Function: code_support-insert_helppages

Calling Sequence:

insert_helppages(L::listlist,Mapleversion::string)

Description:

- code_support is a package that provides various functions to copy, rename, modify, save, etc. Maple help worksheets.
- Examples how to use the remaining functions in the apckage can be found in code_support,examples.
- Procedure **insert_helppages** accomplishes insertion of the help topics into the library archive that has been marched.
- The help topics are given in a list L of type 'lislist' as the first argument to **insert_helppages** while Maple version, for example "M10", is listed as the second argument.

Examples:

```
> restart:with(code_support);
```

Module code_cupport ver. 1.03 for CLIFFORD et al. for Maple 11

Copyright (c) 2002-2008 by Rafal Ablamowicz and Bertfried Fauser. All rights reserved.

Last revised: March 10, 2007

[*NamesInLibrary*, *change_helpfiles*, *change_name*, *copy_file*, *get_TEXT*, *get_dir*, *insert_helppages*, *makeLIST*, *modifyLIST*, *replace_in_file*, *split*]

We will save the help topics in the library archive that exists in the directoy `C:\Maple10/Cliffordlib`.

```
> libname;
```

"C:\Maple11/Cliffordlib", "C:\Maple11/lib", "C:\Brachey.Troy/TNB", "C:\Maple11/SINGULARPLURALlinklib"

```
>
```

Inserting help pages for CLIFFORD:

```
> HDB_LIB_PATH:=convert(libname[1],name);            ##Path to HDB
  library
  HELP_FILE_PATH:=`C:\\Maple11/P11/Cliff11/Help/`;   ##Directory
  where *_M11.mws help files are located
  BROWSER_PATH:=`Mathematics/Algebra/`;              ###Location in
  the browser
  ModuleName  :=`Clifford`;                          ###Name of the
```

```
module
Parent      :=`Clifford,intro`;                    ###Parent for
all help pages in the module
GrandParent :=`Clifford,intro`;                    ###Grand parent
for all help pages in the module
MapleVersion:="M11";                               ### substring of
file names that gives Maple version
##############################################
##list of types defined in the module
##############################################
typesLIST:=["antisymmatrix","clibasmon","climatrix","climon","clip
olynom",

"cliprod","cliscalar","diagmatrix","dfmatrix","evenelement",

"fieldelement","gencomplex","genquatbasis","genquaternion",

"idempotent","nilpotent","oddelement","primitiveidemp","purequatba
sis",
            "quaternion","symmatrix","tensorprod"];
##############################################
##list of types converts defined in the module
##############################################
convertsLIST:=["mlist","str_to_int"];
```

HDB_LIB_PATH := *C:\Maple11/Cliffordlib*

HELP_FILE_PATH := *C:\Maple11/P11/Cliff11/Help/*

BROWSER_PATH := *Mathematics/Algebra/*

ModuleName := *Clifford*

Parent := *Clifford,intro*

GrandParent := *Clifford,intro*

MapleVersion := "M11"

typesLIST := ["antisymmatrix", "clibasmon", "climatrix", "climon", "clipolynom", "cliprod", "cliscalar", "diagmatrix", "dfmatrix", "evenelement", "fieldelement", "gencomplex", "genquatbasis", "genquaternion", "idempotent", "nilpotent", "oddelement", "primitiveidemp", "purequatbasis", "quaternion", "symmatrix", "tensorprod"]

convertsLIST := ["mlist", "str_to_int"]

Step 1: Generating an automatic list from help page files:

```
> Lauto:=makeLIST(ModuleName,Parent,GrandParent,HELP_FILE_PATH,Maple
Version);
```

```
Lauto := [[Clifford,&c, Clifford,intro, ["Clifford,&c", "&c"]],
    [Clifford,adfmatrix, Clifford,intro, ["Clifford,adfmatrix", "adfmatrix"]],
    [Clifford,all_sigs, Clifford,intro, ["Clifford,all_sigs", "all_sigs"]], [
    Clifford,type,antisymmatrix, Clifford,intro, ["Clifford,type,antisymmatrix", "type,antisymmatrix"]
    ], [Clifford,beta_minus, Clifford,intro, ["Clifford,beta_minus", "beta_minus"]],
    [Clifford,beta_plus, Clifford,intro, ["Clifford,beta_plus", "beta_plus"]],
    [Clifford,Bsignature, Clifford,intro, ["Clifford,Bsignature", "Bsignature"]],
    [Clifford,buildm, Clifford,intro, ["Clifford,buildm", "buildm"]],
    [Clifford,bygrade, Clifford,intro, ["Clifford,bygrade", "bygrade"]],
    [Clifford,cbasis, Clifford,intro, ["Clifford,cbasis", "cbasis"]],
    [Clifford,cdfmatrix, Clifford,intro, ["Clifford,cdfmatrix", "cdfmatrix"]],
    [Clifford,cexpQ, Clifford,intro, ["Clifford,cexpQ", "cexpQ"]],
    [Clifford,cexp, Clifford,intro, ["Clifford,cexp", "cexp"]],
    [Clifford,cinv, Clifford,intro, ["Clifford,cinv", "cinv"]],
    [Clifford,type,clibasmon, Clifford,intro, ["Clifford,type,clibasmon", "type,clibasmon"]],
    [Clifford,clibilinear, Clifford,intro, ["Clifford,clibilinear", "clibilinear"]],
    [Clifford,clicollect, Clifford,intro, ["Clifford,clicollect", "clicollect"]],
    [Clifford,clidata, Clifford,intro, ["Clifford,clidata", "clidata"]],
    [Clifford,CLIFFORD_ENV, Clifford,intro, ["Clifford,CLIFFORD_ENV", "CLIFFORD_ENV"]],
    [Clifford,clilinear, Clifford,intro, ["Clifford,clilinear", "clilinear"]],
    [Clifford,type,climatrix, Clifford,intro, ["Clifford,type,climatrix", "type,climatrix"]],
    [Clifford,climinpoly, Clifford,intro, ["Clifford,climinpoly", "climinpoly"]],
    [Clifford,type,climon, Clifford,intro, ["Clifford,type,climon", "type,climon"]],
    [Clifford,cliparse, Clifford,intro, ["Clifford,cliparse", "cliparse"]],
    [Clifford,type,clipolynom, Clifford,intro, ["Clifford,type,clipolynom", "type,clipolynom"]],
    [Clifford,type,cliprod, Clifford,intro, ["Clifford,type,cliprod", "type,cliprod"]],
    [Clifford,cliremove, Clifford,intro, ["Clifford,cliremove", "cliremove"]],
    [Clifford,type,cliscalar, Clifford,intro, ["Clifford,type,cliscalar", "type,cliscalar"]],
    [Clifford,clisolve, Clifford,intro, ["Clifford,clisolve", "clisolve"]],
    [Clifford,clisort, Clifford,intro, ["Clifford,clisort", "clisort"]],
    [Clifford,cliterms, Clifford,intro, ["Clifford,cliterms", "cliterms"]],
    [Clifford,cmulgen, Clifford,intro, ["Clifford,cmulgen", "cmulgen"]],
    [Clifford,cmulNUM, Clifford,intro, ["Clifford,cmulNUM", "cmulNUM"]],
    [Clifford,cmulQ, Clifford,intro, ["Clifford,cmulQ", "cmulQ"]],
    [Clifford,cmulRS, Clifford,intro, ["Clifford,cmulRS", "cmulRS"]],
    [Clifford,cmul, Clifford,intro, ["Clifford,cmul", "cmul"]], [
```

Clifford,cmul_user_defined, *Clifford,intro*, ["Clifford,cmul_user_defined", "cmul_user_defined"]
], [*Clifford,cocycle*, *Clifford,intro*, ["Clifford,cocycle", "cocycle"]], [
Clifford,commutingelements, *Clifford,intro*,
["Clifford,commutingelements", "commutingelements"]],
[*Clifford,conjugation*, *Clifford,intro*, ["Clifford,conjugation", "conjugation"]],
[*Clifford,c_conjug*, *Clifford,intro*, ["Clifford,c_conjug", "c_conjug"]],
[*Clifford,ddfmatrix*, *Clifford,intro*, ["Clifford,ddfmatrix", "ddfmatrix"]],
[*Clifford,type,dfmatrix*, *Clifford,intro*, ["Clifford,type,dfmatrix", "type,dfmatrix"]],
[*Clifford,type,diagmatrix*, *Clifford,intro*, ["Clifford,type,diagmatrix", "type,diagmatrix"]],
[*Clifford,diagonalize*, *Clifford,intro*, ["Clifford,diagonalize", "diagonalize"]],
[*Clifford,displayid*, *Clifford,intro*, ["Clifford,displayid", "displayid"]],
[*Clifford,type,evenelement*, *Clifford,intro*, ["Clifford,type,evenelement", "type,evenelement"]],
[*Clifford,extract*, *Clifford,intro*, ["Clifford,extract", "extract"]],
[*Clifford,factoridempotent*, *Clifford,intro*, ["Clifford,factoridempotent", "factoridempotent"]],
[*Clifford,type,fieldelement*, *Clifford,intro*, ["Clifford,type,fieldelement", "type,fieldelement"]],
[*Clifford,find1str*, *Clifford,intro*, ["Clifford,find1str", "find1str"]],
[*Clifford,findbasis*, *Clifford,intro*, ["Clifford,findbasis", "findbasis"]],
[*Clifford,type,gencomplex*, *Clifford,intro*, ["Clifford,type,gencomplex", "type,gencomplex"]],
[*Clifford,type,genquatbasis*, *Clifford,intro*, ["Clifford,type,genquatbasis", "type,genquatbasis"]], [
Clifford,type,genquaternion, *Clifford,intro*, ["Clifford,type,genquaternion", "type,genquaternion"]
], [*Clifford,gradeinv*, *Clifford,intro*, ["Clifford,gradeinv", "gradeinv"]],
[*Clifford,type,idempotent*, *Clifford,intro*, ["Clifford,type,idempotent", "type,idempotent"]],
[*Clifford,intro*, *Clifford,intro*, ["Clifford,intro", "intro"]],
[*Clifford,isproduct*, *Clifford,intro*, ["Clifford,isproduct", "isproduct"]],
[*Clifford,isVahlenmatrix*, *Clifford,intro*, ["Clifford,isVahlenmatrix", "isVahlenmatrix"]],
[*Clifford,Kfield*, *Clifford,intro*, ["Clifford,Kfield", "Kfield"]],
[*Clifford,LCQ*, *Clifford,intro*, ["Clifford,LCQ", "LCQ"]],
[*Clifford,LC*, *Clifford,intro*, ["Clifford,LC", "LC"]],
[*Clifford,makealiases*, *Clifford,intro*, ["Clifford,makealiases", "makealiases"]],
[*Clifford,makeclibasmon*, *Clifford,intro*, ["Clifford,makeclibasmon", "makeclibasmon"]],
[*Clifford,matKrepr*, *Clifford,intro*, ["Clifford,matKrepr", "matKrepr"]],
[*Clifford,maxgrade*, *Clifford,intro*, ["Clifford,maxgrade", "maxgrade"]],
[*Clifford,maxindex*, *Clifford,intro*, ["Clifford,maxindex", "maxindex"]],
[*Clifford,mdfmatrix*, *Clifford,intro*, ["Clifford,mdfmatrix", "mdfmatrix"]],
[*Clifford,minimalideal*, *Clifford,intro*, ["Clifford,minimalideal", "minimalideal"]],
[*Clifford,convert,mlist*, *Clifford,intro*, ["Clifford,convert,mlist", "convert,mlist"]],

