

- Help For:

'BIGEBRA' - A Maple Package for Clifford and Grassmann Hopf Gebras

Version 1.01 -- designed for Maple 12

(Copyright (c) Rafal Ablamowicz * and Bertfried Fauser§, 1999 - 2008. All rights reserved.)

(*) Department of Mathematics, Box 5054
Tennessee Technological University
Cookeville, TN 38501 USA

Phone: (931) 372-3569, Fax: (931) 372-6353

E-mail: rablamowicz@tntech.edu

URL: <http://math.tntech.edu/rafal/> (home of this package)

(§) Fachbereich Physik, Universität Konstanz
Fach M678, 78457 Konstanz, Germany

Phone: +49 7531 883786, FAX: +49 7531 884266

E-mail: Bertfried.Fauser@uni-konstanz.de

URL: <http://clifford.physik.uni-konstanz.de/~fauser/>

Last revised: December 20, 2008 (BF & RA)

Calling Sequence:

function(args) (if the package was loaded using **with(Bigebra);**)
Bigebra:-function(args) (long form without loading the package)

- Description:

- The BIGEBRA package supplements the CLIFFORD package [Clifford](#) version 8 for Maple 8. If BIGEBRA is loaded using **with(Bigebra);** it loads automatically the CLIFFORD package. BIGEBRA patches the Maple `define/skeleton` and `define/multilinear` routines of Maples [define facility](#) to allow a correct implementation of the [tensor product](#).
- The main purpose of the BIGEBRA package is to allow computations in tensor products of Clifford and Grassmann algebras. For this purpose, a tensor product `&t` is defined which is linear with respect to all non-Clifford elements (constants). This allows to perform calculations in Grassmann/Clifford modules and Grassmann/Clifford bundles. Bi- and Hopf algebraic structures as co-units, co-products, switches etc. are employed. All structures of Grassmann Hopf algebra and Clifford biconvolution are implemented. However, using this device, Grassmann-Cayley algebras and bracket or Peano algebras are also supported. Especially the [meet](#) (of point fields and of plane fields in Plücker coordinatization) is implemented here in a very effective way. The [join](#) (of point fields) is implemented by the wedge of the CLIFFORD package.

- There are several functions which allow the usage of linear operators given in a matrix representation w.r.t. the Grassmann basis. Such operators can act on a single tensor slot, i.e. they are from $\text{End } \wedge V$, or on two adjacent tensor slots, i.e. they are from $\text{End } (\wedge V \ \&t \ \wedge V)$, where $\wedge V$ is the space underlying the Grassmann algebra.
- The BIGEBRA package provides a facility to solve tangle equations [6] for linear operators applied to internal lines of the tangle if the tangle equation has n ingoing and one outgoing line ($n \rightarrow 1$ mapping). This simplifies e.g. the search for Clifford antipodes.
- The Clifford product can be defined in terms of Hopf algebras [8]. BIGEBRA uses the Clifford product of CLIFFORD **cmul** which internally uses by default the cmulRS subroutine based on the Rota-Stein cliffordization technique and Hopf algebraic methods. The Clifford co-product is derived from co-cliffordization in the same way.
- The Clifford co-product needs an additional bilinear form, called co-scalarproduct, which has to be defined as the global $\text{dim}_V \times \text{dim}_V$ matrix **BI**. The dimension has to be specified using the global variable **dim_V** of CLIFFORD. The Clifford co-product needs an *initialization* which is done by calling once the function [make_BI_Id](#). Some **caution** is needed here, since dim_V is set to the maximal value 9 by CLIFFORD and the initialization may take very long in this case, so that dim_V should be set to a smaller value if possible.
- The BIGEBRA package makes use of some global variables, which are stored in the table `_CLIENV`. Currently in use are:
 - `_CLIENV[_SILENT]`, default = unassigned. If 'true' it suppresses lots of startup output.
 - `_CLIENV[_fakenow]`, a flag used to detect if BIGEBRA was already loaded. Needed for patching define.
 - `_CLIENV[_QDEF_PREFACTOR]`, default = -1. Puts q-deformation into the Grassmann coproduct, (beware: ONLY there for now, the q-busines is not yet officially supported and not well tested).
- BIGEBRA can also serve to provide the user a possibility to define various multilinear functions, i.e. tensor products over arbitrary rings, see [define](#).
- The help pages of BIGEBRA are part of the same Maple database file (maple.hdb) which contains help pages for 'CLIFFORD' and should be located in a directory in Maple's `libname[1]` variable. BIGEBRA is supposed to merge with CLIFFORD in a forthcoming version for Maple ver. 6/7.
- BIGEBRA was already successfully used in deriving mathematically and physically relevant results [1,2,3]. Some references are added to provide information about Clifford Hopf gebras.

