Acta Applicandae Mathematicae **16**: 231–242, 1989.
© 1989 *Kluwer Academic Publishers. Printed in the Netherlands.*

Lie Algebra Computations

P. K. H. GRAGERT
Department of Applied Mathematics, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

(Received: 26 July 1988)

Abstract. In the context of prolongation theory, introduced by Wahlquist and Estabrook, computations of a lot of Jacobi identities in (infinite-dimensional) Lie algebras are necessary. These computations can be done (automatically) using 'symbolic computations'. A package written in REDUCE is demonstrated to give an idea of the chosen approach.

AMS subject classifications (1980). 17B99, 35Q99, 58A15, 68C20.

Key words. Lie algebra computations, symbolic computations, REDUCE, prolongation theory.

1. Introduction

Lie algebras occur in different realizations, e.g., as matrices or as vector fields. What will be described here is a different realization, suitable for symbolic computations and, in a sense, very abstract. This is implemented in the symbolic computation language REDUCE and will be demonstrated on a very special subject, to be briefly recalled in the next section.

The text will include examples, imitating a computation session on a terminal, to give the flavour of using the described Lie algebra package. To distinguish input and output in the examples, input will be reproduced in italics, while the answer of the system will be typed in the standard font and as capitals.

2. Prolongation in a Nutshell

One possibility of analyzing nonlinear evolution equations in one space dimension is the prolongation method, introduced by Wahlquist and Estabrook [1, 2].

The method can be briefly recalled as follows: A system of nonlinear differential equations is formulated as a closed exterior differential system of two forms. The underlying manifold is extended (or prolonged) by new coordinates, the prolongation variables. The exterior differential system is prolonged by special 1-forms in the prolongation variables. The condition on these added 1-forms is that the prolonged exterior differential system remains closed. This condition results in an overdetermined system of differential equations which has to be solved. Often, several integration steps are directly possible. Due to the prolongation variables (together with some luck, depending on the evolution system tackled), there remain conditions on the introduced functions which are all of the commutator type.

This is the starting point of the work described in the next sections. Having solved all the equations, one gets a number of 1-forms closely related to the original evolution system. A further analysis of the 1-forms may give a nonlinear superposition formula of solutions or a Bäcklund transformation. For more information, the reader is referred to [1, 2].

3. Equations of Commutator Type

Here it is assumed that the prolongation method applied to an evolution system is already partially, solved which means that all the remaining differential equations to be solved are of the commutator type. To solve these equations, one may proceed as follows: All commutators are interpreted of being Lie products in some base variables, thus turning the differential equations of the commutator type into relations between Lie products. This means that one has to find and compute a Lie algebra, containing all the relations emerging out of the prolongation method. A representation of the Lie algebra (or of a homomorphic image of the sometimes infinite-dimensional Lie algebra) in vector fields, delivers a set of functions for solving the remaining set of differential equations of commutator type; (see e.g., [7]).

Summarizing, the main problem is the computation of a Lie algebra, containing the relations emerging out of the prolongation method.

4. Reason for Symbolic Computations

The point of view taken here is the following: The problem to be solved is the computation of a Lie algebra, given a certain number of relations between some of the Lie products and base variables of the Lie algebra. The only tool is the evaluation of all essential Jacobi identities. By essential Jacobi identity is meant, given a n-dimensional Lie algebra with basis $x_1, \dots, x_n$, that it suffices to check only Jacobi identities of x_i, x_j and x_k, where $0 < i < j < k \leq n$ holds. The number of essential Jacobi identities is $\binom{n}{3}$, a number which grows rapidly with increasing n. Computation of one Jacobi identity contains taking six Lie products, which may give a lot of (easy) algebraic manipulation.

Doing such computations by hand is, though easy, boring and error-prone. Therefore, a computer package in REDUCE has been written to do as many of the computations as possible automatically. It only remains to check the input data and (very often short) commands carefully.

5. Implementation of an Abstract Lie Algebra Package

Lie algebra computations in languages for symbolic computations are often available in the form of matrices and operations with matrices or in the form of vector fields for example, but for the purpose described in the foregoing section, a special way with special advantages is chosen. The realization of the Lie algebra is purely algebraic and the considerations are as follows:

(i) A linear space with its linear operations must be available, including input and output facilities.
(ii) The Lie product must be available as an operation, even if not all structure constants of the Lie algebra are known!

