

- Function: Cliplus:-RCbig - extends the right contraction procedure 'RC' from 'CLIFFORD'

Calling Sequence:

```
RCbig(p1,p2);  
RCbig(p1,p2,name);
```

Parameters:

p1, p2 - any two Clifford polynomials expressed in Grassmann or Clifford basis
name - (optional) parameter of type 'name', 'symbol', 'matrix', or 'array', or
'&*' (numeric, {name,symbol,matrix,array})

- Description:

- This procedure extends procedure [Clifford:-RC](#) from 'CLIFFORD'. Recall, that RC(u,v) was a valid input for 'RC' provided u and v were polynomials in Cl(V,B) expressed in Grassmann basis, that is, expressions of type, [`type/clibasmon`](#), [`type/climon`](#), or [`type/clipolynom`](#). For completeness, procedure 'RC' was also accepting [`type/cliscalar`](#) for u and v.
- After loading 'Cliplus', procedure 'RC' will have the same properties as 'RC' plus the additional versatility afforded by the procedure 'RCbig'. That is, it can now accept polynomial expressions for u and v that contain monomial terms of [`type/cliprod`](#), that is the unevaluated Clifford product '&C'. Notice, that when 'Cliplus' is loaded, definitions of [`type/climon`](#) and [`type/clipolynom`](#) are extended to include monomial terms with expressions '&C'.
- NOTE: When using &C with an optional index, enclose &C in left quotes as in '&C'[K].
- When optional parameter of type 'name' is used, then it replaces B in computations. See examples below.

- Examples:

```
[ > restart:with(Clifford) :
```

```
[ Example 1: Procedure 'RC' gives the right-contraction in the Clifford algebra Cl(B) of any element u by any element v from the right, that is,  $RC(u,v) = u \lfloor v$  in Cl(V,B).
```

```
[ > u:=2*e1we3+e4+Id;v:=2*e1we2we3+e1we2;  
RC(u,v);
```

$$u := 2 e1we3 + e4 + Id$$

$$v := 2 e1we2we3 + e1we2$$

```
Cliffplus has been loaded. Definitions for type/climon and type/clipolynom now include &C and &C[K]. Type ?cliprod for help.
```

$$2 B_{3,1} B_{1,2} Id - 2 B_{1,1} B_{3,2} Id$$

```
[ It is possible to use 'RC' with an optional parameter of type 'name':
```

```
[ > RC(u,v,K);
```

$$2 K_{3,1} K_{1,2} Id - 2 K_{1,1} K_{3,2} Id$$

```
[ For example, it is well known that the Clifford product  $cmul(v,x) = v \lfloor x + wedge(v,x)$  where v
```

is any element in $Cl(V,B)$, and x is a 1-vector in $Cl(V,B)$:

```
> x:=2*e1-2*e2+e3;
```

$$x := 2 e_1 - 2 e_2 + e_3$$

```
> out1:=cmul[B](v,x);
```

$$\begin{aligned} out1 := & (-2 B_{2,2} + 2 B_{2,1} + B_{2,3}) e_1 - (B_{1,3} + 2 B_{1,1} - 2 B_{1,2}) e_2 \\ & + 2 (2 B_{3,1} - 2 B_{3,2} + B_{3,3}) e_1 e_2 - 2 (-2 B_{2,2} + 2 B_{2,1} + B_{2,3}) e_1 e_3 \\ & + 2 (B_{1,3} + 2 B_{1,1} - 2 B_{1,2}) e_2 e_3 + e_1 e_2 e_3 \end{aligned}$$

```
> out2:=RC(v,x,B) + wedge(v,x);
```

