Georg-August-Universität Göttingen

Computing the Baker-Campbell-Haussdorff series and the Zassenhaus product

M. Weyrauch, D. Scholz

Nr. 2008-16

Preprint-Serie des
Instituts für Numerische und Angewandte Mathematik
Lotzestr. 16-18
D - 37083 Göttingen

Computing the Baker-Campbell-Haussdorff Series and the Zassenhaus Product

Michael Weyrauch

Physikalisch-Technische Bundesanstalt,

Bundesallee 100, D-38116 Braunschweig, Germany.

Daniel Scholz

Universität Göttingen, Institut für Numerische und Angewandte Mathematik,

Lotzestraße 16-18, D-37083 Göttingen, Germany.

(Dated: September 30, 2008)

Abstract

The Baker-Campbell-Hausdorff (BCH) series and the Zassenhaus product are of fundamental importance for the theory of Lie groups and their applications in physics. In this paper, various methods for the computation of the terms in these expansions are compared, and a new efficient approach for the calculation of the Zassenhaus terms is introduced. *Mathematica* implementations for the most efficient algorithms are provided together with comparisons of execution times.

Furthermore, we study two maps which translate the polynomial representation of the BCH and Zassenhaus terms into representations in terms of nested commutators. The first of these maps yields the well known Dynkin-Specht-Wever representation of the BCH terms while the second one generates a commutator representation which involves fewer terms than the Dynkin-Specht-Wever representation.

I. INTRODUCTION

The product of the exponentials of two non-commutative variables x and y may be expressed as the exponential of an infinite sum

$$\mathrm{e}^{x}\mathrm{e}^{y} \;=\; \mathrm{e}^{x+y+\sum_{n=2}^{\infty} z_n}. \tag{1}$$

This formula is easily obtained from a formal power series expansion of the exponential. The famous Baker-Campbell-Hausdorff (BCH) theorem [1–3] asserts that the BCH terms z_n may be expressed as linear combinations of nested commutators. This is of fundamental importance for the theory of Lie groups and their applications in physics.

Similarly, the exponential of the sum of two non-commutative variables a and b may be written as an infinite product

$$\mathrm{e}^{a+b} \;=\; \mathrm{e}^{a}\cdot\mathrm{e}^{b}\cdot\prod_{n=2}^{\infty}\mathrm{e}^{c_n}, \tag{2}$$

which is known as the Zassenhaus product. The Zassenhaus exponents c_n may be also expressed as linear combinations of nested commutators as was first shown by Magnus [4] based on unpublished work by Zassenhaus.

The BCH and Zassenhaus theorems proof the existence of a commutator representation, but do not provide a simple way of constructing such a representation. The present paper focuses on the question of how to compute the BCH and Zassenhaus terms efficiently, and then express them in terms of nested commutators. *Mathematica* [5] implementations are provided.

There are several works which addressed these questions before: Methods for the BCH terms based on the formal power series expansions of the exponential function and the logarithm were discussed in Refs. [6–9]. A quite different approach for the calculation of the the coefficients for single monomials in each z_n was presented by Goldberg in Ref. [10]. Using a matrix representation, another approach was developed in Ref. [11].

For the calculation of the Zassenhaus terms a procedure based on the formal power series of the exponential function was given in Ref. [4]. A similar method was discussed by Wilcox [12], and was recently applied in Ref. [13]. An approach based on a matrix representation was suggested by the present authors [14].

Most methods cited above for the calculation of the BCH and Zassenhaus terms initially yield a polynomial representation, and not the desired representation in terms of nested commutators. In a number of independent papers, Dynkin [6], Specht [15], and Wever [16] provided a simple explicit construction of such a commutator representation. This construction is explicitly given by the map Θ defined in Eq. (13) of Section V. However, due to the Jacobi identity $[[x,y],z]+[[y,z],x]+[[z,x],y]=0$ and similar identities for higher order commutators, the commutator representation is not unique. In fact, Oteo [17] formulated a conjecture concerning a commutator representation of the BCH terms, which entails fewer terms than the Dynkin-Specht-Wever representation. This new commutator representation is obtained from the map Φ defined in Eq. (14) in Section V. It was conjectured recently that this different commutator representation also applies to the the Zassenhaus exponents [14]. In the present paper, it is proved that Oteo's map Φ generates indeed a valid nested commutator representation for the BCH and Zassenhaus terms (and more general polynomials as discussed in Section V and the Appendix).

The remainder of the paper is organized as follows: In Sections II and III, various methods for the calculation of the BCH and Zassenhaus terms are discussed. For two algorithms, *Mathematica* implementations are presented. In Section IV we provide comparisons of running times for some of the methods. In the last section, the Dynkin-Specht-Wever theorem is reviewed, and Oteo's map for the translation of a polynomial representation into a nested commutator is presented. The proof that this map indeed yields a valid representation in terms of nested commutators is given in the Appendix. To the best of our knowledge this proof has never been published before. The paper concludes with a brief discussion.

II. METHODS FOR COMPUTING THE BAKER-CAMPBELL-HAUSDORFF SERIES

The exponential function and the logarithm are defined by their formal power series

$$\exp(x) \;=\; \sum_{k=0}^{\infty} \frac{1}{k!} x^k \qquad \text{and} \qquad \log(x) \;=\; \sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{k} (x-1)^k. \tag{3}$$

All infinite series here and in the next section should be understood as formal. The region of convergence has to be studied for each specific choice of variables x and y separately.

