A *Mathematica*™ Package for Computing Operator Product Expansions
(OPEdefs 3.1)

K. Thielemans*

Theoretical Physics Group
Imperial College
London SW7 2BZ (UK)

April 1995

Abstract

A general purpose *Mathematica*^{TM} package for computing Operator Product Expansions of composite operators in meromorphic conformal field theory is described. Given the OPEs for a set of "basic" fields, OPEs of arbitrarily complicated composites can be computed automatically. Normal ordered products are always reduced to a standard form. As an explicit example, the conformal anomaly for superstrings is computed.

The most important extensions with respect to the first version of the package are the ability the check the Jacobi identities, and to compute Poisson brackets ("classical OPEs").

---

[*] Email address : k.thielemans@ic.ac.uk

1 Introduction

Operator Product Expansions (OPEs) are extensively used in conformal field theories, for example in string theory and statistical physics. They are used to evaluate expectation values of several fields with arguments in neighbouring points. One then writes

$$A(z)B(w) = \frac{C(w)}{(z-w)^2} + \frac{D(w)}{z-w} + \ldots \tag{1}$$

where the dots represent the regular terms in the Laurent expansion. The above expression is valid when evaluating expectation values for $z \to w$.

In handling OPEs, the problem of normal ordering requires special attention. In the so called *point splitting regularization* scheme, the normal ordered product of two operators is defined as the zero'th order term in their OPE. The OPE of an operator with a composite follows from a variation on Wick's theorem [1]. However, calculations quickly become long and error prone when composites of composites are involved.

The definition of normal ordering is noncommutative and nonassociative. As a consequence, it is recommended to use a standard order for the operators and introduce a standard way of normal ordering composites of several operators. To this end, there exists a set of prescriptions and again these formulas are conceptually quite simple, but have to be applied recursively in complicated cases.

OPEdefs is written in $Mathematica^{TM}$ [1], an interactive environment for performing symbolic computations. The advantages of writing the package in *Mathematica* are numerous. Its programming language has very powerful pattern matching capabilities. The result of a computation can be transformed with the help of built-in functions (expanding, factoring, collecting terms in specified variables ...) and one may even obtain output in TeX-form. *Mathematica* is running on a lot of machines, from PC's to supercomputers and all versions are completely compatible. And last but not least, it is considerably less time-consuming to program this problem in *Mathematica* than, for instance, in *C*.

In using the package, all "basic" operators of the theory have to be declared to be bosonic or fermionic (parafermions are not supported), and the OPEs of the basic operators must be given. Fields with any (also negative) conformal dimension may be used, the only restriction is that fractional

---

[1] *Mathematica* is a trademark of Wolfram Research Inc. For details, see [2].

powers in the Laurent expansion are not supported. With this input, the package is able to compute OPEs of arbitrarily complicated composites, given enough memory and time. Also, normal ordered products are automatically reduced to standard form (putting operators in the order they are declared and normal ordering them from right to left).

Version 2.0 of *OPEdefs* is already presented in [3]. Version 3.0 is described in [4]. In this paper, version 3.1 is briefly described. A more complete overview of the OPE formalism and of the internals of *OPEdefs* is given in [5] to which I refer for further details.

The paper is organised as follows. In section 2, the necessary rules for computing OPEs are given and the algorithm is discussed briefly. The next section explains how to use the package. Finally, some runtimes are given.

Notation

Input for and output from *Mathematica* is written in `typeset` font. Input lines are preceded by "*In[n] :=*", and corresponding output statements by "*Out[n] =*", as in *Mathematica*.

We use the following notation for OPEs

$$A(z)B(w) = \sum_{n<=h(A,B)} \frac{[AB]_n(w)}{(z-w)^n} \quad , \tag{2}$$

where $h(A,B)$ is some finite number, and is usually given by the sum of the conformal dimensions of A and B. $[AB]_0$ will be called the normal ordered product of A and B.

