

- 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/clipolynomial now include &C and &C[K]. Type
?cliprod for help.
> CLIFFORD_ENV();

`>>> Global variables defined in Clifford:-setup are now available and have these 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 = {complex, indexed, numeric, RootOf, ```, constant, function, rational, mathfunc}
_quatbasis = [[Id, e3we2, elwe3, e2we1], {`Maple has assigned qi:=-e2we3, qj:=elwe3, qk:=-elwe2`}]]
`***** End *****`

`>>> Global variables defined in Cliplus:-setup are now available and have these values: <<<`
`***** Start *****`
macro(cmul = climul)
macro(cmulQ = climul)
macro(`&c` = climul)
macro(`&cQ` = climul)
macro(reversion = clirev)
macro(LC = LCbig)
macro(RC = RCbig)
`Warning, new definitions for type/climon and type/clipolynomial 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 these 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_Clifford_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 October 8, 1995, by Rafal Ablamowicz & Bertfried Fauser, all rights reserved.

Last modified: January 5, 2003, RA/BF.

Function: `dwedge`, `&dw` - Grassmann wedge product for a different filtration

Calling Sequence:

```
c1 := dwedge[K](p1,p2,...,pn)
c1 := &dw[K](p1,p2,...,pn)
```

Parameters:

- p_1, p_2, \dots, p_n - Clifford polynomials (elements of one of these types: `clibasmon`, `climon`, `clipolynom`)
- K - index of type name, symbol, matrix, array, or `&*(numeric, {name, symbol, matrix, array})`

Output:

- c_1 : a Clifford polynomial

Description:

- The dotted-wedge (`dwedge`) 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 .
- Procedure `'dwedge'` requires one index of type name, symbol, matrix, array, `&*(numeric, {name, symbol, matrix, array})`. When the index is a matrix or an array, it must be antisymmetric.
- It can be shown that the Wick-theorem of normal ordering, well known in QFT and many particle QM, is exactly described by this process.
- While being isomorphic as Grassmann algebras and hence being interchangeable, the difference becomes important when further structures are considered. For example, when a Clifford algebra is build over the space of this differently graded Grassmann algebra, or when quantum deformations are modeled within an undeformed Clifford algebra, etc..
- The dotted wedge is a wrapper function which actually uses `'convert/wedge to dwedge'` and `'convert/dwedge to wedge'` to map between the two basis sets. This is possible since the new Grassmann algebra is a cliffordized Grassmann algebra w.r.t. a bilinear form F as stated above.
- The ampersand version of this procedure is called `&dw`.
- **NOTE:** Until 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, 2003.
- [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,

Examples:

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

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, makeclialias]

Example 1: Simple examples first:

```
> dwedge[K](e1+2*e1we3,e4+3*e1we2);
```

```
dwedge[-K](e1+2*e1we3,e4+3*e1we2);
```

```
&dw(e1+2*e1we3,e4+3*e1we2); #default index in `&dw` is F
```

```
&dw[-F](e1+2*e1we3,e4+3*e1we2);
```

$$2 e1we3we4 + e1we4 + (K_{1,4} - 6 K_{1,3} K_{1,2}) Id - 6 K_{1,2} e1we3 - 6 K_{1,3} e1we2 + (2 K_{3,4} - 3 K_{1,2}) e1 - 2 K_{1,4} e3$$

$$2 e1we3we4 + e1we4 - (K_{1,4} + 6 K_{1,3} K_{1,2}) Id + 6 K_{1,2} e1we3 + 6 K_{1,3} e1we2 - (2 K_{3,4} - 3 K_{1,2}) e1 + 2 K_{1,4} e3$$

$$2 e1we3we4 + e1we4 - (-F_{1,4} + 6 F_{1,3} F_{1,2}) Id - 6 F_{1,2} e1we3 - 6 F_{1,3} e1we2 - (-2 F_{3,4} + 3 F_{1,2}) e1 - 2 F_{1,4} e3$$

$$2 e1we3we4 + e1we4 - (F_{1,4} + 6 F_{1,3} F_{1,2}) Id + 6 F_{1,2} e1we3 + 6 F_{1,3} e1we2 + (-2 F_{3,4} + 3 F_{1,2}) e1 + 2 F_{1,4} e3$$

```
>
```

Example 2: Observe that conversion from the undotted wedge basis to the dotted wedge basis using antisymmetric form F and 'dwedge[F]' are related through the following identity:

```
convert(e1we2w...wen,wedge_to_dwedge,F) = dwedge[F](e1,e2,...,en)
```

which can be shown as follows in dim_V <=5:

```
> F:=array(1..9,1..9,antisymmetric):
```

```
> ##when dim_V = 2:
```

```
simplify(dwedge[F](e1,e2)=convert(wedge(e1,e2),wedge_to_dwedge,F));
```

$$e1we2 + F_{1,2} Id = e1we2 + F_{1,2} Id$$

```
> ##when dim_V = 3:
```

```
simplify(dwedge[F](e1,e2,e3)=convert(wedge(e1,e2,e3),wedge_to_dwedge,F));
```

$$e1we2we3 + F_{2,3} e1 - F_{1,3} e2 + F_{1,2} e3 = e1we2we3 + F_{2,3} e1 - F_{1,3} e2 + F_{1,2} e3$$

```
> ##when dim_V = 4:
```

```
simplify(dwedge[F](e1,e2,e3,e4)=convert(wedge(e1,e2,e3,e4),wedge_to_dwedge,F));
```

$$e1we2we3we4 + F_{2,3} e1we4 + F_{2,3} F_{1,4} Id - F_{1,3} e2we4 - F_{1,3} F_{2,4} Id + F_{1,2} e3we4 + F_{1,2} F_{3,4} Id - F_{2,4} e1we3 + F_{1,4} e2we3$$

$$+ F_{3,4} e1we2 = e1we2we3we4 + F_{2,3} e1we4 + F_{2,3} F_{1,4} Id - F_{1,3} e2we4 - F_{1,3} F_{2,4} Id + F_{1,2} e3we4 + F_{1,2} F_{3,4} Id$$

$$- F_{2,4} e1we3 + F_{1,4} e2we3 + F_{3,4} e1we2$$

```
> ##when dim_V = 5:
```

```
simplify(dwedge[F](e1,e2,e3,e4,e5)-convert(wedge(e1,e2,e3,e4,e5),wedge_to_dwedge,F));
```

$$0$$

```
>
```

Example 3: Operation 'dwedge' is associative with Id as a unit:

```
> dwedge[F](dwedge[F](e1,e2),e3);
```

```
dwedge[F](e1,dwedge[F](e2,e3));
```

$$e1we2we3 + F_{2,3} e1 - F_{1,3} e2 + F_{1,2} e3$$

$$e1we2we3 + F_{2,3} e1 - F_{1,3} e2 + F_{1,2} e3$$

```
> dwedge[F](dwedge[F](e1,e2we3),e4);
```

```
dwedge[F](e1,dwedge[F](e2we3,e4));%%-%;
```

$$e1we2we3we4 - F_{1,3} e2we4 + F_{1,2} e3we4 - (F_{1,3} F_{2,4} - F_{1,2} F_{3,4}) Id - F_{2,4} e1we3 + F_{1,4} e2we3 + F_{3,4} e1we2$$

$$e1we2we3we4 - F_{1,3} e2we4 + F_{1,2} e3we4 - (F_{1,3} F_{2,4} - F_{1,2} F_{3,4}) Id - F_{2,4} e1we3 + F_{1,4} e2we3 + F_{3,4} e1we2$$

$$0$$

```
> dwedge[F](dwedge[F](e1,e2we3),e4we5);
```

```
dwedge[F](e1,dwedge[F](e2we3,e4we5));%%-%;
```

$$\begin{aligned}
& F_{2,5} e1we3we4 - (F_{1,3} F_{2,4} - F_{1,2} F_{3,4}) e5 + e1we2we3we4we5 - F_{3,5} e1we2we4 - F_{1,5} e2we3we4 - F_{1,3} e2we4we5 \\
& + F_{1,2} e3we4we5 - F_{2,4} e1we3we5 + F_{1,4} e2we3we5 + F_{3,4} e1we2we5 - (F_{2,4} F_{3,5} - F_{3,4} F_{2,5}) e1 \\
& - (-F_{1,3} F_{2,5} + F_{1,2} F_{3,5}) e4 + (F_{1,4} F_{3,5} - F_{3,4} F_{1,5}) e2 + (-F_{1,4} F_{2,5} + F_{2,4} F_{1,5}) e3 \\
& F_{2,5} e1we3we4 - (F_{1,3} F_{2,4} - F_{1,2} F_{3,4}) e5 + e1we2we3we4we5 - F_{3,5} e1we2we4 - F_{1,5} e2we3we4 - F_{1,3} e2we4we5 \\
& + F_{1,2} e3we4we5 - F_{2,4} e1we3we5 + F_{1,4} e2we3we5 + F_{3,4} e1we2we5 - (F_{2,4} F_{3,5} - F_{3,4} F_{2,5}) e1 \\
& - (-F_{1,3} F_{2,5} + F_{1,2} F_{3,5}) e4 + (F_{1,4} F_{3,5} - F_{3,4} F_{1,5}) e2 + (-F_{1,4} F_{2,5} + F_{2,4} F_{1,5}) e3 \\
& 0
\end{aligned}$$

Finally, for some arbitrary random Clifford polynomials expressed in Grassmann undotted basis:

```

> u:=2+e1-3*e2we3+e4we5we6:
v:=3-4*e1we2we3+e7:
z:=4-2*e3we4+e5we6-e8:
> dwedge[F](Id,u)=u; #unity
dwedge[F](u,Id)=u;
e4we5we6 + 2 Id - 3 e2we3 + e1 = 2 + e1 - 3 e2we3 + e4we5we6
e4we5we6 + 2 Id - 3 e2we3 + e1 = 2 + e1 - 3 e2we3 + e4we5we6
> dwedge[F](dwedge[F](u,v),z):#associativity
dwedge[F](u,dwedge[F](v,z)):%%-%;
0

```

We also have the following **Commutative Diagram 5: Wedge in undotted and dwedge in dotted bases:**

$$\text{wedge}(u,v) = \text{convert}(\text{dwedge}(\text{convert}(u,\text{wedge_to_dwedge},F),\text{convert}(v,\text{wedge_to_dwedge},F)),\text{dwedge_to_wedge},-F)$$

which we show as follows:

```

> uu:=convert(u,wedge_to_dwedge,F); #u converted to dotted basis
vv:=convert(v,wedge_to_dwedge,F); #v converted to dotted basis
uu := 2 Id + e1 - 3 e2we3 - 3 F_{2,3} Id + e4we5we6 + F_{5,6} e4 - F_{4,6} e5 + F_{4,5} e6
vv := 3 Id - 4 e1we2we3 - 4 F_{2,3} e1 + 4 F_{1,3} e2 - 4 F_{1,2} e3 + e7
> out1:=dwedge[F](uu,vv): #dwedge computed w.r.t. F
> out2:=convert(out1,dwedge_to_wedge,-F); #previous result converted back to undotted basis
out2 := -9 e2we3 + e4we5we6we7 + 4 e1we2we3we4we5we6 + 2 e7 - 8 e1we2we3 + 3 e4we5we6 + e1we7 - 3 e2we3we7
+ 6 Id + 3 e1
> out3:=wedge(u,v); #direct computation of the wedge product in undotted basis
out3 := -9 e2we3 + e4we5we6we7 + 4 e1we2we3we4we5we6 + 2 e7 - 8 e1we2we3 + 3 e4we5we6 + e1we7 - 3 e2we3we7
+ 6 Id + 3 e1
> out2-out3; #the same results!
0
>

```

Example 4: (Dotted and undotted wedge bases) First we expand the basis of the original wedge into the dotted wedge and back. For this purpose we choose $\dim_V=3$ and consider $Cl(C,B)$ where the antisymmetric part of B is denoted by F (and its negative by FT), while the symmetric part of B is denoted by g .

```

> dim_V:=3:
F:=array(1..dim_V,1..dim_V,antisymmetric):
g:=array(1..dim_V,1..dim_V,symmetric):
B:=evalm(g+F):
FT:=evalm(-F):
F,FT = evalm(F),evalm(FT);
g,B = evalm(g),evalm(B);
w_bas:=cbasis(dim_V); ## the wedge basis

```

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

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

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

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);
test_wbas:=map(convert,d_bas,dwedge_to_wedge,-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 := [Id, e1, e2, e3, e1we2, e1we3, e2we3, e1we2we3]
```

Note that only the scalar Id and the one vector basis elements ei 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 $e1we2 + F[1,2]*Id$ by $e1de2$ 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. For example, procedure 'dottedcbasis' returns such basis. 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 5: (Commutative Diagram 1: Contraction in dotted and undotted bases) The contraction w.r.t. any bilinear form works on both sets in the **same manner** which can be seen if we re-convert the dotted-wedge basis after the computation into the wedge basis. In a reasonable setting, the antisymmetric bilinear form F would be the antisymmetric part of B . To read more about the left contraction LC in $Cl(B)$, go to the [help page for LC](#) or see [1, 2, 4]. To illustrate this fact, we first compute left contraction by $e1$ of every element in the standard Grassmann wedge basis w_bas with respect to the entire form B :

```
> 'w_bas'=w_bas; #standard Grassmann wedge basis in Cl(B)
w_bas = [Id, e1, e2, e3, e1we2, e1we3, e2we3, e1we2we3]
> w_wout:=map2(LC,e1,w_bas,B);#left contraction LC in Cl(B) w.w.t. B in wedge basis
w_wout := [0, g_{1,1} Id, (g_{1,2} + F_{1,2}) Id, (g_{1,3} + F_{1,3}) Id, g_{1,1} e2 - (g_{1,2} + F_{1,2}) e1, g_{1,1} e3 - (g_{1,3} + F_{1,3}) e1,
```

$$(g_{1,2} + F_{1,2}) e3 - (g_{1,3} + F_{1,3}) e2, g_{1,1} e2we3 - (g_{1,2} + F_{1,2}) e1we3 + (g_{1,3} + F_{1,3}) e1we2]$$

Next, we compute left contraction by e1 of every element in the dotted wedge basis d_bas with respect to the entire form B. Recall from the above that conversion from the wedge basis to the dotted wedge basis used the antisymmetric part F of B:

```
> 'd_bas'=d_bas; #dotted wedge basis in Cl(B)
      d_bas=[Id, e1, e2, e3, e1We2, e1We3, e2We3, e1We2We3]
> w_dout1:=map2(LC,e1,d_bas,B);#left contraction LC in Cl(B) w.w.t. B in dotted wedge basis
w_dout1 := [0, g1,1 Id, (g1,2 + F1,2) Id, (g1,3 + F1,3) Id, g1,1 e2 - (g1,2 + F1,2) e1, g1,1 e3 - (g1,3 + F1,3) e1,
            (g1,2 + F1,2) e3 - (g1,3 + F1,3) e2,
            g1,1 e2we3 - (g1,2 + F1,2) e1we3 + (g1,3 + F1,3) e1we2 + F2,3 g1,1 Id - F1,3 Id g1,2 + F1,2 Id g1,3]
```

Notice that in the above coefficients of g, the symmetric part of B, are mixed with the coefficients of the antisymmetric part F of B. To remove the F coefficients, we need to convert back the above result to the un-dotted standard Grassmann basis using the negative -F, that is, the negative of the antisymmetric part of B in the conversion process:

```
> w_dout:=map(convert,w_dout1,dwedge_to_wedge,-F);#converting back to undotted basis
w_dout := [0, g1,1 Id, Id g1,2 + F1,2 Id, Id g1,3 + F1,3 Id, g1,1 e2 - e1 g1,2 - e1 F1,2, g1,1 e3 - e1 g1,3 - e1 F1,3,
            e3 g1,2 + F1,2 e3 - e2 g1,3 - F1,3 e2, g1,1 e2we3 - e1we3 g1,2 - e1we3 F1,2 + e1we2 g1,3 + e1we2 F1,3]
> map(simplify,w_dout-w_wout);
      [0, 0, 0, 0, 0, 0, 0, 0]
```

>

This computation shows clearly the isomorphy between both pictures. To show that the new structure is nevertheless valuable for other reasons, we proceed with Clifford products.

Example 6: (Commutative Diagram 2: Clifford product in dotted and undotted bases) We can build a Clifford algebra Cl(B) over each basis set, that is, w_wout or w_dout, but with different bilinear forms: when B=g and when B=g+F (following notation from [1, 2, 4]), where g is the symmetric part of B and F is the antisymmetric part of B:

```
> B,g,F=evalm(B),evalm(g),evalm(F); #previously defined
      B, g, F = 
$$\begin{bmatrix} g_{1,1} & g_{1,2} + F_{1,2} & g_{1,3} + F_{1,3} \\ g_{1,2} - F_{1,2} & g_{2,2} & g_{2,3} + F_{2,3} \\ g_{1,3} - F_{1,3} & g_{2,3} - F_{2,3} & g_{3,3} \end{bmatrix}, \begin{bmatrix} g_{1,1} & g_{1,2} & g_{1,3} \\ g_{1,2} & g_{2,2} & g_{2,3} \\ g_{1,3} & g_{2,3} & g_{3,3} \end{bmatrix}, \begin{bmatrix} 0 & F_{1,2} & F_{1,3} \\ -F_{1,2} & 0 & F_{2,3} \\ -F_{1,3} & -F_{2,3} & 0 \end{bmatrix}$$

```

Let us compute some such Clifford products using the facility of [cmul](#) to take a bilinear form (here in matrix form) as index. We will show an example with the following two elements:

```
> w_p1:=e1we2;
      w_p2:=a*e3+b*e2we3;
      w_p1 := e1we2
      w_p2 := a e3 + b e2we3
```

We can then define Clifford product 'cmulg' with respect to the symmetric part g, and another Clifford product 'cmulB' with respect to the entire form B:

```
> cmulg:=proc() RETURN(cmul[g](args)) end;
      cmulB:=proc() RETURN(cmul[B](args)) end;
```

Thus, we are ready to perform computations around our commutative diagram.

First, we compute Clifford product cmul[g] in Cl(g), that is, with respect to the symmetric part g of the bilinear form B, of the two above defined elements w_p1 and w_p2 expressed in undotted Grassmann basis.