Using these power series, we obtain for two non-commutative variables x and y

$$\begin{aligned}\log\left(\mathrm{e}^{x}\mathrm{e}^{y}\right) &= \log\left(\sum_{r=0}^{\infty}\frac{1}{r!}x^{r}\cdot\sum_{s=0}^{\infty}\frac{1}{s!}x^{s}\right)\\ &= \log\left(\left(1+x+\frac{1}{2}x^{2}+\ldots\right)\cdot\left(1+y+\frac{1}{2}y^{2}+\ldots\right)\right)\\ &= \sum_{k=1}^{\infty}\frac{(-1)^{k-1}}{k}\left(x+y+xy+\frac{1}{2}xx+\frac{1}{2}yy+\ldots\right)^{k}. \end{aligned} \tag{4}$$

From this expansion one obtains the polynomials $z_n = z_n(x,y)$ by collecting all summands in Eq. (4) with n factors of x or y, or, expressed differently, by collecting all terms which contain words of length n.

In the literature, several different approaches concerning the question how to calculate the BCH terms can be found. The most important methods will be discussed in the following.

A. Dynkin's method

Dynkin's method or the 'expansion method' (as we like to call it) is based on a direct application of the formal power series expansions of the exponential and logarithm given above. The method was discussed in a seminal paper by Dynkin [6]. From Eq. (4) one obtains

$$z_n = \sum_{k=1}^{n}\left(\sum_{p_1,q_1,\ldots,p_k,q_k}\frac{(-1)^{k-1}}{k}\cdot\frac{1}{\prod_{i=1}^{k}p_i!q_i!}\cdot x^{p_1}y^{q_1}\ldots x^{p_k}y^{q_k}\right) \tag{5}$$

where the second sum is over all possible combinations of $p_1, q_1, \ldots, p_k, q_k \in \mathbb{N}\cup\{0\}$ such that $p_i + q_i > 0$ for $i = 1, \ldots, k$, $\sum_{i=1}^{k}(p_i+q_i) = n$, and $0! = 1$.

The disadvantage of this method is the huge number of possible combinations of the p_i and q_i, e.g. for $n = 8$ we already find $11,144$ allowed combinations. The method yields a polynomial representation of the BCH series which can be translated into a sum of commutators using the maps Θ or Φ discussed in Section V.

Alternatively, using the map Φ in Eq. (5) before performing the sums yields a commutator representation of z_n directly. In fact, such a procedure reduces the number of of possible combinations to be considered in the summations, since we only have to take into account combinations beginning with $(p_1, q_1) = (1, t)$ or with $(p_1, q_1, p_2, q_2) = (1, 0, 0, t)$ and $t \geq 1$. Richtmyer and

Greenspan [8] as well as the textbook by Varadarajan [9] provide an independent proof of the BCH theorem, which lead to a procedure for the calculation of the BCH terms similar to the one just described.

Bose [7] introduced a simple algorithm to calculate the coefficients for single monomials. The idea of his algorithm is to collect all possible combinations of the $(p_1, q_1, \ldots, p_k, q_k)$ which are associated with the same monomial, e.g. $(1, 2)$ and $(1, 1, 0, 1)$ (as well as other combinations) for the monomial xyy.

B. Goldberg's method

An efficient algorithm for the calculation of the BCH terms z_n was published in 1956 by Goldberg [10]. It is based on a generating function for the numerical coefficients of the monomials in each BCH term. For $s \in \mathbb{N}$, one recursively defines the functions $G_1(t) = 1$ and

$$G_s(t) \;=\; \frac{1}{s}\frac{\mathrm{d}}{\mathrm{d}t}\Big(t(t-1)G_{s-1}(t)\Big) \quad \text{for} \quad n > 1.$$

Furthermore, let W_x be a monomial in x and y starting with an x such that $W_x = x^{s_1}y^{s_2}\ldots y^{s_m}$ for m even and $W_x = x^{s_1}y^{s_2}\ldots x^{s_m}$ for m odd with $s_1, \ldots, s_m \in \mathbb{N}$. According to Goldberg [10] the coefficient of such a monomial in a BCH term c_n is given by

$$c_{W_x} \;=\; \int_0^1 t^{m'}(t-1)^{m''}G_{s_1}(t)\ldots G_{s_m}(t)\,\mathrm{d}t \tag{6}$$

with $m' = \lfloor m/2 \rfloor$ and $m'' = \lfloor (m-1)/2 \rfloor$. Here, $\lfloor \cdot \rfloor$ is the floor function. Furthermore, one can show that $g_{W_y} = (-1)^{n-1}g_{W_x}$. A proof of Goldberg's method may be found in the textbook by Reutenauer [18]. Using Goldberg's method, numerical values for the z_n up to $n = 20$ have been listed by Newman and Thompson[19], but they did not provide their code.

In principle we have to calculate 2^n coefficients for the 2^n possible monomials in each z_n. But there is an obvious simplification due to Eq. (6): Each permutation of the $\{s_1, \ldots, s_m\}$ yields an identical coefficient. Therefore, we are allowed to assume that $s_1 \leq \ldots \leq s_m$.