For Poisson (or in fact Dirac) brackets we use

$$\{A(x), B(x_0)\}_{\rm PB} = \sum_{n>0} \frac{(-1)^{n-1}}{(n-1)!} \{AB\}_n(x_0) \partial^{n-1} \delta^{(2)}(x-x_0)\,, \tag{3}$$

where $\delta^{(2)}(x-x_0)$ is the Dirac delta-function on the plane. The derivative is with respect to the x–coordinate. We choose the normalisation factors such that:

$$\{AB\}_n(y) = \int dx^2\ (x-y)^{n-1} \{A(x), B(y)\}_{\rm PB}\,. \tag{4}$$

2 Formulas

In [3], all formulas (extracted from [1]) needed to compute OPEs are given. The more complicated rules can be derived in the following way, see [5] for more details.

Using the definition (2) for the OPEs, and Cauchy's residue formula for contour integrals, we can isolate the contribution of a certain part of the OPE by taking appropriate contour integrals:

$$\begin{aligned}[A[BC]_p]_q(u) = &\oint_{C_u} \frac{dz}{2\pi i}\,(z-u)^{q-1} \\ &\oint_{C_u} \frac{dw}{2\pi i}\,(w-u)^{p-1}\,A(z)B(w)C(u)\,, \end{aligned} \tag{5}$$

where C_u denotes a contour which encircles u once anti-clockwise. We can now use a contour deformation argument relating the contour integral in eq. (5) to a contour integral where the integration over w is performed last. This integral has two terms: one where the z contour is around u, and one where it is around w. We find:

$$[A[BC]_p]_q = (-1)^{|A|\,|B|}[B[AC]_q]_p + \sum_{l>0} \binom{q-1}{l-1}\,[[AB]_l C]_{p+q-l}\,. \tag{6}$$

This equation has to hold (inside correlators) for consistency of the OPE-formalism. It is valid for *any* integers p, q, *i.e.* also negative numbers. However, in practice we use the Jacobi identities eq. (6) for positive p, q as equations for the singular part of the OPEs, while for p or q zero, they define how one should calculate with composites.

OPEdefs 3.1 can calculate OPEs and Poisson brackets. We concentrate on OPEs here and briefly comment which changes are needed for the other case. However, note that the Jacobi identities eq. (6) for positive p, q remain exactly the same.

2.0.1 Computing OPEs

The package should compute OPEs of arbitrarily complicated composites when a set of generators and their OPEs is given. For easy reference, we list the rules we need, aside from linearity. First, there are the rules involving derivatives:

$$[\partial A\, B]_q = -(q-1)[AB]_{q-1} \tag{7}$$

$$[A\,\partial B]_q = (q-1)[AB]_{q-1} + \partial[AB]_q\,. \tag{8}$$

Next, we need the OPE of B with A, given the OPE of A with B:

$$[BA]_q = (-1)^{|A||B|} \sum_{l \geq q} \frac{(-1)^l}{(l-q)!} \partial^{(l-q)} [AB]_l \,. \tag{9}$$

OPEs with composites can be calculated using eq. (6):

$$\begin{aligned} [A\ [BC]_0]_q &= (-1)^{|A||B|}[B\ [AC]_q]_0 + [[AB]_q\ C]_0 \\ &\quad + \sum_{l=1}^{q-1} \binom{q-1}{l} [[AB]_{q-l} C]_l \end{aligned} \tag{10}$$

where $q \geq 1$. Furthermore, we will use:

$$[AA]_0 = -\sum_{l>0} \frac{(-1)^l}{2l!} \partial^l [AA]_l \ , \quad \text{for } A \text{ fermionic} \ . \tag{11}$$

These rules are the only ones needed to compute every OPE in the OPA. Indeed, when computing the OPE of A with B, we apply the following procedure:

- if A and B are generators whose OPE we know, return it as the result.
- apply linearity if necessary.
- if A is an operator with derivatives, use eq. (7).
- if B is an operator with derivatives, use eq. (8).
- if A or B contains a composite, apply eq. (11) if necessary.
- if B is a composite, use eq. (10).
- if A is a composite, use eq. (9).
- if the OPE $B(z)\ A(w)$ is known, compute the OPE $A(z)\ B(w)$ using eq. (9).