[*Clifford,type,nilpotent*, *Clifford,intro*, ["Clifford,type,nilpotent", "type,nilpotent"]],
[*Clifford,type,oddelement*, *Clifford,intro*, ["Clifford,type,oddelement", "type,oddelement"]],
[*Clifford,ord*, *Clifford,intro*, ["Clifford,ord", "ord"]],
[*Clifford,permsign*, *Clifford,intro*, ["Clifford,permsign", "permsign"]], [
Clifford,type,primitiveidemp, *Clifford,intro*,
["Clifford,type,primitiveidemp", "type,primitiveidemp"]],
[*Clifford,pseudodet*, *Clifford,intro*, ["Clifford,pseudodet", "pseudodet"]], [
Clifford,type,purequatbasis, *Clifford,intro*, ["Clifford,type,purequatbasis", "type,purequatbasis"]],
[*Clifford,qdisplay*, *Clifford,intro*, ["Clifford,qdisplay", "qdisplay"]],
[*Clifford,qinv*, *Clifford,intro*, ["Clifford,qinv", "qinv"]],
[*Clifford,qmul*, *Clifford,intro*, ["Clifford,qmul", "qmul"]],
[*Clifford,qnorm*, *Clifford,intro*, ["Clifford,qnorm", "qnorm"]],
[*Clifford,type,quaternion*, *Clifford,intro*, ["Clifford,type,quaternion", "type,quaternion"]],
[*Clifford,q_conjug*, *Clifford,intro*, ["Clifford,q_conjug", "q_conjug"]],
[*Clifford,RCQ*, *Clifford,intro*, ["Clifford,RCQ", "RCQ"]],
[*Clifford,RC*, *Clifford,intro*, ["Clifford,RC", "RC"]],
[*Clifford,rd_clibasmon*, *Clifford,intro*, ["Clifford,rd_clibasmon", "rd_clibasmon"]],
[*Clifford,rd_climon*, *Clifford,intro*, ["Clifford,rd_climon", "rd_climon"]],
[*Clifford,rd_clipolynom*, *Clifford,intro*, ["Clifford,rd_clipolynom", "rd_clipolynom"]],
[*Clifford,reorder*, *Clifford,intro*, ["Clifford,reorder", "reorder"]],
[*Clifford,reversion*, *Clifford,intro*, ["Clifford,reversion", "reversion"]],
[*Clifford,RHnumber*, *Clifford,intro*, ["Clifford,RHnumber", "RHnumber"]],
[*Clifford,rmulm*, *Clifford,intro*, ["Clifford,rmulm", "rmulm"]],
[*Clifford,rot3d*, *Clifford,intro*, ["Clifford,rot3d", "rot3d"]],
[*Clifford,scalarpart*, *Clifford,intro*, ["Clifford,scalarpart", "scalarpart"]],
[*Clifford,setup*, *Clifford,intro*, ["Clifford,setup", "setup"]],
[*Clifford,sexp*, *Clifford,intro*, ["Clifford,sexp", "sexp"]],
[*Clifford,specify_constants*, *Clifford,intro*, ["Clifford,specify_constants", "specify_constants"]],
[*Clifford,spinorKbasis*, *Clifford,intro*, ["Clifford,spinorKbasis", "spinorKbasis"]],
[*Clifford,spinorKrepr*, *Clifford,intro*, ["Clifford,spinorKrepr", "spinorKrepr"]],
[*Clifford,squaremodf*, *Clifford,intro*, ["Clifford,squaremodf", "squaremodf"]],
[*Clifford,convert,str_to_int*, *Clifford,intro*, ["Clifford,convert,str_to_int", "convert,str_to_int"]],
[*Clifford,subs_clipolynom*, *Clifford,intro*, ["Clifford,subs_clipolynom", "subs_clipolynom"]],
[*Clifford,type,symmatrix*, *Clifford,intro*, ["Clifford,type,symmatrix", "type,symmatrix"]],
[*Clifford,type,tensorprod*, *Clifford,intro*, ["Clifford,type,tensorprod", "type,tensorprod"]],
[*Clifford,useproduct*, *Clifford,intro*, ["Clifford,useproduct", "useproduct"]],

[*Clifford,vectorpart*, *Clifford,intro*, ["Clifford,vectorpart", "vectorpart"]],
[*Clifford,version*, *Clifford,intro*, ["Clifford,version", "version"]],
[*Clifford,wedge*, *Clifford,intro*, ["Clifford,wedge", "wedge"]],
[*Clifford,wexp*, *Clifford,intro*, ["Clifford,wexp", "wexp"]]]

```
>
```

Step 2: Modifying, if needed, certain entries in Lauto list:

This is a list of entries that need to have modified aliases.

```
> modsLIST:=[
  [`Clifford,intro`,["Clifford","Clifford,intro","CLIFFORD","Cliffor
  d","clifford"]],
  [`Clifford,&c`,["Clifford,Clifford
  product","&c","&cQ","&w","&q","&cm","&cQm","&wm","&qm","rm","&C"]]
  ,
  [`Clifford,wedge`,["Clifford,wedge","wedge","&w"]],
  [`Clifford,CLIFFORD_ENV`,["Clifford,CLIFFORD_ENV","CLIFFORD_ENV",

  "dim_V","_prolevel","_shortcut_in_minimalideal",

  "_shortcut_in_Kfield","_shortcut_in_spinorKbasis",

  "_shortcut_in_spinorKrepr","_warnings_flag",
                                    "_quatbasis","_scalartypes"]]
  ]:
> for mem in modsLIST do
      Lauto:=modifyLIST(Lauto,op(mem))
  end do:
  Lauto;
```

[[*Clifford,&c*, *Clifford,intro*, ["Clifford,Clifford product", "&c", "&cQ", "&w", "&q", "&cm", "&cQm", "&wm", "&qm", "rm", "&C"]],
[*Clifford,adfmatrix*, *Clifford,intro*, ["Clifford,adfmatrix", "adfmatrix"]],
[*Clifford,all_sigs*, *Clifford,intro*, ["Clifford,all_sigs", "all_sigs"]], [*Clifford,type,antisymmatrix*, *Clifford,intro*, ["Clifford,type,antisymmatrix", "type,antisymmatrix"]], [*Clifford,beta_minus*, *Clifford,intro*, ["Clifford,beta_minus", "beta_minus"]],
[*Clifford,beta_plus*, *Clifford,intro*, ["Clifford,beta_plus", "beta_plus"]],
[*Clifford,Bsignature*, *Clifford,intro*, ["Clifford,Bsignature", "Bsignature"]],
[*Clifford,buildm*, *Clifford,intro*, ["Clifford,buildm", "buildm"]],
[*Clifford,bygrade*, *Clifford,intro*, ["Clifford,bygrade", "bygrade"]],
[*Clifford,cbasis*, *Clifford,intro*, ["Clifford,cbasis", "cbasis"]],

[*Clifford,cdfmatrix*, *Clifford,intro*, ["Clifford,cdfmatrix", "cdfmatrix"]],
[*Clifford,cexpQ*, *Clifford,intro*, ["Clifford,cexpQ", "cexpQ"]],
[*Clifford,cexp*, *Clifford,intro*, ["Clifford,cexp", "cexp"]],
[*Clifford,cinv*, *Clifford,intro*, ["Clifford,cinv", "cinv"]],
[*Clifford,type,clibasmon*, *Clifford,intro*, ["Clifford,type,clibasmon", "type,clibasmon"]],
[*Clifford,clibilinear*, *Clifford,intro*, ["Clifford,clibilinear", "clibilinear"]],
[*Clifford,clicollect*, *Clifford,intro*, ["Clifford,clicollect", "clicollect"]],
[*Clifford,clidata*, *Clifford,intro*, ["Clifford,clidata", "clidata"]], [*Clifford,CLIFFORD_ENV*, *Clifford,intro*, ["Clifford,CLIFFORD_ENV", "CLIFFORD_ENV", "dim_V", "_prolevel", "_shortcut_in_minimalideal", "_shortcut_in_Kfield", "_shortcut_in_spinorKbasis", "_shortcut_in_spinorKrepr", "_warnings_flag", "_quatbasis", "_scalartypes"]],
[*Clifford,clilinear*, *Clifford,intro*, ["Clifford,clilinear", "clilinear"]],
[*Clifford,type,climatrix*, *Clifford,intro*, ["Clifford,type,climatrix", "type,climatrix"]],
[*Clifford,climinpoly*, *Clifford,intro*, ["Clifford,climinpoly", "climinpoly"]],
[*Clifford,type,climon*, *Clifford,intro*, ["Clifford,type,climon", "type,climon"]],
[*Clifford,cliparse*, *Clifford,intro*, ["Clifford,cliparse", "cliparse"]],
[*Clifford,type,clipolynom*, *Clifford,intro*, ["Clifford,type,clipolynom", "type,clipolynom"]],
[*Clifford,type,cliprod*, *Clifford,intro*, ["Clifford,type,cliprod", "type,cliprod"]],
[*Clifford,cliremove*, *Clifford,intro*, ["Clifford,cliremove", "cliremove"]],
[*Clifford,type,cliscalar*, *Clifford,intro*, ["Clifford,type,cliscalar", "type,cliscalar"]],
[*Clifford,clisolve*, *Clifford,intro*, ["Clifford,clisolve", "clisolve"]],
[*Clifford,clisort*, *Clifford,intro*, ["Clifford,clisort", "clisort"]],
[*Clifford,cliterms*, *Clifford,intro*, ["Clifford,cliterms", "cliterms"]],
[*Clifford,cmulgen*, *Clifford,intro*, ["Clifford,cmulgen", "cmulgen"]],
[*Clifford,cmulNUM*, *Clifford,intro*, ["Clifford,cmulNUM", "cmulNUM"]],
[*Clifford,cmulQ*, *Clifford,intro*, ["Clifford,cmulQ", "cmulQ"]],
[*Clifford,cmulRS*, *Clifford,intro*, ["Clifford,cmulRS", "cmulRS"]],
[*Clifford,cmul*, *Clifford,intro*, ["Clifford,cmul", "cmul"]], [*Clifford,cmul_user_defined*, *Clifford,intro*, ["Clifford,cmul_user_defined", "cmul_user_defined"]], [*Clifford,cocycle*, *Clifford,intro*, ["Clifford,cocycle", "cocycle"]], [*Clifford,commutingelements*, *Clifford,intro*, ["Clifford,commutingelements", "commutingelements"]],
[*Clifford,conjugation*, *Clifford,intro*, ["Clifford,conjugation", "conjugation"]],
[*Clifford,c_conjug*, *Clifford,intro*, ["Clifford,c_conjug", "c_conjug"]],
[*Clifford,ddfmatrix*, *Clifford,intro*, ["Clifford,ddfmatrix", "ddfmatrix"]],
[*Clifford,type,dfmatrix*, *Clifford,intro*, ["Clifford,type,dfmatrix", "type,dfmatrix"]],

