# BRST Quantization of String Theories
## Applying REDUCE to High Energy Physics

*Werner M. Seiler*†

Institut für Theoretische Physik, Universität Karlsruhe
D-7500 Karlsruhe 1, West Germany
BITNET: BE04@DKAUNI2

A REDUCE package for commutator calculations in supersymmetric theories (including ordered products) and for infinite sums is presented and applied to the computation of anomalies in string theory.

### 1. Introduction

While the use of computer algebra in general relativity has become fairly common, most applications in high energy physics concern Feynman diagrams. Examples which tend more to model calculations are rare: Castellani reports about a package for supergravity[2)], other authors tried REDUCE in superspace formalism[6,8)].

In this paper, we present a package written in REDUCE 3.3[7)] for commutator calculations in supersymmetric theories, for the handling of ordered products and for the simplification of infinite sums. As an application, anomalies (or Schwinger terms) of constraint algebras in string theory are computed. For commutator calculations Cecchini and Tarlani[3)] (see also their contribution to this workshop) recently presented a COMMON LISP program. But in our case, it is important, for the further processing of the results, to integrate the package into a computer algebra system like REDUCE.

We start with a very short description of the physical background. The following two sections describe shortly the REDUCE package *SUPERCALC* implemented for the calculations. A more explicit description can be found elsewhere[9,10)]. First, we consider the computation of commutators and the handling of ordered expressions, then we regard the simplification of sums. The last section shows a concrete example and gives some conclusions.

### 2. String Theory

In the last years, string theory has attracted a lot of interest as a possible candidate

---

† Supported by Studienstiftung des deutschen Volkes
New address: Institut für Algorithmen und Kognitive Systeme, Universität Karlsruhe

for an unified theory. A striking feature of this theory is the existence of a so-called critical dimension: A consistent quantization is possible only for one special dimension of space-time.

The physical ideas of string theory can be found in the "bible"[5]. Here, we concentrate on the computational aspects of quantization within the BRST formalism. Our basic objects are the Fourier modes of the fundamental fields of the theory. They obey canonical commutation relations, for the bosonic string, for example,

$$[\alpha_m, \alpha_n] = m\delta_{m+n,0} \,. \tag{1}$$

The string Lagrangian is degenerate and gives rise to constraints. Their Fourier modes

$$L_m = \sum_{-\infty}^{\infty} :\alpha_{m-n}\alpha_n: \tag{2}$$

generate the famous Virasoro algebra with relations

$$[L_m, L_n] = (m-n)L_{m+n} + \frac{D}{12}(m^3 - m)\delta_{m+n,0} \,. \tag{3}$$

The colons denote normal ordering defined by the pairings

$$\underbrace{\alpha_m \alpha}_{}{}_n = m\theta(m)\delta_{m+n,0} \,. \tag{4}$$

These expressions already show all characteristics of the calculations: We always deal with bilinear currents like the $L_m$'s; because of the ordering, central extensions or anomalies arise in the algebras of constraints.

In supersymmetric models, the number of modes and therefore the size of the algebras increases. We must work with fermionic modes, too then. Further modes (so-called "ghosts") are introduced by the BRST theory. We call the quantization consistent, if the anomalies vanish. This condition determines the space-time dimension $D$.

To compute the commutator between ordered expressions, we need the theorem of Wick[1], expanding a product in a sum of ordered products:

$$\begin{aligned} A_1 A_2 \ldots A_n = \ & :A_1 A_2 \ldots A_n: + \\ & + \sum_{\text{one pairing}} :A_1 \ldots A_i \ldots A_j \ldots A_n: + \\ & + \sum_{\text{two pairings}} :A_1 \ldots A_i \ldots A_k \ldots A_l \ldots A_j \ldots A_n: + \ldots \end{aligned} \tag{5}$$

A similiar formula exists for the product of two ordered products.

## 3. Commutator Calculations

For commutator calculations and the handling of ordered expressions, *SUPERCALC* provides three procedures: `bracket`, `wick` and `ordprod`. `bracket` uses only the basic, algebraic properties of a graded commutator (with the exception of ordered products, where the definition of a commutator is used):

$$[A,B] = (-)^{\epsilon_A \epsilon_B}\,[B,A] \ , \tag{6a}$$

$$[A+B,C] = [A,C] + [B,C] \ , \tag{6b}$$

$$[AB,C] = A\,[B,C] + (-)^{\epsilon_B \epsilon_C}\,[A,C]\,B \tag{6c}$$

($\epsilon_A$ denotes the grading of the operator $A$). Hence, it can also be used for other structures like Poisson or Jacoby brackets. The fundamental commutation relations are introduced to *SUPERCALC* by means of `LET` rules. If the switch `zerocomm` is set, only the nonvanishing commutators must be given. Any commutator without a defining `LET` rule is then eliminated at once by the simplifier.