Consideration (ii) reveals the problem started from and justifies the adjective 'abstract'.

The following solution is chosen for consideration (i). Univariate polynomials are chosen as objects for a linear space. This choice has a number of advantages:

(1) In REDUCE polynomials are sparsely stored, which means that only nonzero coefficients are stored.
(2) Addition of polynomials is already available and is fast in execution time.
(3) Multiplication by constants is already available and is fast in execution time.
(4) Input as well output of polynomials is already given by REDUCE.
(5) The coefficients (coordinates) may be computed by the system function COEFF.

There is one small disadvantage: a linear polynomial is output without exponent one.

Consideration (ii) is solved by using the ALGEBRAIC OPERATOR concept of REDUCE and a new function, which actually computes a Lie product. That means an identifier, declared ALGEBRAIC OPERATOR, may have any number of parameters and is used to represent Lie products. Such an expression either stands for itself or may get explicitly a value.

```
EXAMPLE: ALGEBRAIC OPERATOR
*algebraic operator lie;
*lie(1, 2);
LIE(1, 2)

*lie(1,3): = x + a*x^3 + 4*lie(1,2) + b*lie(z);
LIE(1,2): = X + A*X^3 + 4*LIE(1,2) + B*LIE(Z)
```

Thus, one can work like this: choose a name for the Lie product and declare this name ALGEBRAIC OPERATOR, take LIE, for example. Then a Lie productable of, say, n basis elements is represented by LIE(i, j); LIE(i, j) representing the Lie product of the ith base element with the jth base element, $1 \leq i \leq n$ and $1 \leq j \leq n$. Naturally, the antisymmetry of the Lie product will be used, i.e., only Lie(i, j)'s will be used with $1 \leq i < j \leq n$.

This choice guarantees a unique representation of Lie products. By this convention, the REDUCE system will successfully simplify expressions containing abstract Lie products.

Furthermore, fixing the polynomial variable to be, for example, X, one can represent any Lie algebra element, also if not all structure constants are known, by a sum of a polynomial in X and a sum of terms, where each term contains exactly one factor with LIE(...) linearly!

A basis of the Lie algebra, say x_i, is isomorphically mapped onto a polynomial by x_i ⟨--⟩ X^i; $i = 1, \ldots, n$.

The last step is writing a function which multiplies two Lie algebra elements of any form using the more-or-less 'filled' productable LIE(i, j). In the package described, this Lie product function is called BLHK (bilinear haak, meaning bilinear bracket (Dutch: haak, English: bracket)). To get unique representations of products of products, the necessary care is taken in writing BLHK and some other underlying functions to get unique representations of the Lie algebra elements, helping to simplify expressions as much as possible!

Besides BLHK, a function is added to compute the essential Jacobi identities. This function is called TESTJAC, which will be shortened to TJ by a DEFINE-statement: DEFINE TJ = TESTJAC; The package must be initialized to a given dimension, because ARRAY's are used to compute coefficients of a polynomial, by calling the function INIBLHK appropriate.