A *Mathematica* (Version 6) implementation of Goldberg's method is given by the following code:

```
g[1]=1;
```

```
g[s_]:=g[s]=Expand[1/s*D[t*(t-1)*g[s-1], t]];
c[w_]:=c[w]=Module[{m, m1, m2, k},
   m = Length[w]; m1 = Floor[m/2]; m2 = Floor[(m-1)/2];
   Integrate[t^m1*(t-1)^m2*Product[g[w[[k]]], {k, m}], {t, 0, 1}] ];

BCH[n_Integer, alph_List] := Module[{p},
   p = Flatten[Permutations/@IntegerPartitions[n], 1];
   Plus@@(c[Sort[#]]*(words[#, alph]-
          (-1)^n*words[#, Reverse[alph]])&/@p)//Expand];
```

The BCH term z_n is returned by `BCH[n,alph]` in terms of the two-letter alphabet `alph`, which is entered as a list of any two letters, e.g. `{"x", "y"}`. The simplification resulting from the ordering property of the s_i discussed above is implemented using `Sort[#]` when calling the coefficient function `c`. In our code, the W_x are represented by the list $\{s_1, s_2, \ldots, s_m\}$ as defined above. The list `p` contains all possible W_x for a given n. A *Mathematica 6* code providing a translation between this representation and the corresponding words written in terms of alphabet `alph` is given by the following code:

```
words[p_List,alph_List]:=StringJoin@(ConstantArray @@@
  Partition[Riffle[p, alph, {1, 2*Length[p], 2}], 2])
```

Note that an implementation which generates the elements of p recursively together with their translation into words is more efficient than the code given above. For very high orders of n such an implementation may be preferable, if the translation into words is really needed.

C. Reinsch's method

Reinsch's method [11] is based on a matrix representation for the matrices e^x and e^y: Let $\sigma_1, \ldots, \sigma_n$ be arbitrary commutative variables and let $F = (F_{ij})$ and $G = (G_{ij})$ be two $(n+1) \times (n+1)$ matrices defined by $F_{ij} = 0$ and $G_{ij} = 0$ for $i > j$ and

$$F_{ij} = \frac{1}{(j-i)!} \quad \text{and} \quad G_{ij} = \frac{1}{(j-i)!} \cdot \prod_{k=i}^{j-1} \sigma_k$$

for $i \leq j$. Furthermore, let T be an operator translating a word $W = \sigma_1^{\mu_1}\sigma_2^{\mu_2}\sigma_3^{\mu_3}\ldots\sigma_n^{\mu_n}$ with $\mu_i \in \{0, 1\}$ for $i = 1, \ldots, n$ into a product of x and y in the following way: If $\mu_i = 0$, replace $\sigma_i^{\mu_i}$

by x, if $\mu_i = 1$, replace $\sigma_i^{\mu_i}$ by y, e.g. $T(\sigma_2\sigma_3\sigma_5) = xyyxy$ for $n = 6$. Then we have

$$z_n \;=\; T\left((\log FG)_{1,n+1}\right) \;=\; T\left(\left(-\sum_{k=1}^{n}\frac{(-1)^k}{k}(FG-I)^k\right)_{1,n+1}\right), \tag{7}$$

where I is the $(n+1)\times(n+1)$ identity matrix and the index $1, n+1$ indicates the upper-right element of the matrix $\log FG$. Note that $\log FG$ can be calculated in finitely many steps because $(FG-I)^m$ is the zero-matrix for $m > n$.

The implementation of this method (as given in Ref. [11]) is quite simple, but we find Goldberg's method superior both in speed and simplicity of implementation (see Section IV). Moreover, one can apply the maps Θ or Φ discussed in Section V only after the whole polynomial z_n has been calculated.

III. METHODS FOR COMPUTING THE ZASSENHAUS PRODUCT

Again, using the formal power series expansion of the exponential function, the Zassenhaus product for two non-commutative variables a and b may be written as

$$\begin{aligned}
\mathrm{e}^{a+b} &= \sum_{k=0}^{\infty}\frac{1}{k!}(a+b)^k \\
&= 1 + a + b + \frac{1}{2}a^2 + \frac{1}{2}ab + \frac{1}{2}ba + \frac{1}{2}b^2 + \ldots && (8)\\
&= \left(1 + a + \frac{a^2}{2} + \ldots\right)\cdot\left(1 + b + \frac{b^2}{2} + \ldots\right)\cdot\prod_{k=2}^{\infty}\left(1 + c_k + \frac{c_k^2}{2} + \ldots\right) && (9)\\
&= \mathrm{e}^{a}\cdot\mathrm{e}^{b}\cdot\mathrm{e}^{c_2}\cdot\mathrm{e}^{c_3}\cdot\mathrm{e}^{c_4}\cdot\ldots\,. && (10)
\end{aligned}$$

Our aim is to compute the polynomials $c_n = c_n(a,b)$ which consist of all summands in Eq. (10) with n factors of a or b, i.e. words of length n.

Methods for calculating the Zassenhaus terms c_n using the power series expansion were discussed by Magnus and Wilcox in Refs. [4, 12]. Recently, Quesne [13] used these methods for calculating the Zassenhaus terms up to c_6.

In the following we present a simple and efficient algorithm related to Wilcox´ method (expansion method). Furthermore, we apply an approach similar to Reinsch's method. This approach was proposed by the present authors in a previous paper [14]. In contrast to the methods for calculating

the BCH terms z_n, the approaches for the Zassenhaus terms are recursive, i.e. one needs to know c_2 to c_{n-1} for the calculation of c_n.