A straightforward recursive implementation of the rules (6a-c) leads to a fairly inefficient algorithm. `bracket` tries to improve the efficiency by first computing the complexity of each argument, and starting with the more involved one. For the most common cases — sums, products and powers — special procedures are called, working iteratively.

`wick` expands a product of operators into a sum of ordered products following the theorem of Wick (5); `ordprod` does the same for a product of ordered products. A depth-first approach is used to generate all terms. This algorithm possesses the highest complexity of the whole package. For a low number $n$ of factors, the number of terms in the expansion is growing approximately exponentially, for a large number with $n^n$. (For $n=5$ we get 25, for $n=8$ already 763 summands!)

## 4. Sums

Most operators of string theory are defined as infinite sums. Kronecker $\delta$'s and step functions occur in the commutators and pairings of the modes. To handle such expressions, *SUPERCALC* provides the operator `ssum(s,n,l,u)` and the procedure `evalsum` to simplify terms with `ssum`. Here `s` denotes the summand, `n` the index and `l,u` the lower resp. upper bound.

The simplification of elementary sums (at present: summands linear or quadratic in the index) is integrated into the REDUCE simplifier and hence performed automatically. For more complex cases, `evalsum` must be invoked. This procedure tries six simplification rules. The first three consider single sums: The summand is checked for an even or odd symmetry, sums over expressions containing a Kronecker $\delta$ are evaluated, and the bounds are adjusted in sums with step functions.

The other three rules work on linear expressions in `ssum`. First, sums over ranges of equal size are collected into a single sum to give REDUCE a chance to simplify the summand. For sums over summands of the same type, two cases are considered: Either the sums cancel each other partially, or their ranges are adjacent. Both times a single sum is generated. All three rules search automatically for index shifts allowing their application.

`evalsum` makes no use of summation theory. It is a completely heuristic approach which suffices for the sums in our calculations. For multiple sums the evaluation can be very time-consuming, because every level must be investigated seperately. The elementary sums are computed by a kind of table look-up. Hence, an extension of this list is easily possible.

Besides the main procedures presented in this and the preceding section, *SUPERCALC* must tackle a lot of minor problems. In particular, a legiable output is important. Further points are e.g. effective control of `LET` rules and the distinction between integer and half integer indices.

## 5. An example

The following figure contains a complete session computing the algebra (3). To get the simplest form of the result, we must give some hints with `LET` rules. A completely automatic calculations (which also detects the necessary index shifts!) would be very difficult to program. Especially the recognition of the Virasoro operator in the final expression, which is very easy in our example, requires nontrival pattern matching in the case of superstrings.

```
setgreater(m,0);
Time: 3 ms
pp:=bracket(normord(alpha(m-j)*alpha(j)),
            normord(alpha(n-k)*alpha(k)));
PP := DELTA          *DELTA      *THETA(-J+M)*THETA(J)*J*(-J+M)
          -J-K+M+N,0       J+K,0
      +DELTA          *DELTA      *THETA(-K+N)*THETA(K)*K*(K-N)
           -J-K+M+N,0       J+K,0
      +DELTA          *THETA(-J+M)*:ALPHA *ALPHA :*(-J+M)
           -J-K+M+N,0                  J      K
      +DELTA          *THETA(-K+N)*:ALPHA *ALPHA :*(K-N)
           -J-K+M+N,0                  J      K
      +DELTA        *DELTA        *THETA(-J+M)*THETA(J)*J*(-J+M)
           -J+K+M,0      J-K+N,0
      +DELTA        *DELTA        *THETA(-K+N)*THETA(K)*K*(K-N)
           -J+K+M,0      J-K+N,0
      +DELTA        *THETA(-J+M)*:ALPHA     *ALPHA :*(-J+M)
           -J+K+M,0                  -K+N      J
      -(DELTA        *THETA(K)*K)*:ALPHA     *ALPHA :
            -J+K+M,0                  -K+N      J
```

```
      +DELTA       *THETA(-K+N)*:ALPHA    *ALPHA :*(K-N)
            J-K+N,0                   -J+M      K
      +DELTA       *THETA(J)*J*:ALPHA    *ALPHA :
            J-K+N,0                  -J+M      K
      +DELTA     *THETA(J)*J*:ALPHA    *ALPHA    :
            J+K,0                  -J+M      -K+N
      -(DELTA     *THETA(K)*K)* :ALPHA    *ALPHA    :
             J+K,0                    -J+M      -K+N
Time: 1378 ms
for all x such that not freeof(x,n) let
  delta(m+n,0)*x=delta(m+n,0)*sub(n=-m,x);
Time: 15 ms
qq:=evalsum ssum(pp,j,-aleph,aleph);
QQ := -(2*DELTA     *THETA(-K)*THETA(K+M)*K)*(K+M)
               M+N,0
      +2*DELTA     *THETA(-K-M)*THETA(K)*K*(K+M)
              M+N,0
      -(THETA(-K)*K)*:ALPHA    *ALPHA   :
                           -K+N      K+M
      -(THETA(-K)*K)*:ALPHA   *ALPHA    :
                           K+M      -K+N
      +2*THETA(-K+N)*:ALPHA      *ALPHA :*(K-N)
                           -K+M+N     K
      +2*THETA(K-N)*:ALPHA      *ALPHA :*(K-N)
                          -K+M+N     K
      -(THETA(K)*K)*:ALPHA    *ALPHA   :
                          -K+N      K+M
      -(THETA(K)*K)*:ALPHA   *ALPHA    :
                          K+M      -K+N
Time: 3068 ms
rr:=evalsum ssum(qq,k,-aleph,aleph);
                      2
RR := (DELTA     *M*(M -1)
            M+N,0
       +6*SSUM(:ALPHA       *ALPHA   :*I:1,I:1,-ALEPH,ALEPH)
                     M+N-I:1      I:1
       -3*SSUM(:ALPHA     *ALPHA     :*I:1,I:1,-ALEPH,ALEPH)
                     M+I:1      N-I:1
       -3*SSUM(:ALPHA     *ALPHA     :*I:1,I:1,-ALEPH,ALEPH)
                     N-I:1      M+I:1
       -(6*N)*SSUM(:ALPHA       *ALPHA   :,I:1,-ALEPH,ALEPH))/3
                         M+N-I:1      I:1
Time: 2934 ms
for all k let
  ssum(normord(alpha(m+k)*alpha(n-k))*k,k,-aleph,aleph)=
    ssum(normord(alpha(m+n-k)*alpha(k))*(k-m),k,-aleph,aleph);
Time: 32 ms
rr/4;
                2
(DELTA     *M*(M -1)
      M+N,0
 +6*SSUM(:ALPHA       *ALPHA   :,I:1,-ALEPH,ALEPH)*(M-N))/12
               M+N-I:1      I:1
Time: 275 ms
```