$$\begin{aligned} out2 := & 4 B_{3,1} e_1 e_2 - 4 B_{2,1} e_1 e_3 + 4 B_{1,1} e_2 e_3 + 2 B_{2,1} e_1 - 2 B_{1,1} e_2 - 4 B_{3,2} e_1 e_2 \\ & + 4 B_{2,2} e_1 e_3 - 4 B_{1,2} e_2 e_3 - 2 B_{2,2} e_1 + 2 B_{1,2} e_2 + 2 B_{3,3} e_1 e_2 - 2 B_{2,3} e_1 e_3 \\ & + 2 B_{1,3} e_2 e_3 + e_1 e_2 e_3 + B_{2,3} e_1 - B_{1,3} e_2 \end{aligned}$$

```
> simplify(out1-out2);
```

$$0$$

We can repeat the above computations for a different form. For example, the Clifford product $cmul[H](v,x)$ in $Cl(V,H)$ can be computed as:

```
> x:=2*e1-2*e2+e3;
```

$$x := 2 e_1 - 2 e_2 + e_3$$

```
> out1:=cmul[H](v,x);
```

$$\begin{aligned} out1 := & (-2 H_{2,2} + 2 H_{2,1} + H_{2,3}) e_1 - (H_{1,3} + 2 H_{1,1} - 2 H_{1,2}) e_2 \\ & + 2 (2 H_{3,1} - 2 H_{3,2} + H_{3,3}) e_1 e_2 - 2 (-2 H_{2,2} + 2 H_{2,1} + H_{2,3}) e_1 e_3 \\ & + 2 (H_{1,3} + 2 H_{1,1} - 2 H_{1,2}) e_2 e_3 + e_1 e_2 e_3 \end{aligned}$$

```
> out2:=RC(v,x,H) + wedge(v,x);
```

$$\begin{aligned} out2 := & 4 H_{3,1} e_1 e_2 - 4 H_{2,1} e_1 e_3 + 4 H_{1,1} e_2 e_3 + 2 H_{2,1} e_1 - 2 H_{1,1} e_2 - 4 H_{3,2} e_1 e_2 \\ & + 4 H_{2,2} e_1 e_3 - 4 H_{1,2} e_2 e_3 - 2 H_{2,2} e_1 + 2 H_{1,2} e_2 + 2 H_{3,3} e_1 e_2 - 2 H_{2,3} e_1 e_3 \\ & + 2 H_{1,3} e_2 e_3 + e_1 e_2 e_3 + H_{2,3} e_1 - H_{1,3} e_2 \end{aligned}$$

```
> simplify(out1-out2);
```

$$0$$

Example 2: 'RC' expects its arguments to be entered in Grassmann basis. Since the unevaluated Clifford basis can also be used in $Cl(V,B)$ instead of the Grassmann basis, we need to load package 'Cliplus' which contains procedure 'RCbig'. 'RCbig' is used internally by 'CLIFFORD' to compute with Clifford polynomials rather than Grassmann polynomials (of `type/clipolynom`). Conversions from one basis to the other can be done with procedures [Cliplus:-clieval](#) and [Cliplus:-cliexpand](#) as follow:

```
> restart:with(Clifford):with(Cliplus);
```

```
v:=2*e1we3+e4+Id;u:=2*e1we2we3+e1we2;
```

Cliplus has been loaded. Definitions for `type/climon` and `type/clipolynom` now include `&C` and `&C[K]`. Type `?cliprod` for help.

[*LCbig, RCbig, clibasis, clieval, cliexpand, climul, clirev, dottedcbasis, dwedge,*

makealiases]

$$v := 2 \text{elwe3} + e4 + \text{Id}$$

$$u := 2 \text{elwe2we3} + \text{elwe2}$$

> 'u'=u; #element u, as defined above, in Grassmann basis
'v'=v; #element v, as defined above, in Grassmann basis

$$u = 2 \text{elwe2we3} + \text{elwe2}$$

$$v = 2 \text{elwe3} + e4 + \text{Id}$$

> uu:=cliexpand(u,K); #u converted to Clifford basis with
'cliexpand'
vv:=cliexpand(v,K); #v converted to Clifford basis with
'cliexpand'

$$uu := 2 \&C_K(e1, e2, e3) - 2 K_{2,3} e1 + 2 K_{1,3} e2 - 2 K_{1,2} e3 + \&C_K(e1, e2) - K_{1,2} \text{Id}$$

$$vv := 2 \&C_K(e1, e3) - 2 K_{1,3} \text{Id} + e4 + \text{Id}$$

> clieval(uu); #uu converted back to Grassmann basis gives the
original u
clieval(vv); #vv converted back to Grassmann basis gives the
original u

$$2 \text{elwe2we3} + \text{elwe2}$$

$$2 \text{elwe3} + e4 + \text{Id}$$

[One can now apply 'RC' to uu and vv:

> out3:=RC(uu,vv,K);

$$\begin{aligned} \text{out3} := & 4 K_{3,1} K_{2,3} e1 - 4 K_{3,1} K_{1,3} e2 - 4 K_{2,1} K_{3,3} e1 + 4 K_{2,1} K_{1,3} e3 + 4 K_{1,1} K_{3,3} e2 \\ & - 4 K_{1,1} K_{2,3} e3 + 2 K_{3,4} \&C_K(e1, e2) - 2 K_{3,4} K_{1,2} \text{Id} - 2 K_{2,4} \&C_K(e1, e3) + 2 K_{2,4} K_{1,3} \text{Id} \\ & + 2 K_{1,4} \&C_K(e2, e3) - 2 K_{1,4} K_{2,3} \text{Id} + 2 \&C_K(e1, e2, e3) - 2 K_{2,3} e1 + 2 K_{1,3} e2 \\ & - 2 K_{1,2} e3 + 2 K_{2,1} K_{1,3} \text{Id} - 2 K_{1,1} K_{2,3} \text{Id} + K_{2,4} e1 - K_{1,4} e2 + \&C_K(e1, e2) - K_{1,2} \text{Id} \end{aligned}$$

[This result, when converted back to the Grassmann basis should give:

> out4:=RC(u,v,K);

$$\begin{aligned} \text{out4} := & 4 K_{3,1} K_{2,3} e1 - 4 K_{3,1} K_{1,3} e2 - 4 K_{2,1} K_{3,3} e1 + 4 K_{2,1} K_{1,3} e3 + 4 K_{1,1} K_{3,3} e2 \\ & - 4 K_{1,1} K_{2,3} e3 + 2 K_{3,4} \text{elwe2} - 2 K_{2,4} \text{elwe3} + 2 K_{1,4} e2\text{we3} + 2 \text{elwe2we3} \\ & + 2 K_{2,1} K_{1,3} \text{Id} - 2 K_{1,1} K_{2,3} \text{Id} + K_{2,4} e1 - K_{1,4} e2 + \text{elwe2} \end{aligned}$$

[so let's convert back 'out3' to Grassmann basis and compare with out4:

> out5:=clieval(out3);

$$\begin{aligned} \text{out5} := & 4 K_{3,1} K_{2,3} e1 - 4 K_{3,1} K_{1,3} e2 - 4 K_{2,1} K_{3,3} e1 + 4 K_{2,1} K_{1,3} e3 + 4 K_{1,1} K_{3,3} e2 \\ & - 4 K_{1,1} K_{2,3} e3 + 2 K_{3,4} \text{elwe2} - 2 K_{2,4} \text{elwe3} + 2 K_{1,4} e2\text{we3} + 2 \text{elwe2we3} \\ & + 2 K_{2,1} K_{1,3} \text{Id} - 2 K_{1,1} K_{2,3} \text{Id} + K_{2,4} e1 - K_{1,4} e2 + \text{elwe2} \end{aligned}$$

> out5-out4;

0

[Now let's see the mixed input:

[> **out6:=RC(u,vv,K);**

$out6 := 4 K_{3,1} K_{2,3} e1 - 4 K_{3,1} K_{1,3} e2 - 4 K_{2,1} K_{3,3} e1 + 4 K_{2,1} K_{1,3} e3 + 4 K_{1,1} K_{3,3} e2$
 $- 4 K_{1,1} K_{2,3} e3 + 2 K_{3,4} \&C_K(e1, e2) - 2 K_{3,4} K_{1,2} Id - 2 K_{2,4} \&C_K(e1, e3) + 2 K_{2,4} K_{1,3} Id$
 $+ 2 K_{1,4} \&C_K(e2, e3) - 2 K_{1,4} K_{2,3} Id + 2 \&C_K(e1, e2, e3) - 2 K_{2,3} e1 + 2 K_{1,3} e2$
 $- 2 K_{1,2} e3 + 2 K_{2,1} K_{1,3} Id - 2 K_{1,1} K_{2,3} Id + K_{2,4} e1 - K_{1,4} e2 + \&C_K(e1, e2) - K_{1,2} Id$

[which should be the same as out5 after conversion to Grassmann basis:

[> **clieval(out6)-out5;**

0

[> **out7:=RC(uu,v,K);**

$out7 := 4 K_{3,1} K_{2,3} e1 - 4 K_{3,1} K_{1,3} e2 - 4 K_{2,1} K_{3,3} e1 + 4 K_{2,1} K_{1,3} e3 + 4 K_{1,1} K_{3,3} e2$
 $- 4 K_{1,1} K_{2,3} e3 + 2 K_{3,4} \&C_K(e1, e2) - 2 K_{3,4} K_{1,2} Id - 2 K_{2,4} \&C_K(e1, e3) + 2 K_{2,4} K_{1,3} Id$
 $+ 2 K_{1,4} \&C_K(e2, e3) - 2 K_{1,4} K_{2,3} Id + 2 \&C_K(e1, e2, e3) - 2 K_{2,3} e1 + 2 K_{1,3} e2$
 $- 2 K_{1,2} e3 + 2 K_{2,1} K_{1,3} Id - 2 K_{1,1} K_{2,3} Id + K_{2,4} e1 - K_{1,4} e2 + \&C_K(e1, e2) - K_{1,2} Id$

[> **out7-out6;**

0

[>

[**Example 3:** Various inputs containing `&C[K]`:

[> **RCbig(`&C`[K](e1,e2),`&C`[K](e3,e4)); ##<<<--- Error because contraction is w.r.t. B**

Error, (in Cliplus:-RCbig) optional (or default B) parameter in RCbig differs from indices encountered in its cliprod arguments. Found these names as indices of &C: {B, K}

[> **RCbig(`&C`[K](e1,e2),`&C`[K](e3,e4),K); ##<<<--- No error because contraction is w.r.t. K**

$K_{1,4} K_{2,3} Id - K_{2,4} K_{1,3} Id + K_{3,4} \&C_K(e1, e2)$

[> **RCbig(`&C`[-K](e1,e2),`&C`[-K](e3,e4),-K); ##<<<--- No error because contraction is w.r.t. -K**

$K_{1,4} K_{2,3} Id - K_{2,4} K_{1,3} Id - K_{3,4} \&C_{-K}(e1, e2)$

[>

[>

See Also: [Clifford:-cmul](#), [Clifford:-clicollect](#), [Cliplus:-clieval](#), [Cliplus:-cliexpand](#), [Clifford:-makealiases](#), [Clifford:-wedge](#)

(c) Copyright 1995-2009, by Rafal Ablamowicz & Bertfried Fauser, all rights reserved.

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

- Function: Cliplus:-setup - the initialization procedure for the package 'Cliplus'

Calling Sequence:

none

Parameters:

none

- Description:

- Procedure 'init' is the initialization procedure for the 'Cliplus' package. It is executed automatically when the package is loaded.
- At the time of loading, the following macros are defined:
 - macro(Clifford:-cmul = Cliplus:-climul) - extends [Clifford:-cmul](#) from 'CLIFFORD'
 - macro(Clifford:-cmulQ = Cliplus:-climul) - extends [Clifford:-cmulQ](#) from 'CLIFFORD'
 - macro(`&c` = Cliplus:-climul) - extends [Clifford:-`&c`](#) from 'CLIFFORD'
 - macro(`&cQ` = Cliplus:-climul) - extends [Clifford:-`&cQ`](#) from 'CLIFFORD'
 - macro(Clifford:-reversion = Cliplus:-clirev) - extends [Clifford:-reversion](#) from 'CLIFFORD'
 - macro(Clifford:-LC = Cliplus:-LCbig) - extends [Clifford:-LC](#) from 'CLIFFORD'
 - macro(Clifford:-RC = Cliplus:-RCbig) - extends [Clifford:-RC](#) from 'CLIFFORD'
- To see all environmental variables that are defined and used by 'CLIFFORD', use procedure [Clifford:-CLIFFORD_ENV](#).
- User can change values of these environmental variables by making simple assignments. See below.

- Examples:

```
> restart:with(Clifford):with(Cliplus):
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
> CLIFFORD_ENV();

`>>> Global variables defined in Clifford:-setup are now available and have the
se values: <<<`
`***** Start *****`
dim_V = 9
_default_Clifford_product = Clifford:-cmulRS
_prolevel = false
_shortcut_in_minimalideal = true
_shortcut_in_Kfield = true
_shortcut_in_spinorKbasis = true
_shortcut_in_spinorKrepr = true
_warnings_flag = true
_scalartypes = {`^`, RootOf, complex, indexed, numeric, constant, function, mat
hfunc, rational}
_quatbasis = [[Id, e3we2, e1we3, e2we1], {`Maple has assigned qi:=-e2we3, qj:=
1we3, qk:=-e1we2`}]
```

```

`***** End *****`

`>>> Global variables defined in Cliplus:-setup are now available and have the
e values: <<<`
`***** Start *****`
macro(Clipford:-cmul = climul)
macro(Clipford:-cmulQ = climul)
macro(`&c` = climul)
macro(`&cQ` = climul)
macro(Clipford:-reversion = clirev)
macro(Clipford:-LC = LCbig)
macro(Clipford:-RC = RCbig)
`Warning, new definitions for type/climon and type/clipolynom now include &C`
`***** End *****`

`***** Start *****`
`>>> There are no new global variables or macros in GTP yet. <<<`
`***** End *****`

`>>> Global variables defined in Octonion:-setup are now available and have the
se values: <<<`

`***** Start *****`
_octbasis = [Id, e1, e2, e3, e4, e5, e6, e7]
_pureoctbasis = [e1, e2, e3, e4, e5, e6, e7]
_default_Fano_triples = [[1, 3, 7], [1, 2, 4], [1, 5, 6], [2, 3, 5], [2, 6, 7],
[3, 4, 6], [4, 5, 7]]
_default_squares = [-Id, -Id, -Id, -Id, -Id, -Id, -Id]
_default_Clipford_product = Clifford:-cmulNUM
`***** End *****`

[ >

```



See Also: [Cliplus:-LCbig](#), [Cliplus:-clirev](#), [Cliplus:-climul](#), [Clifford:-minimalideal](#), [Clifford:-Kfield](#), [Clifford:-spinorKbasis](#), [Clifford:-spinorKrepr](#), [Clifford:-`type/cliscalar`](#), [Clifford:-rmulm](#), [Clifford:-qmul](#), [Clifford:-wedge](#), [Clifford:-cmulQ](#), [Clifford:-cmul](#)

(c) Copyright 1995-2009, by Rafal Ablamowicz & Bertfried Fauser, all rights reserved.

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

Function: Cliplus:-`convert/dwedge_to_wedge`, Cliplus:-`convert/dwedge_to_wedge` - converting between wedge and dotted wedge

Calling Sequence:

```
c1 := convert(p1,wedge_to_dwedge,F)
c2 := convert(p2,dwedge_to_wedge,FT)
```

Parameters:

- p1 - Clifford polynomial expressed in terms of un-dotted standard Grassmann wedge basis (element of one of these types: ``type/clibasmon``, ``type/climon``, ``type/clipolynom``)
- p2 - Clifford polynomial in dotted basis (although still expressed in terms of the standard Grassmann wedge monomials)
- F, FT - argument of type name, symbol, matrix, array, or ``&*<code>(numeric, {name,symbol,matrix,array})</code>`. When F and FT are matrices or arrays, they are expected to be antisymmetric and negative of each other, that is, $FT = \text{linalg}[\text{transpose}](F)$.`
- F is assumed to be, by default, the antisymmetric part of B.