A. Expansion method

Comparing Eqs. (8) and (9) one obtains

$$c_n \;=\; \sum_{t_1,u_1,\ldots,t_n,v_n} \frac{1}{n!}\cdot a^{t_1}b^{u_1}\ldots a^{t_n}b^{u_n} \;-\; \sum_{v_0,\ldots,v_{n-1}} \frac{1}{v_0!v_1!\ldots v_{n-1}!}\cdot a^{v_0}b^{v_1}c_2^{v_2}\ldots c_{n-1}^{v_{n-1}} \qquad (11)$$

where the first sum is over all $t_1, u_1, \ldots, t_n, v_n \in \{0,1\}$ with $t_m + u_m = 1$ for $m = 1, \ldots, n$ and the second sum over all $v_0, \ldots, v_{n-1} \in \mathbb{N} \cup \{0\}$ with

$$v_0 + v_1 + 2\cdot v_2 + 3\cdot v_3 + \ldots + (n-1)\cdot v_{n-1} \;=\; n.$$

The main problem is to compute all allowed combinations for the v_i. As it turns out, the number of allowed combinations is quite small, e.g. for $n = 8$ there are only 66 possible combinations compared to $11,144$ in Dynkin's method for z_8. Although we have to calculate the c_n recursively, the method can be implemented quite efficiently (see Section IV).

Wilcox [12] used the same simple basic idea for the determination of the Zassenhaus terms, namely to collect all summands in Eq. (10) with n factors of a or b. However, in contrast to Wilcox' approach, our method reduces the calculation of the Zassenhaus terms to a simple and well defined combinatorial problem, i.e. the determination of all terms contributing to the finite sums in Eq. (10). A nested commutator representation is then immediately obtained with the help of the maps Θ and Φ given in Section V.

Here we provide a *Mathematica 6* implementation:

```
zassenhaus[2, alph_List]:=zassenhaus[2,alph]=-StringJoin[alph]/2+
                          StringJoin[Reverse[alph]]/2;
zassenhaus[n_/;n>2, alph_List]:=zassenhaus[n, alph]=
  Module[{a,c,e,j,p,t,u,v,w},
  p = IntegerPartitions[n];
  u = Rest[BinCounts[#, {1, n, 1}] & /@ p];
  v = Flatten[(e=t=First[#]; Table[Join[{j=t--,e-j}, Rest[#]],{e+1}])&/@u,1];
  c = tTransform/@Join[alph, Table[zassenhaus[j, alph], {j, 2, n-1}]];
```

```
p = Flatten[Permutations/@p, 1];
w = Plus@@Join[words[#, alph]/n! & /@ p, words[#, Reverse[alph]]/n!&/@p];
a = MapThread[tScalar, {Times@@(1/#!)&/@v,
              tProduct[MapThread[tPower, {c, #}]]&/@v}];
w - tTransform[a]];
```

The coefficients c_n are returned by `Zassenhaus[n,alph]` for all $n \geq 2$ for a two-letter alphabet `alph`, e.g. `{"a", "b"}`. The possible combinations for the v_i to be summed over are contained in the list `v`. The two sums to be subtracted according to Eq. (11) are stored in `w` and `a`.

In order to support the calculation of the product and the power of non-commutative terms c_n, our implementation uses the following routines which must be loaded before the above code can be executed

```
tPower[X_List, 0] := {{1, ""}};
tPower[X_List, 1] := X;
tPower[X_List, n_ /; n > 1]:=Nest[{#[[1,1]]*#[[2,1]], #[[1,2]]<>#[[2, 2]]}&/@
  Flatten[Outer[List, #,X, 1], 1] &, X, n-1];
tProduct[X_List] := Fold[{#[[1, 1]]*#[[2,1]], #[[1, 2]]<>#[[2, 2]]}&/@
  Flatten[Outer[List, #1, #2, 1], 1]&, First[X], Rest[X]];
tScalar[c_, X_List] :={c*First[#], Last[#]}&/@X;
tTransform[X_List]  :=Plus @@ (Times @@@ Flatten[X, 1]);
tTransform[X_String]:= {{1, X}};
tTransform[X_Plus]  :=List@@@(List@@ X);
```

Furthermore, we need the translation between different representations of words given in Subsection II B.

After the full polynomial representation has been obtained, one may apply the maps Θ and Φ given in Section V in order to obtain a representation in terms of nested commutators. However, it is also possible to formulate a method similarly to the method by Richtmyer and Greenspan for the BCH terms discussed above, which directly uses nested commutators.

B. Matrix method

This method was first presented by the present authors in Ref. [14] and is analogous to Reinsch's method for the BCH series. Let $\tau_1, \ldots, \tau_n$ be arbitrary commutative variables and let $J = (J_{ij})$,

$K = (K_{ij})$, and $L = (L_{ij})$ be three $(n+1) \times (n+1)$ matrices defined by $J_{ij} = 0$, $K_{ij} = 0$, and $L_{ij} = 0$ for $i > j$ and

$$J_{ij} \;=\; \frac{1}{(j-i)!} \cdot \prod_{k=i}^{j-1} (1+\tau_k), \qquad K_{ij} \;=\; \frac{(-1)^{i+j}}{(j-i)!}, \qquad \text{and}$$
$$L_{ij} \;=\; \frac{(-1)^{i+j}}{(j-i)!} \cdot \prod_{k=i}^{j-1} \tau_k$$

for $i \leq j$. Furthermore, we define the two $(n+1) \times (n+1)$ matrices P and Q by

$$P_{ij} \;=\; \delta_{i+1,j} \qquad \text{and} \qquad Q_{ij} \;=\; \delta_{i+1,j}\, \tau_i,$$