Literature:

- [1] B. Fauser, On the Hopf-algebraic origin of Wick normal-ordering, J. Phys.A. Math. Gen. 34:105-115, 2001.
- [2] B. Fauser, Z. Oziewicz, Clifford Hopf gebra for two dimensional space, Miscellanea Algebraicae, 2(1):31-42, 2001.
- [3] B. Fauser, A treatise on quantum Clifford algebras, Habilitation, Uni. Konstanz, Jan. 2003.

- [4] F.D. Grosshans, G.-C. Rota, J.A. Stein, Invariant theory and superalgebras, In Conference Board of the Mathematical Sciences Regional Conference Series in Mathematics, number 69, pages i-xxi, 1-80. American Mathematical Society, 1987. M. Barnabei, A. Brini, G.-C. Rota, On the exterior calculus of invariant theory, J. Alg. 96:120-160, 1985.
- [5] V. Lyubashenko, Tangles and Hopf algebras in braided categories, J. Pure Apl. Alg. 98:245-278, 1995.
- [6] J.W. Milnore, J.C. Moore, On the structure of Hopf algebras, Ann. Math., 81:211-264, 1965. M.E. Sweedler, Hopf algebras, W.A. Benjamin, INC, New York, 1969. E. Abe, Hopf algebras, Cambridge Univ. Press, Cambridge, 1980. N. Bourbaki, Elements of Mathematics, Algebra I, Chapters 1-3, Springer-Verlag, Berlin 1989.
- [7] Z. Oziewicz, Clifford Hopf algebra and biuniversal Hopf algebra, Czech. J. Phys. 47(12): 1267-1274, 1997. Z. Oziewicz, Guest Editor's Note, Int. J. Theor. Phys. 40(1):1-13, 2001. Z. Oziewicz, Operad of graphs, convolution and Hopf algebra, Contemp. Math. submitted.
- [8] G.-C. Rota, J.A. Stein, Plethystic Hopf algebras, Proc. Natl. Acad. Sci USA, 91:13057-13061, 1994.
- [9] A. Zaddach, Grassmann Algebra in der Geometrie mit Seitenblicken auf verwandte Strukturen, Bi.-Wissenschaftsverlag, Mannheim, 1994

[-] Load Bigebra in the following way, Clifford has to be loaded manually!

```
[ You can increase the verbosity level of Bigebra setting infolevel[Bigebra]=3 or higher.
[ > restart:with(Clifford):infolevel(Bigebra)=3:with(Bigebra):
[ Increase verbosity by infolevel[function]=val -- use online help > ?Bigebra[h
[ elp]
[ To initialize the Clifford coproduct type:
[ > dim_V:=2:
[   BI:=linalg[matrix](dim_V,dim_V,[a,b,c,d]);
[   make_BI_Id();
[
[ 
$$BI := \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

[ Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
[ clude &C and &C[K]. Type ?cliprod for help.
[ (Id &t Id) + a (e1 &t e1) + c (e2 &t e1) + b (e1 &t e2) + d (e2 &t e2)
[   + (c b - d a) (e1we2 &t e1we2)
[ BI is the dim_V x dim_V matrix of the co-scalarproduct on co-one-vectors, from which the
[ Clifford co-product '&cco' is derived by Rota-Stein co-cliffordization, [2,7,8]. The tensor product
[ '&t' is already defined and ready to use:
[ > &t(e1,&t(e2,e3),e4); ## associativity, i.e. drop 'parentheses'
[   &t(e1, e2, e3, e4)
[ > &t(a*e1+sin(theta)*e3,b*e3-1/x*e1); ## multilinearity
[   
$$a b (e1 \&t e3) - \frac{a (e1 \&t e1)}{x} + \sin(\theta) b (e3 \&t e3) - \frac{\sin(\theta) (e3 \&t e1)}{x}$$

[ >
[ >
```

— Alphabetic listing of available procedures in 'BIGEBRA':

