki. Modern Problems in Mathematics. Recent Progress, **25**, Moscow, pp.3-50 (in Russian).

[4] Grozman, P., Leites, D. (1995a). Defining Relations Associated with the Principal **sl**(2)-subalgebras. To appear.

[5] Grozman, P., Leites, D. (1995b). Lie Superalgebras of Supermatrices of Complex Size. To appear.

[6] Shirshov, A.I. (1962). Some Algorithmic Problems about Lie Algebras. *Sibirsk. Mat. Zh.* **3**, pp.292-296.

[7] Mikhalev, A.A. (1989). The Junction Lemma and the Equality Problem for Color Lie Superalgebras. *Vestnik. Moskov. Univ. Ser. I. Mat. Mekh.* **5**, pp.88-91.

[8] Gragert, P.K.H. (1989). Lie Algebra Computations. *Acta Applicandae Mathematicae* **16**, 231-242.

[9] Akselrod, I.R., Gerdt, V.P., Kovtun, V.E., Robuk, V.N. (1991). Construction of a Lie Algebra by a Subset of Generators and Commutation Relations, in *Computer Algebra in Physical Research*, eds. D.V.Shirkov, V.A.Rostovtsev and V.P.Gerdt, World Scientific Publ.Co., Singapore, pp.306-312.

[10] Gerdt, V.P., Robuk, V.N., Severyanov V.M. (1994). On Construction of Finitely Presented Lie Algebras. Preprint JINR E5-94-302, Dubna. Submitted to *Comput. Maths. & Math. Phys.*.

[11] Roelofs, G.H.M. (1991). *The LIESUPER Package for REDUCE,* Memorandum 943, Univ. of Twente, Netherlands.

[12] Ufnarovsky, V.A. (1990). *Combinatorial and asymptotic methods in algebra.* VINITI. Itogi Nauki i Tekhniki. Modern Problems in

Mathematics. Fundamental Branches, **57**, Moscow, pp.5-177 (in Russian), to appear in EMS-57 (1995) (in English).

[13] Gerdt, V.P., Kornyak,V.V. (1996). Construction of Finitely Presented Lie Algebras and Superalgebras, to appear in *J. Symb. Comp.*.

[14] Gerdt, V.P., Blinkov, Yu.A. (1996). Involutive Bases of Polynomial Ideals. Preprint-Nr.01/1996, Naturwissenschaftlich-Theoretisches Zentrum, Universität Leipzig, 1996. Submitted to *J. Symb. Comp.*.

[15] Bahturin, Yu.A., Mikhalev, A.A., Petrogradsky, V.M., Zaicev, M.V. (1992). *Infinite dimensional Lie superalgebras.* Walter de Gruyter. Berlin- New York.

[16] Mikhalev, A.A., Zolotykh A.A. (1995). *Combinatorial Aspects of Lie Superalgebras*. CRC Press, Boca Raton, New York.

[17] Buchberger, B. (1985). Gröbner bases: an algorithmic method in polynomial ideal theory, in *Recent Trends in Multidimensional System Theory*, ed. N.K.Bose, (D.Reidel), pp.184-232.

[18] Becker, T., Weispfenning, V., Kredel, H. (1993). *Gröbner Bases. A Computational Approach to Commutative Algebra*. Graduate Texts in Mathematics **141**, Springer-Verlag, New York.

[19] Manin, Yu.I., Radul, A.O. (1985). A supersymmetric extension of the Kadomtsev-Petviashvili hierarchy, *Commun. Math. Phys.* **98**, 65-77.

[20] Roelofs, G.H.M. (1993). *Prolongation structures of supersymmetric systems.* Ph.D. Thesis, Univ. of Twente, Enschede, Netherlands.

V.P.Gerdt, V.V.Kornyak,
Laboratory of Computing Techniques and Automation
Joint Institute for Nuclear Research
141980 Dubna, RUSSIA

Received 20 Juny, 1996