where δ_{ij} is the Kronecker delta, i.e. $\delta_{ij} = 0$ for $i \neq j$ and $\delta_{ij} = 1$ for $i = j$. Similar to the operator T, let U be an operator translating a word $W = \tau_1^{\mu_1} \tau_2^{\mu_2} \tau_3^{\mu_3} \ldots \tau_n^{\mu_n}$ with $\mu_i \in \{0, 1\}$ for $i = 1, \ldots, n$ into a product of a and b in the following way: If $\mu_i = 0$, replace $\tau_i^{\mu_i}$ by an a, if $\mu_i = 1$, replace $\tau_i^{\mu_i}$ by b, e.g. $U(\tau_2 \tau_3 \tau_5) = abbaba$ if $n = 6$. Then we have

$$c_n \;=\; U\left(\left(\mathrm{e}^{-C_{n-1}} \cdot \mathrm{e}^{-C_{n-2}} \cdot \ldots \cdot \mathrm{e}^{-C_2} \cdot L \cdot K \cdot J \right)_{1,n+1} \right), \tag{12}$$

where $C_m \;=\; c_m(P, Q)$ is for $m \;<\; n$ the m-th Zassenhaus exponent in terms of the non-commutative matrices P and Q. Again, the index $1, n+1$ indicates the upper-right element of the matrix on the right-hand side.

Using the matrix method, one can apply the maps Θ or Φ discussed in Section V only after the complete polynomial c_n has been obtained. The implementation of the matrix method is rather simple (see Ref. [14] for a *Mathematica* implementation), however in our experience less efficient than the expansion method described before.

IV. COMPUTATIONAL EXPERIENCES

For the BCH series, we compared the running times of our implementation of the three methods discussed above for various values of n. Similarly, for the Zassenhaus product, we compared the expansion method with the matrix method. In all cases, a complete polynomial representation of a BCH or Zassenhaus terms was calculated.

All programs were coded in *Mathematica* 6 and run on a standard personal computer with 1.8 GHz and 1,024 MB of memory. A comparison of the running times for various values of n is given in Table I.

n	Dynkin	Goldberg	Reinsch	Expansion	Matrix
2	0.000	0.004	0.000	0.000	0.000
3	0.000	0.021	0.000	0.005	0.023
4	0.047	0.057	0.000	0.005	0.047
5	0.140	0.083	0.016	0.020	0.141
6	0.360	0.099	0.015	0.037	0.203
7	2.172	0.130	0.078	0.083	0.453
8	44.687	0.240	0.094	0.125	1.125
9	–	0.370	0.188	0.224	3.125
10	–	0.593	0.390	0.614	7.781
11	–	0.912	1.141	1.677	19.406
12	–	1.427	2.313	4.823	46.735
13	–	2.250	6.234	12.828	110.078
14	–	3.557	14.391	36.062	–
15	–	6.036	37.437	99.026	–
16	–	9.844	91.531	–	–
17	–	18.526	–	–	–
18	–	32.276	–	–	–
19	–	65.219	–	–	–
20	–	117.505	–	–	–

TABLE I: Running times for the calculation of BCH and Zassenhaus terms for various values of n.

As expected, the running times for all methods grow exponentially with n. For the BCH terms, Goldberg's method is the most efficient while with Dynkin's formula we could not calculate the BCH terms for $n > 8$ in our time limit of 120 seconds. The method by Goldberg is particularly fast because we can use the symmetry mentioned above.

For the Zassenhaus product, the expansion method is the fastest. However, the calculation of the Zassenhaus terms is more expensive compared to the BCH terms since both methods we studied are recursive.

V. POLYNOMIALS AND NESTED COMMUTATORS

As stated in the introduction, the terms $z_n = z_n(x, y)$ in the BCH series and the terms $c_n = c_n(a, b)$ in the Zassenhaus product may be written as polynomials in x and y or in a and b, respectively. The BCH and Zassenhaus theorems assert that an translation of this representation into linear combinations of nested commutators exists. Obviously, given a commutator representation of z_n or c_n, we obtain the polynomial representation by applying $[x, y] = xy - yx$ iteratively. In this section we want to show how to translate the polynomial representation into a nested commutator representation.

Consider the free associative algebra $\mathcal{R}$ over the field $\mathbb{Q}$ of rational numbers generated by the the non-commutative variables x and y. We will investigate the polynomials $P(x, y)$ contained in the algebra $\mathcal{R}$, which can be expressed as commutators. Such polynomials are called *Lie elements*. Call $[[\cdots[x_1, x_2], \ldots], x_n]$ a *left normal nested commutator* of the x_i with $x_i \in \{x, y\}$. Each polynomial $P \in \mathcal{R}$ may be written as

$$P(x, y) \;=\; \sum a(x_{i_1}, x_{i_2}, \ldots, x_{i_k}) \cdot x_{i_1} x_{i_2} \ldots x_{i_k},$$

where $x_{i_1}, \ldots, x_{i_k} \in \{x, y\}$ and $a(x_{i_1}, x_{i_2}, \ldots, x_{i_k}) \in \mathbb{Q}$. The sum runs over a finite number of factors $(x_{i_1}, x_{i_2}, \ldots, x_{i_k}) \in \{x, y\}^k$, and k in each term is arbitrary. A polynomial with just one summand is called a *monomial*.

We now define a subalgebra $\mathcal{T} \subset \mathcal{R}$ as the smallest subset of $\mathcal{R}$ which contains the following elements (cf. Ref. [6]):

1. $x, y \in \mathcal{T}$,
2. If $P, Q \in \mathcal{T}$, then $\lambda P + \mu Q \in \mathcal{T}$ for all $\lambda, \mu \in \mathbb{Q}$,
3. If $P \in \mathcal{T}$, then $[P, x]$ and $[P, y] \in \mathcal{T}$.