Output:

- c1 : a Clifford polynomial expressed in terms of the un-dotted Grassmann basis
- c2 : a Clifford polynomial in "dotted" basis expressed in terms of the standard Grassmann basis

Description:

- These two functions are used by the dotted-wedge in $Cl(B)$ given by [dwedge](#). The latter accompanies the Grassmann [wedge](#) product, but differs in its graduation. In fact both products are isomorphic as **exterior** products, but rely on different filtrations. The dotted wedge product and the undotted one are related by the process of cliffordization which is used in CLIFFORD internally to compute the Clifford product in $Cl(V,B)$. However, the cliffordization is performed in this case by an antisymmetric bilinear form $B=F$, say $F(x,y)=-F(y,x)$, where x and y are 1-vectors in V .
- **NOTE:** Till now both types of algebras are expanded (formally) over the same basis Grassmann monomials which, according to CLIFFORD's convention, are written as $e_i e_j \dots$. It is the responsibility of the user to keep track which type of wedge he/she is using and which expression is based on which exterior product, dotted or undotted. It is a good idea it to assign such expressions to a descriptive lhs, see below.
- **References:**

[1] Ablamowicz, R.: "Helmstetter formula and rigid motions with CLIFFORD", in "Advances in

Geometric Algebra with Applications in Science and Engineering -- Automatic Theorem proving, Computer Vision, Quantum and Neural Computing, and Robotics", Eds. Eduardo Bayro-Corrochano and Garret Sobczyk, Birkhäuser, 2002.

[2] Ablamowicz, R. and Bertfried Fauser: "On the decomposition of Clifford algebras of arbitrary bilinear form", Rafal Ablamowicz and Bertfried Fauser, in "Clifford Algebras and their Applications in Mathematical Physics", Eds. Rafal Ablamowicz and Bertfried Fauser, Vol. 1: Algebra and Physics, Birkhäuser, Boston, 2000, pages 341--366 (see also <http://www.birkhauser.com/cgi-win/isbn/0-8176-4182-3>).

[3] "Clifford Algebras and their Applications in Mathematical Physics", Eds. Rafal Ablamowicz and Bertfried Fauser, Vol. 1: Algebra and Physics, Birkhäuser, Boston, 2000, (ISBN 0-8176-4182-3) (for more information go to <http://math.tntech.edu/rafal/mexico/mexico.html>).

[4] Ablamowicz, R. and P. Lounesto: "On Clifford algebras of a bilinear form with an antisymmetric part," with P. Lounesto, in "Clifford Algebras with Numeric and Symbolic computations", Eds. R. Ablamowicz, P. Lounesto, and J. Parra, Birkhäuser, Boston, 1996, pages 167-188.

[5] "Clifford Algebras with Numeric and Symbolic Computations", Eds. Rafal Ablamowicz, Pertti Lounesto, and J. Parra, Birkhäuser, Boston, 1996 (ISBN 0-8176-3907-1).

— Examples:

```
> restart:with(Clipford):with(Cliplus);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
[LCbig, RCbig, clibasis, clieval, cliexpand, climul, clirev, dottedcbasis, dwedge,
  makeclialises]
```