[*Clifford,type,diagmatrix*, *Clifford,intro*, ["Clifford,type,diagmatrix", "type,diagmatrix"]],
[*Clifford,diagonalize*, *Clifford,intro*, ["Clifford,diagonalize", "diagonalize"]],
[*Clifford,displayid*, *Clifford,intro*, ["Clifford,displayid", "displayid"]],
[*Clifford,type,evenelement*, *Clifford,intro*, ["Clifford,type,evenelement", "type,evenelement"]],
[*Clifford,extract*, *Clifford,intro*, ["Clifford,extract", "extract"]],
[*Clifford,factoridempotent*, *Clifford,intro*, ["Clifford,factoridempotent", "factoridempotent"]],
[*Clifford,type,fieldelement*, *Clifford,intro*, ["Clifford,type,fieldelement", "type,fieldelement"]],
[*Clifford,find1str*, *Clifford,intro*, ["Clifford,find1str", "find1str"]],
[*Clifford,findbasis*, *Clifford,intro*, ["Clifford,findbasis", "findbasis"]],
[*Clifford,type,gencomplex*, *Clifford,intro*, ["Clifford,type,gencomplex", "type,gencomplex"]],
[*Clifford,type,genquatbasis*, *Clifford,intro*, ["Clifford,type,genquatbasis", "type,genquatbasis"]], [
Clifford,type,genquaternion, *Clifford,intro*, ["Clifford,type,genquaternion", "type,genquaternion"]
], [*Clifford,gradeinv*, *Clifford,intro*, ["Clifford,gradeinv", "gradeinv"]],
[*Clifford,type,idempotent*, *Clifford,intro*, ["Clifford,type,idempotent", "type,idempotent"]],
[*Clifford,intro*, *Clifford,intro*, ["Clifford", "Clifford,intro", "CLIFFORD", "Clifford", "clifford"]],
[*Clifford,isproduct*, *Clifford,intro*, ["Clifford,isproduct", "isproduct"]],
[*Clifford,isVahlenmatrix*, *Clifford,intro*, ["Clifford,isVahlenmatrix", "isVahlenmatrix"]],
[*Clifford,Kfield*, *Clifford,intro*, ["Clifford,Kfield", "Kfield"]],
[*Clifford,LCQ*, *Clifford,intro*, ["Clifford,LCQ", "LCQ"]],
[*Clifford,LC*, *Clifford,intro*, ["Clifford,LC", "LC"]],
[*Clifford,makealiases*, *Clifford,intro*, ["Clifford,makealiases", "makealiases"]],
[*Clifford,makeclibasmon*, *Clifford,intro*, ["Clifford,makeclibasmon", "makeclibasmon"]],
[*Clifford,matKrepr*, *Clifford,intro*, ["Clifford,matKrepr", "matKrepr"]],
[*Clifford,maxgrade*, *Clifford,intro*, ["Clifford,maxgrade", "maxgrade"]],
[*Clifford,maxindex*, *Clifford,intro*, ["Clifford,maxindex", "maxindex"]],
[*Clifford,mdfmatrix*, *Clifford,intro*, ["Clifford,mdfmatrix", "mdfmatrix"]],
[*Clifford,minimalideal*, *Clifford,intro*, ["Clifford,minimalideal", "minimalideal"]],
[*Clifford,convert,mlist*, *Clifford,intro*, ["Clifford,convert,mlist", "convert,mlist"]],
[*Clifford,type,nilpotent*, *Clifford,intro*, ["Clifford,type,nilpotent", "type,nilpotent"]],
[*Clifford,type,oddelement*, *Clifford,intro*, ["Clifford,type,oddelement", "type,oddelement"]],
[*Clifford,ord*, *Clifford,intro*, ["Clifford,ord", "ord"]],
[*Clifford,permsign*, *Clifford,intro*, ["Clifford,permsign", "permsign"]], [
Clifford,type,primitiveidemp, *Clifford,intro*,
["Clifford,type,primitiveidemp", "type,primitiveidemp"]],
[*Clifford,pseudodet*, *Clifford,intro*, ["Clifford,pseudodet", "pseudodet"]], [
Clifford,type,purequatbasis, *Clifford,intro*, ["Clifford,type,purequatbasis", "type,purequatbasis"]],

[*Clifford,qdisplay*, *Clifford,intro*, ["Clifford,qdisplay", "qdisplay"]],
[*Clifford,qinv*, *Clifford,intro*, ["Clifford,qinv", "qinv"]],
[*Clifford,qmul*, *Clifford,intro*, ["Clifford,qmul", "qmul"]],
[*Clifford,qnorm*, *Clifford,intro*, ["Clifford,qnorm", "qnorm"]],
[*Clifford,type,quaternion*, *Clifford,intro*, ["Clifford,type,quaternion", "type,quaternion"]],
[*Clifford,q_conjug*, *Clifford,intro*, ["Clifford,q_conjug", "q_conjug"]],
[*Clifford,RCQ*, *Clifford,intro*, ["Clifford,RCQ", "RCQ"]],
[*Clifford,RC*, *Clifford,intro*, ["Clifford,RC", "RC"]],
[*Clifford,rd_clibasmon*, *Clifford,intro*, ["Clifford,rd_clibasmon", "rd_clibasmon"]],
[*Clifford,rd_climon*, *Clifford,intro*, ["Clifford,rd_climon", "rd_climon"]],
[*Clifford,rd_clipolynom*, *Clifford,intro*, ["Clifford,rd_clipolynom", "rd_clipolynom"]],
[*Clifford,reorder*, *Clifford,intro*, ["Clifford,reorder", "reorder"]],
[*Clifford,reversion*, *Clifford,intro*, ["Clifford,reversion", "reversion"]],
[*Clifford,RHnumber*, *Clifford,intro*, ["Clifford,RHnumber", "RHnumber"]],
[*Clifford,rmulm*, *Clifford,intro*, ["Clifford,rmulm", "rmulm"]],
[*Clifford,rot3d*, *Clifford,intro*, ["Clifford,rot3d", "rot3d"]],
[*Clifford,scalarpart*, *Clifford,intro*, ["Clifford,scalarpart", "scalarpart"]],
[*Clifford,setup*, *Clifford,intro*, ["Clifford,setup", "setup"]],
[*Clifford,sexp*, *Clifford,intro*, ["Clifford,sexp", "sexp"]],
[*Clifford,specify_constants*, *Clifford,intro*, ["Clifford,specify_constants", "specify_constants"]],
[*Clifford,spinorKbasis*, *Clifford,intro*, ["Clifford,spinorKbasis", "spinorKbasis"]],
[*Clifford,spinorKrepr*, *Clifford,intro*, ["Clifford,spinorKrepr", "spinorKrepr"]],
[*Clifford,squaremodf*, *Clifford,intro*, ["Clifford,squaremodf", "squaremodf"]],
[*Clifford,convert,str_to_int*, *Clifford,intro*, ["Clifford,convert,str_to_int", "convert,str_to_int"]],
[*Clifford,subs_clipolynom*, *Clifford,intro*, ["Clifford,subs_clipolynom", "subs_clipolynom"]],
[*Clifford,type,symmatrix*, *Clifford,intro*, ["Clifford,type,symmatrix", "type,symmatrix"]],
[*Clifford,type,tensorprod*, *Clifford,intro*, ["Clifford,type,tensorprod", "type,tensorprod"]],
[*Clifford,useproduct*, *Clifford,intro*, ["Clifford,useproduct", "useproduct"]],
[*Clifford,vectorpart*, *Clifford,intro*, ["Clifford,vectorpart", "vectorpart"]],
[*Clifford,version*, *Clifford,intro*, ["Clifford,version", "version"]],
[*Clifford,wedge*, *Clifford,intro*, ["Clifford,wedge", "wedge", "&w"]],
[*Clifford,wexp*, *Clifford,intro*, ["Clifford,wexp", "wexp"]]]

```
>
```

Step 3: Inserting all help pages into HDB and the browser using the last modified list:

```
> insert_helppages(Lauto,MapleVersion);
Trying to read file C:\Maple11/P11/Cliff11/Help/&c_M11.mws...
Success... inserting topic Clifford,&c from the file:
```

C:\Maple11/P11/Cliff11/Help/&c_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/adfmatrix_M11.mws...
Success... inserting topic Clifford,adfmatrix from the file:
```

C:\Maple11/P11/Cliff11/Help/adfmatrix_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/all_sigs_M11.mws...
Success... inserting topic Clifford,all_sigs from the file:
```

C:\Maple11/P11/Cliff11/Help/all_sigs_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/antisymmatrix_M11.mws...
Success... inserting topic Clifford,type,antisymmatrix from the file:
```

C:\Maple11/P11/Cliff11/Help/antisymmatrix_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/beta_minus_M11.mws...
Success... inserting topic Clifford,beta_minus from the file:
```

C:\Maple11/P11/Cliff11/Help/beta_minus_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/beta_plus_M11.mws...
Success... inserting topic Clifford,beta_plus from the file:
```

C:\Maple11/P11/Cliff11/Help/beta_plus_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/Bsignature_M11.mws...
Success... inserting topic Clifford,Bsignature from the file:
```

C:\Maple11/P11/Cliff11/Help/Bsignature_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/buildm_M11.mws...
Success... inserting topic Clifford,buildm from the file:
```

C:\Maple11/P11/Cliff11/Help/buildm_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/bygrade_M11.mws...
Success... inserting topic Clifford,bygrade from the file:
```

C:\Maple11/P11/Cliff11/Help/bygrade_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/cbasis_M11.mws...
Success... inserting topic Clifford,cbasis from the file:
```

C:\Maple11/P11/Cliff11/Help/cbasis_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/cdfmatrix_M11.mws...
Success... inserting topic Clifford,cdfmatrix from the file:
```

C:\Maple11/P11/Cliff11/Help/cdfmatrix_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/cexpQ_M11.mws...
Success... inserting topic Clifford,cexpQ from the file:
```

C:\Maple11/P11/Cliff11/Help/cexpQ_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/cexp_M11.mws...
Success... inserting topic Clifford,cexp from the file:
```

C:\Maple11/P11/Cliff11/Help/cexp_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/cinv_M11.mws...
Success... inserting topic Clifford,cinv from the file:
```

C:\Maple11/P11/Cliff11/Help/cinv_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/clibasmon_M11.mws...
Success... inserting topic Clifford,type,clibasmon from the file:
```

C:\Maple11/P11/Cliff11/Help/clibasmon_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/clibilinear_M11.mws...
Success... inserting topic Clifford,clibilinear from the file:
```

C:\Maple11/P11/Cliff11/Help/clibilinear_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/clicollect_M11.mws...
Success... inserting topic Clifford,clicollect from the file:
```

C:\Maple11/P11/Cliff11/Help/clicollect_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/clidata_M11.mws...
Success... inserting topic Clifford,clidata from the file:
```

C:\Maple11/P11/Cliff11/Help/clidata_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/CLIFFORD_ENV_M11.mws...
Success... inserting topic Clifford,CLIFFORD_ENV from the file:
```

C:\Maple11/P11/Cliff11/Help/CLIFFORD_ENV_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/clilinear_M11.mws...
Success... inserting topic Clifford,clilinear from the file:
```

C:\Maple11/P11/Cliff11/Help/clilinear_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/climatrix_M11.mws...
Success... inserting topic Clifford,type,climatrix from the file:
```

C:\Maple11/P11/Cliff11/Help/climatrix_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/climinpoly_M11.mws...
Success... inserting topic Clifford,climinpoly from the file:
```

C:\Maple11/P11/Cliff11/Help/climinpoly_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/climon_M11.mws...
Success... inserting topic Clifford,type,climon from the file:
```

C:\Maple11/P11/Cliff11/Help/climon_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/cliparse_M11.mws...
Success... inserting topic Clifford,cliparse from the file:
```

C:\Maple11/P11/Cliff11/Help/cliparse_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/clipolynom_M11.mws...
Success... inserting topic Clifford,type,clipolynom from the file:
```

C:\Maple11/P11/Cliff11/Help/clipolynom_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/cliprod_M11.mws...
Success... inserting topic Clifford,type,cliprod from the file:
```

C:\Maple11/P11/Cliff11/Help/cliprod_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/cliremove_M11.mws...
Success... inserting topic Clifford,cliremove from the file:
```

C:\Maple11/P11/Cliff11/Help/cliremove_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/cliscalar_M11.mws...
Success... inserting topic Clifford,type,cliscalar from the file:
```

C:\Maple11/P11/Cliff11/Help/cliscalar_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/clisolve_M11.mws...
Success... inserting topic Clifford,clisolve from the file:
```

C:\Maple11/P11/Cliff11/Help/clisolve_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/clisort_M11.mws...
Success... inserting topic Clifford,clisort from the file:
```

C:\Maple11/P11/Cliff11/Help/clisort_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/cliterms_M11.mws...
Success... inserting topic Clifford,cliterms from the file:
```