- [&cco](#) -- Clifford co-product on
- [&gco](#) -- The Grassmann co-product w.r.t. the [wedge](#) product.
- [&gco_d](#) -- dotted Grassmann co-product acting on the undotted wedge basis.
- [&gpl_co](#) -- Grassmann-Plücker co-product acting on hyperplanes in Plücker coordinatization.
- [&map](#) -- &map maps a product, i.e. a Clifford valued function of two [Clifford polynoms](#) (a 2->1 mapping) onto two adjacent slots of a [tensor](#).
- [&t](#) -- The tensor product defined in BIGEBRA during loading of the package.
- [&v](#) -- Defined the vee-product, i.e. the [meet](#).
- [tensor polynoms.](#)
- [bracket](#) -- Defines a bracket in the sense of a Peano space [8].
- `cco_monom` -- internal use only.
- [contract](#) -- Contract maps a [cliscalar](#) valued function of two [Clifford polynoms](#) onto two adjacent [tensor](#) slots.
- [define](#) -- Maple 6 'define' still has bugs, so 'define' had to be replaced by a patched code.
New option: give a domain for k-multilinearity.
- [drop_t](#) -- Drops the tensor sign &t in expressions like &t(e1), projects on the first argument in &t(p1,p2,...).
- `eps` -- no longer supported.
- [EV](#) -- EV is the evaluation of a multi-co-vector on a multivector. Multi-co-vectors are described currently (we are sorry to say) by the same Grassmann basis elements. The user is **responsible** to take care in which tensor slot co-vectors reside.
- [gantipode](#) -- Applies the Grassmann antipode to a [tensor](#) slot.
- [gco_unit](#) -- The Grassmann Hopf algebra co-unit.
- [gswitch](#) -- Graded switch of two adjacent slots of a [tensor](#).
- [help](#) -- This page.
- [linop](#) -- Linop defines a linear operator acting on the Grassmann algebra, having a $2^{\dim_V} \times 2^{\dim_V}$ co-contravariant matrix representing it.
- [linop2](#) -- Linop2 defines a linear operator acting on a tensor product of rank two of the Grassmann algebra, having a $4^{\dim_V} \times 4^{\dim_V}$ co-contravariant matrix representing it.

- [list2mat](#) -- List2mat computes from two lists of elements from V^\wedge which are connected as source and target of an linear operator a (possibly unfaithful reducible) matrix representation.
- [list2mat2](#) -- List2mat2 computes from two lists of elements from V^\wedge & V^\wedge which are connected as source and target of an linear operator a (possibly unfaithful reducible) matrix representation.
- [make BI Id](#) -- Initialization routine for the [Clifford co-product](#).
- [&map](#) -- &map maps a product, i.e. a Clifford valued function of two [Clifford polynoms](#) (a 2->1 mapping) onto two adjacent slots of a [tensor](#).
- [mapop](#) -- Mapop applies a linear operator (element of $\text{End } V$) defined by [linop](#) onto **one** single slot of a [tensor](#).
- [mapop2](#) -- Mapop2 applies a linear tensor-operator (element of $\text{End } V \ \&t \ V$) defined by [linop2](#) onto **two** slots of a [tensor](#).
- [meet](#) -- The meet is equivalent to the [&v-\(vee\)-product](#).
- [op2mat](#) -- Op2mat returns a (possibly unfaithful reducible) matrix representation in V^\wedge of a linear operator given as argument.
- [op2mat2](#) -- Op2mat2 returns a (possibly unfaithful reducible) matrix representation in $V^\wedge \ \&t \ V^\wedge$ of a linear operator given as argument.
- [pairing](#) -- A pairing of two [Clifford polynoms](#).
- [peek](#) -- Peek gets a [Clifford polynomial](#) from a [tensor](#) at a certain position.
- [poke](#) -- Poke puts a [Clifford polynomial](#) into a [tensor](#) at a certain position.
- [remove_eq](#) -- Helper function, which allows to remove trivial equations if tangle equations are solved manually.
- [switch](#) -- Switch two adjacent slots of a [tensor](#) (Just a swap).
- [tcollect](#) -- Tcollect collects [cliscalar](#) coefficients in a [tensor](#) expression.
- [tsolve1](#) -- Tsolve1 solves tangle equations with n ingoing and one outgoing line (n-> 1 mappings). It has the ability to solve for operators applied to internal lines of the tangle. Such operators can be defined algebraically or using [linop](#) and [linop2](#).
- [VERSION](#) -- Displays information about the current version of BIGEBRA.

New Types in 'BIGEBRA':

- [type/tensobasmonom](#) - A tensor basis monom having no prefactor.
- [type/tensormonom](#) - A tensor monom which may have a prefactor of type [cliscalar](#).
- [type/tensorpolynom](#) - A sum of tensor monoms.