```
> w_out1:=cmulg(w_p1,w_p2); ## Clifford product w.r.t. g in Cl(g) in wedge basis
      w_out1 := a e1we2we3 - b (-g2,2 g1,3 + g2,3 g1,2) Id + b g2,2 e1we3 - b g1,2 e2we3 - b g2,3 e1we2 + a g2,3 e1 - a g1,3 e2
```

Now, we convert each element p1 and p2 to the dotted wedge basis:

```
> d_p1:=convert(w_p1,wedge_to_dwedge,F);
      d_p2:=convert(w_p2,wedge_to_dwedge,F); #incomplete conversion to e1We2, etc. basis
      d_p1 := e1We2
      d_p2 := a e3 + b e2we3 + b F2,3 Id
```

We now compute the Clifford product of d_p1 and d_p2 in Cl(B) in the dotted wedge basis:

```
> d_out1:=cmulB(d_p1,d_p2); ## Clifford product w.r.t. B=g+F in Cl(B) in dwedge basis
```

$$d_out1 := a e1we2we3 - b (g_{2,3} F_{1,2} - g_{2,2} g_{1,3} - g_{2,2} F_{1,3} + g_{2,3} g_{1,2} + F_{2,3} g_{1,2}) Id + b g_{2,2} e1we3 - b g_{1,2} e2we3 - b g_{2,3} e1we2 + a (g_{2,3} + F_{2,3}) e1 - a (g_{1,3} + F_{1,3}) e2 + F_{1,2} a e3$$

We now convert the above result back to the un-dotted wedge basis:

```
> w_out2:=convert(d_out1,dwedge_to_wedge,-F); ## convert result dwedge-> wedge
```

$$w_out2 := a e1we2we3 + b Id g_{2,2} g_{1,3} - b Id g_{2,3} g_{1,2} + b g_{2,2} e1we3 - b g_{1,2} e2we3 - b g_{2,3} e1we2 + a g_{2,3} e1 - a g_{1,3} e2$$

Finally, we show that this result is the same as before when we computed Clifford product of p1 and p2 in Cl(g):

```
> simplify(w_out1-w_out2); ## show equality !
```

$$0$$

This shows (one can prove this) that the Clifford algebra Cl(g) of the symmetric part g of B using the undotted exterior basis is isomorphic, as an associative algebra, to the Clifford algebra Cl(B) of the entire bilinear form B = g + F spanned by the dotted wedge basis if the antisymmetric part F of B is exactly the same F as is used to connect the two basis sets (cf. [1, 2, 4]).

Example 7: (Commutative Diagram 3: 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,B); #reversion with respect to B
reversion(e1we2,g); #reversion with respect to g (classical result)
```

$$-2 F_{1,2} Id - e1we2$$

$$-e1we2$$

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);
```

$$[Id, e1, e2, e3, -2 F_{1,2} Id - e1we2, -2 F_{1,3} Id - e1we3, -2 F_{2,3} Id - e2we3, -e1we2we3 - 2 F_{2,3} e1 + 2 F_{1,3} e2 - 2 F_{1,2} e3]$$

If instead of B we use a symmetric matrix 'g' defined above, we obtain instead

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

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

Convert now e1we2 to the dotted basis and call it e1We2:

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

$$e1We2$$

Apply reversion to e1We2 with respect to F to get the reversed element in the dotted basis:

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

$$reversed_e1We2 := -F_{1,2} Id - e1we2$$

Observe, that the above element is equal to the negative of e1We2 just like reversing e1we2 with respect to the symmetric part g of B:

```
> reversed_e1We2+e1We2;
```

$$0$$

Finally, convert reversed_e1We2 to the un-dotted standard Grassmann basis to get -e1we2:

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

$$-e1we2$$

The above, of course, can be obtained by applying reversion to e1we2 with respect to the symmetric part of B:

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

$$-e1we2$$

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 the bilinear form 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.

Example 8: It is easy to write a wrapper for the Grassmann co-product too. Since the co-product [&gco](#) makes essential use of the

decomposition of elements into one-vectors, we expect that the dotted-Grassmann co-product will depend on F, that is, then antisymmetric part of B. First we have to load [Bigebra](#) package.

NOTE: `&gco_d` computes the dotted Grassmann co-product in the undotted wedge basis! (The Grassmann co-product on the dotted wedge basis w.r.t. the dotted wedge basis is according to the isomorphism theorem for those algebras identical to the original Grassmann co-product).

```
> with(Bigebra);
Increase verbosity by infolevel[function]=val -- use online help > ?Bigebra[help]
[&cco, &gco, &gco_d, &gco_pl, &map, &v, EV, VERSION, bracket, contract, drop_t, eps, gantipode, gco_unit, gswitch,
  hodge, linop, linop2, lists2mat, lists2mat2, make_BI_Id, mapop, mapop2, meet, op2mat, op2mat2, pairing, peek, poke,
  remove_eq, switch, tcollect, tsolve1]
> w_p1:=e1we2;
  w_p2:=&gco_d(w_p1);
```

$$w_{p1} := e1we2$$

$$w_{p2} := (Id \&t e1we2) + F_{1,2} (Id \&t Id) + (e1 \&t e2) - (e2 \&t e1) + (e1we2 \&t Id)$$

The following examples compose the dotted co-product with dotted and undotted wedge (acting on a wedge basis!!)

```
> dwedge[F](e1,e2);
dwedge[F](e1,e2,e3);
```

$$e1We2$$

$$e1We2We3$$

We then substitute `&dw` and `&w` for the tensor product sign `&t` and evaluate. We will show the results sided by side for easier comparison:

```
> subs(`&t`=`&dw`,&gco_d(Id));simplify(%);
subs(`&t`=`&w`,&gco_d(Id));simplify(%);
```

$$Id \&dw Id$$

$$Id$$

$$Id \&w Id$$

$$Id$$

```
> subs(`&t`=`&dw`,&gco_d(e1));simplify(%);
subs(`&t`=`&w`,&gco_d(e1));simplify(%);
```

$$(Id \&dw e1) + (e1 \&dw Id)$$

$$2 e1$$

$$(Id \&w e1) + (e1 \&w Id)$$

$$2 e1$$

```
> subs(`&t`=`&dw`,&gco_d(e2));simplify(%);
subs(`&t`=`&w`,&gco_d(e2));simplify(%);
```

$$(Id \&dw e2) + (e2 \&dw Id)$$

$$2 e2$$

$$(Id \&w e2) + (e2 \&w Id)$$

$$2 e2$$

```
> subs(`&t`=`&dw`,&gco_d(e1we2));simplify(%);
subs(`&t`=`&w`,&gco_d(e1we2));simplify(%);
```

$$(Id \&dw e1we2) + F_{1,2} (Id \&dw Id) + (e1 \&dw e2) - (e2 \&dw e1) + (e1we2 \&dw Id)$$

$$4 e1we2 + 3 F_{1,2} Id$$

$$(Id \&w e1we2) + F_{1,2} (Id \&w Id) + (e1 \&w e2) - (e2 \&w e1) + (e1we2 \&w Id)$$

$$4 e1we2 + F_{1,2} Id$$

```
> subs(`&t`=`&dw`,&gco_d(e1we3));simplify(%);
subs(`&t`=`&w`,&gco_d(e1we3));simplify(%);
```

$$(Id \&dw e1we3) + F_{1,3} (Id \&dw Id) + (e1 \&dw e3) - (e3 \&dw e1) + (e1we3 \&dw Id)$$

$$4 e1we3 + 3 F_{1,3} Id$$

$$(Id \&w e1we3) + F_{1,3} (Id \&w Id) + (e1 \&w e3) - (e3 \&w e1) + (e1we3 \&w Id)$$

$$4 e1we3 + F_{1,3} Id$$

```
> subs(`&t`=`&dw`,&gco_d(e2we3));simplify(%);
subs(`&t`=`&w`,&gco_d(e2we3));simplify(%);
```

$$(Id \&dw e2we3) + F_{2,3} (Id \&dw Id) + (e2 \&dw e3) - (e3 \&dw e2) + (e2we3 \&dw Id)$$

$$4 e2we3 + 3 F_{2,3} Id$$

$$(Id \&w e2we3) + F_{2,3} (Id \&w Id) + (e2 \&w e3) - (e3 \&w e2) + (e2we3 \&w Id)$$

$$4 e2we3 + F_{2,3} Id$$

```
> subs(`&t`=`&dw`,&gco_d(e1we2we3));simplify(%);
subs(`&t`=`&w`,&gco_d(e1we2we3));simplify(%);
```

$$(Id \&dw e1we2we3) + F_{2,3} (Id \&dw e1) - F_{1,3} (Id \&dw e2) + F_{1,2} (Id \&dw e3) + (e1 \&dw e2we3) + F_{2,3} (e1 \&dw Id)$$

$$- (e2 \&dw e1we3) - F_{1,3} (e2 \&dw Id) + (e1we2 \&dw e3) + (e3 \&dw e1we2) + F_{1,2} (e3 \&dw Id) - (e1we3 \&dw e2)$$

$$+ (e2we3 \&dw e1) + (e1we2we3 \&dw Id)$$

$$6 F_{1,2} e3 + 6 F_{2,3} e1 - 6 F_{1,3} e2 + 8 e1we2we3$$

$$(Id \&w e1we2we3) + F_{2,3} (Id \&w e1) - F_{1,3} (Id \&w e2) + F_{1,2} (Id \&w e3) + (e1 \&w e2we3) + F_{2,3} (e1 \&w Id)$$

$$- (e2 \&w e1we3) - F_{1,3} (e2 \&w Id) + (e1we2 \&w e3) + (e3 \&w e1we2) + F_{1,2} (e3 \&w Id) - (e1we3 \&w e2)$$

$$+ (e2we3 \&w e1) + (e1we2we3 \&w Id)$$

$$8 e1we2we3 + 2 F_{2,3} e1 - 2 F_{1,3} e2 + 2 F_{1,2} e3$$

It is of utmost importance, that in these calculations we find that the usual loop tangle mult \cite Delta_{mult} which come up with the dimension of the spaces involved **fails** here. This might have an strong impact on the renormalization theory in QFT.

Note however, that if we do everything in the same algebra we end up with the correct factor $2^{(\text{grade_of_x})}$ for the dotted bi-vector e1we2:

```
> d_p1:=&gco_d(dwedge[F](e1,e2));
```

$$d_p1 := (Id \&t e1we2) + 2 F_{1,2} (Id \&t Id) + (e1 \&t e2) - (e2 \&t e1) + (e1we2 \&t Id)$$

```
> drop_t(&map(d_p1,1,dwedge[F]));
```

$$4 e1we2 + 4 F_{1,2} Id$$

```
> d_p2:=convert(%,wedge_to_dwedge,F);#comes up with the factor 4 in the dotted basis
'd_p2'=-4*reversion(e1we2);
```

$$d_p2 := 4 e1we2 + 8 F_{1,2} Id$$

$$d_p2 = 4 e1we2 + 8 F_{1,2} Id$$

```
>
```

See Also: [Bigebra:-&gco`](#), [Bigebra:-&cco`](#), [Bigebra:-&t`](#), [Bigebra:-drop t`](#), [Bigebra:-&map`](#)

(c) Copyright October 8, 1995, by Rafal Ablamowicz & Bertfried Fauser, all rights reserved.

Last modified: November 16, 2003, RA/BF.

Function: Cliplus:-clibasis - define a Clifford basis consisting of Clifford monomials in the Clifford algebra $Cl(V)$

Calling Sequence:

```
clibasis(n);  
clibasis[K](n);  
clibasis(n, k);  
clibasis[K](n, k);  
clibasis(n, 'even');  
clibasis[K](n, 'even');
```

Parameters:

n - dimension of the real vector space V where $0 \leq n \leq 9$

k - the k -th grade, $0 \leq k \leq n$

K - (optional) index of type name, symbol, matrix, array, or `&*` (numeric, {name, symbol, matrix, array})`

Description:

- Procedure 'clibasis' writes a Clifford basis given in terms of Clifford monomials for a Clifford algebra $Cl(V)$ over a real vector space V endowed with a bilinear form B , when no optional index is used, or the index specifies the form. The dimension of V is specified as the first argument while the bilinear form may be specified as index.
- It is possible to specify a numeric multiple of the form as in `clibasis[-K](n)`, etc.
- When index is not used, B is the default form used.
- The procedure can be used with one or two arguments. When used with one argument n , it returns an ordered list of all basis elements in the Clifford algebra $Cl(V)$ where the dimension of V is n . When used with two arguments n and k , it returns a list of basis elements in the k -vector subspace of $Cl(V)$.
- The basis elements are given in terms of the inert form `&C`` or `&C`[K]` of the Clifford product. To evaluate, use [Cliplus:-clieval](#).
- To see a basis for $Cl(V)$ given in terms of Grassmann monomials, use [cbasis](#).
- An option 'even' allows one to create a basis in the even subalgebra of the Clifford algebra $Cl(V)$.
- The basis elements can be aliased, e.g., `e1 &C e2` can be aliased as `e12` with the procedure 'Cliplus:-makealiases'. See [Cliplus:-makealiases](#) for more help.
- When $n=0$ then `[Id]` is returned.

Examples:

```
> restart:with(Clipford):with(Cliplus);  
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 ]  
[ Grassmann basis for Cl(3):  
> Gbasis:=cbasis(3);  
[  
      Gbasis := [ Id, e1, e2, e3, e1we2, e1we3, e2we3, e1we2we3 ]  
[ Grassmann basis can be expressed in term of the Clifford basis by using procedure 'cliexpand':  
> map(cliexpand, Gbasis);  
[  
      Id, e1, e2, e3, (e1 &C e2) - B1,2 Id, (e1 &C e3) - B1,3 Id, (e2 &C e3) - B2,3 Id, &C(e1, e2, e3) - B2,3 e1 + B1,3 e2 - B1,2 e3  
      ]  
[ Using optional argument in 'cliexpand', we can convert Grassmann basis to Clifford basis in Cl(K):  
> map(cliexpand, Gbasis, K);  
[ Id, e1, e2, e3, &C_K(e1, e2) - K1,2 Id, &C_K(e1, e3) - K1,3 Id, &C_K(e2, e3) - K2,3 Id,  
      &C_K(e1, e2, e3) - K2,3 e1 + K1,3 e2 - K1,2 e3 ]  
[ Using optional argument in 'cliexpand', we can convert Grassmann basis to Clifford basis in Cl(-K):  
> map(cliexpand, Gbasis, -K);  
[ Id, e1, e2, e3, &C_-K(e1, e2) + K1,2 Id, &C_-K(e1, e3) + K1,3 Id, &C_-K(e2, e3) + K2,3 Id,
```

```

    &C_{-K}(e1, e2, e3) + K_{2,3} e1 - K_{1,3} e2 + K_{1,2} e3]
>
[ Clifford basis for Cl(B) when dim_V = 3:
> dim_V:=3: #dimension of V
> L1:=clibasis(dim_V);      #non-indexed basis in Cl(V,B)
  L2:=clibasis[B](dim_V);   #indexed basis in Cl(V,B)
  L3:=clibasis[K](dim_V);   #indexed basis in Cl(V,K)
  L4:=clibasis[-K](dim_V);  #indexed basis in Cl(V,-K)
  L5:=clibasis[-2*K](dim_V); #indexed basis in Cl(V,-2*K)

      L1 := [Id, e1, e2, e3, e1 &C e2, e1 &C e3, e2 &C e3, &C(e1, e2, e3)]
      L2 := [Id, e1, e2, e3, &C_B(e1, e2), &C_B(e1, e3), &C_B(e2, e3), &C_B(e1, e2, e3)]
      L3 := [Id, e1, e2, e3, &C_K(e1, e2), &C_K(e1, e3), &C_K(e2, e3), &C_K(e1, e2, e3)]
      L4 := [Id, e1, e2, e3, &C_{-K}(e1, e2), &C_{-K}(e1, e3), &C_{-K}(e2, e3), &C_{-K}(e1, e2, e3)]
      L5 := [Id, e1, e2, e3, &C_{-2K}(e1, e2), &C_{-2K}(e1, e3), &C_{-2K}(e2, e3), &C_{-2K}(e1, e2, e3)]

[ Clifford basis can be expressed in terms of the Grassmann basis by using procedure 'clieval':
> L11:=map(clieval,L1);
  L22:=map(clieval,L2);
  L33:=map(clieval,L3);
  L44:=map(clieval,L4);
  L55:=map(clieval,L5);

      L11 := [Id, e1, e2, e3, e1we2 + B_{1,2} Id, e1we3 + B_{1,3} Id, e2we3 + B_{2,3} Id, e1we2we3 + B_{2,3} e1 - B_{1,3} e2 + B_{1,2} e3]
      L22 := [Id, e1, e2, e3, e1we2 + B_{1,2} Id, e1we3 + B_{1,3} Id, e2we3 + B_{2,3} Id, e1we2we3 + B_{2,3} e1 - B_{1,3} e2 + B_{1,2} e3]
      L33 := [Id, e1, e2, e3, e1we2 + K_{1,2} Id, e1we3 + K_{1,3} Id, e2we3 + K_{2,3} Id, e1we2we3 + K_{2,3} e1 - K_{1,3} e2 + K_{1,2} e3]
      L44 := [Id, e1, e2, e3, e1we2 - K_{1,2} Id, e1we3 - K_{1,3} Id, e2we3 - K_{2,3} Id, e1we2we3 - K_{2,3} e1 + K_{1,3} e2 - K_{1,2} e3]
      L55 := [Id, e1, e2, e3, e1we2 - 2 K_{1,2} Id, e1we3 - 2 K_{1,3} Id, e2we3 - 2 K_{2,3} Id, e1we2we3 - 2 K_{2,3} e1 + 2 K_{1,3} e2 - 2 K_{1,2} e3]

[ Clifford basis for the even subalgebra of Cl(3), that is, when dim_V = 3 and B = B (default), K, or -K:
> clibasis(3, 'even');
  clibasis[B](3, 'even');
  clibasis[K](3, 'even');
  clibasis[-K](3, 'even');

      [Id, e1 &C e2, e1 &C e3, e2 &C e3]
      [Id, &C_B(e1, e2), &C_B(e1, e3), &C_B(e2, e3)]
      [Id, &C_K(e1, e2), &C_K(e1, e3), &C_K(e2, e3)]
      [Id, &C_{-K}(e1, e2), &C_{-K}(e1, e3), &C_{-K}(e2, e3)]

[ Clifford basis for the vector subspace of "bivectors" in Cl(3):
> clibasis(3,2);
  clibasis[B](3,2);
  clibasis[K](3,2);
  clibasis[-K](3,2);

      [e1 &C e2, e1 &C e3, e2 &C e3]
      [&C_B(e1, e2), &C_B(e1, e3), &C_B(e2, e3)]
      [&C_K(e1, e2), &C_K(e1, e3), &C_K(e2, e3)]
      [&C_{-K}(e1, e2), &C_{-K}(e1, e3), &C_{-K}(e2, e3)]
>

```

— See Also: [Cliplus:-cliexpand](#), [Cliplus:-clieval](#), [Clifford:-cbasis](#)

(c) Copyright October 8, 1995, by Rafal Ablamowicz & Bertfried Fauser, all rights reserved.
 Last modified: January 5, 2003, RA/BF.

Function: Cliplus:-clieval - evaluates the inert forms $\&C$, $\&C[K]$, or $\&C[-K]$ of the Clifford product in $Cl(B)$ and $Cl(K)$

Calling Sequence:

clieval(p);

Parameters:

p - polynomial whose terms are of type [Clifford:-`type/cliprod`](#).

Description:

- The procedure expands Clifford basis monomials in terms of the Grassmann monomials (see [`type/clibasmon`](#)) used in the package 'CLIFFORD'. It can be applied to any polynomial with such terms.
- The inverse operation to 'clieval' is [Cliplus:-cliexpand](#).

Examples:

```
> restart:with(Clifford):with(Cliplus);
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, makeclialias]
[Example: Some computations with an arbitrary form B:
> p:=e1 &C e2;type(p,cliprod);clieval(p);
      p := e1 &C e2
      true
      e1we2 + B1,2 Id
> p:=`&C`[B](e1,e2);type(p,cliprod);clieval(p);
      p := &CB(e1, e2)
      true
      e1we2 + B1,2 Id
> p:=`&C`[K](e1,e2);type(p,cliprod);clieval(p);
      p := &CK(e1, e2)
      true
      e1we2 + K1,2 Id
> p:=`&C`[-K](e1,e2);type(p,cliprod);clieval(p);
      p := &C-K(e1, e2)
      true
      e1we2 - K1,2 Id
> q:=`&C`[K](e1,e2,e3);clieval(q);
      q := &CK(e1, e2, e3)
      e1we2we3 + K2,3 e1 - K1,3 e2 + K1,2 e3
> q:=`&C`[-K](e1,e2,e3);clieval(q);
      q := &C-K(e1, e2, e3)
      e1we2we3 - K2,3 e1 + K1,3 e2 - K1,2 e3
> q:=`&C`[K](e1,e2,Id);clieval(q);
      q := &CK(e1, e2, Id)
      e1we2 + K1,2 Id
> q:=`&C`[-K](e1,e2,Id);clieval(q);
      q := &C-K(e1, e2, Id)
      e1we2 - K1,2 Id
> p:=Id+2*(e1 &C e2)+3*(e2 &C e3)-&C(e1,e2,e3); p1:=clieval(p);p2:=cliexpand(p1);p2-p;
```

$$p := Id + 2 (e1 \&C e2) + 3 (e2 \&C e3) - \&C(e1, e2, e3)$$

$$p1 := Id + 2 e1we2 + 2 B_{1,2} Id + 3 e2we3 + 3 B_{2,3} Id - e1we2we3 - B_{2,3} e1 + B_{1,3} e2 - B_{1,2} e3$$

$$p2 := Id + 2 (e1 \&C e2) + 3 (e2 \&C e3) - \&C(e1, e2, e3)$$

$$0$$