C:\Maple11/P11/Cliff11/Help/cliterms_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/cmulgen_M11.mws...
Success... inserting topic Clifford,cmulgen from the file:
```

C:\Maple11/P11/Cliff11/Help/cmulgen_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/cmulNUM_M11.mws...
Success... inserting topic Clifford,cmulNUM from the file:
```

C:\Maple11/P11/Cliff11/Help/cmulNUM_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/cmulQ_M11.mws...
```

```
Success... inserting topic Clifford,cmulQ from the file:
```

C:\Maple11/P11/Cliff11/Help/cmulQ_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/cmulRS_M11.mws...
Success... inserting topic Clifford,cmulRS from the file:
```

C:\Maple11/P11/Cliff11/Help/cmulRS_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/cmul_M11.mws...
Success... inserting topic Clifford,cmul from the file:
```

C:\Maple11/P11/Cliff11/Help/cmul_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/cmul_user_defined_M11.mws...
Success... inserting topic Clifford,cmul_user_defined from the file:
```

C:\Maple11/P11/Cliff11/Help/cmul_user_defined_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/cocycle_M11.mws...
Success... inserting topic Clifford,cocycle from the file:
```

C:\Maple11/P11/Cliff11/Help/cocycle_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/commutingelements_M11.mws...
Success... inserting topic Clifford,commutingelements from the file:
```

C:\Maple11/P11/Cliff11/Help/commutingelements_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/conjugation_M11.mws...
Success... inserting topic Clifford,conjugation from the file:
```

C:\Maple11/P11/Cliff11/Help/conjugation_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/c_conjug_M11.mws...
Success... inserting topic Clifford,c_conjug from the file:
```

C:\Maple11/P11/Cliff11/Help/c_conjug_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/ddfmatrix_M11.mws...
Success... inserting topic Clifford,ddfmatrix from the file:
```

C:\Maple11/P11/Cliff11/Help/ddfmatrix_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/dfmatrix_M11.mws...
Success... inserting topic Clifford,type,dfmatrix from the file:
```

C:\Maple11/P11/Cliff11/Help/dfmatrix_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/diagmatrix_M11.mws...
Success... inserting topic Clifford,type,diagmatrix from the file:
```

C:\Maple11/P11/Cliff11/Help/diagmatrix_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/diagonalize_M11.mws...
Success... inserting topic Clifford,diagonalize from the file:
```

C:\Maple11/P11/Cliff11/Help/diagonalize_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/displayid_M11.mws...
Success... inserting topic Clifford,displayid from the file:
```

C:\Maple11/P11/Cliff11/Help/displayid_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/evenelement_M11.mws...
Success... inserting topic Clifford,type,evenelement from the file:
```

C:\Maple11/P11/Cliff11/Help/evenelement_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/extract_M11.mws...
Success... inserting topic Clifford,extract from the file:
```

C:\Maple11/P11/Cliff11/Help/extract_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/factoridempotent_M11.mws...
Success... inserting topic Clifford,factoridempotent from the file:
```

C:\Maple11/P11/Cliff11/Help/factoridempotent_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/fieldelement_M11.mws...
Success... inserting topic Clifford,type,fieldelement from the file:
```

C:\Maple11/P11/Cliff11/Help/fieldelement_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/find1str_M11.mws...
Success... inserting topic Clifford,find1str from the file:
```

C:\Maple11/P11/Cliff11/Help/find1str_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/findbasis_M11.mws...
Success... inserting topic Clifford,findbasis from the file:
```

C:\Maple11/P11/Cliff11/Help/findbasis_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/gencomplex_M11.mws...
Success... inserting topic Clifford,type,gencomplex from the file:
```

C:\Maple11/P11/Cliff11/Help/gencomplex_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/genquatbasis_M11.mws...
Success... inserting topic Clifford,type,genquatbasis from the file:
```

C:\Maple11/P11/Cliff11/Help/genquatbasis_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/genquaternion_M11.mws...
Success... inserting topic Clifford,type,genquaternion from the file:
```

C:\Maple11/P11/Cliff11/Help/genquaternion_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/gradeinv_M11.mws...
Success... inserting topic Clifford,gradeinv from the file:
```

C:\Maple11/P11/Cliff11/Help/gradeinv_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/idempotent_M11.mws...
Success... inserting topic Clifford,type,idempotent from the file:
```

C:\Maple11/P11/Cliff11/Help/idempotent_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/intro_M11.mws...
Success... inserting topic Clifford,intro from the file:
```

C:\Maple11/P11/Cliff11/Help/intro_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/isproduct_M11.mws...
Success... inserting topic Clifford,isproduct from the file:
```

C:\Maple11/P11/Cliff11/Help/isproduct_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/isVahlenmatrix_M11.mws...
Success... inserting topic Clifford,isVahlenmatrix from the file:
```

C:\Maple11/P11/Cliff11/Help/isVahlenmatrix_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/Kfield_M11.mws...
Success... inserting topic Clifford,Kfield from the file:
```

C:\Maple11/P11/Cliff11/Help/Kfield_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/LCQ_M11.mws...
Success... inserting topic Clifford,LCQ from the file:
```

C:\Maple11/P11/Cliff11/Help/LCQ_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/LC_M11.mws...
Success... inserting topic Clifford,LC from the file:
```

C:\Maple11/P11/Cliff11/Help/LC_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/makealiases_M11.mws...
Success... inserting topic Clifford,makealiases from the file:
```

C:\Maple11/P11/Cliff11/Help/makealiases_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/makeclibasmon_M11.mws...
Success... inserting topic Clifford,makeclibasmon from the file:
```

C:\Maple11/P11/Cliff11/Help/makeclibasmon_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/matKrepr_M11.mws...
Success... inserting topic Clifford,matKrepr from the file:
```

C:\Maple11/P11/Cliff11/Help/matKrepr_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/maxgrade_M11.mws...
Success... inserting topic Clifford,maxgrade from the file:
```

C:\Maple11/P11/Cliff11/Help/maxgrade_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/maxindex_M11.mws...
Success... inserting topic Clifford,maxindex from the file:
```

C:\Maple11/P11/Cliff11/Help/maxindex_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/mdfmatrix_M11.mws...
Success... inserting topic Clifford,mdfmatrix from the file:
```

C:\Maple11/P11/Cliff11/Help/mdfmatrix_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/minimalideal_M11.mws...
Success... inserting topic Clifford,minimalideal from the file:
```

C:\Maple11/P11/Cliff11/Help/minimalideal_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/mlist_M11.mws...
Success... inserting topic Clifford,convert,mlist from the file:
```

C:\Maple11/P11/Cliff11/Help/mlist_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/nilpotent_M11.mws...
Success... inserting topic Clifford,type,nilpotent from the file:
```

C:\Maple11/P11/Cliff11/Help/nilpotent_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/oddelement_M11.mws...
Success... inserting topic Clifford,type,oddelement from the file:
```

C:\Maple11/P11/Cliff11/Help/oddelement_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/ord_M11.mws...
Success... inserting topic Clifford,ord from the file:
```

C:\Maple11/P11/Cliff11/Help/ord_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/permsign_M11.mws...
Success... inserting topic Clifford,permsign from the file:
```

C:\Maple11/P11/Cliff11/Help/permsign_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/primitiveidemp_M11.mws...
Success... inserting topic Clifford,type,primitiveidemp from the file:
```

C:\Maple11/P11/Cliff11/Help/primitiveidemp_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/pseudodet_M11.mws...
Success... inserting topic Clifford,pseudodet from the file:
```

C:\Maple11/P11/Cliff11/Help/pseudodet_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/purequatbasis_M11.mws...
Success... inserting topic Clifford,type,purequatbasis from the file:
```

C:\Maple11/P11/Cliff11/Help/purequatbasis_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/qdisplay_M11.mws...
Success... inserting topic Clifford,qdisplay from the file:
```

C:\Maple11/P11/Cliff11/Help/qdisplay_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/qinv_M11.mws...
Success... inserting topic Clifford,qinv from the file:
```

C:\Maple11/P11/Cliff11/Help/qinv_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/qmul_M11.mws...
Success... inserting topic Clifford,qmul from the file:
```

C:\Maple11/P11/Cliff11/Help/qmul_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/qnorm_M11.mws...
Success... inserting topic Clifford,qnorm from the file:
```

C:\Maple11/P11/Cliff11/Help/qnorm_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/quaternion_M11.mws...
```

```
Success... inserting topic Clifford,type,quaternion from the file:
```

C:\Maple11/P11/Cliff11/Help/quaternion_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/q_conjug_M11.mws...
Success... inserting topic Clifford,q_conjug from the file:
```

C:\Maple11/P11/Cliff11/Help/q_conjug_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/RCQ_M11.mws...
Success... inserting topic Clifford,RCQ from the file:
```

C:\Maple11/P11/Cliff11/Help/RCQ_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/RC_M11.mws...
Success... inserting topic Clifford,RC from the file:
```

C:\Maple11/P11/Cliff11/Help/RC_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/rd_clibasmon_M11.mws...
Success... inserting topic Clifford,rd_clibasmon from the file:
```

C:\Maple11/P11/Cliff11/Help/rd_clibasmon_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/rd_climon_M11.mws...
Success... inserting topic Clifford,rd_climon from the file:
```

C:\Maple11/P11/Cliff11/Help/rd_climon_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/rd_clipolynom_M11.mws...
Success... inserting topic Clifford,rd_clipolynom from the file:
```

C:\Maple11/P11/Cliff11/Help/rd_clipolynom_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/reorder_M11.mws...
Success... inserting topic Clifford,reorder from the file:
```

C:\Maple11/P11/Cliff11/Help/reorder_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/reversion_M11.mws...
Success... inserting topic Clifford,reversion from the file:
```

C:\Maple11/P11/Cliff11/Help/reversion_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/RHnumber_M11.mws...
Success... inserting topic Clifford,RHnumber from the file:
```

C:\Maple11/P11/Cliff11/Help/RHnumber_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/rmulm_M11.mws...
Success... inserting topic Clifford,rmulm from the file:
```

C:\Maple11/P11/Cliff11/Help/rmulm_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/rot3d_M11.mws...
Success... inserting topic Clifford,rot3d from the file:
```

C:\Maple11/P11/Cliff11/Help/rot3d_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/scalarpart_M11.mws...
Success... inserting topic Clifford,scalarpart from the file:
```

C:\Maple11/P11/Cliff11/Help/scalarpart_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/setup_M11.mws...
Success... inserting topic Clifford,setup from the file:
```

C:\Maple11/P11/Cliff11/Help/setup_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/sexp_M11.mws...
Success... inserting topic Clifford,sexp from the file:
```

C:\Maple11/P11/Cliff11/Help/sexp_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/specify_constants_M11.mws...
Success... inserting topic Clifford,specify_constants from the file:
```

C:\Maple11/P11/Cliff11/Help/specify_constants_M11.mws

```
Trying to read file C:\Maple11/P11/Cliff11/Help/spinorKbasis_M11.mws...
Success... inserting topic Clifford,spinorKbasis from the file:
```

