Discrete Mathematics and Theoretical Computer Science **1**, 1997, 217–228

An algorithm for analysis of the structure of finitely presented Lie algebras

Vladimir P. Gerdt and Vladimir V. Kornyak

Laboratory of Computing Techniques and Automation, Joint Institute for Nuclear Research, 141980 Dubna, Russia
E-Mail: `gerdt@jinr.dubna.su`

We consider the following problem: what is the most general Lie algebra satisfying a given set of Lie polynomial equations? The presentation of Lie algebras by a finite set of generators and defining relations is one of the most general mathematical and algorithmic schemes of their analysis. That problem is of great practical importance, covering applications ranging from mathematical physics to combinatorial algebra. Some particular applications are construction of prolongation algebras in the Wahlquist–Estabrook method for integrability analysis of nonlinear partial differential equations and investigation of Lie algebras arising in different physical models. The finite presentations also indicate a way to q-quantize Lie algebras. To solve this problem, one should perform a large volume of algebraic transformations which is sharply increased with growth of the number of generators and relations. For this reason, in practice one needs to use a computer algebra tool. We describe here an algorithm for constructing the basis of a finitely presented Lie algebra and its commutator table, and its implementation in the C language. Some computer results illustrating our algorithm and its actual implementation are also presented.

Keywords: Lie algebras, structure analysis

1 Introduction

A *Lie algebra* L is an algebra over a commutative ring K with a unit. The non-commutative and non-associative multiplication in a Lie algebra is called the *Lie product*, and is usually denoted by the commutator [,]. By definition, the Lie product satisfies the following axioms:

$$[u,v] = -[v,u], \qquad skew-symmetry \tag{1}$$

$$[u,[v,w]]+[v,[w,u]]+[w,[u,v]] = 0, \quad Jacobi\ identity \tag{2}$$

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

A finitely presented algebra is determined by a finite number of some of its elements called *generators* subject to a finite number of relations having the form of polynomials in the algebra. Any finite-dimensional algebra is, obviously, a finitely presented one. Nevertheless, the concept of a finite presentation also covers a wide class of infinite-dimensional algebras. Some examples of such algebras having a natural constructive definition in terms of a finite number of generators and relations are:

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

1365–8050 © 1997 Chapman & Hall

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].

The constructions of the last item are of interest in applications to integrable systems, for example, to the Leznov–Saveliev equation [5] also known as the two-dimensional Toda lattice or vector-valued Liouville equation. They also give a prescription for q-quantizing the Lie algebras of matrices.

Below we describe an algorithm for determining the explicit structure of a finitely presented Lie algebra from the defining relations, i.e. for constructing its basis and commutator table, and describe its implementation in C. In fact, our algorithm, if it terminates, produces the Gröbner basis [6] for the case of such non-commutative and non-associative objects as Lie algebras. The algorithm and its actual implementation is illustrated by a rather simple example arising in investigation of the Burgers equation. We also present the table containing computational statistics for Serre relations of all simple Lie algebras up to rank 10.

2 Algorithm

To construct finitely presented Lie algebras, several algorithms were elaborated and implemented in Reduce [7, 8, 9]. The refinement of Gragert's algorithm with extension to superalgebras was carried out by Roelofs [10].

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 algorithm the construction of basis elements and the relation consequences is done separately. Besides, at each step of the new relation generation we perform a Lie multiplication by a single generator only. This procedure allows one to decrease the number of Jacobi identities to be verified. From our computational experiments (see Sect. 4) this approach seems to be rather efficient. Before a more detailed algorithm description we explain some terms used in the text.

The set $X = \{x_1, x_2, \ldots, x_k\}$ of *generators* is a set of Lie 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 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 algebra L is called *finitely presented* if both sets X and R are finite.

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

Any finitely presented Lie 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 algebra, i.e. a set of Lie monomials which are not expressible in terms of others by means of (1–2).

It is known that a suitable basis of a free Lie algebra can be formed by *regular* Lie monomials [11, 12]. 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 Bahturin *et al*. [11]).

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

In the present algorithm we use Hall ordering because it is compatible with the natural grading by length or by positive weight. The use of Shirshov ordering may be more convenient for analysis of a Lie monomial structure. However, as in the associative case, this ordering typically decreases the efficiency because a Lie monomial may contain a greater submonomial that complicates the structure of data and algorithm.

The algorithm below, if it terminates, produces the complete set of relations. The left hand sides of the latter form a set of Lie polynomials which is often called a Gröbner basis (see Ufnatovsky [6]).

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.

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.