```
> p:=Id+2*`&C`[K](e1,e2)+3*`&C`[K](e2,e3)-`&C`[-K](e1,e2,e3);
p1:=clieval(p);p2:=cliexpand(p1,K);p2-p;
```

$$p := Id + 2 \&C_K(e1, e2) + 3 \&C_K(e2, e3) - \&C_{-K}(e1, e2, e3)$$

$$p1 := Id + 2 e1we2 + 2 K_{1,2} Id + 3 e2we3 + 3 K_{2,3} Id - e1we2we3 + K_{2,3} e1 - K_{1,3} e2 + K_{1,2} e3$$

$$p2 := Id + 2 \&C_K(e1, e2) + 3 \&C_K(e2, e3) - \&C_K(e1, e2, e3) + 2 K_{2,3} e1 - 2 K_{1,3} e2 + 2 K_{1,2} e3$$

$$-\&C_K(e1, e2, e3) + 2 K_{2,3} e1 - 2 K_{1,3} e2 + 2 K_{1,2} e3 + \&C_{-K}(e1, e2, e3)$$

```
> p:=Id+2*`&C`[-K](e1,e2)+3*`&C`[-K](e2,e3)-`&C`[-K](e1,e2,e3);
p1:=clieval(p);p2:=cliexpand(p1,-K);p2-p;
```

$$p := Id + 2 \&C_{-K}(e1, e2) + 3 \&C_{-K}(e2, e3) - \&C_{-K}(e1, e2, e3)$$

$$p1 := Id + 2 e1we2 - 2 K_{1,2} Id + 3 e2we3 - 3 K_{2,3} Id - e1we2we3 + K_{2,3} e1 - K_{1,3} e2 + K_{1,2} e3$$

$$p2 := Id + 2 \&C_{-K}(e1, e2) + 3 \&C_{-K}(e2, e3) - \&C_{-K}(e1, e2, e3)$$

$$0$$

[Conversions between the Grassmann basis and the Clifford bases can be done as follows:

```
> L:=clibasis(3); #Clifford basis for Cl(3)
```

$$L := [Id, e1, e2, e3, e1 \&C e2, e1 \&C e3, e2 \&C e3, \&C(e1, e2, e3)]$$

```
> L1:=map(clieval,L); #above Clifford basis expanded in terms of the Grassmann basis in Cl(3)
```

$$L1 := [Id, e1, e2, e3, e1we2 + B_{1,2} Id, e1we3 + B_{1,3} Id, e2we3 + B_{2,3} Id, e1we2we3 + B_{2,3} e1 - B_{1,3} e2 + B_{1,2} e3]$$

```
> L2:=map(cliexpand,L1); #obtaining back the Clifford basis
```

$$L2 := [Id, e1, e2, e3, e1 \&C e2, e1 \&C e3, e2 \&C e3, \&C(e1, e2, e3)]$$

```
> L:=cbasis(3); #Grassmann basis for Cl(3)
```

$$L := [Id, e1, e2, e3, e1we2, e1we3, e2we3, e1we2we3]$$

```
> L1:=map(cliexpand,L); #conversion to a Clifford basis w.r.t. B
```

```
L1 := [
```

$$Id, e1, e2, e3, (e1 \&C e2) - B_{1,2} Id, (e1 \&C e3) - B_{1,3} Id, (e2 \&C e3) - B_{2,3} Id, \&C(e1, e2, e3) - B_{2,3} e1 + B_{1,3} e2 - B_{1,2} e3$$

```
]
```

[It is also possible to have the B form listed explicitly:

```
> L1:=map(cliexpand,L,B); #conversion to a Clifford basis w.r.t. B
```

$$L1 := [Id, e1, e2, e3, \&C_B(e1, e2) - B_{1,2} Id, \&C_B(e1, e3) - B_{1,3} Id, \&C_B(e2, e3) - B_{2,3} Id,$$

$$\&C_B(e1, e2, e3) - B_{2,3} e1 + B_{1,3} e2 - B_{1,2} e3]$$

```
> L2:=map(clieval,L1); #conversion back to the Grassmann basis
```

$$L2 := [Id, e1, e2, e3, e1we2, e1we3, e2we3, e1we2we3]$$

```
>
```

See Also: [Cliplus:-cliexpand](#), [Cliplus:-clibasis](#), [Clifford:-makealiases](#)

(c) Copyright October 8, 1995, by Rafal Ablamowicz & Bertfried Fauser, all rights reserved.

Last modified: January 5, 2003, RA/BF.

Function: Cliplus:-cliexpand - expresses Grassmann basis in terms of the Clifford basis in Cl(B), Cl(K), or Cl(-K)

Calling Sequence:

```
cliexpand(p);  
cliexpand(p,K);
```

Parameters:

p - polynomial whose terms are of type [Clifford:-`type/clibasmon`](#).
K - (optional) argument of type name, symbol, matrix, array, or `&*(numeric,{name,symbol,matrix,array})`

Description:

- The procedure converts any Clifford polynomial expressed in terms of the Grassmann basis monomials into a polynomial expressed in terms of the Clifford monomials, that is, terms of type [Clifford:-`type/cliprod`](#).
- It can take an optional argument K of type name, symbol, matrix, array, or `&*(numeric,{name,symbol,matrix,array})` in which case the Clifford basis is in Cl(K). The default is K=B.
- The inverse operation to 'cliexpand' is [Cliplus:-clieval](#).

Examples:

```
> restart:with(Clifford):with(Cliplus);  
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, makeclialias]  
[ Example 1: Some computations with a symbolic form B:  
> L:=cbasis(3); #Grassmann basis for Cl(3)  
L := [Id, e1, e2, e3, e1we2, e1we3, e2we3, e1we2we3]  
> map(cliexpand,L); #the above basis expressed in terms of the Clifford basis in Cl(B)  
[  
Id, e1, e2, e3, (e1 &C e2) - B1,2 Id, (e1 &C e3) - B1,3 Id, (e2 &C e3) - B2,3 Id, &C(e1, e2, e3) - B2,3 e1 + B1,3 e2 - B1,2 e3  
]  
> map(cliexpand,L,B); #the above basis expressed in terms of the Clifford basis in  
Cl(B)  
[Id, e1, e2, e3, &C_B(e1, e2) - B1,2 Id, &C_B(e1, e3) - B1,3 Id, &C_B(e2, e3) - B2,3 Id,  
&C_B(e1, e2, e3) - B2,3 e1 + B1,3 e2 - B1,2 e3]  
> map(cliexpand,L,K); #the above basis expressed in terms of the Clifford basis in  
Cl(K)  
[Id, e1, e2, e3, &C_K(e1, e2) - K1,2 Id, &C_K(e1, e3) - K1,3 Id, &C_K(e2, e3) - K2,3 Id,  
&C_K(e1, e2, e3) - K2,3 e1 + K1,3 e2 - K1,2 e3]  
> map(cliexpand,L,-K); #the above basis expressed in terms of the Clifford basis in  
Cl(-K)  
[Id, e1, e2, e3, &C_-K(e1, e2) + K1,2 Id, &C_-K(e1, e3) + K1,3 Id, &C_-K(e2, e3) + K2,3 Id,  
&C_-K(e1, e2, e3) + K2,3 e1 - K1,3 e2 + K1,2 e3]  
> p:=1-2*e2+e3we4; #Clifford polynomial expressed in Grassmann basis  
p := 1 - 2 e2 + e3we4  
> p1:=cliexpand(p); #polynomial p expressed now in Clifford basis in Cl(B)  
p1:=cliexpand(p,B); #polynomial p expressed now in Clifford basis in Cl(B)  
p1 := 1 - 2 e2 + (e3 &C e4) - B3,4 Id  
p1 := 1 - 2 e2 + &C_B(e3, e4) - B3,4 Id  
> p2:=clieval(p1); #polynomial p1 expressed back in Grassmann basis should equal p  
p2 := Id - 2 e2 + e3we4
```

Note: 'Id' denotes the unit element in any Clifford algebra, that is, $1 = 1*Id$, and therefore $p = p2$. Recall that procedure [Clifford:-displayid](#) displays any scalar s as $s*Id$. For example,

```
> pp:=displayid(p);
                                     pp := Id - 2 e2 + e3we4
```

Example 2: Computations with another symbolic form H:

```
> L:=cbasis(3); #Grassmann basis for Cl(3)
                                     L := [Id, e1, e2, e3, e1we2, e1we3, e2we3, e1we2we3]
> map(cliexpand,L,H); #the above basis expressed in terms of the Clifford basis in Cl(H)
[Id, e1, e2, e3, &CH(e1, e2) - H1,2 Id, &CH(e1, e3) - H1,3 Id, &CH(e2, e3) - H2,3 Id,
&CH(e1, e2, e3) - H2,3 e1 + H1,3 e2 - H1,2 e3]
> p:=1-2*e2+e3we4; #Clifford polynomial expressed in Grassmann basis
                                     p := 1 - 2 e2 + e3we4
> p1:=cliexpand(p,H); #polynomial p expressed now in Clifford basis
                                     p1 := 1 - 2 e2 + &CH(e3, e4) - H3,4 Id
> p2:=clieval(p1); #polynomial p1 expressed back in Grassmann basis should equal p
                                     p2 := Id - 2 e2 + e3we4
```

Example 3: Computations with another symbolic form -H:

```
> L:=cbasis(3); #Grassmann basis for Cl(3)
                                     L := [Id, e1, e2, e3, e1we2, e1we3, e2we3, e1we2we3]
> map(cliexpand,L,-H); #the above basis expressed in terms of the Clifford basis in Cl(-H)
[Id, e1, e2, e3, &C-H(e1, e2) + H1,2 Id, &C-H(e1, e3) + H1,3 Id, &C-H(e2, e3) + H2,3 Id,
&C-H(e1, e2, e3) + H2,3 e1 - H1,3 e2 + H1,2 e3]
> p:=1-2*e2+e3we4; #Clifford polynomial expressed in Grassmann basis
                                     p := 1 - 2 e2 + e3we4
> p1:=cliexpand(p,-H); #polynomial p expressed now in Clifford basis
                                     p1 := 1 - 2 e2 + &C-H(e3, e4) + H3,4 Id
> p2:=clieval(p1); #polynomial p1 expressed back in Grassmann basis should equal p
                                     p2 := Id - 2 e2 + e3we4
```

[This shows that 'cliexpand' is the inverse operation to 'clieval'.

See Also: [Clifford:-cbasis](#), [Clifford:-`type/cliproduct`](#), [Clifford:-`type/clibasmon`](#), [Cliquplus:-clieval](#), [Cliquplus:-clibasis](#), [Clifford:-makealiases](#)

(c) Copyright October 8, 1995, by Rafal Ablamowicz & Bertfried Fauser, all rights reserved.

Last modified: January 5, 2003, RA/BF.

Function: Cliplus:-climul - extends Clifford multiplication 'cmul' to polynomials expressed in the Clifford bases e.g., $\&C(e1,e2)$, $\&C[K](e1,e2,e3)$

Calling Sequence:

```
cmul(p1, p2,..., pn);
climul(p1, p2,..., pn);
cmul[K](p1, p2,..., pn);
climul[K](p1, p2,..., pn);
```

Parameters:

p1, p2, ..., pn - polynomials whose terms are of type [Clifford:-type/cliprod](#).
K - (optional) index of type name, symbol, matrix, array, or $\&^{*}$ (numeric, {name,symbol,matrix,array})

Description:

- This procedure extends procedure [cmul](#) from 'CLIFFORD' to polynomials in Cl(B) or Cl(K) expressed in terms of the Clifford basis written in terms of the unevaluated Clifford product $\&C$ or $\&C[K]$ or $\&C[-K]$ respectively.
- Notice that the default is K=B. In this case, the basis can be displayed and used as $\&C(e1,e2)$, etc., or, with B listed explicitly as index as in $\&C[B](e1,e2)$. Not internal mechanism exists to convert from one form to the other although they give the same Clifford basis in Cl(B).
- User can use now the modified procedure 'cmul' which now can be applied to a new type of input, that is, [type/cliprod](#).
- It is also possible to mix Grassmann basis with the Clifford basis.
- When package 'Cliplus' is loaded, definitions of [type/climon](#) and [type/clipolynom](#) are modified so that expressions like $2*\&C(e1,e2)$, $a*\&C[K](e3,e2)$, etc. are of [type/climon](#) while sums of such expressions, e.g., $2*\&C(e1,e2) + 4*\&C(e3,e2)$, $2*\&C[B](e1,e2) + 4*\&C[B](e3,e2)$, $2*\&C[K](e1,e2) + 4*\&C[K](e3,e2)$ are of [type/clipolynom](#). This is needed so that procedures [Cliplus:-clieval](#) and [Cliplus:-cliexpand](#) from Cliplus work properly.
- **Notice** that although procedures [cmulQ](#) and its ampersand form [&cQ](#) are extended by climul, they don't compute correctly (off-diagonal entries in B are included) unless the form is diagonal. See **Example 2** below. Procedure 'climul' was meant to extend 'cmul' and $\&c$ only.

Examples:

```
[ > restart:bench:=time():with(Clifford):
[ Example 1: Let's do first some type checking.
[ > p1:=e1 &C e2;p2:=2*p1;p3:=p2+3*\&C(e1,e3,e4);
      p1 := e1 &C e2
      p2 := 2 (e1 &C e2)
      p3 := 2 (e1 &C e2) + 3 &C(e1, e3, e4)
[ > p1,type(p1,cliprod);p2,type(p2,cliprod);p3,type(p3,cliprod);
      e1 &C e2,true
      2 (e1 &C e2),false
      2 (e1 &C e2) + 3 &C(e1, e3, e4),false
[ > p1,type(p1,clibasmon);p2,type(p2,clibasmon);p3,type(p3,clibasmon);
      e1 &C e2,false
      2 (e1 &C e2),false
      2 (e1 &C e2) + 3 &C(e1, e3, e4),false
[ > p1,type(p1,climon);p2,type(p2,climon);p3,type(p3,climon);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now include &C and &C[K]. Type
?cliprod for help.
      e1 &C e2,false
      2 (e1 &C e2),true
      2 (e1 &C e2) + 3 &C(e1, e3, e4),false
```

```
> p1,type(p1,clipolynom);p2,type(p2,clipolynom);p3,type(p3,clipolynom);
      e1 &C e2,false
      2 (e1 &C e2),false
      2 (e1 &C e2) + 3 &C(e1, e3, e4), true
```

Now let's see index cliprods:

```
> p1:=`&C`[B](e1,e2);p2:=2*p1;p3:=p2+3*`&C`[K](e3,e4);p4:=`&C`[-B](e1,e2);
      p1 := &CB(e1, e2)
      p2 := 2 &CB(e1, e2)
      p3 := 2 &CB(e1, e2) + 3 &CK(e3, e4)
      p4 := &C-B(e1, e2)
```

```
> p1,type(p1,cliprod);p2,type(p2,cliprod);
p3,type(p3,cliprod);p4,type(p4,cliprod);
      &CB(e1, e2), true
      2 &CB(e1, e2), false
      2 &CB(e1, e2) + 3 &CK(e3, e4), false
      &C-B(e1, e2), true
```

```
> p1,type(p1,clibasmon);p2,type(p2,clibasmon);
p3,type(p3,clibasmon);p4,type(p4,clibasmon);
      &CB(e1, e2), false
      2 &CB(e1, e2), false
      2 &CB(e1, e2) + 3 &CK(e3, e4), false
      &C-B(e1, e2), false
```

We can turn off printing of the warning messages by changing the default value of the environmental variable `_warnings_flag` via [CLIFFORD ENV](#):

```
> _warnings_flag:=false;
p1,type(p1,climon);p2,type(p2,climon);p3,type(p3,climon);p4,type(p4,climon);
p1,type(p1,clipolynom);p2,type(p2,clipolynom);p3,type(p3,clipolynom);
      _warnings_flag := false
      &CB(e1, e2), false
      2 &CB(e1, e2), true
      2 &CB(e1, e2) + 3 &CK(e3, e4), false
      &C-B(e1, e2), false
      &CB(e1, e2), false
      2 &CB(e1, e2), false
      2 &CB(e1, e2) + 3 &CK(e3, e4), true
```

Thus, in the main package CLIFFORD (version 6), expressions p1, p2, and p3 are not of type 'clibasmon', 'climon', or 'clipolynom', p2 and p3 are not of type 'cliprod', and only p1 is of type 'cliprod'. Furthermore, procedure 'cmul' cannot expressions containing types 'cliprod' unless it is modified by loading the supplementary package 'Cliplus'. Loading is done automatically now:

```
> cmul(p1,p1); #cmul can handle new `type/cliprod`
      B2,1 &CB(e1, e2) + B1,2 &CB(e1, e2) - Id B2,2 B1,1
> cmul(p1,p1);cmul(p1,p2);cmul[-B](p4,p4);
      B2,1 &CB(e1, e2) + B1,2 &CB(e1, e2) - Id B2,2 B1,1
      2 B2,1 &CB(e1, e2) + 2 B1,2 &CB(e1, e2) - 2 Id B2,2 B1,1
      -B2,1 &C-B(e1, e2) - B1,2 &C-B(e1, e2) - Id B2,2 B1,1
```

and the result is expressed in terms of the Clifford basis.

Notice that an error message is returned when various different names (indices) for the unevaluated Clifford products are used:

```
> 'p3'=p3;
cmul(p3,p3); ###<<<- Intended error message
```

$$p3 = 2 \&C_B(e1, e2) + 3 \&C_K(e3, e4)$$

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

This is because expression p3 contains terms $\&C[B](e1,e2)$ and $\&C[K](e1,e2,e3)$ with different indices. However, when all indices are the same, computation proceeds:

```
> p4:=2*`&C`[K](e1,e2)- 3*`&C`[K](e3,e4)+Id;
p5:=2*`&C`[-K](e1,e2)- 3*`&C`[-K](e3,e4)+Id;
```

$$p4 := 2 \&C_K(e1, e2) - 3 \&C_K(e3, e4) + Id$$

$$p5 := 2 \&C_{-K}(e1, e2) - 3 \&C_{-K}(e3, e4) + Id$$

```
> cmul[K](p4,p4);
cmul[-K](p5,p5);
```

$$\begin{aligned} & -6 K_{3,1} \&C_K(e2, e4) + 6 K_{4,2} K_{1,3} Id - 6 K_{4,2} \&C_K(e1, e3) + 6 K_{3,2} \&C_K(e1, e4) + 6 K_{4,1} \&C_K(e2, e3) - 9 Id K_{4,4} K_{3,3} \\ & + 4 K_{2,1} \&C_K(e1, e2) + 4 K_{1,2} \&C_K(e1, e2) + 9 K_{4,3} \&C_K(e3, e4) + 9 K_{3,4} \&C_K(e3, e4) - 12 \&C_K(e1, e2, e3, e4) \\ & + 4 \&C_K(e1, e2) - 6 K_{3,2} K_{1,4} Id + 6 K_{2,3} \&C_K(e1, e4) - 6 K_{2,4} \&C_K(e1, e3) - 6 K_{1,3} \&C_K(e2, e4) + 6 K_{1,4} \&C_K(e2, e3) \\ & + Id + 6 K_{3,1} K_{2,4} Id - 6 Id K_{2,3} K_{1,4} - 6 Id K_{4,1} K_{3,2} + 6 Id K_{2,4} K_{1,3} + 6 Id K_{4,2} K_{3,1} - 6 K_{4,1} K_{2,3} Id - 6 \&C_K(e3, e4) \\ & - 4 Id K_{2,2} K_{1,1} \end{aligned}$$

$$\begin{aligned} & 6 K_{4,2} \&C_{-K}(e1, e3) + 6 K_{4,2} K_{1,3} Id + 6 K_{3,1} \&C_{-K}(e2, e4) - 6 K_{4,1} \&C_{-K}(e2, e3) - 4 K_{2,1} \&C_{-K}(e1, e2) \\ & - 4 K_{1,2} \&C_{-K}(e1, e2) - 9 Id K_{4,4} K_{3,3} - 9 K_{4,3} \&C_{-K}(e3, e4) - 9 K_{3,4} \&C_{-K}(e3, e4) - 12 \&C_{-K}(e1, e2, e3, e4) \\ & + 4 \&C_{-K}(e1, e2) - 6 \&C_{-K}(e3, e4) - 6 K_{3,2} K_{1,4} Id - 6 K_{2,3} \&C_{-K}(e1, e4) + 6 K_{1,3} \&C_{-K}(e2, e4) + 6 K_{2,4} \&C_{-K}(e1, e3) \\ & - 6 K_{1,4} \&C_{-K}(e2, e3) + Id + 6 K_{3,1} K_{2,4} Id - 6 Id K_{2,3} K_{1,4} - 6 Id K_{4,1} K_{3,2} + 6 Id K_{2,4} K_{1,3} + 6 Id K_{4,2} K_{3,1} - 6 K_{4,1} K_{2,3} Id \\ & - 6 K_{3,2} \&C_{-K}(e1, e4) - 4 Id K_{2,2} K_{1,1} \end{aligned}$$