Example: (Dotted and undotted wedge bases) Let us first expand the basis of the original wedge into the dotted wedge and back. For this purpose we choose $\dim_V=3$ and set up a antisymmetric bilinear form F and its negative FT :

```
> convert(elwe2,wedge_to_dwedge,K);
          elwe2 + K1,2 Id
> convert(%,dwedge_to_wedge,-K);
          elwe2
> dim_V:=3:
F:=array(1..dim_V,1..dim_V,antisymmetric):
F:=evalm(F);
FT:=linalg[transpose](F);
```

$$F = \begin{bmatrix} 0 & F_{1,2} & F_{1,3} \\ -F_{1,2} & 0 & F_{2,3} \\ -F_{1,3} & -F_{2,3} & 0 \end{bmatrix}$$

$$FT := \begin{bmatrix} 0 & -F_{1,2} & -F_{1,3} \\ F_{1,2} & 0 & -F_{2,3} \\ F_{1,3} & F_{2,3} & 0 \end{bmatrix}$$

```
> w_bas:=cbasis(dim_V); ## the wedge basis
g:=array(1..dim_V,1..dim_V,symmetric):
B:=evalm(g+F);
```

```
w_bas := [Id, e1, e2, e3, e1we2, e1we3, e2we3, e1we2we3]
```

$$B := \begin{bmatrix} g_{1,1} & g_{1,2} + F_{1,2} & g_{1,3} + F_{1,3} \\ g_{1,2} - F_{1,2} & g_{2,2} & g_{2,3} + F_{2,3} \\ g_{1,3} - F_{1,3} & g_{2,3} - F_{2,3} & g_{3,3} \end{bmatrix}$$

Now we map the convert function onto this basis to get the dotted-wedge basis (and back to test that this device works properly)

```
> d_bas:=map(convert,w_bas,wedge_to_dwedge,F);
```

```
d_bas := [Id, e1, e2, e3, e1we2 + F_{1,2} Id, e1we3 + F_{1,3} Id, e2we3 + F_{2,3} Id,
e1we2we3 + F_{2,3} e1 - F_{1,3} e2 + F_{1,2} e3]
```

```
> test_wbas:=map(convert,d_bas,dwedge_to_wedge,-F);
```

```
test_wbas := [Id, e1, e2, e3, e1we2, e1we3, e2we3, e1we2we3]
```

Note that only the scalar Id and the one vector basis elements e_i are unaltered and that the other basis elements of higher grade pick up additional terms of lower grade (which preserves the filtration).

It is possible to define aliases for the dotted wedge basis "monomials" similar to the Grassmann basis monomials used by 'CLIFFORD'. For example, we could denote the element $e_1we_2 + F_{1,2}Id$ by $e1we2$ or $e1We2$, and similarly for other elements:

```
> alias(e1We2=e1we2 + F[1,2]*Id,
e1We3=e1we3 + F[1,3]*Id,
e2We3=e2we3 + F[2,3]*Id,
e1We2We3=e1we2we3+F[2,3]*e1-F[1,3]*e2+F[1,2]*e3);
```

```
e1We2, e1We3, e2We3, e1We2We3
```

and then Maple will display automatically dotted basis in d_bas in terms of the aliases:

```
> d_bas;
```

```
[Id, e1, e2, e3, e1We2, e1We3, e2We3, e1We2We3]
```

While command 'cbasis' displays basis elements in the Grassmann basis by default, it is not difficult to write a new procedure that would display the dotted basis instead. This procedure is called 'dottedcbasis'. Since we have defined aliases above, output from 'dottedcbasis' will be automatically converted to aliases:

```
> dottedcbasis[F](3);
```

```
[Id, e1, e2, e3, e1We2, e1We3, e2We3, e1We2We3]
```

```
> dottedcbasis[F](3, 'even');
```

```
[Id, e1We2, e1We3, e2We3]
```

```
> dottedcbasis[F] (3,2) ;
```

```
[e1We2, e1We3, e2We3]
```

With the procedure 'findbasis' which returns linearly independent elements from a list, we can verify that the above lists contain linearly independent elements:

```
> findbasis(dottedcbasis[F] (3)) ;
```

```
[Id, e1, e2, e3, e1We2, e1We3, e2We3, e1We2We3]
```

```
> findbasis(dottedcbasis[F] (3, 'even')) ;
```

```
[Id, e1We2, e1We3, e2We3]
```

```
> findbasis(dottedcbasis[F] (3,2)) ;
```

```
[e1We2, e1We3, e2We3]
```