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 [6] and commutative [13, 14] 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 (2) for the relation r:

$$[[u, v], r] = [u, [v, r]] - [v, [u, r]] \tag{3}$$

Here there are three alternatives:

   (a) Both the left- and right-hand sides of (3) are reduced to zero. In this case a new relation is not produced.

(b) Both sides of (3) 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 (3) recursively, the process of generating new relations is reduced to successive multiplication by generators.

(c) The right-hand side of (3) 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* (1–2) 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 and output data for the algorithm are:

Input. The ordered set of generators $X = \{x_1, x_2, \ldots\}$ with prescribed 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.

Output. The interreduced set of consequences of the input relations $\tilde{R} = \{\tilde{r}_1, \tilde{r}_2, \ldots\}$;
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;
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.

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, then the truncated output makes sense only if all $\tilde{r}_i \in \tilde{R}$ are homogeneous. In this case we obtain a part of the whole Gröbner basis, and the set E forms a subbasis of the Lie 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.

3 Implementation and Sample Session

The algorithm has been implemented in the C language. The source code has a total length of almost 7500 lines and contains about 150 C functions realizing: top level algorithms, Lie algebra operations, manipulation with scalar polynomials, multiprecision integer arithmetic, substitutions, list processing, input and output handling, etc.

The following session file has been produced on a 66 Mhz MS-DOS based AT/486 computer. We use here 32bit *GCC* compiler and *GO32* DOS extender, though for the considered example the 16bit *Borland C++ 3.1* environment is quite sufficient (and takes twice smaller space for the internal data structures). That illustrative example, arising in investigation of symmetries of well-known in mathematical physics Burgers equation leads to relatively compact output.

The relations contain three generators X, Y and T and five parameters $c_1, \ldots c_5$ which allow us to illustrate the classification aspects of the problem.

Note that the program asks for the output form of Lie monomials. In this example we choose the *right-normed* arrangement for non-associative monomials. It means, for instance, that Lie monomials $[x,[x,[y,x]]$ and $[[x,y],[x,[x,y]]]$ are presented in the output as x^2yx and $(xy)x^2y$, respectively. Such notations are more compact and expressive especially for high degree Lie monomials, and widely used by algebraists. Otherwise, one can choose the standard output with explicit square brackets printed out. Input data can be entered from either a keyboard or from a separate file.

```
Enter name of existing or new input file -> burggen.in
Input data:
Generators: X Y T;
Parameters: c_1 c_2 c_3 c_4 c_5;
Relations:
2 [[Y,X],Y] + c_2 [Y, X] + 2 c_5 Y;
```

```
[[Y,X],X] + c_1 [Y, X] + [T,Y] + c_4 Y;
[T,X] + c_3 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 -> 1
Weight for T -> 2

Enter limiting number for relations -> 100

Initial relations:

(1)  XT - c  Y = 0
           3

           2
(2)  YT - X Y + c  XY - c  Y = 0
                 1        4

(3)  2 YXY - c  XY + 2 c  Y = 0
              2         5

*** Possible parameter branching in relation:

             2         2        2
    4 c  c  X Y - (8 c   + c  c  ) XY + 2 c  c  c  Y
       5  2           5     4  2           5  4  2

    *** Parametric coefficient at leading term of relation:

    c  c
     5  2

*** Possible parameter branching in relation:

                      2                     2
    4 c  c  YT - (8 c   - 4 c  c  c  + c  c  ) XY - 2 c  c  c  Y
       5  2          5       5  2  1    4  2           5  4  2

    *** Parametric coefficient at leading term of relation:

    c  c
     5  2

*** Possible parameter branching in relation:

    c  XY - 2 c  Y
     2         5

    *** Parametric factor of relation:
```

$c_5 \, c_2$

*** Parametric coefficient at leading term of relation:

c_2

*** Possible parameter branching in relation:

$c_2^2 \, YT - (4 \, c_5^2 - 2 \, c_5 \, c_2 \, c_1 + c_4 \, c_2^2) \, Y$

*** Parametric factor of relation:

c_5

*** Parametric coefficient at leading term of relation:

c_2

Non-zero parametric coefficients:

(1) c_2

(2) c_5

Reduced relations:

(1) $c_2 \, XY - 2 \, c_5 \, Y = 0$

(2) $XT - c_3 \, Y = 0$

(3) $c_2^2 \, YT - (4 \, c_5^2 - 2 \, c_5 \, c_2 \, c_1 + c_4 \, c_2^2) \, Y = 0$

Basis elements:

(1) $E_1 = X$

(2) $E_2 = Y$

(3) $E_3 = T$

```
Non-zero commutators of basis elements:

(1)  [E , E ] = 2 c /c  E
       1   2       5  2  2

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

                      2                      2    2
(3)  [E , E ] = (4 c    - 2 c  c  c  + c  c  )/c   E
       2   3        5       5  2  1    4  2    2   2

Dimensions of homogeneous components:

dim G   = 2
     1

dim G   = 1
     2

Time: 0.01 sec
Number of relations:    12  Relation space:         96 bytes
Number of ordinals:     19  Ordinal space:         228 bytes
Number of nodes:       153  Node space:           1836 bytes
Total space: 2160 bytes
```

Here E_i are basis elements. In the case of an infinite-dimensional algebra the program prints out only those commutators which can be expressed in terms of the basis elements computed.

It can easily be seen that for the generic values of parameters we have a three-dimensional nilpotent Lie algebra. The branching of the algebra structure is possible at the values of parameters $c_2 = 0$ and $c_5 = 0$. The computations with these particular values show that the choice $(c_2 \neq 0, c_5 = 0)$ gives also a three-dimensional algebra, the choice $(c_2 = 0, c_5 \neq 0)$ gives zero-dimensional algebra, the choice $(c_2 = 0, c_5 = 0)$ leads to the most interesting case of the infinite-dimensional algebra indicating the complete integrability of the Burgers equation.

4 Serre Relations for Simple Lie Algebras

In Table 1 we present the results of application of the program to *Serre relations* (see, for example, Mikhalev and Zolotykh [12]) for all simple Lie algebras up to rank 10. The timings are presented for the above mentioned 66 Mhz MS-DOS based AT/486 computer.

Any (semi)simple complex Lie algebra L possesses a *Gauss decomposition* $L = E \oplus H \oplus F$, where H is a commutative *Cartan subalgebra*, and E and F are *positive* and *negative* nilpotent subalgebras, respectively. This decomposition is compatible with the following relations containing *Cartan elements* h_i and *Chevalley generators* e_i, f_i corresponding to positive and negative simple roots of the algebra:

$$[h_i, h_j] = 0, \tag{4}$$
$$[e_i, f_j] = \delta_{ij} h_j, \tag{5}$$
$$[h_i, e_j] = a_{ji} e_j, \tag{6}$$

$$[h_i, f_j] = -a_{ji} f_j, \tag{7}$$
$$(ad\, e_i)^{1-a_{ji}} e_j = 0, \tag{8}$$
$$(ad\, f_i)^{1-a_{ji}} f_j = 0, \tag{9}$$

where a_{ij} is the *Cartan matrix*, $i, j = 1, \ldots, r = rank\ L$. Note that for Kac–Moody algebras just the same relations hold with slightly different Cartan matrices.

Relations (8–9) include only Chevalley generators corresponding to positive and negative subalgebras E and F containing the principal part of the information about the algebra. These relations taken separately define subalgebras E and F.

One can see that calculation of the exceptional algebra E_8 is the most difficult task among those included in Table 1. The number of initial relations here is 290. The program generates the Gröbner basis which contains 23074 relations involving Lie monomials up to degree 58 while the Lie algebra basis elements go up to 29th degree. The task requires 3 min 14 sec of computing time and 815516 bytes of memory.

Unlike the whole set of Serre relations for E_8 the separate processing of relations (8) (or (9)) gives an example with twelve new relations arising at Step 2 of the algorithm in addition to 5508 relations derived at Step 1. Note that for E_8 one needs only 15 sec and 186096 bytes. Our computational experience shows that similar situations with extra relations generated at Step 2 are rather rare. On the other hand Step 2 takes usually much less computing time than Step 1. That is why in pursuit of efficiency we postpone the analysis of those special situations to Step 2.

The content of the columns in Table 1 is as follows:

Dim is the dimension of the algebra,
N_{in} is the number of input relations,
N_{GB} is the number of relations in the Gröbner basis,
N_{comm} is the number of nonzero commutators,
D_{GB} is the maximum degree of the Lie monomials in Gröbner basis,
Space is the maximum memory occupied by the computed structures,
Time is the running time without expenses for input-output operations.

5 Conclusion

Unlike commutative algebra, where such a universal tool for analysis of polynomial ideals as Buchberger's algorithm for computing the Gröbner basis has been developed [13, 14], its generalizations to non-commutative [6, 15, 16], and especially to non-associative algebras are still far from being practically useful. Moreover, because of very serious mathematical and algorithmic problems still to be solved, there are only a few packages implementing the non-commutative Gröbner basis technique, and none of them so far is able to deal with non-associative algebras.