```
> &c[K](p4,p4); #short for of the Clifford product can be used as well
&c[-K](p5,p5); #short for of the Clifford product can be used as well
```

$$\begin{aligned} & -6 K_{3,1} \&C_K(e2, e4) + 6 K_{4,2} K_{1,3} Id - 6 K_{4,2} \&C_K(e1, e3) + 6 K_{3,2} \&C_K(e1, e4) + 6 K_{4,1} \&C_K(e2, e3) - 9 Id K_{4,4} K_{3,3} \\ & + 4 K_{2,1} \&C_K(e1, e2) + 4 K_{1,2} \&C_K(e1, e2) + 9 K_{4,3} \&C_K(e3, e4) + 9 K_{3,4} \&C_K(e3, e4) - 12 \&C_K(e1, e2, e3, e4) \\ & + 4 \&C_K(e1, e2) - 6 K_{3,2} K_{1,4} Id + 6 K_{2,3} \&C_K(e1, e4) - 6 K_{2,4} \&C_K(e1, e3) - 6 K_{1,3} \&C_K(e2, e4) + 6 K_{1,4} \&C_K(e2, e3) \\ & + Id + 6 K_{3,1} K_{2,4} Id - 6 Id K_{2,3} K_{1,4} - 6 Id K_{4,1} K_{3,2} + 6 Id K_{2,4} K_{1,3} + 6 Id K_{4,2} K_{3,1} - 6 K_{4,1} K_{2,3} Id - 6 \&C_K(e3, e4) \\ & - 4 Id K_{2,2} K_{1,1} \end{aligned}$$

$$\begin{aligned} & 6 K_{4,2} \&C_{-K}(e1, e3) + 6 K_{4,2} K_{1,3} Id + 6 K_{3,1} \&C_{-K}(e2, e4) - 6 K_{4,1} \&C_{-K}(e2, e3) - 4 K_{2,1} \&C_{-K}(e1, e2) \\ & - 4 K_{1,2} \&C_{-K}(e1, e2) - 9 Id K_{4,4} K_{3,3} - 9 K_{4,3} \&C_{-K}(e3, e4) - 9 K_{3,4} \&C_{-K}(e3, e4) - 12 \&C_{-K}(e1, e2, e3, e4) \\ & + 4 \&C_{-K}(e1, e2) - 6 \&C_{-K}(e3, e4) - 6 K_{3,2} K_{1,4} Id - 6 K_{2,3} \&C_{-K}(e1, e4) + 6 K_{1,3} \&C_{-K}(e2, e4) + 6 K_{2,4} \&C_{-K}(e1, e3) \\ & - 6 K_{1,4} \&C_{-K}(e2, e3) + Id + 6 K_{3,1} K_{2,4} Id - 6 Id K_{2,3} K_{1,4} - 6 Id K_{4,1} K_{3,2} + 6 Id K_{2,4} K_{1,3} + 6 Id K_{4,2} K_{3,1} - 6 K_{4,1} K_{2,3} Id \\ & - 6 K_{3,2} \&C_{-K}(e1, e4) - 4 Id K_{2,2} K_{1,1} \end{aligned}$$

```
> Cliplus:-climul[K](p4,p4); #climul can be used directly too
climul[-K](p5,p5); #THIS DOES NOT WORK since Cliplus has not been loaded yet!
```

$$\begin{aligned} & -6 K_{3,1} \&C_K(e2, e4) + 6 K_{4,2} K_{1,3} Id - 6 K_{4,2} \&C_K(e1, e3) + 6 K_{3,2} \&C_K(e1, e4) + 6 K_{4,1} \&C_K(e2, e3) - 9 Id K_{4,4} K_{3,3} \\ & + 4 K_{2,1} \&C_K(e1, e2) + 4 K_{1,2} \&C_K(e1, e2) + 9 K_{4,3} \&C_K(e3, e4) + 9 K_{3,4} \&C_K(e3, e4) - 12 \&C_K(e1, e2, e3, e4) \\ & + 4 \&C_K(e1, e2) - 6 K_{3,2} K_{1,4} Id + 6 K_{2,3} \&C_K(e1, e4) - 6 K_{2,4} \&C_K(e1, e3) - 6 K_{1,3} \&C_K(e2, e4) + 6 K_{1,4} \&C_K(e2, e3) \\ & + Id + 6 K_{3,1} K_{2,4} Id - 6 Id K_{2,3} K_{1,4} - 6 Id K_{4,1} K_{3,2} + 6 Id K_{2,4} K_{1,3} + 6 Id K_{4,2} K_{3,1} - 6 K_{4,1} K_{2,3} Id - 6 \&C_K(e3, e4) \\ & - 4 Id K_{2,2} K_{1,1} \end{aligned}$$

$$climul_{-K}(2 \&C_{-K}(e1, e2) - 3 \&C_{-K}(e3, e4) + Id, 2 \&C_{-K}(e1, e2) - 3 \&C_{-K}(e3, e4) + Id)$$

```
> cmul[B](p1,p1,p1);
```

$$B_{1,2}^2 \&C_B(e1, e2) + B_{2,1}^2 \&C_B(e1, e2) - B_{2,2} B_{1,1} \&C_B(e1, e2) - B_{1,2} Id B_{2,2} B_{1,1} + 2 B_{2,1} B_{1,2} \&C_B(e1, e2)$$

```

- B_{2,1} Id B_{2,2} B_{1,1}
> &c[B](p1,p1,p1); #short for of the Clifford product can be used as well
B_{1,2}^2 &C_B(e1, e2) + B_{2,1}^2 &C_B(e1, e2) - B_{2,2} B_{1,1} &C_B(e1, e2) - B_{1,2} Id B_{2,2} B_{1,1} + 2 B_{2,1} B_{1,2} &C_B(e1, e2)
- B_{2,1} Id B_{2,2} B_{1,1}

```

Example 2: Procedure 'climul' extends 'cmulQ' and '&cQ' also except that off-diagonal terms from B are included unless B is a diagonal matrix. For example,

```

> B:='B':p1;
&C_B(e1, e2)
> cmulQ(p1,p1);
&cQ(p1,p1);
B_{2,1} &C_B(e1, e2) + B_{1,2} &C_B(e1, e2) - Id B_{2,2} B_{1,1}
B_{2,1} &C_B(e1, e2) + B_{1,2} &C_B(e1, e2) - Id B_{2,2} B_{1,1}

```

while the correct result should be only $-B[2,2]*B[1,1]*Id$. However, if B is defined as a diagonal matrix then result is correct:

```

> B:=matrix(2,2,[a,0,0,b]);
B := [ a  0 ]
      [ 0  b ]
> cmulQ(p1,p1);&cQ(p1,p1);
-b a Id
-b a Id

```

Example 3: Notice the following changes in type definitions when 'Cliplus' is loaded:

```

> p1,type(p1,cliprod);p2,type(p2,cliprod);
p3,type(p3,cliprod); #no change here
&C_B(e1, e2), true
2 &C_B(e1, e2), false
2 &C_B(e1, e2) + 3 &C_K(e3, e4), false
> p1,type(p1,clibasmon);p2,type(p2,clibasmon);
p3,type(p3,clibasmon); #no change here
&C_B(e1, e2), false
2 &C_B(e1, e2), false
2 &C_B(e1, e2) + 3 &C_K(e3, e4), false
> p1,type(p1,climon);p2,type(p2,climon);
p3,type(p3,climon); #changes here
&C_B(e1, e2), false
2 &C_B(e1, e2), true
2 &C_B(e1, e2) + 3 &C_K(e3, e4), false
> p1,type(p1,clipolynom);p2,type(p2,clipolynom);
p3,type(p3,clipolynom); #changes here
&C_B(e1, e2), false
2 &C_B(e1, e2), false
2 &C_B(e1, e2) + 3 &C_K(e3, e4), true

```

Example 4: Let's see some more computations:

```

> B:='B':
p1:=Id+2*(e1 &C e2) + 3*&C(e1,e2,e3);
p1 := Id + 2 (e1 &C e2) + 3 &C(e1, e2, e3)
> cmul(p1,p2);

```

```

-4 Id B2,2 B1,1 - 6 e3 B2,2 B1,1 + 6 e2 B3,2 B1,1 + 6 B1,3 B2,2 e1 - 6 B1,2 B3,2 e1 + 6 B2,3 B1,1 e2 + 4 B2,1 &CB(e1, e2)
+ 4 B1,2 &CB(e1, e2) - 6 B2,1 B2,3 e1 + 6 B1,2 &CB(e1, e2, e3) + 6 B2,1 &CB(e1, e2, e3) + 2 &CB(e1, e2) - 6 B2,3 B1,2 e1
- 6 e1 B3,2 B2,1 + 6 e1 B3,1 B2,2

```

It is also possible to use the infix form of 'cmul':

```

> p1 &c p2;
-4 Id B2,2 B1,1 - 6 e3 B2,2 B1,1 + 6 e2 B3,2 B1,1 + 6 B1,3 B2,2 e1 - 6 B1,2 B3,2 e1 + 6 B2,3 B1,1 e2 + 4 B2,1 &CB(e1, e2)
+ 4 B1,2 &CB(e1, e2) - 6 B2,1 B2,3 e1 + 6 B1,2 &CB(e1, e2, e3) + 6 B2,1 &CB(e1, e2, e3) + 2 &CB(e1, e2) - 6 B2,3 B1,2 e1
- 6 e1 B3,2 B2,1 + 6 e1 B3,1 B2,2

```

One use 'clicollect' to collect the above output:

```

> clicollect(%);
2 (2 B2,1 + 2 B1,2 + 1) &CB(e1, e2) + 6 (B1,2 + B2,1) &CB(e1, e2, e3) - 4 Id B2,2 B1,1
- 6 (B1,2 B3,2 - B3,1 B2,2 + B2,1 B2,3 + B3,2 B2,1 - B1,3 B2,2 + B2,3 B1,2) e1 + 6 B1,1 (B3,2 + B2,3) e2 - 6 e3 B2,2 B1,1

```

Mixed basis elements are also allowed:

```

> p2:=p1+2*e2we3;
p2 := Id + 2 (e1 &C e2) + 3 &C(e1, e2, e3) + 2 e2we3
> cmul(p2,p2);
-4 B2,3 Id - 9 Id B3,3 B2,2 B1,1 + 6 e3 B3,1 B2,2 - 6 e3 B3,2 B2,1 + 6 e2 B3,3 B2,1 - 12 e1 B3,3 B2,2 - 4 Id B3,3 B2,2
+ 6 B1,2 B3,3 e2 - 6 B1,2 B2,3 e3 + 4 Id B1,3 B2,2 - 4 Id B2,3 B1,2 - 6 B1,2 B3,2 e3 + 4 Id B3,1 B2,2 - 4 Id B3,2 B2,1
- 4 Id B2,2 B1,1 - 12 e3 B2,2 B1,1 + 6 e2 B3,2 B1,1 + 6 B1,3 B2,2 e1 - 6 B1,2 B3,2 e1 + 6 B2,3 B1,1 e2 + 12 B3,2 &CB(e1, e2, e3)
+ 4 B2,1 &CB(e1, e2) + 4 B1,2 &CB(e1, e2) - 9 B1,2 B3,2 &CB(e1, e3) + 9 B3,1 B2,2 &CB(e1, e3) - 9 B2,1 B2,3 &CB(e1, e3)
- 9 B3,2 B2,1 &CB(e1, e3) + 9 B1,3 B2,2 &CB(e1, e3) - 9 B2,3 B1,2 &CB(e1, e3) + 4 &CB(e2, e3) - 6 B2,1 B2,3 e1
+ 6 &CB(e1, e2, e3) - 4 B2,1 B2,3 Id + 9 B3,2 B1,1 &CB(e2, e3) + 9 B2,3 B1,1 &CB(e2, e3) - 4 B2,3 &CB(e2, e3) + 4 B2,32 Id
+ 4 B2,1 &CB(e2, e3) + 4 B1,2 &CB(e2, e3) + 4 B3,2 &CB(e2, e3) + 12 B1,2 &CB(e1, e2, e3) + 12 B2,1 &CB(e1, e2, e3)
- 4 B2,3 &CB(e1, e2) + 4 B3,2 &CB(e1, e2) + 9 B3,3 B1,2 &CB(e1, e2) - 4 B3,2 B1,2 Id + 9 B3,3 B2,1 &CB(e1, e2) + Id
+ 4 &CB(e1, e2) - 6 B2,3 B1,2 e1 - 6 B2,3 B2,1 e3 - 6 e1 B3,2 B2,1 + 6 e1 B3,1 B2,2 + 6 B1,3 B2,2 e3

```

Polynomial p2 has three terms, two expressed the Clifford basis and one in the Grassmann basis. By the way, to express p2 solely in terms of the Clifford basis, use procedure 'cliexpand':

```

> Cliplus:-cliexpand(p2);
Id + 2 (e1 &C e2) + 3 &C(e1, e2, e3) + 2 (e2 &C e3) - 2 B2,3 Id

```

while to express p2 solely in terms of the Grassmann basis, use 'clieval':

```

> Cliplus:-clieval(p2);
Id + 2 e1we2 + 2 B1,2 Id + 3 e1we2we3 + 3 B2,3 e1 - 3 B1,3 e2 + 3 B1,2 e3 + 2 e2we3

```

Let's see how 'cmul' now handles mixed input:

```

> clicollect(cmul(p2,p1));
-(9 B1,2 B3,2 + 9 B3,2 B2,1 - 9 B2,2 B1,3 + 9 B2,3 B2,1 + 4 B2,2 - 9 B3,1 B2,2 + 9 B2,3 B1,2) &CB(e1, e3) -
(9 B3,3 B2,2 B1,1 + 4 B3,2 B2,1 + 4 B1,2 B3,2 + 4 B2,3 B2,1 - 1 - 4 B3,1 B2,2 + 2 B2,3 + 4 B2,3 B1,2 - 4 B2,2 B1,3 + 4 B2,2 B1,1) Id
- 6 (-B2,2 B1,3 + B1,2 B3,2 + B3,2 B2,1 + B2,3 B1,2 - B3,1 B2,2 + B2,3 B2,1 + B3,3 B2,2) e1
+ 6 (B3,3 B2,1 + B3,2 B1,1 + B2,3 B1,1 + B3,3 B1,2) e2
- 6 (-B3,1 B2,2 + B2,3 B1,2 - B2,2 B1,3 + B1,2 B3,2 + B3,2 B2,1 + B2,3 B2,1 + 2 B2,2 B1,1) e3
+ (4 + 9 B3,3 B2,1 + 4 B2,1 + 4 B3,2 + 4 B1,2 + 9 B3,3 B1,2) &CB(e1, e2)
+ (4 B2,1 + 2 + 4 B1,2 + 9 B3,2 B1,1 + 9 B2,3 B1,1) &CB(e2, e3) + 6 (B3,2 + 2 B2,1 + 1 + 2 B1,2) &CB(e1, e2, e3)

```

Finally, let's see the multiplication table for the basis Clifford monomials in Cl(3):

```

> B:=linalg[diag](1,1,1);
> Clifford_basis:=Cliplus:-clibasis(3);
Clifford_basis := [Id, e1, e2, e3, e1 &C e2, e1 &C e3, e2 &C e3, &C(e1, e2, e3)]
> M:=matrix(8,8,(i,j)->cmul(Clifford_basis[i],Clifford_basis[j]));

```

```
> evalm(M);
```

```
[ Id, e1, e2, e3, &CB(e1, e2), &CB(e1, e3), &CB(e2, e3), &CB(e1, e2, e3)
  e1, Id, e1we2, e1we3, e2, e3, &CB(e1, e2, e3), &CB(e2, e3)
  e2, -e1we2, Id, e2we3, -e1, -&CB(e1, e2, e3), e3, -&CB(e1, e3)
  e3, -e1we3, -e2we3, Id, &CB(e1, e2, e3), -e1, -e2, &CB(e1, e2)
  &CB(e1, e2), -e2, e1, &CB(e1, e2, e3), -Id, -&CB(e2, e3), &CB(e1, e3), -e3
  &CB(e1, e3), -e3, -&CB(e1, e2, e3), e1, &CB(e2, e3), -Id, -&CB(e1, e2), e2
  &CB(e2, e3), &CB(e1, e2, e3), -e3, e2, -&CB(e1, e3), &CB(e1, e2), -Id, -e1
  &CB(e1, e2, e3), &CB(e2, e3), -&CB(e1, e3), &CB(e1, e2), -e3, e2, -e1, -Id]
```

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

```
Worksheet took 5.249000 seconds to compute on AMD Athlon 2700+ 1GB RAM machine with Win XP Professional
```

```
>
```

See Also: [Clifford:-cmul](#), [Clifford:-clicollect](#), [Clifford:-`type/cliproduct`](#), [Cliplus:-clieval](#), [Cliplus:-cliexpand](#), [Cliplus:-clibasis](#)

(c) Copyright October 8, 1995, by Rafal Ablamowicz & Bertfried Fauser, all rights reserved.

Last modified: January 5, 2003, RA/BF.

Function: Cliplus:-clirev - extends procedure 'reversion' to polynomials expressed in the Clifford basis in Cl(B) or Cl(K)

Calling Sequence:

```
reversion(p);  
clirev(p);  
reversion(p,K);  
clirev(p,K)
```

Parameters:

p - polynomial whose terms are of type [Clifford:-`type/cliprod`](#).
k - (optional) argument of type name, symbol, matrix, or array, or type `&*(numeric,{name,symbol,array,matrix})`

Description:

- This procedure extends procedure [Clifford:-reversion](#) from 'CLIFFORD' to polynomials in Cl(B) expressed in terms of the Clifford basis in Cl(B).
- User can use now the modified procedure 'reversion' which now can be applied to a new type of input, that is, [`type/cliprod`](#).
- Observe that a second optional argument K may now be used of type name, symbol, matrix, array, or type `&*(numeric,{name,symbol,array,matrix})`: in that case, reversion is done with respect to the form K or its numeric multiple. When the optional argument is not specified, reversion is done, by default, with respect to the bilinear form B.
- It is also possible to mix Grassmann basis with the Clifford basis.
- When applied to a Clifford monomial, it reverses the order of the basis 1-vectors. For example, when reversion is applied to the basis element $e_1 \&C e_2 \&C e_3$, it will return $e_3 \&C e_2 \&C e_1$. Then, extend it by linearity to polynomial expressions.

Examples:

```
> restart:bench:=time():with(Clifford):with(Cliplus):  
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now include &C and &C[K]. Type  
?cliprod for help.  
> p:=e1 &C e2 &C e3;  
p1:=e1 &C e2;  
p2:=`&C`(e1,e2we3);  
  
p := (e1 &C e2) &C e3  
p1 := e1 &C e2  
p2 := e1 &C e2we3  
  
> "clirev(p)"=clirev(p);  
"reversion(p)"=reversion(p);  
"clirev(p1)"=clirev(p1);  
"reversion(p1)"=reversion(p1);  
"clirev(p2)"=clirev(p2);  
"reversion(p2)"=reversion(p2);  
  
"clirev(p)" = &C(e3, e2, e1)  
"reversion(p)" = &C(e3, e2, e1)  
"clirev(p1)" = e2 &C e1  
"reversion(p1)" = e2 &C e1  
"clirev(p2)" = e2we3 &C e1  
"reversion(p2)" = e2we3 &C e1  
  
> pp:=`&C`[B](e1,e2,e3);  
p11:=`&C`[K](e1,e2,e3);  
p22:=`&C`[-K](e1,e2,e3);  
  
pp := &CB(e1, e2, e3)  
p11 := &CK(e1, e2, e3)
```

$$p22 := \&C_{-K}(e1, e2, e3)$$