This list should be used recursively until none of the rules applies, which means that the OPE has been calculated. The order in which we check the rules is in this case not important, but we will check them in a "top-down" order. Note that to compute an OPE of a composite with a generator, first eq. (9) is used, and eq. (10) in the next step.

The algorithm used to compute Poisson brackets is essentially the same. Some small changes in the rule eq. (10) are needed, namely all double contractions have to be dropped. Also, eq. (11) is changed to $[AA]_0 = 0$ when A is fermionic.

2.0.2 Simplifying composites

We now discuss how we can reduce normal ordered products to a standard form. We define an order on the generators and their derivatives, *e.g.* lexicographic ordering. Given a composite, we apply the rules given below until all composites are normal ordered from right to left and the operators are ordered, *i.e.* $[A[B[C\ldots]_0]_0]_0$, and $A \leq B \leq C$. The relevant formulas are:

$$\partial[AB]_0 = [A\ \partial B]_0 + [\partial A\ B]_0 \tag{12}$$

$$[AB]_{-q} = \frac{1}{q!}[(\partial^q A)\ B]_0\,, \qquad q \geq 1 \tag{13}$$

$$[BA]_0 = (-1)^{|A||B|}[AB]_0 + (-1)^{|A||B|}\sum_{l\geq 1}\frac{(-1)^l}{l!}\partial^l[AB]_l \tag{14}$$

$$[A\ [BC]_0]_0 = (-1)^{|A||B|}[B\ [AC]_0]_0 + \tag{15}$$

$$[([AB]_0 - (-1)^{|A||B|}[BA]_0)\ C]_0 \tag{16}$$

For the case of Poisson brackets, the rules (14) and (16) drastically simplify, only the first terms remain.

2.0.3 Improvements

The rules given in this section up to now are sufficient to compute any OPE, and to reorder any composite into a standard form. However, some shortcuts exist [2], and where introduced in *OPEdefs* 3.x.

$[A\ [BC]_0]_q$ can be computed using eq. (10), but an alternative is:

$$\begin{aligned}[A\ [BC]_0]_q &= (-1)^{|A||B|}\Big([B\ [AC]_q]_0 + \sum_{l\geq 0}\frac{(-1)^{l+q}}{l!}[\partial^l[BA]_{l+q}\ C]_0 \\ &\qquad + \sum_{l=1}^{q-1}(-1)^l[[BA]_l\ C]_{q-l}\Big)\,. \end{aligned} \tag{17}$$

This rule is more convenient when we know the OPE $B(z)A(w)$ while $A(z)B(w)$ has to be computed using eq. (9).

---

[2] Eqs. (17)- (19) follow from the familiar argument of splitting the paths of a contour-integral, and using $[A\ B]_{-n} = \frac{1}{n!}[A^{(n)}\ B]_0$ for positive n, where $A^{(n)}$ denotes the n-th derivative of A.

Similarly, to compute an OPE where the first operator is a composite, the algorithm as presented above uses eqs. (9) and (10). However, the next rule implements this in one step:

$$\begin{aligned}[[AB]_0\ C]_q &= \sum_{l\geq 0}\frac{1}{l!}[\partial^l A\ [BC]_{l+q}]_0 + (-1)^{|A||B|}\sum_{l\geq 0}\frac{1}{l!}[\partial^l B\ [AC]_{l+q}]_0 \\ &\quad +(-1)^{|A||B|}\sum_{l=1}^{q-1}[B\ [AC]_{q-l}],, \qquad (18)\end{aligned}$$

where $q \geq 1$ and:

$$\begin{aligned}[[AB]_0 C]_0 &= [A[BC]_0]_0 + \\ &\quad \sum_{l>0}\frac{1}{l!}[\partial^l A\ [BC]_l]_0 + (-1)^{|A||B|}\sum_{l>0}\frac{1}{l!}[\partial^l B\ [AC]_l]_0\,. \qquad (19)\end{aligned}$$

In *OPEdefs* 3.1 these two last rules are applied when C is not a composite itself.