Before going on, the reader is kindly requested to follow the REDUCE session, demonstrating the features of the abstract Lie algebra package. Please pay attention to how the default values (LIE, X) of the Lie productable and the polynomial variable are changed, demonstrating the flexibility of the package, and to the functions PRINTTAFEL, printing the productable, and LIE!@REVAL, which reorders arguments of an abstract product, according to the adopted order preference. An example of basic Lie algebra computations is as follows:

```
*fload blhk,opcoef;                    comment loading of the package;
EXECUTE: INIBLKH(MAXIMAL DIMENSION); *iniblhk 4;
D!@LIE = 4
THE ARRAYS: A!@K1, A!@K2 and !@JANAX ARE INITIALIZED
INIBLHK END

*off msg; define tj = testjac;
*blhk(x,x^3);
LIE(1,3)
```

```
*blhk(x^3,x);
-LIE(1,3)

*blhk(a*lie(1,3) + x,x^2 + lie(d));
LIE(1,2) - LIE(LIE(D),1) + LIE(LIE(1,3),2)*A +
LIE(LIE(1,3),LIE(D))*A

*lie(1,4) : = b*x^2;
LIE(1,4) := x2*B

*tj(2,3,4):
LIE(LIE(3,4),2) - LIE(LIE(2,4),3) + LIE(LIE(2,3),4)

*tj(1,3,4);
-LIE(2,3)*B + LIE(LIE(3,4),1) + LIE(LIE(1,3),4)

*lie(1,3) := x^4;
LIE(1,3) := X4

*tj(1,3,4);
-LIE(2,3)*B + LIE(LIE(3,4),1)

*lie(1,2) := 0,
LIE(1,2) := 0

*lie(3,4) := x^3;
LIE(3,4) := x3

*tj(1,3,4);
-LIE(2,3)*B - X4

*lie(2,3) := -x^4/b;
LIE(2,3) := (-x4)/B

*tj(1,2,3);
LIE(2,4) + X2

*lie(2,4) := -x^2;
LIE(2,4) := -X2

*for i := 1:2 do for J := i + 1:3 do for k := j + 1:4 do
write i, " ",j," ",k,"  ".tj(i,j,k);
1  2  3    0
1  2  4    0
1  3  4    0
2  3  4    0
*comment all 4 Jacobi identities are zero!;

*printtafel 4$
LIE(1,2) := 0;
LIE(1,2) := X4
LIE(1,4):=X^2*B;
LIE(2,3):=(-X^4)/B;
LIE(2,4):= -X^2;
LIE(3,4):=X^3;

*comment Lie algebra; if B is not zero! Implicitly assumed!;
*lisp !@lie:='newlie;
NEWLIE

*blhk(x^5,x^2);
DECLARE NEWLIE OPERATOR? (Y/N)
?y
-NEWLIE(2,4)

*lisp !@lievar:='y;
Y

*blhk(x,x+x^2);
***** COM!@LIE: X IS NOT AN ALGEBRA ELEMENT
*comment error message, if one tries to compute Lie products with expres-
sions not to be considered lie algebra elements;

*blhk(y,y+y^2);
NEWLIE(1,2)

*!@liereval newlie (1,newlie(1,2)) ;
-NEWLIE(NEWLIE(1,2),1)
```

6. Automatic Evaluation of Jacobi Identities

Before programming some useful functions for helping, finding, and constructing a Lie algebra, which contains the Lie relations resulting from the prolongation approach, the Lie relations will be analyzed in theory and will give all available computational possibilities.

First, some notations are introduced: Let L be an n-dimensional Lie algebra over a field K. Let x_i, $i=1,\ldots,n$, be a basis of L. The Lie product $[x_i, x_j]$ will be written as LIE(i, j), to relate this analysis to the already-introduced REDUCE package for Lie algebras. The Jacobi identity

$$[[x_i, x_j], x_k]+[[x_j, x_k], x_i i]+[[x_k, x_i i], x_j]=0$$

will be written as $J(i, j, k)=0$, or, equivalently as

$$\text{LIE(LIE}(i, j), k)+\text{LIE(LIE}(j, k), i)-\text{LIE(LIE}(i, k), j),$$

omitting '=0'. This is the notation used in REDUCE. The following index sets are used:

$I = \{1, \ldots, n\}$,
$\mathrm{EP} = \{p = (i, j) \mid i, j \in I, i < j\}$ essential index pairs
$\mathrm{ET} = \{w = (i, j, k) \mid i, j, k \in I, i < j < k\}$ essential index triples.

By this notation, J may be regarded as a function on ET into L, LIE(p), $p \in \mathrm{EP}$, may be regarded as the productable of an anticommuting algebra, which will be a Lie algebra, if $J(w) = 0$ for all $w \in \mathrm{ET}$.