Therefore, the elements of $\mathcal{T}$ are linear combinations of left normal nested commutators of x and y. Note that due to $[x, P] = -[P, x]$ all Lie elements contained in $\mathcal{R}$ are elements of $\mathcal{T}$ and each element of $\mathcal{T}$ is a Lie element.

Furthermore, we define two maps $\Theta, \Phi : \mathcal{R} \to \mathcal{T}$ as follows:

$$\begin{aligned}\Theta\left(P(x,y)\right) &= \Theta\left(\sum a(x_{i_1}, x_{i_2}, \ldots, x_{i_k}) \cdot x_{i_1} x_{i_2} \ldots x_{i_k}\right) \\ &= \sum \frac{a(x_{i_1}, x_{i_2}, \ldots, x_{i_k})}{k} \cdot [[\cdots [x_{i_1}, x_{i_2}], \ldots], x_{i_k}] \end{aligned} \tag{13}$$

and

$$\begin{aligned}\Phi\left(P(x,y)\right) &= \Phi\left(\sum a(x_{i_1} x_{i_2} \ldots x_{i_k}) \cdot x_{i_1} x_{i_2} \ldots x_{i_k}\right) \\ &= \sum \frac{a(x, y, x_{i_3}, x_{i_4}, \ldots, x_{i_k})}{n_x(x_{i_3} x_{i_4} \ldots x_{i_k}) + 1} \cdot [[\cdots [[[x, y], x_{i_3}], x_{i_4}], \ldots], x_{i_k}]. \end{aligned} \tag{14}$$

Here, $n_x(x_{i_3} x_{i_4} \ldots x_{i_k})$ is the number of x generators in the set $\{x_{i_3}, x_{i_4}, \ldots, x_{i_k}\}$, e.g. $n_x(xyxxy) = 3$. Note that the map Φ only involves terms starting with xy.

For example, for the polynomial

$$z_3 = \frac{1}{12}xxy - \frac{1}{6}xyx + \frac{1}{12}xyy + \frac{1}{12}yxx - \frac{1}{6}yxy + \frac{1}{12}yyx$$

the maps Θ and Φ yield

$$\Theta(z_3) = -\frac{1}{18}[[x,y],x] + \frac{1}{36}[[x,y],y] + \frac{1}{36}[[y,x],x] - \frac{1}{18}[[y,x],y], \tag{15}$$

$$\Phi(z_3) = -\frac{1}{12}[[x,y],x] + \frac{1}{12}[[x,y],y]. \tag{16}$$

For the map Θ, the following theorem has been proved independently by Dynkin [6], Specht [15], and Wever [16].

Theorem 1 *For all $P \in \mathcal{T}$, it holds that $\Theta(P) = P$.*

In other words: If we know that a polynomial $P(x, y)$ can be written in terms of commutators, then we can construct such a representation using the map Θ defined in Eq. (13).

As conjectured in Refs. [17] and [14] without proof, we also have the following result:

Corollary 1 *For all $P \in \mathcal{T}$, it holds that $\Phi(P) = P$.*

Using this corollary we obtain a commutator representation which contains fewer terms than the Dynkin-Specht-Wever representation. A proof of this Corollary can be found in the Appendix. Note that $\Theta(P) = \Phi(P) = P$ for all $P \in \mathcal{T}$ and $z_n, c_n \in \mathcal{T}$ for all $n \geq 2$.

VI. CONCLUSION

For the calculation of the Lie elements z_n of the Baker-Campbell-Hausdorff (BCH) series or the c_n of the Zassenhaus product, efficient methods are required. Our aim in this paper was to analyze several of such methods and to provide efficient implementations.

For the BCH series, we compared three different methods (Dynkin's, Goldberg's, and Reinsch's method). For the Zassenhaus product, we discussed an expansion and a matrix method. Our computational experiences show that Goldberg's method is most efficient for calculating the BCH terms, and the expansion method appears to be the best for the calculation of the Zassenhaus terms.

In our approach the BCH and Zassenhaus terms are obtained in a polynomial representation. A translation into a nested commutator representation is provided by the famous Dynkin-Specht-Wever theorem. Another translation into a commutator representation is stated in Corollary 1 and proven in the Appendix. It yields a representation with fewer terms than the Dynkin-Specht-Wever representation.

Both methods we studied for the Zassenhaus product are recursive. Non-recursive, potentially faster methods similar to Goldberg's method for the BCH terms are not known. It is an open question if a generating function (similar to Eq. (6)) for the coefficients in the Zassenhaus terms can be constructed.

APPENDIX A: PROOF OF COROLLARY 1

We will first collect a number of elementary properties of the Lie elements which may be easily proven by induction, see e.g. Ref. [15].

Lemma 1 *Left normal nested commutators* $[[\cdots[x_1, x_2], \ldots], x_n]$ *may be expanded as*

$$[[\cdots[x_1, x_2], \ldots], x_n] \;=\; x_1 x_2 \ldots x_n + \ldots, \tag{17}$$

and the terms not written out explicitly do not start with x_1*.*

Lemma 2 *Each arbitrarily nested commutator can be written as a linear combination of left normal nested commutators.*

Lemma 3 *For a left normal nested commutator* $[[\cdots[x_1,x_2],\ldots],x_n]$ *and a given* $k\in\{1,\ldots,n\}$ *there exists a* $c(x_k,x_{j_2},\ldots,x_{j_n})\in\mathbb{Q}$ *such that*

$$[[\cdots[x_1,x_2],\ldots],x_n] \;=\; \sum c(x_k,x_{j_2},\ldots,x_{j_n})\cdot[[\cdots[x_k,x_{j_2}],\ldots],x_{j_n}]. \tag{18}$$

The sum is over all permutations $j_2,\ldots,j_n$ *of the numbers* $1,\ldots,k-1,k+1,\ldots,n$.