3 User's Guide

This section is intended as a user's guide to the package *OPEdefs* 3.1. Explicit examples are given for most operations. Note that *OPEdefs* 3.1 requires *Mathematica* 1.2 or later.

We introduce some special notations. Input for and output from *Mathematica* is written in `typeset` font. Input lines are preceded by "*In[n] :=*", and corresponding output statements by "*Out[n] =*", as in *Mathematica*.

As *OPEdefs* is implemented as a *Mathematica* package, it has to be loaded *before* any of its global symbols is used. Loading the package a second time will clear all previous definitions of operators and OPEs, as well as all stored intermediate results. Assuming that the package is located in the *Mathematica*-path, *e.g.* in your current directory, issue:

In[1] := `<<OPEdefs.m`

After loading *OPEdefs* into *Mathematica*, help for all the global symbols is provided using the standard help-mechanism, *e.g.* `?OPE`.

Now, you need to declare the operators that will be used. If you want to define bosonic operators `T` and `J[i]` (any index could be used), and fermionic operators `psi[i]`, the corresponding statements are:

```
In[2] :=    Bosonic[T, J[i_]]
In[3] :=    Fermionic[psi[i_]]
```

The order of the declarations fixes also the ordering of operators used by the program:

$$\texttt{T < J[1]' < J[1] < J[2] < J[i] < psi[1] < ...} \qquad (20)$$

By default, derivatives of an operator are considered "smaller" than the operator itself. This can be reversed using the global options `NOOrdering` (see below).

Finally, the non-regular OPEs between the basic operators have to be given. An OPE can be specified in two different ways.
The first way is by listing the operators that occur at the poles, the first operator in the list is the one at the highest non-zero pole, the last operator has to be the one at the first order pole, *e.g.* :

```
In[4] :=    OPE[T, T] = MakeOPE[{c/2 One, 0, 2T, T' }];
```

Note the operator `One` which specifies the unit-operator.
The second way is by giving the OPE as a Laurent series expansion, adding the symbol `Ord` which specifies the (implicit) arguments of the operators for which the OPE is defined[3]. The arguments for the operators can be any *Mathematica* expression.
Warning: it is important that the operators occuring as arguments of `OPE` in a definition should be given in standard order (20), otherwise wrong results will be generated.

The following statements define a $\widehat{SU(2)}_k$-Kač–Moody algebra:

```
In[5] :=    OPE[J[i_],J[i_]] :=
             MakeOPE[-k/2 (z-w)^-2+ Ord[z,w,0]]
In[6] :=    OPE[J[1],J[2]] =
             MakeOPE[J[3][w](z-w)^-1 +Ord[z,w,0]];
In[7] :=    OPE[J[2],J[3]] =
             MakeOPE[J[1][w](z-w)^-1 + Ord[z,w,0]];
In[8] :=    OPE[J[1],J[3]] =
             MakeOPE[-J[2][w](z-w)^-1 + Ord[z,w,0]];
```

In fact, with the above definitions, one has to use always the explicit indices $1, 2, 3$ for the currents J. If we would compute an OPE with current `J[i]`

---

[3] The first time you use this syntax, you may notice an unexpected delay. This is because *Mathematica* is loading the `Series` package.

where the index i is not $1, 2$ or 3, wrong results will be given. One can circumvent this peculiarity by reformulating the definitions.