Any $J(w) = J(i, j, k)$, $w \in \mathrm{ET}$, will be called an essential Jacobi identity. The Lie relations from the prolongation approach, in general, look like

$$R_j : \sum d_i^j X^i + \sum \beta_p^j \mathrm{LIE}(p) = 0.$$

After (algebraically) solving each relation R_j for a (not yet solved) LIE(p_i), the result is, so to speak, a Lie algebra productable, which is incomplete: Some of the products $[x_i, x_j] = \mathrm{LIE}(i, j)$ may not be known, some may be known, and all others are partly known, because they are related to not-yet-known essential Lie products. (*Reminder*: The products LIE(p), $p \in \mathrm{EP}$, are called essential.) Thus, the problem to be solved may be stated as follows: given an incomplete productable of an anticommuting algebra, fill in all unknown (essential) products, such that the result becomes a Lie algebra. Solving this problem now means: check all $J(w)$, $w \in \mathrm{ET}$, whether the values are the zero-element of the algebra or not. If any $J(w)$ is not (yet) zero, one gets a condition on the (yet) unknown products LIE(p).

Now the different possibilities of a $J(w)$ have to be discussed, giving all computational possibilities for solving the above-stated problem.

The most general form of a $J(w)$ looks like

$$J(w) = \sum_{i \in IX(w)} d_i X^i + \sum_{p \in IP(w)} \beta_p \mathrm{LIE}(p) + \sum_{qk \in IT(w)} \gamma_{qk} \mathrm{LIE}(\mathrm{LIE}(q), k)) = 0,$$

where the following sets are introduced

$$\left.\begin{array}{l} \mathrm{IX}(w) \subset I \\ \mathrm{IP}(w) \subset \mathrm{EP} \\ \mathrm{IT}(w) = \{pk \mid p \in \mathrm{EP}, k \in I\} \end{array}\right\} \quad \text{for any } w \in \mathrm{ET}.$$

These index sets will change while solving the problem.

If $J(w) = 0$ holds, the three index sets IX(w), IP(w) and IT(w) are empty sets, the goal of the problem!

At an intermediate stage of the solving process for a fixed $w \in \mathrm{ET}$, four different cases can be recognized.

(*Case* 1) $\mathrm{IX}(w) = \mathrm{IP}(w) = \mathrm{IT}(w) = \emptyset$. This means, that the Jacobi identity already holds for x_i, x_j, x_k; ($(i, j, k) = w$), and nothing has to be done.

(*Case* 2) $\mathrm{IX}(w) \neq \emptyset$, $\mathrm{IP}(w) = \mathrm{IT}(w) = \emptyset$. This means, that there exists a linear relation between the base elements x_i. This is a contradiction and may only be solved by reducing the dimension of the Lie algebra L by at least one.

(*Case* 3) $\mathrm{IP}(w) \neq \emptyset$, $\mathrm{IT}(w) = \emptyset$. This means, there exists a linear relation between (yet) unknown Lie products, therefore solve $J(w) = 0$ algebraically for any LIE(p), $p \in \mathrm{IP}(w)$.

(*Case* $\mathrm{IT}(w) \neq \emptyset$. This means, that $J(w)$ contains a term involving a Lie product with one factor a (not yet) known Lie product. Such a Lie product will be called nested Lie product. This case needs special attention.

(Case 1) and (Case 3) are suitable for automatic calculations and indeed a function LOSOP (solve in dutch) is written to carry out these calculations. Some intelligence is built into the constituent functions of LOSOP, namely to check only the $J(w)$'s which have not been checked earlier in the computational process and which will not belong to (Case 4). If (Case 2) is encountered, the automatic process is stopped, because an assumption is violated. To adjust the state of the system in (Case 2) is sometimes easy and sometimes difficult. The information, the linear dependency, is in itself valuable information. One should feed in this information at an appropriate (earlier) point into the computations and resume the solving process.

If LOSOP reaches its normal end, either a Lie algebra is found or the productable is still incomplete which means all not (yet) checked Jacobi identities $J(w)$ are of (Case 4).

In the latter case, one may check the consequences of three different approaches to break the tie: (a) artificially introduce a linear dependence; (b) artificially introduce a value for an unknown Lie product – zero or a linear expression of the basis with or without parameters (to be determined). (c) artificially introduce a new basis variable for an unknown Lie product extending the dimension of the Lie algebra by 1.