```
                    C:\Maple11/P11/Cliff11/Help/spinorKbasis_M11.mws
Trying to read file C:\Maple11/P11/Cliff11/Help/spinorKrepr_M11.mws...
Success... inserting topic Clifford,spinorKrepr from the file:
                    C:\Maple11/P11/Cliff11/Help/spinorKrepr_M11.mws
Trying to read file C:\Maple11/P11/Cliff11/Help/squaremodf_M11.mws...
Success... inserting topic Clifford,squaremodf from the file:
                    C:\Maple11/P11/Cliff11/Help/squaremodf_M11.mws
Trying to read file C:\Maple11/P11/Cliff11/Help/str_to_int_M11.mws...
Success... inserting topic Clifford,convert,str_to_int from the file:
                    C:\Maple11/P11/Cliff11/Help/str_to_int_M11.mws
Trying to read file C:\Maple11/P11/Cliff11/Help/subs_clipolynom_M11.mws...
Success... inserting topic Clifford,subs_clipolynom from the file:
                    C:\Maple11/P11/Cliff11/Help/subs_clipolynom_M11.mws
Trying to read file C:\Maple11/P11/Cliff11/Help/symmatrix_M11.mws...
Success... inserting topic Clifford,type,symmatrix from the file:
                    C:\Maple11/P11/Cliff11/Help/symmatrix_M11.mws
Trying to read file C:\Maple11/P11/Cliff11/Help/tensorprod_M11.mws...
Success... inserting topic Clifford,type,tensorprod from the file:
                    C:\Maple11/P11/Cliff11/Help/tensorprod_M11.mws
Trying to read file C:\Maple11/P11/Cliff11/Help/useproduct_M11.mws...
Success... inserting topic Clifford,useproduct from the file:
                    C:\Maple11/P11/Cliff11/Help/useproduct_M11.mws
Trying to read file C:\Maple11/P11/Cliff11/Help/vectorpart_M11.mws...
Success... inserting topic Clifford,vectorpart from the file:
                    C:\Maple11/P11/Cliff11/Help/vectorpart_M11.mws
Trying to read file C:\Maple11/P11/Cliff11/Help/version_M11.mws...
Success... inserting topic Clifford,version from the file:
                    C:\Maple11/P11/Cliff11/Help/version_M11.mws
Trying to read file C:\Maple11/P11/Cliff11/Help/wedge_M11.mws...
Success... inserting topic Clifford,wedge from the file:
                    C:\Maple11/P11/Cliff11/Help/wedge_M11.mws
Trying to read file C:\Maple11/P11/Cliff11/Help/wexp_M11.mws...
Success... inserting topic Clifford,wexp from the file:
                    C:\Maple11/P11/Cliff11/Help/wexp_M11.mws
***********************
Finished inserting 110 file topics into the HDB and Browser
***********************
```

```
> #?climatrix
>
```

Inserting help pages for Bigebra:

```
> HDB_LIB_PATH:=convert(libname[1],name);
  HELP_FILE_PATH:=`C:\\Maple11/P11/Bigebra11/Help/`;  ##Directory
  where *_M11.mws help files are located
  BROWSER_PATH:=`Mathematics/Algebra/`;
  ModuleName  :=`Bigebra`;
  Parent      :=`Bigebra,help`;
```

```
GrandParent :=`Clifford,intro`;
MapleVersion:="M11";
typesLIST:=["tensorpolynom"];
convertsLIST:=[];
```

HDB_LIB_PATH := *C:\Maple11/Cliffordlib*

HELP_FILE_PATH := *C:\Maple11/P11/Bigebra11/Help/*

BROWSER_PATH := *Mathematics/Algebra/*

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

ModuleName := *Bigebra*

Parent := *Bigebra,help*

GrandParent := *Clifford,intro*

MapleVersion := "M11"

typesLIST := ["tensorpolynom"]

convertsLIST := [ ]

Step 1: Generating an automatic list from help page files:

```
> Lauto:=makeLIST(ModuleName,Parent,GrandParent,HELP_FILE_PATH,Maple
  Version);
```

Lauto := [[*Bigebra,&cco*, *Bigebra,help*, ["Bigebra,&cco", "&cco"]],
[*Bigebra,&gco_d*, *Bigebra,help*, ["Bigebra,&gco_d", "&gco_d"]],
[*Bigebra,&gco*, *Bigebra,help*, ["Bigebra,&gco", "&gco"]],
[*Bigebra,&gco_pl*, *Bigebra,help*, ["Bigebra,&gco_pl", "&gco_pl"]],
[*Bigebra,&map*, *Bigebra,help*, ["Bigebra,&map", "&map"]],
[*Bigebra,&t*, *Bigebra,help*, ["Bigebra,&t", "&t"]],
[*Bigebra,&v*, *Bigebra,help*, ["Bigebra,&v", "&v"]],
[*Bigebra,bracket*, *Bigebra,help*, ["Bigebra,bracket", "bracket"]],
[*Bigebra,contract*, *Bigebra,help*, ["Bigebra,contract", "contract"]],
[*Bigebra,define*, *Bigebra,help*, ["Bigebra,define", "define"]],
[*Bigebra,drop_t*, *Bigebra,help*, ["Bigebra,drop_t", "drop_t"]],
[*Bigebra,EV*, *Bigebra,help*, ["Bigebra,EV", "EV"]],
[*Bigebra,gantipode*, *Bigebra,help*, ["Bigebra,gantipode", "gantipode"]],
[*Bigebra,gco_unit*, *Bigebra,help*, ["Bigebra,gco_unit", "gco_unit"]],
[*Bigebra,gswitch*, *Bigebra,help*, ["Bigebra,gswitch", "gswitch"]],
[*Bigebra,help*, *Clifford,intro*, ["Bigebra,help", "help"]],
[*Bigebra,init*, *Bigebra,help*, ["Bigebra,init", "init"]],
[*Bigebra,linop2*, *Bigebra,help*, ["Bigebra,linop2", "linop2"]],

[*Bigebra,linop*, *Bigebra,help*, ["Bigebra,linop", "linop"]],
[*Bigebra,lists2mat2*, *Bigebra,help*, ["Bigebra,lists2mat2", "lists2mat2"]],
[*Bigebra,lists2mat*, *Bigebra,help*, ["Bigebra,lists2mat", "lists2mat"]],
[*Bigebra,make_BI_Id*, *Bigebra,help*, ["Bigebra,make_BI_Id", "make_BI_Id"]],
[*Bigebra,mapop2*, *Bigebra,help*, ["Bigebra,mapop2", "mapop2"]],
[*Bigebra,mapop*, *Bigebra,help*, ["Bigebra,mapop", "mapop"]],
[*Bigebra,meet*, *Bigebra,help*, ["Bigebra,meet", "meet"]],
[*Bigebra,op2mat2*, *Bigebra,help*, ["Bigebra,op2mat2", "op2mat2"]],
[*Bigebra,op2mat*, *Bigebra,help*, ["Bigebra,op2mat", "op2mat"]],
[*Bigebra,pairing*, *Bigebra,help*, ["Bigebra,pairing", "pairing"]],
[*Bigebra,peek*, *Bigebra,help*, ["Bigebra,peek", "peek"]],
[*Bigebra,poke*, *Bigebra,help*, ["Bigebra,poke", "poke"]],
[*Bigebra,remove_eq*, *Bigebra,help*, ["Bigebra,remove_eq", "remove_eq"]],
[*Bigebra,switch*, *Bigebra,help*, ["Bigebra,switch", "switch"]],
[*Bigebra,tcollect*, *Bigebra,help*, ["Bigebra,tcollect", "tcollect"]],
[*Bigebra,tensorbasmonom*, *Bigebra,help*, ["Bigebra,tensorbasmonom", "tensorbasmonom"]],
[*Bigebra,tensormonom*, *Bigebra,help*, ["Bigebra,tensormonom", "tensormonom"]], [
Bigebra,type,tensorpolynom, *Bigebra,help*,
["Bigebra,type,tensorpolynom", "type,tensorpolynom"]],
[*Bigebra,tsolve1*, *Bigebra,help*, ["Bigebra,tsolve1", "tsolve1"]],
[*Bigebra,VERSION*, *Bigebra,help*, ["Bigebra,VERSION", "VERSION"]]]

```
>
```

Step 2: Modifying, if needed, certain entries in Lauto list:

```
> modsLIST:=[
  [`Bigebra,&cco`,["Bigebra,Clifford co-product","&cco"]],
  [`Bigebra,&gco_d`,["Bigebra,Grassmann dotted
  co-product","&gco_d"]],
  [`Bigebra,&gco`,["Bigebra,Grassmann co-product","&gco"]],
  [`Bigebra,&gco_d`,["Bigebra,Grassmann dotted
  co-product","&gco_d"]],
  [`Bigebra,&gco`,["Bigebra,Grassmann co-product","&gco"]],
  [`Bigebra,&gco_pl`,["Bigebra,Grassmann-Pluecker co-product",

  "Bigebra,Grassmann-Pluecker co-product","&gco_pl"]],
  [`Bigebra,&v`,["Bigebra,meet","Bigebra,join","&v","meet","join"]],

  [`Bigebra,bracket`,["Bigebra,Peano bracket","Bigebra,volume
  form","Bigebra,bracket","bracket"]],
```

```
[`Bigebra,EV`,["Bigebra,eval","Bigebra,EV","EV"]],
[`Bigebra,gantipode`,["Bigebra,gantipode","gantipode","antipode"]]
,
[`Bigebra,help`,["Bigebra","Bigebra,intro","Bigebra,start","Bigebr
a,help"]],
[`Bigebra,linop2`,["Bigebra,linop2","linop2","linop","Bigebra,lino
p"]],
[`Bigebra,linop`,["Bigebra,linop","linop2","linop","Bigebra,linop2
"]],
[`Bigebra,lists2mat2`,["Bigebra,lists2mat2","lists2mat2"]],
[`Bigebra,lists2mat`,["Bigebra,lists2mat","lists2mat"]],
[`Bigebra,mapop2`,["Bigebra,mapop2","mapop2","Bigebra,mapop","mapo
p"]],
[`Bigebra,mapop`,["Bigebra,mapop","mapop","Bigebra,mapop2","mapop2
"]],
[`Bigebra,meet`,["Bigebra,meet","Bigebra,join","&v","meet","join"]
],
[`Bigebra,op2mat`,["Bigebra,op2mat","op2mat"]],
[`Bigebra,op2mat2`,["Bigebra,op2mat2","op2mat2"]],
[`Bigebra,pairing`,["Bigebra,dual
product","Bigebra,pairing","pairing"]],
[`Bigebra,switch`,["crossing","Bigebra,switch","switch"]],
[`Bigebra,type,tensorpolynom`,