This justifies the elaboration of other algorithmic techniques. Those described in earlier work [7–10] have been already applied to a number of problems in mathematical physics.

Table 1

Algebra	Dim	N_{in}	N_{GB}	N_{comm}	D_{GB}	Space, *bytes*	Time, *seconds*
A_2	8	17	24	21	4	1188	< 1
A_3	15	40	84	60	6	3612	< 1
A_4	24	72	218	126	8	8716	< 1
A_5	35	113	473	225	10	18088	< 1
A_6	48	163	908	363	12	33700	1
A_7	63	222	1594	546	14	57908	3
A_8	80	290	2614	780	16	93452	6
A_9	99	367	4063	1071	18	143456	10
A_{10}	120	453	6048	1425	20	211428	19
B_2	10	17	35	28	6	1672	< 1
B_3	21	40	149	106	10	6160	< 1
B_4	36	72	441	263	14	17148	< 1
B_5	55	113	1047	522	18	39180	2
B_6	78	163	2153	906	22	78544	5
B_7	105	222	3981	1441	26	142620	12
B_8	136	290	6792	2150	30	240024	26
B_9	171	367	10904	3057	34	381256	50
B_{10}	210	453	16683	4185	38	578364	98
C_3	21	40	138	106	10	5772	< 1
C_4	36	72	411	263	14	16032	< 1
C_5	55	113	968	522	18	36304	2
C_6	78	163	2007	906	22	73320	6
C_7	105	222	3756	1441	26	134652	14
C_8	136	290	6439	2150	30	227672	28
C_9	171	367	10398	3057	34	363720	54
C_{10}	210	453	15999	4185	38	554832	94
D_4	28	72	283	179	10	11336	< 1
D_5	45	113	726	389	14	27768	1
D_6	66	163	1573	713	18	58292	3
D_7	91	222	3034	1174	22	109964	8
D_8	120	290	5355	1798	26	190932	19
D_9	153	367	8817	2608	30	310452	37
D_{10}	190	453	13762	3628	34	479792	69
G_2	14	17	73	56	10	3200	< 1
F_4	52	72	858	544	22	32832	2
E_6	78	163	2186	1003	22	80740	5
E_7	133	222	6389	2527	34	230140	29
E_8	248	290	23074	7710	58	815516	194

The algorithm presented above reveals some common features with involutive techniques in commutative algebra [17]. Similar to the above consideration, in the involutive approach to Gröbner bases construction only multiplications by independent variables (prolongations) rather than S-polynomials are used. In the noncommutative case an analog of a Buchberger S-polynomial is a *composition* [6]. Thus, to construct a Gröbner basis one can perform all possible compositions and reduce them modulo the current relation set. However, in the nonassociative Lie algebra case such a computational scheme meets very serious combinatorial obstacles. To recognize reducibility of the particular composition one has to do recursive transformations based on Jacobi identities. Our present algorithm, like that in Gerdt and Blinkov [17], successively combines prolongations and reductions preventing massive recursive resimplifications.

Our C code can be easily generalized to handle Lie superalgebras.

Acknowledgments

We are grateful to A. Cohen, P. Gragert, V. Robuk, M.Roelofs and especially to V. Ufnarovsky for fruitful discussions and useful remarks. This work was supported in part by the INTAS project No. 93-893.

References

[1] Kac, V. G. (1990). *Infinite Dimensional Lie Algebras (3rd ed.).* Cambridge University Press.

[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. and Leites, D. (1995). Defining Relations Associated with the Principal $\mathbf{sl}(2)$-subalgebras.

[5] Leznov, A. and Saveliev, M. (1991) *Group-Theoretical Methods for Integration of Dynamical Systems*. Birkhäuser.

[6] 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).)

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

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

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

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

[11] Bahturin, Yu. A., Mikhalev, A. A., Petrogradsky, V. M. and Zaicev, M. V. (1992). *Infinite dimensional Lie superalgebras.* Walter de Gruyter.

[12] Mikhalev, A. A. and Zolotykh A. A. (1995). *Combinatorial Aspects of Lie Superalgebras*. CRC Press.

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

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

[15] Mora, T. (1988). Seven Variations on Standard Bases. Preprint No.45, Dip. di Mathematica, Universita di Genova, Italy.

[16] Kandri-Rody, A. and Weispfenning, V. (1990). Non-commutative Gröbner bases in Algebras of Solvable Type. *J. Symb. Comp.* **9**, 1–26.

[17] Gerdt, V. P. and Blinkov, Yu. A. (1995). Involutive Polynomial Bases, *Publication IT-95-271*, LIFL USTL, Lille. (Submitted to *J. Symb. Comp.*)