A normal ordered product $[AB]_0$ is entered in the form `NO[A,B]`. Multiple composites can be entered using only one `NO` head, *e.g.* `NO[A,B,C]`. This input is effectively translated into `NO[A, NO[B, C]]`. All output is normal ordered with the same convention, *i.e.* from right to left (input can be in any order). Also, the operators in composites will always be ordered according to the standard order (20).

As an example, we can define the Sugawara energy-momentum tensor for $\widehat{SU(2)}_k$. The *Mathematica* output of an OPE is a list of the operators at the poles.

```
In[9] :=    Ts = -1/(k+2)(NO[J[1],J[1]]+
                 NO[J[2],J[2]]+NO[J[3],J[3]]);
In[10] :=   OPESimplify[OPE[Ts, J[1]]]
Out[10] =   << 2|| J[1] ||1|| J[1]' >>
```

Warning: when computing OPEs with composites, or when reordering composites, *OPEdefs* remembers by default some intermediate results. Thus, it is dangerous to change the definition of the basic OPEs after some calculations have been performed. For example, consider a constant a in an OPE. If calculations are performed after assigning a value to a, the intermediate results are stored with this value. Changing a afterwards will give wrong results.

The other globally defined functions available from the package are:

- `OPEOperator[operator_, parity_]` provides a more general way to declare an operator than `Bosonic` and `Fermionic`. The second argument is the parity of the operator such that $(-1)^{\texttt{parity}}$ is $+1$ for a boson, and -1 for a fermion. It can be a symbolic constant. This is mainly useful for declaring a *bc*-system of unspecified parity, or a Kač–Moody algebra based on a super-Lie algebra. In such cases, the operator can contain a named pattern:
  ```
  In[11] :=   OPEOperator[J[i_],parity[i]]
  ```
  If one wants to declare more operators, one can group each operator and its parity in a list:
  ```
  In[12] :=   OPEOperator[{b[i_],parity[i]},{c[i_],parity[i]}]
  ```
  See also `SetOPEOptions[ParityMethod, _]`.

- `OPEPole[n_][ope_]` gets a single pole term of an OPE:
  ```
  In[13] :=   OPEPole[2][Out[10]]
  ```

```
Out[13] = J[1]
```

`OPEPole[n_][A_,B_]` can also be used to compute only one pole term of an OPE:

```
In[14] :=  Factor[OPEPole[4][Ts, Ts]]
Out[14] =  (3 k One)/(2 (2 + k))
```

`OPEPole` can also give terms in the regular part of the OPE:

```
In[15] :=  OPEPole[-1][T, T]
Out[15] =  NO[T', T]
```

- `MaxPole[ope_]` gives the order of the highest pole in the OPE.
- `OPEParity[A]` returns an even (odd) integer of A is bosonic (fermionic).
- `OPESimplify[ope_, function_]` "collects" all terms in `ope` with the same operator and applies `function` on the coefficients. When no second argument is given, the coefficients are `Expand`ed.

  ```
  In[16] :=  OPESimplify[OPE[J[1], NO[J[2], J[1]]]]
  Out[16] =  << 2|| (1 - k/2) J[2] ||1||
                NO[J[1], J[3]] + J[2]' >>
  ```

  `OPESimplify[pole_, function_]` does the same simplifications on sums of operators.
- `OPEMap[function_, ope_]` maps `function` to all poles of `ope`.
- `GetCoefficients[expr_]` returns a list of all coefficients of operators in `expr` which can be (a list of) OPEs or poles.
- `OPEJacobi[op1_,op2_,op3_]` computes the Jacobi-identities (6) for the singular part of the OPEs of the three arguments. Due to the nature of eq. (6), the computing time will be smallest (in most cases) when `op1` $\leq$ `op2` $\leq$ `op3` in the order (20). Note that it is sufficient to check the Jacobi-identities with the generators of the OPA.

  The result of `OPEJacobi` is a double list of operators. It is generated by

  ```
  Table[OPEPole[n][A,OPEPole[m][B,C]] +
              corrections, {m, maxm},{n,maxn}]
  ```

  All elements of the list should be zero up to null operators for the OPA to be associative.
- `Delta[i_,j_]` is the Kronecker delta symbol δ_{ij}.