```
> reversion(pp); # reversion of pp w.r.t. B (use implicitly)
reversion(pp,B); # reversion of pp w.r.t. B (use explicitly)
&C(e3, e2, e1)
&C(e3, e2, e1)
```

Reversion of expression p11 defined above has to be done with respect to the same index. Otherwise, an error message will be printed:

```
> reversion(p11); ###<<<<-Desired error message
Error, (in Cliplus:-clierev) optional (or default B) parameter in clierev differs from indices encountered in its cliprod arguments. Found these names as indices of &C: {B, K}
```

```
> reversion(p11,K); # reversion of p11 w.r.t. K can be done because it matches index in
`&C`[K]
```

$$\&C_K(e3, e2, e1)$$

```
> reversion(p22,-K); # reversion of p22 w.r.t. -K
&C_{-K}(e3, e2, e1)
```

```
>
```

Notice in the above that since macro(reversion=clierev) is defined at the time when the package 'Cliplus' is loaded, user can enter name 'reversion', yet maple returns always its macro name 'clierev'. For example,

```
> 'reversion';
```

clierev

When 'reversion' acts on Clifford polynomial p expressed in the Grassmann basis, it returns reversed p in terms of the Grassmann basis:

```
> p:=e1we2-1+2*e2+e1we2we3; #original p expressed in the Grassmann basis
```

$$p := e1we2 - 1 + 2 e2 + e1we2we3$$

```
> p1:=reversion(p); #reversed p expressed in the Grassmann basis
```

$$p1 := -e1we2 + B_{2,1} Id - B_{1,2} Id - Id + B_{3,2} e1 - B_{2,3} e1 - B_{3,1} e2 + 2 e2 + B_{1,3} e2 + B_{2,1} e3 - B_{1,2} e3 - e1we2we3$$

When 'reversion' acts on Clifford polynomial p expressed in the Clifford basis, it returns reversed p in terms of the Clifford basis:

```
> p2:=cliexpand(p); #polynomial p expressed in the Clifford basis
```

$$p2 := (e1 \&C e2) - B_{1,2} Id - 1 + 2 e2 + \&C(e1, e2, e3) - B_{2,3} e1 + B_{1,3} e2 - B_{1,2} e3$$

```
> reversion(p2); #reversed p2 expressed in the Clifford basis
```

$$(e2 \&C e1) - B_{1,2} Id - 1 + 2 e2 + \&C(e3, e2, e1) - B_{2,3} e1 + B_{1,3} e2 - B_{1,2} e3$$

```
> p3:=clieval(%); #converting reversed p to the Grassmann basis
```

$$p3 := -e1we2 + B_{2,1} Id - B_{1,2} Id - Id + B_{3,2} e1 - B_{2,3} e1 - B_{3,1} e2 + 2 e2 + B_{1,3} e2 + B_{2,1} e3 - B_{1,2} e3 - e1we2we3$$

p1 and p3 now should be the same:

```
> p1 - p3;
```

$$0$$

'reversion' can take an optional argument K of type name, symbol, matrix, array, and it will then act on Clifford polynomial p in Cl(K). It will return reversed p in terms of the Grassmann basis using coefficients of the form K:

```
> p:=e1we2-1+2*e2+e1we2we3; #original p expressed in the Grassmann basis
```

$$p := e1we2 - 1 + 2 e2 + e1we2we3$$

```
> p1:=reversion(p,K); #reversed p expressed in the Grassmann basis
```

$$p1 := -e1we2 + K_{2,1} Id - K_{1,2} Id - Id + K_{3,2} e1 - K_{2,3} e1 - K_{3,1} e2 + 2 e2 + K_{1,3} e2 + K_{2,1} e3 - K_{1,2} e3 - e1we2we3$$

When 'reversion' acts on Clifford polynomial p expressed in the Clifford basis, it returns reversed p in terms of the Clifford basis:

```
> p2:=cliexpand(p,K); #polynomial p expressed in the Clifford basis
```

$$p2 := \&C_K(e1, e2) - K_{1,2} Id - 1 + 2 e2 + \&C_K(e1, e2, e3) - K_{2,3} e1 + K_{1,3} e2 - K_{1,2} e3$$

```
> reversion(p2); ###<<<<-Desired error message
```

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

The above error message is due to the fact that 'reversion' performs reversion with respect to the default form B while it has encountered index K as in $\&C[K](e1,e2)$:

```
> reversion(p2,K);
```

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

```
> p3:=clieval(%); #converting reversed p to the Grassmann basis
      p3 := -e1we2 + K_{2,1} Id - K_{1,2} Id - Id + K_{3,2} e1 - K_{2,3} e1 - K_{3,1} e2 + 2 e2 + K_{1,3} e2 + K_{2,1} e3 - K_{1,2} e3 - e1we2we3
```

p1 and p3 now should be the same:

```
> p1 - p3;
```

0

Reversion is always an anti-automorphism in the Clifford algebra Cl(B):

```
> p1:=4*(e2 &C e3) - 3*(e1 &C e2 &C e3) + 2*e3;
```

$$p1 := 4(e2 \&C e3) - 3((e1 \&C e2) \&C e3) + 2 e3$$

```
> p2:=1-e1+e3+2*(e1 &C e2);
```

$$p2 := 1 - e1 + e3 + 2(e1 \&C e2)$$

```
> p1p2:=cmul(p1,p2);
```

$$\begin{aligned} p1p2 := & -6 B_{2,1} \&C_B(e1, e2, e3) + 2 \&C_B(e1, e3) + 6 B_{1,2} B_{2,3} e1 + 6 B_{2,1} B_{2,3} e1 - 3 B_{2,1} \&C_B(e1, e3) \\ & - 3 B_{1,2} \&C_B(e1, e3) - 8 B_{2,2} \&C_B(e1, e3) + 4 \&C_B(e2, e3) + 3 B_{1,1} \&C_B(e2, e3) + 8 B_{1,2} \&C_B(e2, e3) \\ & + 8 B_{2,1} \&C_B(e2, e3) + 3 B_{3,1} \&C_B(e1, e2) + 8 B_{3,2} \&C_B(e1, e2) + 8 B_{2,3} \&C_B(e1, e2) - 3 B_{3,3} \&C_B(e1, e2) \\ & + 3 B_{1,3} \&C_B(e1, e2) - 2 B_{3,1} Id + 4 B_{3,3} e2 - 2 B_{1,3} Id + 2 B_{3,3} Id + 6 B_{1,2} B_{3,2} e1 - 6 e2 B_{3,2} B_{1,1} - 6 B_{1,2} \&C_B(e1, e2, e3) \\ & - 6 e1 B_{3,1} B_{2,2} - 6 B_{1,3} B_{2,2} e1 - 8 B_{2,1} B_{2,3} Id + 8 B_{2,2} B_{1,3} Id + 6 e3 B_{2,2} B_{1,1} + 8 Id B_{3,1} B_{2,2} + 2 e3 - 6 B_{2,3} B_{1,1} e2 \\ & - 3 \&C_B(e1, e2, e3) - 8 Id B_{3,2} B_{2,1} + 4 B_{1,2} e3 - 8 B_{3,2} B_{1,2} Id - 8 B_{2,3} Id B_{1,2} - 4 B_{3,2} e1 - 4 B_{2,3} e1 + 6 e1 B_{3,2} B_{2,1} \\ & + 4 B_{2,1} e3 \end{aligned}$$

We want to show that

$$\text{reversion}(\text{cmul}(p1,p2)) = \text{cmul}(\text{reversion}(p2),\text{reversion}(p1)).$$

```
> revp1p2:=clicollect(reversion(p1p2));
```

$$\begin{aligned} \text{revp1p2} := & -(3 B_{1,2} + 8 B_{2,2} + 3 B_{2,1} - 2)(e3 \&C e1) + (3 B_{1,1} + 8 B_{1,2} + 4 + 8 B_{2,1})(e3 \&C e2) \\ & - (3 B_{3,3} - 3 B_{3,1} - 8 B_{3,2} - 8 B_{2,3} - 3 B_{1,3})(e2 \&C e1) - 3(2 B_{2,1} + 2 B_{1,2} + 1) \&C(e3, e2, e1) \\ & + 2(4 B_{3,1} B_{2,2} + B_{3,3} - B_{3,1} - 4 B_{3,2} B_{2,1} - B_{1,3} - 4 B_{1,2} B_{2,3} + 4 B_{1,3} B_{2,2} - 4 B_{2,1} B_{2,3} - 4 B_{1,2} B_{3,2}) Id \\ & - 2(-3 B_{2,1} B_{2,3} + 3 B_{3,1} B_{2,2} + 3 B_{1,3} B_{2,2} + 2 B_{3,2} - 3 B_{1,2} B_{2,3} - 3 B_{1,2} B_{3,2} + 2 B_{2,3} - 3 B_{3,2} B_{2,1}) e1 \\ & - 2(-2 B_{3,3} + 3 B_{3,2} B_{1,1} + 3 B_{2,3} B_{1,1}) e2 + 2(3 B_{2,2} B_{1,1} + 2 B_{1,2} + 1 + 2 B_{2,1}) e3 \end{aligned}$$

```
> p11:=reversion(p1);
```

$$p11 := 4(e3 \&C e2) - 3 \&C(e3, e2, e1) + 2 e3$$

```
> p22:=reversion(p2);
```

$$p22 := 1 - e1 + e3 + 2(e2 \&C e1)$$

```
> Revp1p2:=cmul(p22,p11);
```

$$\begin{aligned} \text{Revp1p2} := & 6 B_{2,1} \&C_B(e1, e2, e3) - 3 B_{3,3} B_{1,2} Id - 2 \&C_B(e1, e3) + 3 B_{2,1} \&C_B(e1, e3) + 3 B_{1,2} \&C_B(e1, e3) \\ & + 8 B_{2,2} \&C_B(e1, e3) - 4 \&C_B(e2, e3) - 6 e3 B_{2,1}^2 - 3 B_{1,1} \&C_B(e2, e3) - 8 B_{1,2} \&C_B(e2, e3) - 8 B_{2,1} \&C_B(e2, e3) \\ & - 3 B_{3,1} \&C_B(e1, e2) - 8 B_{3,2} \&C_B(e1, e2) - 8 B_{2,3} \&C_B(e1, e2) + 3 B_{3,3} \&C_B(e1, e2) - 3 B_{1,3} \&C_B(e1, e2) + 4 B_{3,3} e2 \\ & + 2 B_{3,3} Id + 4 B_{2,3} Id + 6 B_{2,1} B_{1,3} e2 + 6 B_{1,2} B_{1,3} e2 + 3 B_{1,3} e2 - 6 B_{1,2}^2 e3 - 6 B_{3,2} B_{1,1} e2 + 3 B_{1,1} B_{2,3} Id \\ & + 6 B_{1,2} \&C_B(e1, e2, e3) - 6 B_{3,1} B_{2,2} e1 - 12 e3 B_{2,1} B_{1,2} - 3 B_{3,3} Id B_{2,1} - 6 B_{1,3} B_{2,2} e1 + 3 B_{3,2} B_{1,1} Id + 4 B_{3,2} Id \\ & + 3 B_{3,1} e2 + 8 B_{2,1} B_{2,3} Id + 6 B_{2,2} B_{1,1} e3 + 6 B_{3,1} e2 B_{2,1} + 6 B_{3,1} B_{1,2} e2 + 2 e3 - 6 B_{2,3} B_{1,1} e2 + 3 \&C_B(e1, e2, e3) \\ & + 8 B_{3,2} B_{2,1} Id + B_{1,2} e3 + 8 B_{3,2} B_{1,2} Id + 8 B_{2,3} B_{1,2} Id - 7 B_{3,2} e1 - 7 B_{2,3} e1 + B_{2,1} e3 \end{aligned}$$

```
> clieval(Revp1p2-revp1p2);
```

0

The same computation as above except this time with respect to some arbitrary form K:

```
> p1:=4*`&C`[K](e2,e3) - 3*`&C`[K](e1,e2,e3) + 2*e3;
```

$$p1 := 4 \&C_K(e2, e3) - 3 \&C_K(e1, e2, e3) + 2 e3$$

```
> p2:=1-e1+e3+2*`&C`[K](e1,e2);
```

$$p2 := 1 - e1 + e3 + 2 \&C_K(e1, e2)$$

> **p1p2:=cmul[K](p1,p2);**

$$\begin{aligned} p1p2 := & -4 K_{2,3} e1 + 4 K_{1,2} e3 + 4 K_{2,1} e3 + 6 K_{2,1} K_{2,3} e1 - 6 K_{2,1} \&C_K(e1, e2, e3) - 6 K_{1,2} \&C_K(e1, e2, e3) \\ & + 6 K_{1,2} K_{2,3} e1 + 4 \&C_K(e2, e3) - 3 \&C_K(e1, e2, e3) + 2 \&C_K(e1, e3) - 3 K_{1,2} \&C_K(e1, e3) - 8 K_{2,2} \&C_K(e1, e3) \\ & - 3 K_{2,1} \&C_K(e1, e3) - 8 K_{2,1} K_{2,3} Id + 8 K_{2,2} K_{1,3} Id + 8 K_{2,1} \&C_K(e2, e3) + 3 K_{1,1} \&C_K(e2, e3) + 8 K_{1,2} \&C_K(e2, e3) \\ & + 8 K_{2,3} \&C_K(e1, e2) + 8 K_{3,2} \&C_K(e1, e2) + 3 K_{3,1} \&C_K(e1, e2) + 3 K_{1,3} \&C_K(e1, e2) - 8 K_{3,2} K_{1,2} Id \\ & - 3 K_{3,3} \&C_K(e1, e2) - 4 K_{3,2} e1 - 6 K_{2,3} K_{1,1} e2 + 2 e3 + 8 Id K_{3,1} K_{2,2} - 8 Id K_{3,2} K_{2,1} + 6 e1 K_{3,2} K_{2,1} - 8 K_{2,3} Id K_{1,2} \\ & - 6 e2 K_{3,2} K_{1,1} + 6 K_{1,2} K_{3,2} e1 - 6 K_{1,3} K_{2,2} e1 + 4 K_{3,3} e2 - 2 K_{1,3} Id + 2 K_{3,3} Id - 2 K_{3,1} Id - 6 e1 K_{3,1} K_{2,2} \\ & + 6 e3 K_{2,2} K_{1,1} \end{aligned}$$

We want to show that

$$\text{reversion}(\text{cmul}(p1,p2)) = \text{cmul}(\text{reversion}(p2),\text{reversion}(p1)).$$

> **revp1p2:=clicollect(reversion(p1p2,K));**

$$\begin{aligned} revp1p2 := & (8 K_{3,2} - 3 K_{3,3} + 3 K_{1,3} + 8 K_{2,3} + 3 K_{3,1}) \&C_K(e2, e1) - 3 (2 K_{2,1} + 2 K_{1,2} + 1) \&C_K(e3, e2, e1) \\ & + (3 K_{1,1} + 8 K_{1,2} + 4 + 8 K_{2,1}) \&C_K(e3, e2) - (3 K_{1,2} + 3 K_{2,1} + 8 K_{2,2} - 2) \&C_K(e3, e1) \\ & - 2 (4 K_{1,2} K_{2,3} + 4 K_{2,1} K_{2,3} - 4 K_{2,2} K_{1,3} + 4 K_{3,2} K_{1,2} + K_{3,1} - 4 K_{3,1} K_{2,2} + 4 K_{3,2} K_{2,1} + K_{1,3} - K_{3,3}) Id \\ & + 2 (-2 K_{2,3} - 3 K_{3,1} K_{2,2} + 3 K_{3,2} K_{2,1} + 3 K_{2,1} K_{2,3} + 3 K_{1,2} K_{2,3} - 3 K_{2,2} K_{1,3} + 3 K_{3,2} K_{1,2} - 2 K_{3,2}) e1 \\ & - 2 (3 K_{3,2} K_{1,1} - 2 K_{3,3} + 3 K_{2,3} K_{1,1}) e2 + 2 (1 + 2 K_{2,1} + 2 K_{1,2} + 3 K_{2,2} K_{1,1}) e3 \end{aligned}$$

> **p11:=reversion(p1,K);**

$$p11 := 4 \&C_K(e3, e2) - 3 \&C_K(e3, e2, e1) + 2 e3$$

> **p22:=reversion(p2,K);**

$$p22 := 1 - e1 + e3 + 2 \&C_K(e2, e1)$$

> **Revplp2:=cmul[K](p22,p11);**

$$\begin{aligned} Revplp2 := & -7 K_{2,3} e1 + 6 K_{2,1} K_{1,3} e2 + 6 K_{1,2} K_{1,3} e2 + 4 K_{3,2} Id - 6 K_{1,2}^2 e3 + 3 K_{1,3} e2 + K_{1,2} e3 + 3 K_{3,1} e2 + K_{2,1} e3 \\ & + 4 K_{2,3} Id + 6 K_{2,1} \&C_K(e1, e2, e3) + 6 K_{1,2} \&C_K(e1, e2, e3) - 3 K_{3,3} K_{1,2} Id - 4 \&C_K(e2, e3) + 3 \&C_K(e1, e2, e3) \\ & - 2 \&C_K(e1, e3) + 3 K_{3,2} K_{1,1} Id + 6 K_{3,1} e2 K_{2,1} + 6 K_{3,1} K_{1,2} e2 + 3 K_{1,2} \&C_K(e1, e3) + 8 K_{2,2} \&C_K(e1, e3) \\ & + 3 K_{2,1} \&C_K(e1, e3) + 8 K_{2,1} K_{2,3} Id - 3 K_{3,3} Id K_{2,1} - 8 K_{2,1} \&C_K(e2, e3) - 3 K_{1,1} \&C_K(e2, e3) - 8 K_{1,2} \&C_K(e2, e3) \\ & - 6 e3 K_{2,1}^2 - 8 K_{2,3} \&C_K(e1, e2) - 8 K_{3,2} \&C_K(e1, e2) - 3 K_{3,1} \&C_K(e1, e2) - 3 K_{1,3} \&C_K(e1, e2) + 8 K_{3,2} K_{1,2} Id \\ & + 3 K_{3,3} \&C_K(e1, e2) - 7 K_{3,2} e1 - 6 K_{2,3} K_{1,1} e2 + 2 e3 + 8 K_{3,2} K_{2,1} Id + 8 K_{2,3} K_{1,2} Id - 6 K_{3,2} K_{1,1} e2 - 6 K_{1,3} K_{2,2} e1 \\ & + 4 K_{3,3} e2 + 2 K_{3,3} Id - 12 e3 K_{2,1} K_{1,2} - 6 K_{3,1} K_{2,2} e1 + 6 K_{2,2} K_{1,1} e3 + 3 K_{1,1} K_{2,3} Id \end{aligned}$$

> **clieval(Revplp2-revp1p2);**

$$0$$

Another short example:

> **p1:=Id+2*(e1 &C e2) + 3*&C(e1,e2,e3);**

$$p1 := Id + 2 (e1 \&C e2) + 3 \&C(e1, e2, e3)$$

> **reversion(p1);**

$$Id + 2 (e2 \&C e1) + 3 \&C(e3, e2, e1)$$

> **p2:=p1+e2we3we4;**

$$p2 := Id + 2 (e1 \&C e2) + 3 \&C(e1, e2, e3) + e2we3we4$$

Mixed input can be used in 'reversion':

> **reversion(p2);**

$$Id + 2 (e2 \&C e1) + 3 \&C(e3, e2, e1) - e2we3we4 + B_{3,2} e4 - B_{2,3} e4 + B_{4,3} e2 - B_{3,4} e2 - B_{4,2} e3 + B_{2,4} e3$$

To express the above output solely in terms of the Clifford basis, use 'cliexpand':

> **cliexpand(%);**

$$Id + 2 (e2 \&C e1) + 3 \&C(e3, e2, e1) - \&C(e2, e3, e4) + B_{3,2} e4 + B_{4,3} e2 - B_{4,2} e3$$

[To express the above output solely in terms of the Grassmann basis, use 'clieval':

[> **clieval(%);**

$Id - 2 e1we2 + 2 B_{2,1} Id - 3 e1we2we3 - 3 B_{3,1} e2 + 3 B_{2,1} e3 + 3 B_{3,2} e1 - e2we3we4 - B_{3,4} e2 + B_{2,4} e3 - B_{2,3} e4 + B_{3,2} e4$
 $+ B_{4,3} e2 - B_{4,2} e3$

[> **printf("Worksheet took %f seconds to compute on AMD Athlon 2700+ 1GB RAM Win XP Prof\n",time()-bench);**

[Worksheet took 3.264000 seconds to compute on AMD Athlon 2700+ 1GB RAM machine with Win XP Prof

[>

See Also: [Clifford:-cmul](#), [Clifford:-clicollect](#), [Clifford:-`type/cliproduct`](#), [Cliplus:-clieval](#), [Cliplus:-cliexpand](#), [Clifford:-reversion](#)

(c) Copyright October 8, 1995, by Rafal Ablamowicz & Bertfried Fauser, all rights reserved.

Last modified: January 5, 2003, RA/BF.

Function: Cliplus:-dottedcbasis - create dotted Grassmann basis (expressed in terms of the Grassmann standard wedge basis)

Calling Sequence:

```
L := dottedcbasis[K](n);
L := dottedcbasis[K](n,'even');
L := dottedcbasis[K](n,k);
```

Parameters:

- n - positive integer between 1 and 9
- k - non-negative integer between 0 and n
- K - antisymmetric matrix, or a name, symbol, array, `&*(numeric,{name,symbol,array,matrix})

Output:

- L : a list of Clifford [clibasmons](#), [climons](#) and/or [clipolynoms](#) that make up dotted wedge basis

Description:

- This function is similar to [Clifford:-cbasis](#): while the latter gives standard Grassmann wedge basis, this procedure returns a basis for the Clifford algebra $Cl(V,B)$ where $\dim_V = n$. When the second positive integer k is used, the basis returned contains only those basis polynomials of maximum grade exactly equal to k.
- **NOTE:** Until now both types of algebras are expanded (formally) over the same basis Grassmann monomials which, according to CLIFFORD's convention, are written as eiwej... . 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 to assign such expressions to a descriptive lhs, see below.

Examples:

```
> restart;with(Clipford):with(Cliplus);
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, makeclialias]
```

Example 1: Procedure 'dottedcbasis' displays dotted basis for $Cl(V)$ while procedure [cbasis](#) displays Grassmann basis for $Cl(V)$:

```
> dottedcbasis[F](3);
dottedcbasis[-F](3);
[Id, e1, e2, e3, e1we2 + F1,2 Id, e1we3 + F1,3 Id, e2we3 + F2,3 Id, e1we2we3 + F2,3 e1 - F1,3 e2 + F1,2 e3]
[Id, e1, e2, e3, e1we2 - F1,2 Id, e1we3 - F1,3 Id, e2we3 - F2,3 Id, e1we2we3 - F2,3 e1 + F1,3 e2 - F1,2 e3]
> cbasis(3);
[Id, e1, e2, e3, e1we2, e1we3, e2we3, e1we2we3]
```

No error message appears when unassigned symbol 'g' is used but when 'g' gets assigned a non-antisymmetric matrix, an error will be reported:

```
> dottedcbasis[g](3);
[Id, e1, e2, e3, e1we2 + g1,2 Id, e1we3 + g1,3 Id, e2we3 + g2,3 Id, e1we2we3 + g2,3 e1 - g1,3 e2 + g1,2 e3]
> g:=array(1..3,1..3,symmetric):
> dottedcbasis[g](3); ## <<<-- Expected error message
Error, (in Cliplus:-dottedcbasis[g]) index is expected to be an antisymmetric matrix or array, or, name
or symbol
>
```

Example 2: (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 $-F = FT$, respectively:

```
> dim_V:=3:
F:=array(1..dim_V,1..dim_V,antisymmetric):
FT:=linalg[transpose](F):
F,FT = evalm(F),evalm(FT);
```

```
w_bas:=cbasis(dim_V); ## the wedge basis
B:=evalm(g+F);
```

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

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

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

[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);
test_wbas:=map(convert,d_bas,dwedge_to_wedge,FT);
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 := [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 $e1we2 + F[1,2]*Id$ by $e1de2$ 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]
>
```

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

- Function: `Cliplus:-`dwedge``, `Cliplus:-`&dw`` - Grassmann wedge product for a different filtration

Calling Sequence:

```
c1 := dwedge[K](p1,p2,...,pn)
c1 := &dw[K](p1,p2,...,pn)
```

Parameters:

- p_1, p_2, \dots, p_n - Clifford polynoms (elements of one of these types: ``type/clibasmon``, ``type/climon``, ``type/clipolynom``)
- K - index of type name, symbol, matrix, array, or ``&*(numeric, {name,symbol,matrix,array})``

Output:

- c_1 : a Clifford polynom

- Description:

- The dotted-wedge (`dwedge`) 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 .
- Procedure `'dwedge'` requires one index of type name, symbol, matrix, array, ``&*(numeric, {name,symbol,matrix,array})``. When the index is a matrix or an array, it must be antisymmetric.
- It can be shown that the Wick-theorem of normal ordering, well known in QFT and many particle QM, is exactly described by this process.
- While being isomorphic as Grassmann algebras and hence being interchangeable, the difference becomes important when further structures are considered. For example, when a Clifford algebra is build over the space of this differently graded Grassmann algebra, or when quantum deformations are modeled within an undeformed Clifford algebra, etc..
- The dotted wedge is a wrapper function which actually uses ``convert/wedge_to_dwedge`` and ``convert/dwedge_to_wedge`` to map between the two basis sets. This is possible since the new Grassmann algebra is a cliffordized Grassmann algebra w.r.t. a bilinear form F as stated above.
- The ampersand version of this procedure is called ``&dw``.
- **NOTE:** Until 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, 2003.
- [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,

Examples:

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

Clipplus 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, makeclialias]

Example 1: Simple examples first:

```
> dwedge[K](e1+2*e1we3,e4+3*e1we2);
```

```
dwedge[-K](e1+2*e1we3,e4+3*e1we2);
```

```
&dw(e1+2*e1we3,e4+3*e1we2); #default index in `&dw` is F
```

```
&dw[-F](e1+2*e1we3,e4+3*e1we2);
```

$$2 e1we3we4 + e1we4 + (K_{1,4} - 6 K_{1,3} K_{1,2}) Id - 6 K_{1,2} e1we3 - 6 K_{1,3} e1we2 + (2 K_{3,4} - 3 K_{1,2}) e1 - 2 K_{1,4} e3$$

$$2 e1we3we4 + e1we4 - (K_{1,4} + 6 K_{1,3} K_{1,2}) Id + 6 K_{1,2} e1we3 + 6 K_{1,3} e1we2 - (2 K_{3,4} - 3 K_{1,2}) e1 + 2 K_{1,4} e3$$

$$2 e1we3we4 + e1we4 - (-F_{1,4} + 6 F_{1,3} F_{1,2}) Id - 6 F_{1,2} e1we3 - 6 F_{1,3} e1we2 - (-2 F_{3,4} + 3 F_{1,2}) e1 - 2 F_{1,4} e3$$

$$2 e1we3we4 + e1we4 - (F_{1,4} + 6 F_{1,3} F_{1,2}) Id + 6 F_{1,2} e1we3 + 6 F_{1,3} e1we2 + (-2 F_{3,4} + 3 F_{1,2}) e1 + 2 F_{1,4} e3$$

```
>
```

Example 2: Observe that conversion from the undotted wedge basis to the dotted wedge basis using antisymmetric form F and 'dwedge[F]' are related through the following identity:

```
convert(e1we2w...wen,wedge_to_dwedge,F) = dwedge[F](e1,e2,...,en)
```

which can be shown as follows in dim_V <=5:

```
> F:=array(1..9,1..9,antisymmetric):
```

```
> ##when dim_V = 2:
```

```
simplify(dwedge[F](e1,e2)=convert(wedge(e1,e2),wedge_to_dwedge,F));
```

$$e1we2 + F_{1,2} Id = e1we2 + F_{1,2} Id$$

```
> ##when dim_V = 3:
```

```
simplify(dwedge[F](e1,e2,e3)=convert(wedge(e1,e2,e3),wedge_to_dwedge,F));
```

$$e1we2we3 + F_{2,3} e1 - F_{1,3} e2 + F_{1,2} e3 = e1we2we3 + F_{2,3} e1 - F_{1,3} e2 + F_{1,2} e3$$

```
> ##when dim_V = 4:
```

```
simplify(dwedge[F](e1,e2,e3,e4)=convert(wedge(e1,e2,e3,e4),wedge_to_dwedge,F));
```

$$e1we2we3we4 + F_{2,3} e1we4 + F_{2,3} F_{1,4} Id - F_{1,3} e2we4 - F_{1,3} F_{2,4} Id + F_{1,2} e3we4 + F_{1,2} F_{3,4} Id - F_{2,4} e1we3 + F_{1,4} e2we3$$

$$+ F_{3,4} e1we2 = e1we2we3we4 + F_{2,3} e1we4 + F_{2,3} F_{1,4} Id - F_{1,3} e2we4 - F_{1,3} F_{2,4} Id + F_{1,2} e3we4 + F_{1,2} F_{3,4} Id$$

$$- F_{2,4} e1we3 + F_{1,4} e2we3 + F_{3,4} e1we2$$

```
> ##when dim_V = 5:
```

```
simplify(dwedge[F](e1,e2,e3,e4,e5)-convert(wedge(e1,e2,e3,e4,e5),wedge_to_dwedge,F));
```

0

```
>
```

Example 3: Operation 'dwedge' is associative with Id as a unit:

```
> dwedge[F](dwedge[F](e1,e2),e3);
```

```
dwedge[F](e1,dwedge[F](e2,e3));
```

$$e1we2we3 + F_{2,3} e1 - F_{1,3} e2 + F_{1,2} e3$$

$$e1we2we3 + F_{2,3} e1 - F_{1,3} e2 + F_{1,2} e3$$

```
> dwedge[F](dwedge[F](e1,e2we3),e4);
```

```
dwedge[F](e1,dwedge[F](e2we3,e4));%%-%;
```

$$e1we2we3we4 - F_{1,3} e2we4 + F_{1,2} e3we4 - (F_{1,3} F_{2,4} - F_{1,2} F_{3,4}) Id - F_{2,4} e1we3 + F_{1,4} e2we3 + F_{3,4} e1we2$$

$$e1we2we3we4 - F_{1,3} e2we4 + F_{1,2} e3we4 - (F_{1,3} F_{2,4} - F_{1,2} F_{3,4}) Id - F_{2,4} e1we3 + F_{1,4} e2we3 + F_{3,4} e1we2$$

0

```
> dwedge[F](dwedge[F](e1,e2we3),e4we5);
```

```
dwedge[F](e1,dwedge[F](e2we3,e4we5));%%-%;
```

$$\begin{aligned}
& F_{2,5} e1we3we4 - (F_{1,3} F_{2,4} - F_{1,2} F_{3,4}) e5 + e1we2we3we4we5 - F_{3,5} e1we2we4 - F_{1,5} e2we3we4 - F_{1,3} e2we4we5 \\
& + F_{1,2} e3we4we5 - F_{2,4} e1we3we5 + F_{1,4} e2we3we5 + F_{3,4} e1we2we5 - (F_{2,4} F_{3,5} - F_{3,4} F_{2,5}) e1 \\
& - (-F_{1,3} F_{2,5} + F_{1,2} F_{3,5}) e4 + (F_{1,4} F_{3,5} - F_{3,4} F_{1,5}) e2 + (-F_{1,4} F_{2,5} + F_{2,4} F_{1,5}) e3 \\
& F_{2,5} e1we3we4 - (F_{1,3} F_{2,4} - F_{1,2} F_{3,4}) e5 + e1we2we3we4we5 - F_{3,5} e1we2we4 - F_{1,5} e2we3we4 - F_{1,3} e2we4we5 \\
& + F_{1,2} e3we4we5 - F_{2,4} e1we3we5 + F_{1,4} e2we3we5 + F_{3,4} e1we2we5 - (F_{2,4} F_{3,5} - F_{3,4} F_{2,5}) e1 \\
& - (-F_{1,3} F_{2,5} + F_{1,2} F_{3,5}) e4 + (F_{1,4} F_{3,5} - F_{3,4} F_{1,5}) e2 + (-F_{1,4} F_{2,5} + F_{2,4} F_{1,5}) e3 \\
& 0
\end{aligned}$$

Finally, for some arbitrary random Clifford polynomials expressed in Grassmann undotted basis:

```

> u:=2+e1-3*e2we3+e4we5we6:
v:=3-4*e1we2we3+e7:
z:=4-2*e3we4+e5we6-e8:
> dwedge[F](Id,u)=u; #unity
dwedge[F](u,Id)=u;
e4we5we6 + 2 Id - 3 e2we3 + e1 = 2 + e1 - 3 e2we3 + e4we5we6
e4we5we6 + 2 Id - 3 e2we3 + e1 = 2 + e1 - 3 e2we3 + e4we5we6
> dwedge[F](dwedge[F](u,v),z):#associativity
dwedge[F](u,dwedge[F](v,z)):%%-%;
0

```

We also have the following **Commutative Diagram 5: Wedge in undotted and dwedge in dotted bases:**

$$\text{wedge}(u,v) = \text{convert}(\text{dwedge}(\text{convert}(u,\text{wedge_to_dwedge},F),\text{convert}(v,\text{wedge_to_dwedge},F)),\text{dwedge_to_wedge},-F)$$

which we show as follows:

```

> uu:=convert(u,wedge_to_dwedge,F); #u converted to dotted basis
vv:=convert(v,wedge_to_dwedge,F); #v converted to dotted basis
uu := 2 Id + e1 - 3 e2we3 - 3 F_{2,3} Id + e4we5we6 + F_{5,6} e4 - F_{4,6} e5 + F_{4,5} e6
vv := 3 Id - 4 e1we2we3 - 4 F_{2,3} e1 + 4 F_{1,3} e2 - 4 F_{1,2} e3 + e7
> out1:=dwedge[F](uu,vv): #dwedge computed w.r.t. F
> out2:=convert(out1,dwedge_to_wedge,-F); #previous result converted back to undotted basis
out2 := -9 e2we3 + e4we5we6we7 + 4 e1we2we3we4we5we6 + 2 e7 - 8 e1we2we3 + 3 e4we5we6 + e1we7 - 3 e2we3we7
+ 6 Id + 3 e1
> out3:=wedge(u,v); #direct computation of the wedge product in undotted basis
out3 := -9 e2we3 + e4we5we6we7 + 4 e1we2we3we4we5we6 + 2 e7 - 8 e1we2we3 + 3 e4we5we6 + e1we7 - 3 e2we3we7
+ 6 Id + 3 e1
> out2-out3; #the same results!
0
>

```

Example 4: (Dotted and undotted wedge bases) First we expand the basis of the original wedge into the dotted wedge and back. For this purpose we choose $\dim_V=3$ and consider $Cl(C,B)$ where the antisymmetric part of B is denoted by F (and its negative by FT), while the symmetric part of B is denoted by g .

```

> dim_V:=3:
F:=array(1..dim_V,1..dim_V,antisymmetric):
g:=array(1..dim_V,1..dim_V,symmetric):
B:=evalm(g+F):
FT:=evalm(-F):
F,FT = evalm(F),evalm(FT);
g,B = evalm(g),evalm(B);
w_bas:=cbasis(dim_V); ## the wedge basis

```

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

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

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

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);
test_wbas:=map(convert,d_bas,dwedge_to_wedge,-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 := [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 $e1we2 + F[1,2]*Id$ by $e1de2$ 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. For example, procedure 'dottedcbasis' returns such basis. 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 5: (Commutative Diagram 1: Contraction in dotted and undotted bases) The contraction w.r.t. any bilinear form works on both sets in the **same manner** which can be seen if we re-convert the dotted-wedge basis after the computation into the wedge basis. In a reasonable setting, the antisymmetric bilinear form F would be the antisymmetric part of B . To read more about the left contraction LC in $Cl(B)$, go to the [help page for LC](#) or see [1, 2, 4]. To illustrate this fact, we first compute left contraction by $e1$ of every element in the standard Grassmann wedge basis w_bas with respect to the entire form B :

```
> 'w_bas'=w_bas; #standard Grassmann wedge basis in Cl(B)
w_bas = [Id, e1, e2, e3, e1we2, e1we3, e2we3, e1we2we3]
> w_wout:=map2(LC,e1,w_bas,B);#left contraction LC in Cl(B) w.w.t. B in wedge basis
w_wout := [0, g_{1,1} Id, (g_{1,2} + F_{1,2}) Id, (g_{1,3} + F_{1,3}) Id, g_{1,1} e2 - (g_{1,2} + F_{1,2}) e1, g_{1,1} e3 - (g_{1,3} + F_{1,3}) e1,
```

$$(g_{1,2} + F_{1,2}) e3 - (g_{1,3} + F_{1,3}) e2, g_{1,1} e2we3 - (g_{1,2} + F_{1,2}) e1we3 + (g_{1,3} + F_{1,3}) e1we2]$$

Next, we compute left contraction by e1 of every element in the dotted wedge basis d_bas with respect to the entire form B. Recall from the above that conversion from the wedge basis to the dotted wedge basis used the antisymmetric part F of B:

```
> 'd_bas'=d_bas; #dotted wedge basis in Cl(B)
      d_bas=[Id, e1, e2, e3, e1We2, e1We3, e2We3, e1We2We3]
> w_dout1:=map2(LC,e1,d_bas,B);#left contraction LC in Cl(B) w.w.t. B in dotted wedge basis
w_dout1 := [0, g1,1 Id, (g1,2 + F1,2) Id, (g1,3 + F1,3) Id, g1,1 e2 - (g1,2 + F1,2) e1, g1,1 e3 - (g1,3 + F1,3) e1,
            (g1,2 + F1,2) e3 - (g1,3 + F1,3) e2,
            g1,1 e2we3 - (g1,2 + F1,2) e1we3 + (g1,3 + F1,3) e1we2 + F2,3 g1,1 Id - F1,3 Id g1,2 + F1,2 Id g1,3]
```

Notice that in the above coefficients of g, the symmetric part of B, are mixed with the coefficients of the antisymmetric part F of B. To remove the F coefficients, we need to convert back the above result to the un-dotted standard Grassmann basis using the negative -F, that is, the negative of the antisymmetric part of B in the conversion process:

```
> w_dout:=map(convert,w_dout1,dwedge_to_wedge,-F);#converting back to undotted basis
w_dout := [0, g1,1 Id, Id g1,2 + F1,2 Id, Id g1,3 + F1,3 Id, g1,1 e2 - e1 g1,2 - e1 F1,2, g1,1 e3 - e1 g1,3 - e1 F1,3,
            e3 g1,2 + F1,2 e3 - e2 g1,3 - F1,3 e2, g1,1 e2we3 - e1we3 g1,2 - e1we3 F1,2 + e1we2 g1,3 + e1we2 F1,3]
> map(simplify,w_dout-w_wout);
      [0, 0, 0, 0, 0, 0, 0, 0]
```

>

This computation shows clearly the isomorphy between both pictures. To show that the new structure is nevertheless valuable for other reasons, we proceed with Clifford products.