We now want to prove Corollary 1: For all $P\in\mathfrak{T}$, it holds that $\Phi(P)=P$ (see Section V). Note that our proof is purely algebraic, and all infinite series and products are formal. We do not introduce a topology and, therefore, cannot address questions of convergence. Our proof heavily relies on Dynkin's methods discussed in Ref. [6].

Proof. According to Lemma 2, each polynomial contained in $\mathfrak{T}$ can be written as a linear combination of left normal nested commutators. Therefore, it is sufficient to prove our assertion for

$$P(x,y) \;=\; [[\cdots[x_{i_1},x_{i_2}],\ldots],x_{i_k}]$$

with $x_{i_1},\ldots,x_{i_k}\in\{x,y\}$, i.e. we only need to show that $\Phi(P(x,y))=P(x,y)$. Expanding the commutators by repeated application of $[x_i,x_j]=x_ix_j-x_jx_i$ we obtain

$$[[\cdots[x_1,x_2],\ldots],x_n] \;=\; \sum a(x_{i_1},x_{i_2},\ldots,x_{i_n})\cdot x_{i_1}x_{i_2}\ldots x_{i_n}, \tag{19}$$

where the sum is over all permutations $i_1,\ldots,i_n$ of the numbers $1,\ldots,n$, and $a(x_{i_1},x_{i_2},\ldots,x_{i_n})\in\mathbb{Q}$.

According to Lemma 3, for a fixed $k\in\{1,\ldots,n\}$ it holds that

$$[[\cdots[x_1,x_2],\ldots],x_n] \;=\; \sum c(x_k,x_{j_2}\ldots x_{j_n})\cdot[[\cdots[x_k,x_{j_2}],\ldots],x_{j_n}]. \tag{20}$$

Here, the summation runs over all permutations $j_2,\ldots,j_n$ of the numbers $1,\ldots,k-1,k+1,\ldots,n$.

Furthermore, according to Lemma 1 it holds that

$$[[\cdots[x_k,x_{j_2}],\ldots],x_{j_n}] \;=\; x_kx_{j_2}\ldots x_{j_n}+\ldots, \tag{21}$$

where the terms not written out explicitly do not start with x_k. Using Eqns. (19) and (20) one obtains

$$\sum a(x_{i_1}x_{i_2}\ldots x_{i_n})\cdot x_{i_1}x_{i_2}\ldots x_{i_n} \;=\; \sum c(x_k,x_{j_2}\ldots x_{j_n})\cdot[[\cdots[x_k,x_{j_2}],\ldots],x_{j_n}],$$

which, according to Eq. (21), may be written as

$$\sum a(x_{i_1}, x_{i_2}, \ldots, x_{i_n}) \cdot x_{i_1} x_{i_2} \ldots x_{i_n} \;=\; \sum c(x_k, x_{j_2}, \ldots, x_{j_n}) \cdot (x_k x_{j_2} \ldots x_{j_n} + \ldots).$$

Consequently, it holds that $a(x_k, x_{j_2}, \ldots, x_{j_n}) = c(x_k, x_{j_2}, \ldots, x_{j_n})$, and for arbitrary but fixed k one finds

$$[[\cdots[x_1, x_2], \ldots], x_n] \;=\; \sum a(x_k, x_{j_2}, \ldots, x_{j_n}) \cdot [[\cdots[x_k, x_{j_2}], \ldots], x_{j_n}], \tag{22}$$

and the sum is over all permutations $j_2, \ldots, j_n$ of the numbers $1, \ldots, k-1, k+1, \ldots n$.

Now we write down as many copies of Eq. (22) as there are x generators in the set $\{x_1, \ldots x_n\}$, one copy for each $x_k = x$, and add them up:

$$\begin{aligned} n_x(x_1 \ldots x_n) \cdot [[\cdots[x_1, x_2], \ldots], x_n] \;&=\; \sum a(x_{i_1}, \ldots, x_{i_n}) \cdot [[\cdots[x_{i_1}, x_{i_2}], \ldots], x_{i_n}] \\ &=\; \sum a(x, x_{i_2}, \ldots, x_{i_n}) \cdot [[\cdots[x, x_{i_2}], \ldots], x_{i_n}], \end{aligned} \tag{23}$$

and here the sum is over all permutations $i_2, \ldots, i_n$ of the numbers $2, \ldots, n$ and $x_{i_1} = x$. The symbol $n_x(x_1 \ldots x_n)$ denotes the number of x generators in the set $\{x_1, \ldots, x_n\}$.