- `ClearOPESavedValues[]` clears all stored intermediate results, but not the definition of the operators and their OPEs. To clear everything, reload the package.

- `OPEToSeries[ope_]` converts an OPE to a Laurent series expansion in `z` and `w`. The arguments can be set to `x` and `y` with:
  In[17] := `SetOPEOptions[SeriesArguments, {x, y}]`

- `TeXForm[ope_]` gives TEXoutput for an OPE. The same arguments are used as in `OPEToSeries`.

- `OPESave[filename_]` (with `filename` a string between double quotes) saves the intermediate results that *OPEdefs* remembers to file (see the option `OPESaving` below).

- `SetOPEOptions` is a function to set the global options of the package. The current options are:

  - `SetOPEOptions[SeriesArguments, {arg1_, arg2_}]` : sets arguments to be used by `TeXForm` and `OPEToSeries`. One can use any Mathematica expression for `arg1` and `arg2`.
  - `SetOPEOptions[NOOrdering, n_]` : if `n` is negative, order higher derivatives to the left (default), if `n` is positive, order them to the right.
  - `SetOPEOptions[ParityMethod, 0|1]` : makes it possible to use operators of an unspecified parity. When the second argument is 0 (default), all operators have to be declared to be bosonic or fermionic. When the argument is 1, `OPEOperator` can be used with a symbolic parity. Note that in this case, powers of -1 are used to compute signs, which is slightly slower than the boolean function which is used by the first method.
    This option is not normally needed as the use of `OPEOperator` with a non-integer second argument sets this option automatically.
  - `SetOPEOptions[OPESaving, boolean_]` : if `boolean` evaluates to `True` (default), *OPEdefs* stores the intermediate results when computing OPEs of composites and when reordering composites. This option is useful if *Mathematica* runs short of memory in a

large calculation, or when computing with dummy indices[4].

- `SetOPEOptions[OPEMethod, method_]` : with the parameter `method` equal to `QuantumOPEs` enables normal OPE computations (default setting), while `ClassicalOPEs` enables Poisson bracket computations. Using this option implicitly calls `ClearOPESavedValues[]`.

4 Example : The conformal anomaly in superstring theory

We consider only one free boson field X and one free fermion field ψ because additional free fields will have exactly the same OPEs and commute with each other. We denote ∂X with `J` and ψ with `psi` (we normalise them such that they have a $+1$ in their OPEs. The ghosts are a fermionic b, c system (operators `b,c`) and a bosonic β, γ system (operators `B,G`). b has conformal dimension 2 and β has 3/2. It is now a trivial task to compute the conformal anomaly:

```
In[1] :=    <<OPEdefs.m
In[2] :=    Bosonic[J,B,G]; Fermionic[b,c,psi];
            OPE[J,J] = MakeOPE[{One, 0}];
            OPE[psi,psi] = MakeOPE[{One}];
            OPE[b,c] = MakeOPE[{One}];
            OPE[B,G] = MakeOPE[{One}];
            Tb = 1/2 NO[J,J]; Tf = -1/2 NO[psi,psi’];
            Tbc = -2 NO[b,c’] - NO[b’,c];
            TBG = 3/2 NO[B,G’] + 1/2 NO[B’,G];

In[3] :=    OPESimplify[OPE[Tb,Tb]]
Out[3] =    << 4|| One/2 ||3|| 0 ||2|| NO[J, J] ||1||
                  NO[J’, J] >>
In[4] :=    OPESimplify[OPE[Tf,Tf]]
Out[4] =    << 4|| One/4 ||3|| 0 ||2|| NO[psi’, psi] ||1||
                  NO[psi’’, psi]/2>>
In[5] :=    OPESimplify[OPE[Tbc,Tbc]]
Out[5] =    << 4|| -13 One ||3|| 0 ||2||
                  -4 NO[b, c’] - 2 NO[b’, c]||1||
                  -2 NO[b, c’’] - 3 NO[b’, c’] - NO[b’’, c] >>
```