A complete session computing the Virasoro algebra (3). The output is slightly edited for better readibility. The exection times are those of a Siemens 7880 mainframe under TSO. `aleph` is a keyword for infinity.

Similiar problems occur in many applications of computer algebra. They demonstrate the importance of interactive usage. But this requires reasonably execution times! In REDUCE, we achieve this only by working in the symbolic (RLISP) mode and by a close integration into the system. Both demand a detailed knowledge of the internal structure of REDUCE (for which no documentation besides the source code exists).

The main reason to use computer algebra for the calculations presented here lies in the reliability of the results. It does not take too much time to commute two bilinear currents like (2). But the large number of similiar calculations with only slight changes (e.g. fermionic instead of bosonic modes) inevitably leads to errors.

A comparison of this work with the results of Gorman *et al.*[4] leads to an interesting point. There, a fairly general formula for the anomalies in Virasoro and related Kac-Moody algebras is derived. But its application is more tedious than a computer-aided, brute force calculation. An implementation of their expression would be rather difficult and would result in a highly specialized program, whereas the procedures of *SUPERCALC* can be used for many different tasks.

## 6. Acknowledgments

It's a pleasure to thank Göktürk Üçoluk for many valuable hints about REDUCE, and Marcus Scholl for many discussions about BRST and string theory.

## 7. References

[1] N.N. Bogoljubov, D.V. Shirkov: Introduction to the Theory of Quantized Fields, Interscience Publisher, New York 1959

[2] L. Castellani, Int. J. Mod. Phys. A3(1988)1435

[3] R. Cecchini, M. Tarlani, Comp. Phys. Comm. 52(1989)283

[4] N. Gorman, W. McGlinn, L. O'Raifeartaigh, D. Williams, Int. J. Mod. Phys. A4(1989)1235

[5] M.B. Green, J.H. Schwarz, E. Witten: Superstring Theory, vol. I&II, Cambridge University Press, Cambridge (UK) 1987

[6] R. Grimm, H. Kühnelt, Comp. Phys. Comm. 20(1980)77

[7] A.C. Hearn: REDUCE 3.3 – User Manual, RAND Publication CP78, The RAND Corporation, Santa Monica 1987

[8] R.P. dos Santos, J. Symb. Comp. 7(1989)523

[9] W.M. Seiler: Die Bestimmung der kritischen Dimensionen von String und Superstring mit REDUCE, master thesis (in german), Karlsruhe 1989

[10] W.M. Seiler: BRST Quantization of String Theories with REDUCE, to appear