 See Also: [Clifford:-setup](#), [Clifford:-version](#), [Bigebra:-VERSION](#)

(c) Copyrights 1999-2009, by Rafal Ablamowicz & Bertfried Fauser, all rights reserved.
Last modified: December 20, 2008/BF/RA.

- Function: Bigebra:-VERSION - prints information about Bigebra and the version

Calling Sequence:

VERSION()

Parameters:

- none.

Output:

- none.

- Description:

- VERSION() displays information about the [Bigebra package](#)

- Examples:

```
> restart:with(Bigebra) :
```

```
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h  
elp]
```

```
VERSION is a function hence parentheses are needed after the name!
```

```
> VERSION() ;
```

```
<=====>
```

```
°°Bi-Gebra Package VERSION 1.01 for Clifford version 11°°  
by Rafal Ablamowicz($) and Bertfried Fauser(*)  
(c) Dec-16-99 / Nov-16-2002 / Mar-3-1007 / Aug-14-2007 / Dec-20-2007 / Jun  
e-20-2008 / December-20-2008  
Available from http://math.tntech.edu/rafal/
```

```
($) Department of Mathematics, Box 5054  
Tennessee Technological University  
Cookeville, TN 38505, U.S.A.  
Email: rablamowicz@tntech.edu  
URL: http://math.tntech.edu/rafal/
```

```
(*) Universit"at Konstanz  
Fachbereich Physik, Fach M678  
78457 Konstanz, Germany  
Email: Bertfried.Fauser@uni-konstanz.de  
URL: http://clifford.physik.uni-konstanz.de/~fauser/
```

```
Online help available with:  
> ?Bigebra  
or use 'help' menu and search for topics
```

```
Copyright (c) Rafal Ablamowicz, Bertfried Fauser 1999-2009.  
All rights reserved. See also > ?Bigebra[help]
```

```
BUG-REPORTS to Bertfried Fauser
```

[<----->

[>

[>

- See Also: [Bigebra helppage](#)

(c) Copyright 1999-2009, by Rafal Ablamowicz & Bertfried Fauser, all rights reserved.

Last modified: December 20, 2008 /BF/RA.

- Function: `&cco` - Clifford co-product

Calling Sequence:

```
t1 = &cco(p1,i)
```

```
t1 = &cco(c1)
```

Parameters:

- `p1` : a tensorpolynom (element of ``type/tensorpolynom``) of rank not less than `i` in each factor
`i` : the slot number (first slot from the left is 1) on which the co-product acts
- `c1` : a Clifford polynom (element of one of these types: ``type/clipolynom``, ``type/climon``, ``type/clibasmon``)

Output:

- `t1` : a tensor polynom

Global variables:

- `BI_Id` - set by `make_BI_Id`
- `dim_V` - the dimension of the one-vector space `V`

WARNING:

The Clifford co-product takes only one 'factor' (and one parameter), the **infix form** makes no sense with this function and yields *unpredictable nonsense*.

- Description:

- Like the Clifford product, Clifford co-product needs a bilinear form defined on the base space of the Grassmann algebra. In the case of the co-product, this form is tied to co-one-vectors, so it is called co-scalar product. Since we deal with finite dimensional spaces, the dimension of the covector space is `dim_V`, the same as for the vector space `V` used by `CLIFFORD`. Hence we use the *global variable* `dim_V`, which has to be assigned. The matrix of the Clifford co-product w.r.t. the co-one-vector basis (in abuse of language also denoted by `e1`, see remarks in `EV`) is stored in `BI`. The elements of `BI` can be assigned freely, without any restrictions or relations to the matrix of the Clifford scalar product `B`. `BI` can be singular or non symmetric or even zero, in which case the Clifford co-product reduces to the `Grassmann co-product`.
- The Clifford co-product is based on Rota-Stein co-cliffordization `\[2,3,7,8\]`. This is the categorical dual of the Rota-Stein cliffordization of the Grassmann algebra which leads to the Clifford co-product. In Sweedler notation, the formula for Clifford co-product is:

$$\Delta_{\&c}(x) = (\text{wedge } \&t \text{ wedge})(\text{Id } \&t \text{ BI}_{(1)} \&t \text{ BI}_{(2)} \&t \text{ Id})\Delta(x)$$

where Δ [c] is the Clifford co-product, Δ is the [Grassmann co-product](#). The factor BI_Id = BI_(1) &t BI_(2) (in fact internally represented by a list of type [[sign,BI_(1),BI_(2)] , ...]) has to be precomputed using [make BI_Id](#).