["Bigebra,type,tensorpolynom","Bigebra,type,tensorbasmonom","Bigeb
ra,type,tensormonom"]]
]:
> for mem in modsLIST do
    Lauto:=modifyLIST(Lauto,op(mem))
end do:
Lauto;
```

[[*Bigebra,&cco*, *Bigebra,help*, ["Bigebra,Clifford co-product", "&cco"]],
[*Bigebra,&gco_d*, *Bigebra,help*, ["Bigebra,Grassmann dotted co-product", "&gco_d"]],
[*Bigebra,&gco*, *Bigebra,help*, ["Bigebra,Grassmann co-product", "&gco"]], [*Bigebra,&gco_pl*, *Bigebra,help*, [
"Bigebra,Grassmann-Pluecker co-product", "Bigebra,Grassmann-Pluecker co-product", "&gco_pl"
]], [*Bigebra,&map*, *Bigebra,help*, ["Bigebra,&map", "&map"]],
[*Bigebra,&t*, *Bigebra,help*, ["Bigebra,&t", "&t"]],
[*Bigebra,&v*, *Bigebra,help*, ["Bigebra,meet", "Bigebra,join", "&v", "meet", "join"]], [
Bigebra,bracket, *Bigebra,help*,
["Bigebra,Peano bracket", "Bigebra,volume form", "Bigebra,bracket", "bracket"]],

[*Bigebra,contract*, *Bigebra,help*, ["Bigebra,contract", "contract"]],
[*Bigebra,define*, *Bigebra,help*, ["Bigebra,define", "define"]],
[*Bigebra,drop_t*, *Bigebra,help*, ["Bigebra,drop_t", "drop_t"]],
[*Bigebra,EV*, *Bigebra,help*, ["Bigebra,eval", "Bigebra,EV", "EV"]],
[*Bigebra,gantipode*, *Bigebra,help*, ["Bigebra,gantipode", "gantipode", "antipode"]],
[*Bigebra,gco_unit*, *Bigebra,help*, ["Bigebra,gco_unit", "gco_unit"]],
[*Bigebra,gswitch*, *Bigebra,help*, ["Bigebra,gswitch", "gswitch"]],
[*Bigebra,help*, *Clifford,intro*, ["Bigebra", "Bigebra,intro", "Bigebra,start", "Bigebra,help"]],
[*Bigebra,init*, *Bigebra,help*, ["Bigebra,init", "init"]],
[*Bigebra,linop2*, *Bigebra,help*, ["Bigebra,linop2", "linop2", "linop", "Bigebra,linop"]],
[*Bigebra,linop*, *Bigebra,help*, ["Bigebra,linop", "linop2", "linop", "Bigebra,linop2"]],
[*Bigebra,lists2mat2*, *Bigebra,help*, ["Bigebra,lists2mat2", "lists2mat2"]],
[*Bigebra,lists2mat*, *Bigebra,help*, ["Bigebra,lists2mat", "lists2mat"]],
[*Bigebra,make_BI_Id*, *Bigebra,help*, ["Bigebra,make_BI_Id", "make_BI_Id"]],
[*Bigebra,mapop2*, *Bigebra,help*, ["Bigebra,mapop2", "mapop2", "Bigebra,mapop", "mapop"]],
[*Bigebra,mapop*, *Bigebra,help*, ["Bigebra,mapop", "mapop", "Bigebra,mapop2", "mapop2"]],
[*Bigebra,meet*, *Bigebra,help*, ["Bigebra,meet", "Bigebra,join", "&v", "meet", "join"]],
[*Bigebra,op2mat2*, *Bigebra,help*, ["Bigebra,op2mat2", "op2mat2"]],
[*Bigebra,op2mat*, *Bigebra,help*, ["Bigebra,op2mat", "op2mat"]],
[*Bigebra,pairing*, *Bigebra,help*, ["Bigebra,dual product", "Bigebra,pairing", "pairing"]],
[*Bigebra,peek*, *Bigebra,help*, ["Bigebra,peek", "peek"]],
[*Bigebra,poke*, *Bigebra,help*, ["Bigebra,poke", "poke"]],
[*Bigebra,remove_eq*, *Bigebra,help*, ["Bigebra,remove_eq", "remove_eq"]],
[*Bigebra,switch*, *Bigebra,help*, ["crossing", "Bigebra,switch", "switch"]],
[*Bigebra,tcollect*, *Bigebra,help*, ["Bigebra,tcollect", "tcollect"]],
[*Bigebra,tensorbasmonom*, *Bigebra,help*, ["Bigebra,tensorbasmonom", "tensorbasmonom"]],
[*Bigebra,tensormonom*, *Bigebra,help*, ["Bigebra,tensormonom", "tensormonom"]], [
Bigebra,type,tensorpolynom, *Bigebra,help*,
["Bigebra,type,tensorpolynom", "Bigebra,type,tensorbasmonom", "Bigebra,type,tensormonom"]],
[*Bigebra,tsolve1*, *Bigebra,help*, ["Bigebra,tsolve1", "tsolve1"]],
[*Bigebra,VERSION*, *Bigebra,help*, ["Bigebra,VERSION", "VERSION"]]]

```
>
```

Step 3: Inserting all help pages into HDB and the browser using the last modified list:

```
> insert_helppages(Lauto,MapleVersion);
Trying to read file C:\Maple11/P11/Bigebra11/Help/&cco_M11.mws...
Success... inserting topic Bigebra,&cco from the file:
                    C:\Maple11/P11/Bigebra11/Help/&cco_M11.mws
Trying to read file C:\Maple11/P11/Bigebra11/Help/&gco_d_M11.mws...
```

```
Success... inserting topic Bigebra,&gco_d from the file:
```

C:\Maple11/P11/Bigebra11/Help/&gco_d_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/&gco_M11.mws...
Success... inserting topic Bigebra,&gco from the file:
```

C:\Maple11/P11/Bigebra11/Help/&gco_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/&gco_pl_M11.mws...
Success... inserting topic Bigebra,&gco_pl from the file:
```

C:\Maple11/P11/Bigebra11/Help/&gco_pl_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/&map_M11.mws...
Success... inserting topic Bigebra,&map from the file:
```

C:\Maple11/P11/Bigebra11/Help/&map_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/&t_M11.mws...
Success... inserting topic Bigebra,&t from the file:
```

C:\Maple11/P11/Bigebra11/Help/&t_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/&v_M11.mws...
Success... inserting topic Bigebra,&v from the file:
```

C:\Maple11/P11/Bigebra11/Help/&v_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/bracket_M11.mws...
Success... inserting topic Bigebra,bracket from the file:
```

C:\Maple11/P11/Bigebra11/Help/bracket_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/contract_M11.mws...
Success... inserting topic Bigebra,contract from the file:
```

C:\Maple11/P11/Bigebra11/Help/contract_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/define_M11.mws...
Success... inserting topic Bigebra,define from the file:
```

C:\Maple11/P11/Bigebra11/Help/define_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/drop_t_M11.mws...
Success... inserting topic Bigebra,drop_t from the file:
```

C:\Maple11/P11/Bigebra11/Help/drop_t_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/EV_M11.mws...
Success... inserting topic Bigebra,EV from the file:
```

C:\Maple11/P11/Bigebra11/Help/EV_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/gantipode_M11.mws...
Success... inserting topic Bigebra,gantipode from the file:
```

C:\Maple11/P11/Bigebra11/Help/gantipode_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/gco_unit_M11.mws...
Success... inserting topic Bigebra,gco_unit from the file:
```

C:\Maple11/P11/Bigebra11/Help/gco_unit_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/gswitch_M11.mws...
Success... inserting topic Bigebra,gswitch from the file:
```

C:\Maple11/P11/Bigebra11/Help/gswitch_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/help_M11.mws...
Success... inserting topic Bigebra,help from the file:
```

C:\Maple11/P11/Bigebra11/Help/help_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/init_M11.mws...
Success... inserting topic Bigebra,init from the file:
```

C:\Maple11/P11/Bigebra11/Help/init_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/linop2_M11.mws...
Success... inserting topic Bigebra,linop2 from the file:
```

C:\Maple11/P11/Bigebra11/Help/linop2_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/linop_M11.mws...
Success... inserting topic Bigebra,linop from the file:
```

C:\Maple11/P11/Bigebra11/Help/linop_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/lists2mat2_M11.mws...
Success... inserting topic Bigebra,lists2mat2 from the file:
```

C:\Maple11/P11/Bigebra11/Help/lists2mat2_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/lists2mat_M11.mws...
Success... inserting topic Bigebra,lists2mat from the file:
```

C:\Maple11/P11/Bigebra11/Help/lists2mat_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/make_BI_Id_M11.mws...
Success... inserting topic Bigebra,make_BI_Id from the file:
```

C:\Maple11/P11/Bigebra11/Help/make_BI_Id_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/mapop2_M11.mws...
Success... inserting topic Bigebra,mapop2 from the file:
```

C:\Maple11/P11/Bigebra11/Help/mapop2_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/mapop_M11.mws...
Success... inserting topic Bigebra,mapop from the file:
```

C:\Maple11/P11/Bigebra11/Help/mapop_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/meet_M11.mws...
Success... inserting topic Bigebra,meet from the file:
```

C:\Maple11/P11/Bigebra11/Help/meet_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/op2mat2_M11.mws...
Success... inserting topic Bigebra,op2mat2 from the file:
```

C:\Maple11/P11/Bigebra11/Help/op2mat2_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/op2mat_M11.mws...
Success... inserting topic Bigebra,op2mat from the file:
```

C:\Maple11/P11/Bigebra11/Help/op2mat_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/pairing_M11.mws...
Success... inserting topic Bigebra,pairing from the file:
```

C:\Maple11/P11/Bigebra11/Help/pairing_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/peek_M11.mws...
Success... inserting topic Bigebra,peek from the file:
```

C:\Maple11/P11/Bigebra11/Help/peek_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/poke_M11.mws...
Success... inserting topic Bigebra,poke from the file:
```

C:\Maple11/P11/Bigebra11/Help/poke_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/remove_eq_M11.mws...
Success... inserting topic Bigebra,remove_eq from the file:
```

C:\Maple11/P11/Bigebra11/Help/remove_eq_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/switch_M11.mws...
Success... inserting topic Bigebra,switch from the file:
```

C:\Maple11/P11/Bigebra11/Help/switch_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/tcollect_M11.mws...
Success... inserting topic Bigebra,tcollect from the file:
```

C:\Maple11/P11/Bigebra11/Help/tcollect_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/tensorbasmonom_M11.mws...
Success... inserting topic Bigebra,tensorbasmonom from the file:
```

C:\Maple11/P11/Bigebra11/Help/tensorbasmonom_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/tensormonom_M11.mws...
Success... inserting topic Bigebra,tensormonom from the file:
```

C:\Maple11/P11/Bigebra11/Help/tensormonom_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/tensorpolynom_M11.mws...
Success... inserting topic Bigebra,type,tensorpolynom from the file:
```

C:\Maple11/P11/Bigebra11/Help/tensorpolynom_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/tsolve1_M11.mws...
Success... inserting topic Bigebra,tsolve1 from the file:
```

C:\Maple11/P11/Bigebra11/Help/tsolve1_M11.mws

```
Trying to read file C:\Maple11/P11/Bigebra11/Help/VERSION_M11.mws...
Success... inserting topic Bigebra,VERSION from the file:
```

C:\Maple11/P11/Bigebra11/Help/VERSION_M11.mws

```
************************
Finished inserting 38 file topics into the HDB and Browser
************************
```

```
> #?tensorpolynom
```

```
>
```

Inserting help pages for Cliplus:

```
> HDB_LIB_PATH:=convert(libname[1],name);
  HELP_FILE_PATH:=`C:\\Maple11/P11/Cliplus11/Help/`;
  BROWSER_PATH:=`Mathematics/Algebra/`;
  ModuleName    :=`Cliplus`;
  Parent        :=`Cliplus,setup`;
  GrandParent   :=`Clifford,intro`;
  MapleVersion:="M11";
  typesLIST:=[];
  convertsLIST:=["wedge_to_dwedge","dwedge_to_wedge"];
```

$$HDB\_LIB\_PATH := C:\backslash Maple11/Cliffordlib$$

$$HELP\_FILE\_PATH := C:\backslash Maple11/P11/Cliplus11/Help/$$

$$BROWSER\_PATH := Mathematics/Algebra/$$

$$ModuleName := Cliplus$$

$$Parent := Cliplus,setup$$

$$GrandParent := Clifford,intro$$

$$MapleVersion := \text{"M11"}$$

$$typesLIST := [\ ]$$

$$convertsLIST := [\text{"wedge\_to\_dwedge"}, \text{"dwedge\_to\_wedge"}]$$

Step 1: Generating an automatic list from help page files:

```
> Lauto:=makeLIST(ModuleName,Parent,GrandParent,HELP_FILE_PATH,Maple
  Version);
```

```
Lauto := [[Cliplus,&dw, Cliplus,setup, ["Cliplus,&dw", "&dw"]],
    [Cliplus,clibasis, Cliplus,setup, ["Cliplus,clibasis", "clibasis"]],
    [Cliplus,clieval, Cliplus,setup, ["Cliplus,clieval", "clieval"]],
    [Cliplus,cliexpand, Cliplus,setup, ["Cliplus,cliexpand", "cliexpand"]],
    [Cliplus,climul, Cliplus,setup, ["Cliplus,climul", "climul"]],
    [Cliplus,clirev, Cliplus,setup, ["Cliplus,clirev", "clirev"]],
    [Cliplus,dottedcbasis, Cliplus,setup, ["Cliplus,dottedcbasis", "dottedcbasis"]],
    [Cliplus,dwedge, Cliplus,setup, ["Cliplus,dwedge", "dwedge"]], [
    Cliplus,convert,dwedge_to_wedge, Cliplus,setup,
    ["Cliplus,convert,dwedge_to_wedge", "convert,dwedge_to_wedge"]],
    [Cliplus,LCbig, Cliplus,setup, ["Cliplus,LCbig", "LCbig"]],
    [Cliplus,makeclialiases, Cliplus,setup, ["Cliplus,makeclialiases", "makeclialiases"]],
    [Cliplus,RCbig, Cliplus,setup, ["Cliplus,RCbig", "RCbig"]],
    [Cliplus,setup, Clifford,intro, ["Cliplus,setup", "setup"]], [Cliplus,convert,wedge_to_dwedge,
    Cliplus,setup, ["Cliplus,convert,wedge_to_dwedge", "convert,wedge_to_dwedge"]]]
```

```
>
```