Example 6: (Commutative Diagram 2: Clifford product in dotted and undotted bases) We can build a Clifford algebra Cl(B) over each basis set, that is, w_wout or w_dout, but with different bilinear forms: when B=g and when B=g+F (following notation from [1, 2, 4]), where g is the symmetric part of B and F is the antisymmetric part of B:

```
> B,g,F=evalm(B),evalm(g),evalm(F); #previously defined
      B, g, F = 
$$\begin{bmatrix} g_{1,1} & g_{1,2} + F_{1,2} & g_{1,3} + F_{1,3} \\ g_{1,2} - F_{1,2} & g_{2,2} & g_{2,3} + F_{2,3} \\ g_{1,3} - F_{1,3} & g_{2,3} - F_{2,3} & g_{3,3} \end{bmatrix}, \begin{bmatrix} g_{1,1} & g_{1,2} & g_{1,3} \\ g_{1,2} & g_{2,2} & g_{2,3} \\ g_{1,3} & g_{2,3} & g_{3,3} \end{bmatrix}, \begin{bmatrix} 0 & F_{1,2} & F_{1,3} \\ -F_{1,2} & 0 & F_{2,3} \\ -F_{1,3} & -F_{2,3} & 0 \end{bmatrix}$$

```

Let us compute some such Clifford products using the facility of [cmul](#) to take a bilinear form (here in matrix form) as index. We will show an example with the following two elements:

```
> w_p1:=e1we2;
      w_p2:=a*e3+b*e2we3;
      w_p1 := e1we2
      w_p2 := a e3 + b e2we3
```

We can then define Clifford product 'cmulg' with respect to the symmetric part g, and another Clifford product 'cmulB' with respect to the entire form B:

```
> cmulg:=proc() RETURN(cmul[g](args)) end;
      cmulB:=proc() RETURN(cmul[B](args)) end;
```

Thus, we are ready to perform computations around our commutative diagram.

First, we compute Clifford product cmul[g] in Cl(g), that is, with respect to the symmetric part g of the bilinear form B, of the two above defined elements w_p1 and w_p2 expressed in undotted Grassmann basis.

```
> w_out1:=cmulg(w_p1,w_p2); ## Clifford product w.r.t. g in Cl(g) in wedge basis
      w_out1 := a e1we2we3 - b (-g2,2 g1,3 + g2,3 g1,2) Id + b g2,2 e1we3 - b g1,2 e2we3 - b g2,3 e1we2 + a g2,3 e1 - a g1,3 e2
```

Now, we convert each element p1 and p2 to the dotted wedge basis:

```
> d_p1:=convert(w_p1,wedge_to_dwedge,F);
      d_p2:=convert(w_p2,wedge_to_dwedge,F); #incomplete conversion to e1We2, etc. basis
      d_p1 := e1We2
      d_p2 := a e3 + b e2we3 + b F2,3 Id
```

We now compute the Clifford product of d_p1 and d_p2 in Cl(B) in the dotted wedge basis:

```
> d_out1:=cmulB(d_p1,d_p2); ## Clifford product w.r.t. B=g+F in Cl(B) in dwedge basis
```

$$d_out1 := a e1we2we3 - b (g_{2,3} F_{1,2} - g_{2,2} g_{1,3} - g_{2,2} F_{1,3} + g_{2,3} g_{1,2} + F_{2,3} g_{1,2}) Id + b g_{2,2} e1we3 - b g_{1,2} e2we3 - b g_{2,3} e1we2 + a (g_{2,3} + F_{2,3}) e1 - a (g_{1,3} + F_{1,3}) e2 + F_{1,2} a e3$$

We now convert the above result back to the un-dotted wedge basis:

```
> w_out2:=convert(d_out1,dwedge_to_wedge,-F); ## convert result dwedge-> wedge
```

$$w_out2 := a e1we2we3 + b Id g_{2,2} g_{1,3} - b Id g_{2,3} g_{1,2} + b g_{2,2} e1we3 - b g_{1,2} e2we3 - b g_{2,3} e1we2 + a g_{2,3} e1 - a g_{1,3} e2$$

Finally, we show that this result is the same as before when we computed Clifford product of p1 and p2 in Cl(g):

```
> simplify(w_out1-w_out2); ## show equality !
```

$$0$$

```
>
```

This shows (one can prove this) that the Clifford algebra Cl(g) of the symmetric part g of B using the undotted exterior basis is isomorphic, as an associative algebra, to the Clifford algebra Cl(B) of the entire bilinear form B = g + F spanned by the dotted wedge basis if the antisymmetric part F of B is exactly the same F as is used to connect the two basis sets (cf. [1, 2, 4]).

Example 7: (Commutative Diagram 3: 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,B); #reversion with respect to B
reversion(e1we2,g); #reversion with respect to g (classical result)
```

$$-2 F_{1,2} Id - e1we2$$

$$-e1we2$$

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);
```

$$[Id, e1, e2, e3, -2 F_{1,2} Id - e1we2, -2 F_{1,3} Id - e1we3, -2 F_{2,3} Id - e2we3, -e1we2we3 - 2 F_{2,3} e1 + 2 F_{1,3} e2 - 2 F_{1,2} e3]$$

If instead of B we use a symmetric matrix 'g' defined above, we obtain instead

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

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

Convert now e1we2 to the dotted basis and call it e1We2:

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

$$e1We2$$

Apply reversion to e1We2 with respect to F to get the reversed element in the dotted basis:

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

$$reversed_e1We2 := -F_{1,2} Id - e1we2$$

Observe, that the above element is equal to the negative of e1We2 just like reversing e1we2 with respect to the symmetric part g of B:

```
> reversed_e1We2+e1We2;
```

$$0$$

Finally, convert reversed_e1We2 to the un-dotted standard Grassmann basis to get -e1we2:

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

$$-e1we2$$

The above, of course, can be obtained by applying reversion to e1we2 with respect to the symmetric part of B:

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

$$-e1we2$$

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 the bilinear form 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.

Example 8: It is easy to write a wrapper for the Grassmann co-product too. Since the co-product [&gco](#) makes essential use of the

decomposition of elements into one-vectors, we expect that the dotted-Grassmann co-product will depend on F, that is, then antisymmetric part of B. First we have to load [Bigebra](#) package.

NOTE: `&gco_d` computes the dotted Grassmann co-product in the undotted wedge basis! (The Grassmann co-product on the dotted wedge basis w.r.t. the dotted wedge basis is according to the isomorphism theorem for those algebras identical to the original Grassmann co-product).

```
> with(Bigebra);
Increase verbosity by infolevel[function]=val -- use online help > ?Bigebra[help]
[&cco, &gco, &gco_d, &gco_pl, &map, &v, EV, VERSION, bracket, contract, drop_t, eps, gantipode, gco_unit, gswitch,
  hodge, linop, linop2, lists2mat, lists2mat2, make_BI_Id, mapop, mapop2, meet, op2mat, op2mat2, pairing, peek, poke,
  remove_eq, switch, tcollect, tsolve1]
> w_p1:=e1we2;
w_p2:=&gco_d(w_p1);
```

$$w_{p1} := e1we2$$

$$w_{p2} := (Id \&t e1we2) + F_{1,2} (Id \&t Id) + (e1 \&t e2) - (e2 \&t e1) + (e1we2 \&t Id)$$

The following examples compose the dotted co-product with dotted and undotted wedge (acting on a wedge basis!!)

```
> dwedge[F](e1,e2);
dwedge[F](e1,e2,e3);
```

$$e1We2$$

$$e1We2We3$$

We then substitute `&dw` and `&w` for the tensor product sign `&t` and evaluate. We will show the results sided by side for easier comparison:

```
> subs(`&t`=`&dw`,&gco_d(Id));simplify(%);
subs(`&t`=`&w`,&gco_d(Id));simplify(%);
```

$$Id \&dw Id$$

$$Id$$

$$Id \&w Id$$

$$Id$$

```
> subs(`&t`=`&dw`,&gco_d(e1));simplify(%);
subs(`&t`=`&w`,&gco_d(e1));simplify(%);
```

$$(Id \&dw e1) + (e1 \&dw Id)$$

$$2 e1$$

$$(Id \&w e1) + (e1 \&w Id)$$

$$2 e1$$

```
> subs(`&t`=`&dw`,&gco_d(e2));simplify(%);
subs(`&t`=`&w`,&gco_d(e2));simplify(%);
```

$$(Id \&dw e2) + (e2 \&dw Id)$$

$$2 e2$$

$$(Id \&w e2) + (e2 \&w Id)$$

$$2 e2$$

```
> subs(`&t`=`&dw`,&gco_d(e1we2));simplify(%);
subs(`&t`=`&w`,&gco_d(e1we2));simplify(%);
```

$$(Id \&dw e1we2) + F_{1,2} (Id \&dw Id) + (e1 \&dw e2) - (e2 \&dw e1) + (e1we2 \&dw Id)$$

$$4 e1we2 + 3 F_{1,2} Id$$

$$(Id \&w e1we2) + F_{1,2} (Id \&w Id) + (e1 \&w e2) - (e2 \&w e1) + (e1we2 \&w Id)$$

$$4 e1we2 + F_{1,2} Id$$

```
> subs(`&t`=`&dw`,&gco_d(e1we3));simplify(%);
subs(`&t`=`&w`,&gco_d(e1we3));simplify(%);
```

$$(Id \&dw e1we3) + F_{1,3} (Id \&dw Id) + (e1 \&dw e3) - (e3 \&dw e1) + (e1we3 \&dw Id)$$

$$4 e1we3 + 3 F_{1,3} Id$$

$$(Id \&w e1we3) + F_{1,3} (Id \&w Id) + (e1 \&w e3) - (e3 \&w e1) + (e1we3 \&w Id)$$

$$4 e1we3 + F_{1,3} Id$$

```
> subs(`&t`=`&dw`,&gco_d(e2we3));simplify(%);
subs(`&t`=`&w`,&gco_d(e2we3));simplify(%);
```

$$(Id \&dw e2we3) + F_{2,3} (Id \&dw Id) + (e2 \&dw e3) - (e3 \&dw e2) + (e2we3 \&dw Id)$$

$$4 e2we3 + 3 F_{2,3} Id$$

$$(Id \&w e2we3) + F_{2,3} (Id \&w Id) + (e2 \&w e3) - (e3 \&w e2) + (e2we3 \&w Id)$$

$$4 e2we3 + F_{2,3} Id$$

```
> subs(`&t`=`&dw`,&gco_d(e1we2we3));simplify(%);
subs(`&t`=`&w`,&gco_d(e1we2we3));simplify(%);
```

$$(Id \&dw e1we2we3) + F_{2,3} (Id \&dw e1) - F_{1,3} (Id \&dw e2) + F_{1,2} (Id \&dw e3) + (e1 \&dw e2we3) + F_{2,3} (e1 \&dw Id)$$

$$- (e2 \&dw e1we3) - F_{1,3} (e2 \&dw Id) + (e1we2 \&dw e3) + (e3 \&dw e1we2) + F_{1,2} (e3 \&dw Id) - (e1we3 \&dw e2)$$

$$+ (e2we3 \&dw e1) + (e1we2we3 \&dw Id)$$

$$6 F_{1,2} e3 + 6 F_{2,3} e1 - 6 F_{1,3} e2 + 8 e1we2we3$$

$$(Id \&w e1we2we3) + F_{2,3} (Id \&w e1) - F_{1,3} (Id \&w e2) + F_{1,2} (Id \&w e3) + (e1 \&w e2we3) + F_{2,3} (e1 \&w Id)$$

$$- (e2 \&w e1we3) - F_{1,3} (e2 \&w Id) + (e1we2 \&w e3) + (e3 \&w e1we2) + F_{1,2} (e3 \&w Id) - (e1we3 \&w e2)$$

$$+ (e2we3 \&w e1) + (e1we2we3 \&w Id)$$

$$8 e1we2we3 + 2 F_{2,3} e1 - 2 F_{1,3} e2 + 2 F_{1,2} e3$$

It is of utmost importance, that in these calculations we find that the usual loop tangle mult \cite Delta_{mult} which come up with the dimension of the spaces involved **fails** here. This might have an strong impact on the renormalization theory in QFT.

Note however, that if we do everything in the same algebra we end up with the correct factor $2^{(\text{grade_of_x})}$ for the dotted bi-vector e1we2:

```
> d_p1:=&gco_d(dwedge[F](e1,e2));
```

$$d_p1 := (Id \&t e1we2) + 2 F_{1,2} (Id \&t Id) + (e1 \&t e2) - (e2 \&t e1) + (e1we2 \&t Id)$$

```
> drop_t(&map(d_p1,1,dwedge[F]));
```

$$4 e1we2 + 4 F_{1,2} Id$$

```
> d_p2:=convert(%,wedge_to_dwedge,F);#comes up with the factor 4 in the dotted basis
'd_p2'=-4*reversion(e1we2);
```

$$d_p2 := 4 e1we2 + 8 F_{1,2} Id$$

$$d_p2 = 4 e1we2 + 8 F_{1,2} Id$$

```
>
```

See Also: [Bigebra:-`&gco`](#), [Bigebra:-`&cco`](#), [Bigebra:-`&t`](#), [Bigebra:-drop t](#), [Bigebra:-`&map`](#)

(c) Copyright October 8, 1995, by Rafal Ablamowicz & Bertfried Fauser, all rights reserved.

Last modified: November 16, 2003, 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 ``&*(numeric,{name,symbol,matrix,array})`. When `F` and `FT` are matrices or arrays, they are expected to be antisymmetric and negative of each other, that is, `FT = linalg[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:** Until now both types of algebras are expanded (formally) over the same basis Grassmann monomials which, according to `CLIFFORD`'s convention, are written as `eiwej...`. 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, 2003.
- [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(Clifford):with(Cliplus);
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, 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 an antisymmetric bilinear form F and its negative FT :

```
> convert(e1we2, wedge_to_dwedge, K);
```

$$e1we2 + K_{1,2} Id$$

```
> convert(%, dwedge_to_wedge, -K);
```

$$e1we2$$

```
> 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 $e1we2 + F[1,2]*Id$ by $e1de2$ 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)
      -e1we2 - 2 F1,2 Id
      -e1we2 - 2 F1,2 Id
      -e1we2 - 2 F1,2 Id
      -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, -e1we2 - 2 F1,2 Id, -e1we3 - 2 F1,3 Id, -e2we3 - 2 F2,3 Id, -2 F2,3 e1 + 2 F1,3 e2 - e1we2we3 - 2 F1,2 e3]
      [Id, e1, e2, e3, -e1we2 - 2 F1,2 Id, -e1we3 - 2 F1,3 Id, -e2we3 - 2 F2,3 Id, -2 F2,3 e1 + 2 F1,3 e2 - e1we2we3 - 2 F1,2 e3]
```

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 $e1we2$ to the dotted basis and call it $e1We2$:

```
> convert(e1we2,wedge_to_dwedge,F);
      e1We2
```

Apply reversion to $e1We2$ with respect to F to get the reversed element in the dotted basis:

```
> reversed_e1We2:=reversion(e1We2,F);
      reversed_e1We2 := -e1we2 - F1,2 Id
```

Observe, that the above element equals the negative of $e1We2$ just like reversing $e1we2$ with respect to the symmetric part g of B:

```
> reversed_e1We2+e1We2;
      0
```

Finally, convert $reversed_e1We2$ to the un-dotted standard Grassmann basis to get $-e1we2$:

```
> convert(reversed_e1We2,dwedge_to_wedge,-F);
      -e1we2
```

The above, of course, can be obtained by applying reversion to $e1we2$ with respect to the symmetric part of B:

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

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 the bilinear form 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 October 8, 1995, by Rafal Ablamowicz & Bertfried Fauser, all rights reserved.
Last modified: January 5, 2003, RA/BF.

- Function: Cliplus:-LCbig - extends the left contraction procedure 'LC' from 'CLIFFORD'

Calling Sequence:

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

Parameters:

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

- Description:

- This procedure extends procedure [Clifford:-LC](#) from 'CLIFFORD'. Recall, that LC(u,v) was a valid input for 'LC' 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/clipolynomial`](#). For completeness, procedure 'LC' was also accepting [`type/cliscalar`](#) for u and v.
- After loading 'Cliplus', procedure 'LC' will have the same properties as 'LC' plus the additional versatility afforded by the procedure 'LCbig'. 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/clipolynomial`` 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:bench:=time():with(Clifford):
```

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

```
> u:=2*elwe3+e4+Id;v:=2*e1we2we3+e1we2;  
LC(u,v);
```

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

```
v := 2 e1we2we3 + e1we2
```

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

```
4 B3,1 B1,2 e3 - 4 B3,1 B1,3 e2 - 4 B3,2 B1,1 e3 + 4 B3,2 B1,3 e1 + 4 B3,3 B1,1 e2 - 4 B3,3 B1,2 e1 + 2 B3,1 B1,2 Id - 2 B3,2 B1,1 Id  
+ 2 B4,1 e2we3 - 2 B4,2 e1we3 + 2 B4,3 e1we2 + B4,1 e2 - B4,2 e1 + 2 e1we2we3 + e1we2
```

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

```
> LC(u,v,K);
```

```
4 K3,1 K1,2 e3 - 4 K3,1 K1,3 e2 - 4 K3,2 K1,1 e3 + 4 K3,2 K1,3 e1 + 4 K3,3 K1,1 e2 - 4 K3,3 K1,2 e1 + 2 K3,1 K1,2 Id  
- 2 K3,2 K1,1 Id + 2 K4,1 e2we3 - 2 K4,2 e1we3 + 2 K4,3 e1we2 + K4,1 e2 - K4,2 e1 + 2 e1we2we3 + e1we2
```

```
[ For example, it is well known that the Clifford product cmul(x,v) = x  $\lrcorner$  v + wedge(x,v) 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 e1 - 2 e2 + e3
```

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

```
out1 := -2 (-2 B2,2 + 2 B1,2 + B3,2) e1we3 + 2 (2 B1,1 + B3,1 - 2 B2,1) e2we3 + 2 (-2 B2,3 + 2 B1,3 + B3,3) e1we2  
+ e1we2we3 - (-2 B2,2 + 2 B1,2 + B3,2) e1 + (2 B1,1 + B3,1 - 2 B2,1) e2
```

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

```
out2 := 4 B1,1 e2we3 - 4 B1,2 e1we3 + 4 B1,3 e1we2 - 4 B2,1 e2we3 + 4 B2,2 e1we3 - 4 B2,3 e1we2 + 2 B3,1 e2we3  
- 2 B3,2 e1we3 + 2 B3,3 e1we2 + 2 B1,1 e2 - 2 B1,2 e1 - 2 B2,1 e2 + 2 B2,2 e1 + e1we2we3 + B3,1 e2 - B3,2 e1
```