- The Clifford co-product is associative by construction, but it is not (graded) co-commutative.
- Clifford co-products lead to non-connected co-modules, see [Milnore and Moore](#). This is an important difference with Grassmann co-products [3,6].
- There is a sort of asymmetry in the BIGEBRA package, since the Clifford product is not (yet) computed using Rota-Stein cliffordization. The Clifford product is simply taken from CLIFFORD. This may cause problems if one tries to compute with q-deformed Clifford co-products. However, q-deformed Grassmann co-products are available by setting the global variable _CLIENV[_QDEF_PREFACTOR] e.g. to -q.

TO DO:

- The Clifford product has to be based on Rota-Stein cliffordization and a q-wedge product has to be created.

Examples:

```
> restart:bench:=time():with(Clifford):with(Bigebra):

Increase verbosity by infolevel[`function`=val -- use online help > ?Bigebr[h
elp]
> dim_V:=2:
BI:=linalg[matrix](dim_V,dim_V,[a,b,c,d]): #co-scalarproduct
m1:=make_BI_Id(): #remember this
result in m1
Climon has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
The Clifford co-product of Clifford polynomials needs no slot index:
> c1:= &cco(e1);
c2:= &cco(&t(e1),1); # the same, &t(e1) = e1
c3:= &cco(&t(e1),2); # the same, the slot is ignored here

c1 := (Id &t e1) - b (e1 &t e1we2) - d (e2 &t e1we2) + (e1 &t Id) + c (e1we2 &t e1)
      + d (e1we2 &t e2)
c2 := (Id &t e1) - b (e1 &t e1we2) - d (e2 &t e1we2) + (e1 &t Id) + c (e1we2 &t e1)
      + d (e1we2 &t e2)
c3 := (Id &t e1) - b (e1 &t e1we2) - d (e2 &t e1we2) + (e1 &t Id) + c (e1we2 &t e1)
      + d (e1we2 &t e2)

Reduction to the Grassmann co-product is obtained by letting all parameters of the
co-scalarproduct go to zero:
> subs(a=0,b=0,c=0,d=0,m1); ## &cco(Id) -> &gco(Id)
```

```

&gco (Id) ;
subs (a=0 ,b=0 ,c=0 ,d=0 ,c2) ; ## &cco (e1) -> &gco (e1)
&gco (e1) ;

```

$$\begin{aligned}
& Id \ \&t \ Id \\
& Id \ \&t \ Id \\
& (Id \ \&t \ e1) + (e1 \ \&t \ Id) \\
& (Id \ \&t \ e1) + (e1 \ \&t \ Id)
\end{aligned}$$

One can either change the co-product, or substitute the general parameters to get the result for another co-scalarproduct.

```

> BI:=linalg[matrix] (dim_V ,dim_V , [1 ,b ,b ,1]) : #new
co-scalarproduct
make_BI_Id() :

```

Compute once more &cco(e1) and compare it with the substituted result:

```

> c4:=&cco (e1) ;
c5:=subs (a=1 ,b=b ,c=b ,d=1 ,c1) ;

```

$$\begin{aligned}
c4 := & (Id \ \&t \ e1) - b (e1 \ \&t \ e1we2) - (e2 \ \&t \ e1we2) + (e1 \ \&t \ Id) + b (e1we2 \ \&t \ e1) \\
& + (e1we2 \ \&t \ e2)
\end{aligned}$$

$$\begin{aligned}
c5 := & (Id \ \&t \ e1) - b (e1 \ \&t \ e1we2) - (e2 \ \&t \ e1we2) + (e1 \ \&t \ Id) + b (e1we2 \ \&t \ e1) \\
& + (e1we2 \ \&t \ e2)
\end{aligned}$$

```

> &cco (Id) ;
subs (a=1 ,b=b ,c=b ,d=1 ,m1) ;

```

$$\begin{aligned}
& (Id \ \&t \ Id) + (e1 \ \&t \ e1) + b (e2 \ \&t \ e1) + b (e1 \ \&t \ e2) + (e2 \ \&t \ e2) \\
& + (b^2 - 1) (e1we2 \ \&t \ e1we2)
\end{aligned}$$

$$\begin{aligned}
& (Id \ \&t \ Id) + (e1 \ \&t \ e1) + b (e2 \ \&t \ e1) + b (e1 \ \&t \ e2) + (e2 \ \&t \ e2) \\
& + (b^2 - 1) (e1we2 \ \&t \ e1we2)
\end{aligned}$$