Step 2: Modifying, if needed, certain entries in Lauto list:

```
> modsLIST:=[
  [`Cliplus,&dw`,["Cliplus,dwedge","dwedge","&dw","Cliplus,&dw"]]
  ]:
> for mem in modsLIST do
      Lauto:=modifyLIST(Lauto,op(mem))
  end do:
  Lauto;
```

```
[[Cliplus,&dw, Cliplus,setup, ["Cliplus,dwedge", "dwedge", "&dw", "Cliplus,&dw"]],
    [Cliplus,clibasis, Cliplus,setup, ["Cliplus,clibasis", "clibasis"]],
    [Cliplus,clieval, Cliplus,setup, ["Cliplus,clieval", "clieval"]],
    [Cliplus,cliexpand, Cliplus,setup, ["Cliplus,cliexpand", "cliexpand"]],
    [Cliplus,climul, Cliplus,setup, ["Cliplus,climul", "climul"]],
    [Cliplus,clirev, Cliplus,setup, ["Cliplus,clirev", "clirev"]],
    [Cliplus,dottedcbasis, Cliplus,setup, ["Cliplus,dottedcbasis", "dottedcbasis"]],
    [Cliplus,dwedge, Cliplus,setup, ["Cliplus,dwedge", "dwedge"]], [
    Cliplus,convert,dwedge_to_wedge, Cliplus,setup,
    ["Cliplus,convert,dwedge_to_wedge", "convert,dwedge_to_wedge"]],
    [Cliplus,LCbig, Cliplus,setup, ["Cliplus,LCbig", "LCbig"]],
    [Cliplus,makeclialiases, Cliplus,setup, ["Cliplus,makeclialiases", "makeclialiases"]],
    [Cliplus,RCbig, Cliplus,setup, ["Cliplus,RCbig", "RCbig"]],
    [Cliplus,setup, Clifford,intro, ["Cliplus,setup", "setup"]], [Cliplus,convert,wedge_to_dwedge,
```

Cliplus,setup, ["Cliplus,convert,wedge_to_dwedge", "convert,wedge_to_dwedge"]]]

```
>
```

Step 3: Inserting all help pages into HDB and the browser using the last modified list:

```
> insert_helppages(Lauto,MapleVersion);
Trying to read file C:\Maple11/P11/Cliplus11/Help/&dw_M11.mws...
Success... inserting topic Cliplus,&dw from the file:
```

C:\Maple11/P11/Cliplus11/Help/&dw_M11.mws

```
Trying to read file C:\Maple11/P11/Cliplus11/Help/clibasis_M11.mws...
Success... inserting topic Cliplus,clibasis from the file:
```

C:\Maple11/P11/Cliplus11/Help/clibasis_M11.mws

```
Trying to read file C:\Maple11/P11/Cliplus11/Help/clieval_M11.mws...
Success... inserting topic Cliplus,clieval from the file:
```

C:\Maple11/P11/Cliplus11/Help/clieval_M11.mws

```
Trying to read file C:\Maple11/P11/Cliplus11/Help/cliexpand_M11.mws...
Success... inserting topic Cliplus,cliexpand from the file:
```

C:\Maple11/P11/Cliplus11/Help/cliexpand_M11.mws

```
Trying to read file C:\Maple11/P11/Cliplus11/Help/climul_M11.mws...
Success... inserting topic Cliplus,climul from the file:
```

C:\Maple11/P11/Cliplus11/Help/climul_M11.mws

```
Trying to read file C:\Maple11/P11/Cliplus11/Help/clirev_M11.mws...
Success... inserting topic Cliplus,clirev from the file:
```

C:\Maple11/P11/Cliplus11/Help/clirev_M11.mws

```
Trying to read file C:\Maple11/P11/Cliplus11/Help/dottedcbasis_M11.mws...
Success... inserting topic Cliplus,dottedcbasis from the file:
```

C:\Maple11/P11/Cliplus11/Help/dottedcbasis_M11.mws

```
Trying to read file C:\Maple11/P11/Cliplus11/Help/dwedge_M11.mws...
Success... inserting topic Cliplus,dwedge from the file:
```

C:\Maple11/P11/Cliplus11/Help/dwedge_M11.mws

```
Trying to read file C:\Maple11/P11/Cliplus11/Help/dwedge_to_wedge_M11.mws...
Success... inserting topic Cliplus,convert,dwedge_to_wedge from the file:
```

C:\Maple11/P11/Cliplus11/Help/dwedge_to_wedge_M11.mws

```
Trying to read file C:\Maple11/P11/Cliplus11/Help/LCbig_M11.mws...
Success... inserting topic Cliplus,LCbig from the file:
```

C:\Maple11/P11/Cliplus11/Help/LCbig_M11.mws

```
Trying to read file C:\Maple11/P11/Cliplus11/Help/makeclialiases_M11.mws...
Success... inserting topic Cliplus,makeclialiases from the file:
```

C:\Maple11/P11/Cliplus11/Help/makeclialiases_M11.mws

```
Trying to read file C:\Maple11/P11/Cliplus11/Help/RCbig_M11.mws...
Success... inserting topic Cliplus,RCbig from the file:
```

C:\Maple11/P11/Cliplus11/Help/RCbig_M11.mws

```
Trying to read file C:\Maple11/P11/Cliplus11/Help/setup_M11.mws...
Success... inserting topic Cliplus,setup from the file:
```

C:\Maple11/P11/Cliplus11/Help/setup_M11.mws

```
Trying to read file C:\Maple11/P11/Cliplus11/Help/wedge_to_dwedge_M11.mws...
Success... inserting topic Cliplus,convert,wedge_to_dwedge from the file:
```

C:\Maple11/P11/Cliplus11/Help/wedge_to_dwedge_M11.mws

```
***********************
Finished inserting 14 file topics into the HDB and Browser
***********************
```

```
> #?setup
```

Inserting help pages for GTP:

```
> HDB_LIB_PATH:=convert(libname[1],name);
  HELP_FILE_PATH:=`C:\\Maple11/P11/GTP11/Help/`;
  BROWSER_PATH:=`Mathematics/Algebra/`;
  ModuleName   :=`GTP`;
  Parent       :=`Clifford,setup`;
  GrandParent  :=`Clifford,intro`;
  MapleVersion:="M11";
  typesLIST:=["gradedeven","gradedmonom","gradedodd","gradedpolynom"
  ];
  converts:=[];
```

$$HDB\_LIB\_PATH := C:\backslash Maple11/Cliffordlib$$

$$HELP\_FILE\_PATH := C:\backslash Maple11/P11/GTP11/Help/$$

$$BROWSER\_PATH := Mathematics/Algebra/$$

$$ModuleName := GTP$$

$$Parent := Clifford,setup$$

$$GrandParent := Clifford,intro$$

$$MapleVersion := \text{"M11"}$$

$$typesLIST := [\text{"gradedeven"}, \text{"gradedmonom"}, \text{"gradedodd"}, \text{"gradedpolynom"}]$$

$$converts := [\ ]$$

```
>
```

Step 1: Generating an automatic list from help page files:

```
> Lauto:=makeLIST(ModuleName,Parent,GrandParent,HELP_FILE_PATH,Maple
  Version);
```

$Lauto := [[GTP,\&t, Clifford,setup, [\text{"GTP,\&t"}, \text{"\&t"}]],$
$[GTP,cmulB, Clifford,setup, [\text{"GTP,cmulB"}, \text{"cmulB"}]],$
$[GTP,gbasis, Clifford,setup, [\text{"GTP,gbasis"}, \text{"gbasis"}]],$
$[GTP,gcollect, Clifford,setup, [\text{"GTP,gcollect"}, \text{"gcollect"}]],$
$[GTP,gprod, Clifford,setup, [\text{"GTP,gprod"}, \text{"gprod"}]],$
$[GTP,type,gradedeven, Clifford,setup, [\text{"GTP,type,gradedeven"}, \text{"type,gradedeven"}]],$
$[GTP,type,gradedmonom, Clifford,setup, [\text{"GTP,type,gradedmonom"}, \text{"type,gradedmonom"}]],$
$[GTP,type,gradedodd, Clifford,setup, [\text{"GTP,type,gradedodd"}, \text{"type,gradedodd"}]],$
$[GTP,type,gradedpolynom, Clifford,setup, [\text{"GTP,type,gradedpolynom"}, \text{"type,gradedpolynom"}]]$
$, [GTP,gradedprod, Clifford,setup, [\text{"GTP,gradedprod"}, \text{"gradedprod"}]],$

[*GTP,grade*, *Clifford,setup*, ["GTP,grade", "grade"]],
[*GTP,tensorrank*, *Clifford,setup*, ["GTP,tensorrank", "tensorrank"]]]

```
>
```

Step 2: Modifying, if needed, certain entries in Lauto list:

```
> modsLIST:=[
  [`GTP,cmulB`,["GTP,cmulB","cmulB","cmul"]]
  ]:
> for mem in modsLIST do
      Lauto:=modifyLIST(Lauto,op(mem))
  end do:
  Lauto;
```

[[*GTP,&t*, *Clifford,setup*, ["GTP,&t", "&t"]],
[*GTP,cmulB*, *Clifford,setup*, ["GTP,cmulB", "cmulB", "cmul"]],
[*GTP,gbasis*, *Clifford,setup*, ["GTP,gbasis", "gbasis"]],
[*GTP,gcollect*, *Clifford,setup*, ["GTP,gcollect", "gcollect"]],
[*GTP,gprod*, *Clifford,setup*, ["GTP,gprod", "gprod"]],
[*GTP,type,gradedeven*, *Clifford,setup*, ["GTP,type,gradedeven", "type,gradedeven"]],
[*GTP,type,gradedmonom*, *Clifford,setup*, ["GTP,type,gradedmonom", "type,gradedmonom"]],
[*GTP,type,gradedodd*, *Clifford,setup*, ["GTP,type,gradedodd", "type,gradedodd"]],
[*GTP,type,gradedpolynom*, *Clifford,setup*, ["GTP,type,gradedpolynom", "type,gradedpolynom"]]
, [*GTP,gradedprod*, *Clifford,setup*, ["GTP,gradedprod", "gradedprod"]],
[*GTP,grade*, *Clifford,setup*, ["GTP,grade", "grade"]],
[*GTP,tensorrank*, *Clifford,setup*, ["GTP,tensorrank", "tensorrank"]]]

```
>
```

Step 3: Inserting all help pages into HDB and the browser using the last modified list:

```
> insert_helppages(Lauto,MapleVersion);
Trying to read file C:\Maple11/P11/GTP11/Help/&t_M11.mws...
Success... inserting topic GTP,&t from the file:
                        C:\Maple11/P11/GTP11/Help/&t_M11.mws
Trying to read file C:\Maple11/P11/GTP11/Help/cmulB_M11.mws...
Success... inserting topic GTP,cmulB from the file:
                        C:\Maple11/P11/GTP11/Help/cmulB_M11.mws
Trying to read file C:\Maple11/P11/GTP11/Help/gbasis_M11.mws...
Success... inserting topic GTP,gbasis from the file:
                        C:\Maple11/P11/GTP11/Help/gbasis_M11.mws
Trying to read file C:\Maple11/P11/GTP11/Help/gcollect_M11.mws...
Success... inserting topic GTP,gcollect from the file:
                        C:\Maple11/P11/GTP11/Help/gcollect_M11.mws
Trying to read file C:\Maple11/P11/GTP11/Help/gprod_M11.mws...
Success... inserting topic GTP,gprod from the file:
                        C:\Maple11/P11/GTP11/Help/gprod_M11.mws
Trying to read file C:\Maple11/P11/GTP11/Help/gradedeven_M11.mws...
Success... inserting topic GTP,type,gradedeven from the file:
```