All three methods (a), (b) and (c) have been used successsfully on different problems, although it must be said, that up to now, there is no theory which guarantees a finite-dimensional nontrivial solution.

Example using LOSOP for 2 given Lie relations:

```
LIE(1,2) - LIE(2,3) = 0
LIE(1,3) - X^2      = 0
```

Compute a Lie algebra, containing these relations. There are infinite many solutions, here the most general 4 dimensional Lie algebra will be computed.

```
*fload blhk, opcoef, jana, losop;     comment load the package!;
EXECUTE: INIBLKH(MAXIMAL DIMENSION);
TO TAKE OWN DECISIONS IN JACOBIANA, EXECUTE:
!@JANADECIDE := 2;
```

```
*iniblhk 6;
D!@LIE = 6
THE ARRAYS: A!@K1, A!@K2 AND !@JANAX ARE INITIALIZED
INIBLHK END

*lie(1,2):= lie(2,3)$         comment $ suppresses output!;

*lie(1,3):= x^2S
*define tj = testjac;

*tj(1,2,3);
LIE(LIE(2,3),3) + LIE(LIE(2,3),1)

*comment (case 4), e.g. introduce a new variable!;
*lie(2,3):= x^4
*losop;
1,PART
1  2  3                  comment essential triple;
- LIE(3,4) - LIE(1,4)    comment J(1,2,3);
SUCCESS WITH LIE(1,4)    comment j(1,2,3) = 0 solved for LIE(1,4)
LOSOP END

*comment all other J(w) are of (case 4) type, therefore look for them by
hand;
*tj(1,2,4);
LIE(LIE(3,4),2 + LIE(LIE(2,4),1)   comment two different nested Lie
                                   products, nasty, look for another;

*tj(1,3,4);
LIe(2,4) + LIE(LIE(3,4),3) + LIE(3,4),1)

*comment (case 4) easy one, introduce a linear combination for lie(3,4),
using the helpfunction EL;

*Lie(3,4):= el(4,a);

LIE(3,4):= X^4*A4 + X^3*A3 + X^2*A2 + X*A1

*losop;                      comment automatic check of J(w)'s;
1. PART
1  3  4                      comment essential triple;
LIE(2,4) + X^2*(A1 - A3)     comment J(1,3,4);

SUCCESS WITH LIE(2,4)        comment relation solved for
LIE(2,4);
```

```
2. PART                          comment second automatic loop of LOSOP;
1  2  4                                 comment essential triple;
2*X^4*(A1 - A3) + X^2*A4*(A1 - A3)      comment J(1,2,4);
A1*A4   :=   A3*A4                      comment condition on paramaters
A1   :=   A3                            comment condition on parameters;
2  3  4                                 comment essential triple
0
LOSOP END                               comment J(2,3,4) = 0 holds already;
```

**comment condition a1*a4 = a3*a4 became superfluous;*

```
*printtafel 4;                                comment show the solution!;
LIE(1,2) := X^4;
LIE(1,3) := X^2;
LIE(1,4) := -X^4*A4 - X^3*A3 - X^2*A2 - X*A3;
LIE(2,3) := X^4;
LIE(2,4) := 0;
LIE(3,4) := X^4*A^4 + X^3*A3 + A^2*A2 + X*A3;
```

**comment a three parameter solution, the most general 4 dimensional one!;*

7. Application of the Lie Algebra Package to the Prolongation Approach of the Korteweg–de Vries Differential Equation

The prolongation of the Korteweg–de Vries differential equation (KdV) is discussed in the literature [1, 4, 7], and will not be repeated here. The success of van Eck in explicitly describing the infinite-dimensional Lie algebra of the KdV, was dependent on the results computed by the package described in the foregoing sections. The idea here is to summarize the computational process to find the crucial value sof $[x_4, x_8]$, necessary in the proof of van Eck [4]. The Lie relations for the KdV of the prolongation approach are the following:

$$\mathrm{LIE}(1,3) = \mathrm{LIE}(2,3) = \mathrm{LIE}(1,4) = \mathrm{LIE}(2,6) = 0$$
$$\mathrm{LIE}(1,2) + x^7 = 0;\ \mathrm{LIE}(1,7) - x^5 = 0;\ \mathrm{LIE}(2,7) - x^6 = 0;$$
$$\mathrm{LIE}(3,4) + x^8 = 0;\ \mathrm{LIE}(2,4) + x^9 = 0;\ \mathrm{LIE}(1,5) - x^9 = 0;$$
$$\mathrm{LIE}(1,6) + x^7 - x^8 = 0$$