Example 2: (Commutative Diagram: Reversion in dotted and undotted bases) We proceed to show that the expansion of the Clifford basis elements into the dotted or undotted exterior products has also implications for other well known operations such as e.g. the Clifford reversion. Only if the bilinear form is symmetric, we find that the reversion is grade preserving, otherwise it reflects only the filtration (i.e. is in general a sum of terms of the same and lower degrees).

```
> reversion(e1we2) ; #reversion w.r.t. B (implicit)
reversion(e1we2,B) ; #reversion w.r.t. B (explicit - only
antisymmetric part F matters)
reversion(e1we2,F) ; #reversion w.r.t. B (explicit - only
antisymmetric part F matters)
reversion(e1we2,g) ; #reversion w.r.t. g (classical result)
```

```
-2 F1,2 Id - e1we2
```

```
-2 F1,2 Id - e1we2
```

```
-2 F1,2 Id - e1we2
```

```
-e1we2
```

Observe in the above that only when $B[1,2]=B[2,1]$, the result is $-e1we2$ known from the theory of classical Clifford algebras. Likewise,

```
> cbas:=cbasis(3) ;
```

```
cbas := [Id, e1, e2, e3, e1we2, e1we3, e2we3, e1we2we3]
```

```
> map(reversion,cbas,B) ; #explicit use of B = g + F
map(reversion,cbas,F) ; #one use the antisymmetric part of B
only
```

```
[Id, e1, e2, e3, -2 F1,2 Id - e1we2, -2 F1,3 Id - e1we3, -2 F2,3 Id - e2we3,
-2 F2,3 e1 + 2 F1,3 e2 - 2 F1,2 e3 - e1we2we3]
```

```
[Id, e1, e2, e3, -2 F1,2 Id - e1we2, -2 F1,3 Id - e1we3, -2 F2,3 Id - e2we3,
-2 F2,3 e1 + 2 F1,3 e2 - 2 F1,2 e3 - e1we2we3]
```

If instead of B we use the symmetric part g of B, we obtain instead

```
> map(reversion,cbas,g) ;
```

```
[Id, e1, e2, e3, -e1we2, -e1we3, -e2we3, -e1we2we3]
```

[Convert now e_1we_2 to the dotted basis and call it e_1We_2 :

[> `convert(e1we2,wedge_to_dwedge,F);`

e_1We_2

[Apply reversion to e_1We_2 with respect to F to get the reversed element in the dotted basis:

[> `reversed_e1We2:=reversion(e1We2,F);`

$reversed_e1We_2 := -F_{1,2} Id - e_1we_2$

[Observe, that the above element equals the negative of e_1We_2 just like reversing e_1we_2 with respect to the symmetric part of B (called 'g' above):

[> `reversed_e1We2+e1We2;`

0

[Finally, convert $reversed_e_1We_2$ to the un-dotted standard Grassmann basis to get $-e_1we_2$:

[> `convert(reversed_e1We2,dwedge_to_wedge,-F);`

$-e_1we_2$

[The above, of course, can be obtained by applying reversion to e_1we_2 with respect to the symmetric part of B :

[> `reversion(e1we2,g); #reversion with respect to the symmetric part g of B`

$-e_1we_2$

[This shows that the dotted wedge basis is the particular basis which is stable under the Clifford reversion computed with respect to F , the antisymmetric part of B . This requirement allows one to distinguish Clifford algebras $Cl(g)$ which have a symmetric bilinear form g from those which do not have such symmetric bilinear form but a more general form B instead. We call the former **classical Clifford algebras** while we use the term **quantum Clifford algebras** for the general non-necessarily-symmetric case.

[>

[>

– See Also: [Bigebra:-help](#), [Cliplus:-dwedge](#), [Clifford:-reversion](#), [Clifford:-cbasis](#)

(c) Copyright 1995-2009, by Rafal Ablamowicz & Bertfried Fauser, all rights reserved.

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