---

[4] No mechanism to use dummy indices is built-in in *OPEdefs*. I wrote a separate package *Dummies* to handle this.

```
In[6] :=    OPESimplify[OPE[TBG,TBG] - MakeOPE[{2 TBG,TBG'}]
Out[6] =    << 4|| 11 One/2 ||3|| 0 ||2|| 0 ||1|| 0 >>
```

We see that each bosonic (fermionic) field will contribute a central charge 1 (1/2) to the total central charge of the theory. The b, c system contributes -26, and the β, γ system 11. This gives the well known relation for the critical dimensions of the bosonic string $D_b - 26 = 0$ and the superstring $3/2D_s - 26 + 11 = 0$. Moreover, we can easily verify that the energy–momentum tensors obey the Virasoro algebra .

The reader without experience in CFT is invited at this point to take out some time and compute the OPE for T_{BG}, for instance, by hand. Although this computation is rather trivial with *OPEdefs*, the same calculation was attempted in [6] using the mode–algebra. There it proved not to be possible to compute the Virasoro algebra automatically due to difficulties with the infinite sums in the normal ordered products.

5 Performance

In [3], a free field realization for $\widehat{B_2}$ level k using (bosonic) β, γ systems was constructed. In Table 1, we tabulate CPU times for computing an OPE of two of the currents, and the Sugawara tensor for this realization. The first time given in the table is the time for evaluating the statement after loading the package and defining the realization. The time between brackets is measured when the statement is repeated. Note that version 2.0 of *Mathematica*is roughly 1.4 times slower than version 1.2!

Table 1: CPU time for the computation of the OPE of the currents corresponding to the positive simple root of $\widehat{B_2}$ (statement 9) and the computation of the Sugawara tensor (statement 11) (see Ref. [3]) for *Mathematica* running on a PC 386 (25 Mhz).

Mathematica-version *OPEdefs*-version	1.2 2.0	1.2 3.1	2.0 3.1
`In[9]`	23.5 (4.5) s	14.9 (2.8) s	19.3 (3.8) s
`In[11]`	43.2 (11.6) s	31.3 (9.4) s	40.7 (12.1) s

6 How to get it, and the future

If you are interested in *OPEdefs*, you can get it by Email from the author. Please put a reference to [3] in your paper when you use it. Questions, remarks and improvements are welcome. Already in testing-phase is *OPEconf*, a package which enables you to work with (quasi-)primaries and conformal blocks.

7 Acknowledgements

Many improvements implemented in *OPEdefs* 3.0 are suggested by K. Hornfeck, who also helped me testing this version.

References

[1] F. Bais, P. Bouwknegt, M. Surridge, K. Schoutens, Nucl. Phys. **B304** (1988) 348.
A. Sevrin, W. Troost, A. Van Proeyen, P. Spindel, Nucl. Phys. **B311** 465 (1988).

[2] *Mathematica, A system for Doing Mathematics by Computer*, S. Wolfram, Addison-Wesley Publishing Company, Inc.

[3] K. Thielemans, Int. J. Mod. Phys. C Vol. **2**, No. 3, 787 (1991).

[4] K. Thielemans, *New computing techniques in Physics Research II*, proceedings of the Second International Workshop on Software Engineering, Artificial Intelligence and Expert Systems in High Energy and Nuclear Physics, ed. D. Perret-Gallix, World Scientific (1992).

[5] K. Thielemans, *An Algorithmic Approach to Operator Product Expansions, W-algebras and W-strings*, PhD thesis KU Leuven, june 1994.

[6] W.M. Seiler, *SUPERCALC, a REDUCE Package for commutator calculations*, Karlsruhe preprint KA-THEP-20/90.