From Eqns. (19) and (23) we obtain using $[x, x] = 0$

$$\begin{aligned} \Phi(P(x_1, \ldots, x_n)) \;&=\; \Phi\left(\sum a(x_{i_1}, x_{i_2}, \ldots, x_{i_n}) \cdot x_{i_1} x_{i_2} \ldots x_{i_n}\right) \\ &=\; \sum \frac{a(x, y, x_{i_3}, \ldots, x_{i_n})}{n_x(x_{i_3} x_{i_4} \ldots x_{i_n}) + 1} \cdot [[\cdots[[[x, y], x_{i_3}], x_{i_4}], \ldots], x_{i_n}] \\ &=\; \sum \frac{a(x, x_{i_2}, \ldots, x_{i_n})}{n_x(x_{i_3} x_{i_4} \ldots x_{i_n}) + 1} \cdot [[\cdots[[x, x_{i_1}], x_{i_3}], \ldots], x_{i_n}] \\ &=\; \sum \frac{a(x, x_{i_2}, \ldots, x_{i_n})}{n_x(x_{i_1} x_{i_2} \ldots x_{i_n})} \cdot [[\cdots[[x, x_{i_1}], x_{i_3}], \ldots], x_{i_n}] \\ &=\; \frac{1}{n_x(x_1 \ldots x_n)} \cdot \sum a(x, x_{i_2}, \ldots, x_{i_n}) \cdot [[\cdots[[x, x_{i_1}], x_{i_3}], \ldots], x_{i_n}] \\ &=\; [[\cdots[x_1, x_2], \ldots], x_n], \end{aligned}$$

which proves Corollary 1.

---

[1] H. Baker, Proc. London Math. Soc. **s2-3**, 24 (1905).

[2] J. Campbell, Proc. London Math. Soc. **s1-29**, 14 (1897).

[3] F. Hausdorff, Ber. Verh. Saechs. Akad. Wiss., Leipzig, Math.-Phys. Kl. **58**, 19 (1906).

[4] W. Magnus, Commun. Pure Appl. Math. **7**, 649 (1954).

[5] Wolfram Research, Inc., *Mathematica Edition: Version 6* (Wolfram Research, Inc., 2008).

[6] E. Dynkin, Dokl. Akad. Nauk. SSSR **57**, 323 (1947), in Russian. An English translation may be found in "Selected papers of E.B. Dynkin with commentary", E.B. Dynkin, A.A. Yushkevich, G.M. Seitz, A.L. Onishchik, editors. American Mathematical Society, Providence, R.I., and International Press, Cambridge, Mass., 2000.

[7] A. Bose, J. Math. Phys. **30**, 2035 (1989).

[8] R. Richtmyer and S. Greenspan, Commun. Pure Appl. Math. **18**, 107 (1965).

[9] V. Varadarajan, *Lie–Groups, Lie–Algebras, and Their Representations* (Prentice-Hall, Englewood Cliffs, N.J., 1974), 1st ed.

[10] K. Goldberg, Duke Math. J. **23**, 13 (1956).

[11] M. Reinsch, J. Math. Phys. **41**, 2434 (2000).

[12] R. Wilcox, J. Math. Phys. **8**, 962 (1966).

[13] C. Quesne, Int. J. Theor. Phys. **43**, 545 (2004).

[14] D. Scholz and M. Weyrauch, J. Math. Phys. **47**, 033505 (2006).

[15] W. Specht, Math. Zeitschr. **51**, 367 (1948).

[16] F. Wever, J. reine angew. Math. **187**, 44 (1947).

[17] J. Oteo, J. Math. Phys. **32**, 419 (1991).

[18] C. Reutenauer, *Free Lie Algebras* (Clarendon Press, Oxford, 1993), 1st ed.

[19] M. Newman and R. C. Thompson, Math. Comp. **48**, 265 (1987).

Institut für Numerische und Angewandte Mathematik
Universität Göttingen
Lotzestr. 16-18
D - 37083 Göttingen

Telefon: 0551/394512
Telefax: 0551/393944

Email: trapp@math.uni-goettingen.de URL: http://www.num.math.uni-goettingen.de

Verzeichnis der erschienenen Preprints 2008:

2008-01	M. Körner, A. Schöbel	Weber problems with high-speed curves
2008-02	S. Müller, R. Schaback	A Newton Basis for Kernel Spaces
2008-03	H. Eckel, R. Kress	Nonlinear integral equations for the complete electrode model in inverse impedance tomography
2008-04	M. Michaelis, A. Schöbel	Integrating Line Planning, Timetabling, and Vehicle Scheduling: A customer-oriented approach
2008-05	O. Ivanyshyn, R. Kress, P. Serranho	Huygen's principle and iterative methods in inverse obstacle scattering
2008-06	F. Bauer, T. Hohage, A. Munk	Iteratively regularized Gauss-Newton method for nonlinear inverse problems with random noise
2008-07	R. Kress, N. Vintonyak	Iterative methods for planar crack reconstruction in semi-infinite domains
2008-08	M. Uecker, T. Hohage, K.T. Block, J. Frahm	Image reconstruction by regularized nonlinear inversion - Joint estimation of coil sensitivities and image content
2008-09	M. Schachtebeck, A. Schöbel	IP-based Techniques for Delay Management with Priority Decisions
2008-10	S. Cicerone, G. Di Stefano, M. Schachtebeck, A. Schöbel	Dynamic Algorithms for Recoverable Robustness Problems
2008-11	M. Braack, G. Lube	Finite elements with local projection stabilization for incompressible flow problems
2008-12	T. Knopp, X.Q. Zhang, R. Kessler, G. Lube	Calibration of a finite volume discretization and of model parameters for incompressible large eddy-type simulations
2008-13	P. Knobloch, G. Lube	Local projection stabilization for advection-diffusion-reaction problems: One-level vs. two-level approach

2008-14	G. Lube, B. Tews	Distributed and boundary control of singularly perturbed advection-diffusion-reaction problems
2008-15	G. Lube, B. Tews	Optimal control of singularly perturbed advection-diffusion-reaction problems
2008-16	M. Weyrauch, D. Scholz	Computing the Baker-Campbell-Haussdorff series and the Zassenhaus product