These relations are the input to the package. Calling LOSOP gives:

```
              w                  J(w)
1. part  1  2  7          LIE(2,5) - x8 + x7
         1  2  6          LIE(6,7) + LIE(2,8) - x6
         1  2  3          LIE(3,7)
         1  3  4          LIE(1,8)
         1  2  4          LIE(4,7) + LIE(1,9)
         2  3  4          -LIE(6,7) - LIE(3,9) + x6
```

```
    2. part 1  3  7        LIE(3,5)
            1  2  5        LIE(5,7) + LIE(2,9) + x5
            2  3  7        LIE(3,6)
    3. part 1  3  6        LIE(3,8)
            1  3  5        -LIE(6,7) + x6
            2  3  6        0
            2  3  5        0
    4. part 1  6  7        -LIE(7,8) - LIE(5,6) - x8 + x7
            1  3  8        0
            1  2  8        LIE(5,6) + x8 - x7
            2  6  7        0
            2  3  8        0
            3  6  7        0
    5. part 1  7  8        LIE(5,8)
            1  5  6        -LIE(6,9) - LIE(5,7) - x5
            2  7  8        LIE(6,8)
            2  5  6        0
            3  7  8        0
            3  5  6        0
    6. part 1  6  8        0
            1  5  8        -LIE(8,9)
            2  6  8        0
            2  5  8        0
            3  6  8        0
            3  5  8        0
            5  6  8        0
            6  7  8        0
    7. part 3  8  9        0
                           LOSOP END
```

Thirty-four Jacobi identities were checked and evaluated automatically! Now extending the dimension up to 18, more than 100 Jacobi identities have to be evaluated to give an inconsistency, which gives here the valuable information, after introducing x^9 and x^{10} for LIE(1,5), respectively LIE(5,7), that LIE(5,10) must be $-x^9$. Hereafter, expanding the dimension again up to 12, delivers the crucial relation LIE(4,8) = 0. For this step again more than 50 Jacobi identities were evaluated automatically. These computational steps are omitted, because the way of working should already be understood.

8. Some Concluding Remarks

The Lie algebra package is written in REDUCE 2, although REDUCE 3.3 is now sold. Because there are some essential differences between these releases, all the written functions of the package have to be adjusted; this work will be finished soon.

Other functions have also been developed, e.g., the computation of the Killing form of a Lie algebra, the computation of the solvable radical of a Lie algebra, and a base transformation of a Lie algebra, even if not all the structure constants are known, and it is interesting in the discussed field of application. To implement such functions means more or less using linear algebra but, nevertheless, it is worthwhile using these tools in the appropriate circumstances.

References

1. Wahlquist, H. D. and Estabroook, F. B.: Prolongation structures of nonlinear evolution equations I, *J. Math. Phys.* **16** (1975), 1-7.
2. Estabroook, F. B. and Wahlquist, H. D.: Prolongation structures of nonlinear evolution equations II, *J. Math. Phys.* **18** (1976), 1293–1297.
3. van Eck, H. N.: Lie groups in prolongation theory, PhD thesis, University of Twente, The Netherlands.
4. van Eck, H. N.: The explicit form of the lie algebra of Wahlquist and Estabrook, a presentation problem, *Proc. Kon. Ned. Akad. Wetensch., Ser. A* **86** (1983), 165–172.
5. van Eck, H. N., Gragert, P. K. H., and Martini, R.: The explicit structure of the nonlinear Schrödinger prolongation algebra.
6. Shadwick, W. F.: The KdV Prolongation Algebra. *J. Math. Phys.* **21** (1980).
7. Gragert, P. K. H.: Symbolic computations in prolongation theory, PhD thesis, University of Twente, The Netherlands, 1981.