Reduction to the Grassmann co-product as a test:

```

> subs (a=0 ,b=0 ,c=0 ,d=0 ,m1) ; ## &cco (Id) -> &gco (Id)
&gco (Id) ;
subs (a=0 ,b=0 ,c=0 ,d=0 ,c2) ; ## &cco (e1) -> &gco (e1)
&gco (e1) ;

```

$$\begin{aligned}
& Id \ \&t \ Id \\
& Id \ \&t \ Id \\
& (Id \ \&t \ e1) + (e1 \ \&t \ Id) \\
& (Id \ \&t \ e1) + (e1 \ \&t \ Id)
\end{aligned}$$

Co-associativity of the Clifford co-product:

```
> &cco (&cco (&t (e1) , 1) , 1) ;
&cco (&cco (&t (e1) , 1) , 2) ;
```

```
-&t(e2, Id, elwe2) + b &t(elwe2, Id, el) + b &t(Id, elwe2, el) - &t(el, elwe2, elwe2)
- &t(Id, e2, elwe2) + &t(Id, el, Id) + &t(Id, Id, el) + (b^2 - 1) &t(elwe2, elwe2, el)
+ &t(el, el, el) + 2 b &t(el, e2, el) + &t(el, e2, e2) + &t(Id, elwe2, e2)
- b &t(el, elwe2, Id) + &t(e2, e2, el) - &t(e2, el, e2) + &t(elwe2, Id, e2)
+ &t(elwe2, e2, Id) + &t(el, Id, Id) + b^2 &t(el, elwe2, elwe2) - &t(e2, elwe2, Id)
- b &t(el, Id, elwe2) - b^2 &t(elwe2, el, elwe2) + &t(elwe2, el, elwe2)
- b &t(Id, el, elwe2) + b &t(elwe2, el, Id)
-&t(e2, Id, elwe2) + b &t(elwe2, Id, el) + b &t(Id, elwe2, el) - &t(Id, e2, elwe2)
+ &t(Id, el, Id) + &t(Id, Id, el) + &t(el, el, el) + 2 b &t(el, e2, el) + &t(el, e2, e2)
+ &t(Id, elwe2, e2) - b &t(el, elwe2, Id) + &t(e2, e2, el) - &t(e2, el, e2)
+ &t(elwe2, Id, e2) + b^2 &t(elwe2, elwe2, el) - &t(elwe2, elwe2, el)
+ &t(elwe2, e2, Id) + &t(el, Id, Id) + (b^2 - 1) &t(el, elwe2, elwe2) - &t(e2, elwe2, Id)
- b &t(el, Id, elwe2) - b^2 &t(elwe2, el, elwe2) + &t(elwe2, el, elwe2)
- b &t(Id, el, elwe2) + b &t(elwe2, el, Id)
```

```
> simplify(tcollect(%-%)) ;
```

0

Note however that acting on different slots of the same tensor gives different answers:

```
> res1 := &cco (&t (e1, e2) , 1) ;
res2 := &cco (&t (e1, e2) , 2) ;
print(`res1 - res2 = 0 is ` , evalb(tcollect(res1-res2)=0)) ;
```

```
res1 := &t(Id, el, e2) - b &t(el, elwe2, e2) - &t(e2, elwe2, e2) + &t(el, Id, e2)
+ b &t(elwe2, el, e2) + &t(elwe2, e2, e2)
```

```
res2 := &t(el, Id, e2) + &t(el, el, elwe2) + b &t(el, e2, elwe2) + &t(el, e2, Id)
- &t(el, elwe2, el) - b &t(el, elwe2, e2)
```

res1 - res2 = 0 is ,false

If the index is not in the range of the tensor slots, an error occurs so the user has to account for that.

```
> &cco (&t (e1, e2) , 3) ; ####<<<<-- Intended error
Error, (in Bigebra:-&cco) invalid subscript selector
```

```
> printf("Worksheet took %f seconds to compute on AMD Athlon
2700+ 1GB RAM machine\n", time() - bench) ;
```

```
Worksheet took 1.297000 seconds to compute on AMD Athlon 2700+ 1GB RAM machine
```

```
>
```

[>

 **See Also:** [Bigebra:-`&gco`](#), [Bigebra:-`&t`](#), [Bigebra:-drop_t](#)

(c) Copyright 1999-2009, by Rafal Ablamowicz & Bertfried Fauser, all rights reserved.

Last modified: December 20, 2008 /BF/RA.