```
> out1-out2;
```

```
-2 (-2 B2,2 + 2 B1,2 + B3,2) e1we3 + 2 (2 B1,1 + B3,1 - 2 B2,1) e2we3 + 2 (-2 B2,3 + 2 B1,3 + B3,3) e1we2  
- (-2 B2,2 + 2 B1,2 + B3,2) e1 + (2 B1,1 + B3,1 - 2 B2,1) e2 - 4 B1,1 e2we3 + 4 B1,2 e1we3 - 4 B1,3 e1we2 + 4 B2,1 e2we3
```

$$-4 B_{2,2} e1we3 + 4 B_{2,3} e1we2 - 2 B_{3,1} e2we3 + 2 B_{3,2} e1we3 - 2 B_{3,3} e1we2 - 2 B_{1,1} e2 + 2 B_{1,2} e1 + 2 B_{2,1} e2 - 2 B_{2,2} e1 - B_{3,1} e2 + B_{3,2} e1$$

>

We can repeat the above computations using a different form. For example, it is well known that the Clifford product $\text{cmul}(x,v) = x _ v + \text{wedge}(x,v)$ where v is any element in $\text{Cl}(V,H)$, and x is a 1-vector in $\text{Cl}(V,H)$:

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

$$x := 2 e1 - 2 e2 + e3$$

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

$$\text{out1} := 2(-2 H_{1,2} - H_{3,2} + 2 H_{2,2}) e1we3 - 2(-H_{3,1} - 2 H_{1,1} + 2 H_{2,1}) e2we3 - 2(2 H_{2,3} - 2 H_{1,3} - H_{3,3}) e1we2 + e1we2we3 + (-2 H_{1,2} - H_{3,2} + 2 H_{2,2}) e1 - (-H_{3,1} - 2 H_{1,1} + 2 H_{2,1}) e2$$

> **out2:=LC(x,v,H)+ wedge(x,v);**

$$\text{out2} := 4 H_{1,1} e2we3 - 4 H_{1,2} e1we3 + 4 H_{1,3} e1we2 + 2 H_{1,1} e2 - 2 H_{1,2} e1 - 4 H_{2,1} e2we3 + 4 H_{2,2} e1we3 - 4 H_{2,3} e1we2 - 2 H_{2,1} e2 + 2 H_{2,2} e1 + 2 H_{3,1} e2we3 - 2 H_{3,2} e1we3 + 2 H_{3,3} e1we2 + H_{3,1} e2 - H_{3,2} e1 + e1we2we3$$

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

$$0$$

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

Worksheet took .361000 seconds to compute on AMD Athlon 2700+ 1GB RAM Win XP Prof

>

Example 2: 'LC' expects its arguments to be entered in Grassmann basis. Since the unevaluated Clifford basis can also be used in $\text{Cl}(V,B)$ instead of the Grassmann basis, we need to load package 'Cliplus' which contains procedure 'LCbig'. 'LCbig' 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:bench:=time():with(Clifford):with(Cliplus);**

u:=2*e1we3+e4+Id;v:=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, makeclialias]

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

$$v := 2 e1we2we3 + e1we2$$

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

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

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

$$v = 2 e1we2we3 + e1we2$$

> **uu:=cliexpand(u,B); #u converted to Clifford basis with 'cliexpand'**

vv:=cliexpand(v,B); #v converted to Clifford basis with 'cliexpand'

$$uu := 2 \&C_B(e1, e3) - 2 B_{1,3} Id + e4 + Id$$

$$vv := 2 \&C_B(e1, e2, e3) - 2 B_{2,3} e1 + 2 B_{1,3} e2 - 2 B_{1,2} e3 + \&C_B(e1, e2) - B_{1,2} Id$$

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

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

$$2 e1we3 + e4 + Id$$

$$2 e1we2we3 + e1we2$$

One can now apply 'LC' to uu and vv:

> **out3:=LC(uu,vv,B);**

$$\text{out3} := 4 B_{3,1} B_{1,2} e3 - 4 B_{3,1} B_{1,3} e2 - 4 B_{3,2} B_{1,1} e3 + 4 B_{3,2} B_{1,3} e1 + 4 B_{3,3} B_{1,1} e2 - 4 B_{3,3} B_{1,2} e1 + 2 B_{3,1} B_{1,2} Id - 2 B_{3,2} B_{1,1} Id + 2 B_{4,1} \&C_B(e2, e3) - 2 B_{4,1} B_{2,3} Id - 2 B_{4,2} \&C_B(e1, e3) + 2 B_{4,2} B_{1,3} Id + 2 B_{4,3} \&C_B(e1, e2) - 2 B_{4,3} B_{1,2} Id + B_{4,1} e2 - B_{4,2} e1 + 2 \&C_B(e1, e2, e3) - 2 B_{2,3} e1 + 2 B_{1,3} e2 - 2 B_{1,2} e3 + \&C_B(e1, e2) - B_{1,2} Id$$

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

> **out4:=LC(u,v,B);**

$$\text{out4} := 4 B_{3,1} B_{1,2} e3 - 4 B_{3,1} B_{1,3} e2 - 4 B_{3,2} B_{1,1} e3 + 4 B_{3,2} B_{1,3} e1 + 4 B_{3,3} B_{1,1} e2 - 4 B_{3,3} B_{1,2} e1 + 2 B_{3,1} B_{1,2} Id - 2 B_{3,2} B_{1,1} Id + 2 B_{4,1} e2we3 - 2 B_{4,2} e1we3 + 2 B_{4,3} e1we2 + B_{4,1} e2 - B_{4,2} e1 + 2 e1we2we3 + e1we2$$

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

```
[ > out5:=clieval(out3);
out5 := 4 B3,1 B1,2 e3 - 4 B3,1 B1,3 e2 - 4 B3,2 B1,1 e3 + 4 B3,2 B1,3 e1 + 4 B3,3 B1,1 e2 - 4 B3,3 B1,2 e1 + 2 B3,1 B1,2 Id
- 2 B3,2 B1,1 Id + 2 B4,1 e2we3 - 2 B4,2 e1we3 + 2 B4,3 e1we2 + B4,1 e2 - B4,2 e1 + 2 e1we2we3 + e1we2
[ > out5-out4;
0
[ >
```

[Now let's see the mixed input:

```
[ > out6:=LC(u,vv,B);
out6 := 4 B3,1 B1,2 e3 - 4 B3,1 B1,3 e2 - 4 B3,2 B1,1 e3 + 4 B3,2 B1,3 e1 + 4 B3,3 B1,1 e2 - 4 B3,3 B1,2 e1 + 2 B3,1 B1,2 Id
- 2 B3,2 B1,1 Id + 2 B4,1 &CB(e2, e3) - 2 B4,1 B2,3 Id - 2 B4,2 &CB(e1, e3) + 2 B4,2 B1,3 Id + 2 B4,3 &CB(e1, e2)
- 2 B4,3 B1,2 Id + B4,1 e2 - B4,2 e1 + 2 &CB(e1, e2, e3) - 2 B2,3 e1 + 2 B1,3 e2 - 2 B1,2 e3 + &CB(e1, e2) - B1,2 Id
```

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

```
[ > clieval(out6)-out5;
0
[ > out7:=LC(uu,v,B);
out7 := 4 B3,1 B1,2 e3 - 4 B3,1 B1,3 e2 - 4 B3,2 B1,1 e3 + 4 B3,2 B1,3 e1 + 4 B3,3 B1,1 e2 - 4 B3,3 B1,2 e1 + 2 B3,1 B1,2 Id
- 2 B3,2 B1,1 Id + 2 B4,1 &CB(e2, e3) - 2 B4,1 B2,3 Id - 2 B4,2 &CB(e1, e3) + 2 B4,2 B1,3 Id + 2 B4,3 &CB(e1, e2)
- 2 B4,3 B1,2 Id + B4,1 e2 - B4,2 e1 + 2 &CB(e1, e2, e3) - 2 B2,3 e1 + 2 B1,3 e2 - 2 B1,2 e3 + &CB(e1, e2) - B1,2 Id
[ > out7-out6;
0
```

[Note these special cases:

```
[ > LC(0,2);
0
[ > p:=a*Id+2*e1we3;
p := a Id + 2 e1we3
[ > LC(0,p);
0
[ > LC(2,3);
6
[ > LC(2*Id,3);
6 Id
[ > LC(2*Id,3*Id);
6 Id
[ >
```

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

```
[ > LCbig('&C'[K](e1,e2), '&C'[K](e3,e4)); ##<<<--- Error because contraction is w.r.t. B
Error, (in Cliplus:-LCbig) optional (or default B) parameter in LCbig differs from indices encountered
in its cliprod arguments. Found these names as indices of &C: {B, K}
[ > LCbig('&C'[K](e1,e2), '&C'[K](e3,e4),K); ##<<<--- No error because contraction is
w.r.t. K
K2,3 K1,4 Id - K2,4 K1,3 Id + K1,2 &CK(e3, e4)
[ > LCbig('&C'[-K](e1,e2), '&C'[-K](e3,e4),-K); ##<<<--- No error because contraction is
w.r.t. -K
K2,3 K1,4 Id - K2,4 K1,3 Id - K1,2 &C-K(e3, e4)
[ > printf("Worksheet took %f seconds to compute on AMD Athlon 2700+ 1GB RAM Win XP
Prof\n",time()-bench);
Worksheet took .861000 seconds to compute on AMD Athlon 2700+ 1GB RAM Win XP Prof
[ >
```

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

(c) Copyright October 8, 1995, by Rafal Ablamowicz & Bertfried Fauser, all rights reserved.
Last modified: January 5, 2003, RA/BF.

Function: Cliplus:-makeclialias - define aliases for Clifford basis monomials in Cl(K)

Calling Sequence:

```
makeclialias(p);  
makeclialias[K](p);  
makeclialias(p,s);  
makeclialias[K](p,s);
```

Parameters:

p - positive integer such that $1 \leq p \leq 9$
s - (optional) parameter 'ordered' or "ordered"
K - (optional) index for the inert Clifford product $\&C$

Description:

- The procedure defines and displays aliases for Clifford basis monomials which are of type 'cliprod'. See [Clifford:-cliprod](#) for more information.
- It is possible to use optional index, e.g., K. In this case, Clifford basis will be expressed in terms of $\&C[K](e1,e2)$, etc.
- Positive integer p defines the dimension of the vector space V whose basis elements generate Cl(V).
- In order to evaluate these aliases, use 'eval' command.
- When used with the optional argument, it defines only aliases for ordered monomials. This is time and memory saving feature.
- This procedure is similar to [Clifford:-makealiases](#) from the 'CLIFFORD' package which defines aliases for Grassmann monomials.

Examples:

```
> restart:with(Clifford):with(Cliplus);  
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, makeclialias]  
> type(e1 &C e2,cliprod);  
type(`&C`[K](e1,e2),cliprod);  
type(`&C`[-K](e1,e2),cliprod);  
  
true  
true  
true  
  
> makeclialias(2);  
makeclialias[K](2);  
makeclialias[-K](2);  
  
alias(e12 = e1 &C e2, e21 = e2 &C e1)  
alias(e12 = &C_K(e1, e2), e21 = &C_K(e2, e1))  
alias(e12 = &C_-K(e1, e2), e21 = &C_-K(e2, e1))  
  
[ Command 'eval' is not built-in into 'makeclialias' (like it is not built-in into the procedure 'makealiases') in order to display  
aliases before they are evaluated. Once aliases are evaluated, there is no way for Maple to show what these aliases represent.  
> eval(%);  
  
e12, e21  
  
[ The above is a list of all aliases currently defined in this worksheet. Notice that Maple also displays I which is the Maple alias for  
sqrt(-1).  
> restart:with(Clifford):with(Cliplus);  
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, makeclialias]  
> makeclialias(3, 'ordered');  
  
alias(e12 = e1 &C e2, e13 = e1 &C e3, e23 = e2 &C e3, e123 = &C(e1, e2, e3))
```

```

[ > eval(%);
                                     e12, e13, e23, e123
[ > makeclialiases[X](3, 'ordered');
      alias(e12 = &CX(e1, e2), e13 = &CX(e1, e3), e23 = &CX(e2, e3), e123 = &CX(e1, e2, e3))
[ > eval(%);
                                     e12, e13, e23, e123
[ > makeclialiases[-X](3, 'ordered');
      alias(e12 = &C-X(e1, e2), e13 = &C-X(e1, e3), e23 = &C-X(e2, e3), e123 = &C-X(e1, e2, e3))
[ > eval(%);
                                     e12, e13, e23, e123
[ >

```

[This time, we have only defined aliases for ordered Clifford monomials.

See Also: [Cliplus:-clieval](#), [Cliplus:-clibasis](#), [Clifford:-makealiases](#)

(c) Copyright October 8, 1995, by Rafal Ablamowicz & Bertfried Fauser, all rights reserved.
 Last modified: January 5, 2003, RA/BF.

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
```

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

```
2 B3,1 B1,2 Id - 2 B1,1 B3,2 Id
```

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

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

```
2 K3,1 K1,2 Id - 2 K1,1 K3,2 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 e1 - 2 e2 + e3
```

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

```
out1 := -2 (-2 B2,2 + 2 B2,1 + B2,3) e1we3 + 2 (B1,3 + 2 B1,1 - 2 B1,2) e2we3 + 2 (2 B3,1 - 2 B3,2 + B3,3) e1we2 + e1we2we3  
+ (-2 B2,2 + 2 B2,1 + B2,3) e1 - (B1,3 + 2 B1,1 - 2 B1,2) e2
```

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

```
out2 := 4 B3,1 e1we2 - 4 B2,1 e1we3 + 4 B1,1 e2we3 - 4 B3,2 e1we2 + 4 B2,2 e1we3 - 4 B1,2 e2we3 + 2 B3,3 e1we2  
- 2 B2,3 e1we3 + 2 B1,3 e2we3 + 2 B2,1 e1 - 2 B1,1 e2 - 2 B2,2 e1 + 2 B1,2 e2 + B2,3 e1 - B1,3 e2 + e1we2we3
```

```
[ > 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 e1 - 2 e2 + e3
```

```

> out1:=cmul[H](v,x);
out1 := -2(-2 H2,2 + 2 H2,1 + H2,3) e1we3 + 2(H1,3 + 2 H1,1 - 2 H1,2) e2we3 + 2(2 H3,1 - 2 H3,2 + H3,3) e1we2
+ e1we2we3 + (-2 H2,2 + 2 H2,1 + H2,3) e1 - (H1,3 + 2 H1,1 - 2 H1,2) e2
> out2:=RC(v,x,H)+ wedge(v,x);
out2 := 4 H3,1 e1we2 - 4 H2,1 e1we3 + 4 H1,1 e2we3 + 2 H2,1 e1 - 2 H1,1 e2 - 4 H3,2 e1we2 + 4 H2,2 e1we3 - 4 H1,2 e2we3
- 2 H2,2 e1 + 2 H1,2 e2 + 2 H3,3 e1we2 - 2 H2,3 e1we3 + 2 H1,3 e2we3 + e1we2we3 + H2,3 e1 - H1,3 e2
> 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(Cliifford):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, makeclialias]

v := 2 e1we3 + e4 + Id

u := 2 e1we2we3 + e1we2

```

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

```

```

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

```

u = 2 e1we2we3 + e1we2

v = 2 e1we3 + e4 + 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} Id

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

```

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

```

```

clieval(vv); #vv converted back to Grassmann basis gives the original v

```

2 e1we2we3 + e1we2

2 e1we3 + e4 + Id

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

```

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

```

```

out3 := 4 K3,1 K2,3 e1 - 4 K3,1 K1,3 e2 - 4 K2,1 K3,3 e1 + 4 K2,1 K1,3 e3 + 4 K1,1 K3,3 e2 - 4 K1,1 K2,3 e3 + 2 K3,4 &CK(e1, e2)
- 2 K3,4 K1,2 Id - 2 K2,4 &CK(e1, e3) + 2 K2,4 K1,3 Id + 2 K1,4 &CK(e2, e3) - 2 K1,4 K2,3 Id + 2 &CK(e1, e2, e3) - 2 K2,3 e1
+ 2 K1,3 e2 - 2 K1,2 e3 + 2 K2,1 K1,3 Id - 2 K1,1 K2,3 Id + K2,4 e1 - K1,4 e2 + &CK(e1, e2) - K1,2 Id

```

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

```

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

```

```

out4 := 4 K3,1 K2,3 e1 - 4 K3,1 K1,3 e2 - 4 K2,1 K3,3 e1 + 4 K2,1 K1,3 e3 + 4 K1,1 K3,3 e2 - 4 K1,1 K2,3 e3 + 2 K3,4 e1we2
- 2 K2,4 e1we3 + 2 K1,4 e2we3 + 2 e1we2we3 + 2 K2,1 K1,3 Id - 2 K1,1 K2,3 Id + K2,4 e1 - K1,4 e2 + e1we2

```

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

```

> out5:=clieval(out3);

```

```

out5 := 4 K3,1 K2,3 e1 - 4 K3,1 K1,3 e2 - 4 K2,1 K3,3 e1 + 4 K2,1 K1,3 e3 + 4 K1,1 K3,3 e2 - 4 K1,1 K2,3 e3 + 2 K3,4 e1we2
- 2 K2,4 e1we3 + 2 K1,4 e2we3 + 2 e1we2we3 + 2 K2,1 K1,3 Id - 2 K1,1 K2,3 Id + K2,4 e1 - K1,4 e2 + e1we2

```

```

> out5-out4;

```

0

Now let's see the mixed input:

```

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

```

```

out6 := 4 K3,1 K2,3 e1 - 4 K3,1 K1,3 e2 - 4 K2,1 K3,3 e1 + 4 K2,1 K1,3 e3 + 4 K1,1 K3,3 e2 - 4 K1,1 K2,3 e3 + 2 K3,4 &CK(e1, e2)
- 2 K3,4 K1,2 Id - 2 K2,4 &CK(e1, e3) + 2 K2,4 K1,3 Id + 2 K1,4 &CK(e2, e3) - 2 K1,4 K2,3 Id + 2 &CK(e1, e2, e3) - 2 K2,3 e1

```

```

[ + 2 K1,3 e2 - 2 K1,2 e3 + 2 K2,1 K1,3 Id - 2 K1,1 K2,3 Id + K2,4 e1 - K1,4 e2 + &CK(e1, e2) - K1,2 Id
[ which should be the same as out5 after conversion to Grassmann basis:
[ > clieval(out6)-out5;
[
[ 0
[ > out7:=RC(uu,v,K);
[ out7 := 4 K3,1 K2,3 e1 - 4 K3,1 K1,3 e2 - 4 K2,1 K3,3 e1 + 4 K2,1 K1,3 e3 + 4 K1,1 K3,3 e2 - 4 K1,1 K2,3 e3 + 2 K3,4 &CK(e1, e2)
[ - 2 K3,4 K1,2 Id - 2 K2,4 &CK(e1, e3) + 2 K2,4 K1,3 Id + 2 K1,4 &CK(e2, e3) - 2 K1,4 K2,3 Id + 2 &CK(e1, e2, e3) - 2 K2,3 e1
[ + 2 K1,3 e2 - 2 K1,2 e3 + 2 K2,1 K1,3 Id - 2 K1,1 K2,3 Id + K2,4 e1 - K1,4 e2 + &CK(e1, e2) - K1,2 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
[
[ 
$$K_{1,4} K_{2,3} Id - K_{2,4} K_{1,3} Id + K_{3,4} &C_K(e_1, e_2)$$

[ > RCbig(`&C`[-K](e1,e2),`&C`[-K](e3,e4),-K); ##<<<--- No error because contraction is
[ w.r.t. -K
[
[ 
$$K_{1,4} K_{2,3} Id - K_{2,4} K_{1,3} Id - K_{3,4} &C_{-K}(e_1, e_2)$$

[ >

```

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

(c) Copyright October 8, 1995, by Rafal Ablamowicz & Bertfried Fauser, all rights reserved.
 Last modified: January 5, 2003, 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 ``&*(numeric,{name,symbol,matrix,array})`. When `F` and `FT` are matrices or arrays, they are expected to be antisymmetric and negative of each other, that is, `FT = linalg[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:** Until now both types of algebras are expanded (formally) over the same basis Grassmann monomials which, according to `CLIFFORD`'s convention, are written as `eiwej...`. 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, 2003.
- [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(Clifford):with(Cliplus);
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, 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 an antisymmetric bilinear form F and its negative FT :

```
> convert(e1we2, wedge_to_dwedge, K);
```

$$e1we2 + K_{1,2} Id$$

```
> convert(%, dwedge_to_wedge, -K);
```

$$e1we2$$

```
> 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 $e1we2 + F[1,2]*Id$ by $e1de2$ 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)
      -e1we2 - 2 F1,2 Id
      -e1we2 - 2 F1,2 Id
      -e1we2 - 2 F1,2 Id
      -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, -e1we2 - 2 F1,2 Id, -e1we3 - 2 F1,3 Id, -e2we3 - 2 F2,3 Id, -2 F2,3 e1 + 2 F1,3 e2 - e1we2we3 - 2 F1,2 e3]
      [Id, e1, e2, e3, -e1we2 - 2 F1,2 Id, -e1we3 - 2 F1,3 Id, -e2we3 - 2 F2,3 Id, -2 F2,3 e1 + 2 F1,3 e2 - e1we2we3 - 2 F1,2 e3]
```

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 $e1we2$ to the dotted basis and call it $e1We2$:

```
> convert(e1we2,wedge_to_dwedge,F);
      e1We2
```

Apply reversion to $e1We2$ with respect to F to get the reversed element in the dotted basis:

```
> reversed_e1We2:=reversion(e1We2,F);
      reversed_e1We2 := -e1we2 - F1,2 Id
```

Observe, that the above element equals the negative of $e1We2$ just like reversing $e1we2$ with respect to the symmetric part of B (called 'g' above):

```
> reversed_e1We2+e1We2;
      0
```

Finally, convert $reversed_e1We2$ to the un-dotted standard Grassmann basis to get $-e1we2$:

```
> convert(reversed_e1We2,dwedge_to_wedge,-F);
      -e1we2
```

The above, of course, can be obtained by applying reversion to $e1we2$ with respect to the symmetric part of B:

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

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: [Algebra:-help](#), [Clifford:-dwedge](#), [Clifford:-reversion](#), [Clifford:-cbasis](#)

(c) Copyright October 8, 1995, by Rafal Ablamowicz & Bertfried Fauser, all rights reserved.

Last modified: January 5, 2003, RA/BF.