C:\Maple11/P11/GTP11/Help/gradedeven_M11.mws

```
Trying to read file C:\Maple11/P11/GTP11/Help/gradedmonom_M11.mws...
Success... inserting topic GTP,type,gradedmonom from the file:
```

C:\Maple11/P11/GTP11/Help/gradedmonom_M11.mws

```
Trying to read file C:\Maple11/P11/GTP11/Help/gradedodd_M11.mws...
Success... inserting topic GTP,type,gradedodd from the file:
```

C:\Maple11/P11/GTP11/Help/gradedodd_M11.mws

```
Trying to read file C:\Maple11/P11/GTP11/Help/gradedpolynom_M11.mws...
Success... inserting topic GTP,type,gradedpolynom from the file:
```

C:\Maple11/P11/GTP11/Help/gradedpolynom_M11.mws

```
Trying to read file C:\Maple11/P11/GTP11/Help/gradedprod_M11.mws...
Success... inserting topic GTP,gradedprod from the file:
```

C:\Maple11/P11/GTP11/Help/gradedprod_M11.mws

```
Trying to read file C:\Maple11/P11/GTP11/Help/grade_M11.mws...
Success... inserting topic GTP,grade from the file:
```

C:\Maple11/P11/GTP11/Help/grade_M11.mws

```
Trying to read file C:\Maple11/P11/GTP11/Help/tensorrank_M11.mws...
Success... inserting topic GTP,tensorrank from the file:
```

C:\Maple11/P11/GTP11/Help/tensorrank_M11.mws

```
***********************
Finished inserting 12 file topics into the HDB and Browser
***********************
```

```
> ##?cmulB
>
> ##############
```

Inserting help pages for Octonion:

```
> HDB_LIB_PATH:=convert(libname[1],name);
  HELP_FILE_PATH:=`C:\\Maple11/P11/Octonion11/Help/`;
  BROWSER_PATH:=`Mathematics/Algebra/`;
  ModuleName   :=`Octonion`;
  Parent       :=`Octonion,setup`;
  GrandParent  :=`Clifford,intro`;
  MapleVersion:="M11";
  typesLIST    :=["Fano_triples","octonion"];
  convertsLIST:=[];
```

HDB_LIB_PATH := C:\Maple11/Cliffordlib

HELP_FILE_PATH := C:\Maple11/P11/Octonion11/Help/

BROWSER_PATH := Mathematics/Algebra/

ModuleName := Octonion

Parent := Octonion,setup

GrandParent := Clifford,intro

$$MapleVersion := \text{"M11"}$$
$$typesLIST := [\text{"Fano\_triples"}, \text{"octonion"}]$$
$$convertsLIST := [\ ]$$

Step 1: Generating an automatic list from help page files:

```
> Lauto:=makeLIST(ModuleName,Parent,GrandParent,HELP_FILE_PATH,Maple
  Version);
```

Lauto := [[*Octonion,associator*, *Octonion,setup*, ["Octonion,associator", "associator"]],
[*Octonion,commutator*, *Octonion,setup*, ["Octonion,commutator", "commutator"]],
[*Octonion,def_omultable*, *Octonion,setup*, ["Octonion,def_omultable", "def_omultable"]], [
Octonion,type,Fano_triples, *Octonion,setup*, ["Octonion,type,Fano_triples", "type,Fano_triples"]]
, [*Octonion,type,octonion*, *Octonion,setup*, ["Octonion,type,octonion", "type,octonion"]],
[*Octonion,oinv*, *Octonion,setup*, ["Octonion,oinv", "oinv"]],
[*Octonion,omultable*, *Octonion,setup*, ["Octonion,omultable", "omultable"]],
[*Octonion,omul*, *Octonion,setup*, ["Octonion,omul", "omul"]],
[*Octonion,onorm*, *Octonion,setup*, ["Octonion,onorm", "onorm"]],
[*Octonion,oversion*, *Octonion,setup*, ["Octonion,oversion", "oversion"]],
[*Octonion,o_conjug*, *Octonion,setup*, ["Octonion,o_conjug", "o_conjug"]],
[*Octonion,Phi*, *Octonion,setup*, ["Octonion,Phi", "Phi"]],
[*Octonion,purevectorpart*, *Octonion,setup*, ["Octonion,purevectorpart", "purevectorpart"]],
[*Octonion,realpart*, *Octonion,setup*, ["Octonion,realpart", "realpart"]],
[*Octonion,setup*, *Clifford,intro*, ["Octonion,setup", "setup"]]]

```
>
```

Step 2: Modifying, if needed, certain entries in Lauto list:

```
> modsLIST:=[
  [`Octonion,omul`,["Octonion,omul","omul","&o","octonion","Octonion
  "]]
  ]:
```

```
> for mem in modsLIST do
      Lauto:=modifyLIST(Lauto,op(mem))
  end do:
  Lauto;
```

[[*Octonion,associator*, *Octonion,setup*, ["Octonion,associator", "associator"]],
[*Octonion,commutator*, *Octonion,setup*, ["Octonion,commutator", "commutator"]],
[*Octonion,def_omultable*, *Octonion,setup*, ["Octonion,def_omultable", "def_omultable"]], [
Octonion,type,Fano_triples, *Octonion,setup*, ["Octonion,type,Fano_triples", "type,Fano_triples"]]
, [*Octonion,type,octonion*, *Octonion,setup*, ["Octonion,type,octonion", "type,octonion"]],
[*Octonion,oinv*, *Octonion,setup*, ["Octonion,oinv", "oinv"]],

[*Octonion,omultable*, *Octonion,setup*, ["Octonion,omultable", "omultable"]],
[*Octonion,omul*, *Octonion,setup*, ["Octonion,omul", "omul", "&o", "octonion", "Octonion"]],
[*Octonion,onorm*, *Octonion,setup*, ["Octonion,onorm", "onorm"]],
[*Octonion,oversion*, *Octonion,setup*, ["Octonion,oversion", "oversion"]],
[*Octonion,o_conjug*, *Octonion,setup*, ["Octonion,o_conjug", "o_conjug"]],
[*Octonion,Phi*, *Octonion,setup*, ["Octonion,Phi", "Phi"]],
[*Octonion,purevectorpart*, *Octonion,setup*, ["Octonion,purevectorpart", "purevectorpart"]],
[*Octonion,realpart*, *Octonion,setup*, ["Octonion,realpart", "realpart"]],
[*Octonion,setup*, *Clifford,intro*, ["Octonion,setup", "setup"]]]

```
>
```

Step 3: Inserting all help pages into HDB and the browser using the last modified list:

```
> insert_helppages(Lauto,MapleVersion);
Trying to read file C:\Maple11/P11/Octonion11/Help/associator_M11.mws...
Success... inserting topic Octonion,associator from the file:
```

C:\Maple11/P11/Octonion11/Help/associator_M11.mws

```
Trying to read file C:\Maple11/P11/Octonion11/Help/commutator_M11.mws...
Success... inserting topic Octonion,commutator from the file:
```

C:\Maple11/P11/Octonion11/Help/commutator_M11.mws

```
Trying to read file C:\Maple11/P11/Octonion11/Help/def_omultable_M11.mws...
Success... inserting topic Octonion,def_omultable from the file:
```

C:\Maple11/P11/Octonion11/Help/def_omultable_M11.mws

```
Trying to read file C:\Maple11/P11/Octonion11/Help/Fano_triples_M11.mws...
Success... inserting topic Octonion,type,Fano_triples from the file:
```

C:\Maple11/P11/Octonion11/Help/Fano_triples_M11.mws

```
Trying to read file C:\Maple11/P11/Octonion11/Help/octonion_M11.mws...
Success... inserting topic Octonion,type,octonion from the file:
```

C:\Maple11/P11/Octonion11/Help/octonion_M11.mws

```
Trying to read file C:\Maple11/P11/Octonion11/Help/oinv_M11.mws...
Success... inserting topic Octonion,oinv from the file:
```

C:\Maple11/P11/Octonion11/Help/oinv_M11.mws

```
Trying to read file C:\Maple11/P11/Octonion11/Help/omultable_M11.mws...
Success... inserting topic Octonion,omultable from the file:
```

C:\Maple11/P11/Octonion11/Help/omultable_M11.mws

```
Trying to read file C:\Maple11/P11/Octonion11/Help/omul_M11.mws...
Success... inserting topic Octonion,omul from the file:
```

C:\Maple11/P11/Octonion11/Help/omul_M11.mws

```
Trying to read file C:\Maple11/P11/Octonion11/Help/onorm_M11.mws...
Success... inserting topic Octonion,onorm from the file:
```

C:\Maple11/P11/Octonion11/Help/onorm_M11.mws

```
Trying to read file C:\Maple11/P11/Octonion11/Help/oversion_M11.mws...
Success... inserting topic Octonion,oversion from the file:
```

C:\Maple11/P11/Octonion11/Help/oversion_M11.mws

```
Trying to read file C:\Maple11/P11/Octonion11/Help/o_conjug_M11.mws...
Success... inserting topic Octonion,o_conjug from the file:
```

C:\Maple11/P11/Octonion11/Help/o_conjug_M11.mws

```
Trying to read file C:\Maple11/P11/Octonion11/Help/Phi_M11.mws...
Success... inserting topic Octonion,Phi from the file:
```

C:\Maple11/P11/Octonion11/Help/Phi_M11.mws

```
Trying to read file C:\Maple11/P11/Octonion11/Help/purevectorpart_M11.mws...
Success... inserting topic Octonion,purevectorpart from the file:
```

C:\Maple11/P11/Octonion11/Help/purevectorpart_M11.mws

```
Trying to read file C:\Maple11/P11/Octonion11/Help/realpart_M11.mws...
Success... inserting topic Octonion,realpart from the file:
```

C:\Maple11/P11/Octonion11/Help/realpart_M11.mws

```
Trying to read file C:\Maple11/P11/Octonion11/Help/setup_M11.mws...
Success... inserting topic Octonion,setup from the file:
```

C:\Maple11/P11/Octonion11/Help/setup_M11.mws

```
***********************
Finished inserting 15 file topics into the HDB and Browser
***********************
```

```
> ##############################################
  ##list of types defined in the module
  ##############################################
  typesLIST:=["antisymmatrix","clibasmon","climatrix","climon","clip
  olynom",

  "cliprod","cliscalar","diagmatrix","dfmatrix","evenelement",

  "fieldelement","gencomplex","genquatbasis","genquaternion",

  "idempotent","nilpotent","oddelement","primitiveidemp","purequatba
  sis",
              "quaternion","symmatrix","tensorprod"];
  ##############################################
  ##list of types converts defined in the module
  ##############################################
  convertsLIST:=["mlist","str_to_int"];
```

typesLIST := ["antisymmatrix", "clibasmon", "climatrix", "climon", "clipolynom", "cliprod", "cliscalar", "diagmatrix", "dfmatrix", "evenelement", "fieldelement", "gencomplex", "genquatbasis", "genquaternion", "idempotent", "nilpotent", "oddelement", "primitiveidemp", "purequatbasis", "quaternion", "symmatrix", "tensorprod"]

convertsLIST := ["mlist", "str_to_int"]

```
>
```

Inserting help pages for SchurFkt:

```
> HDB_LIB_PATH:=convert(libname[1],name);
  HELP_FILE_PATH:=`C:\\Maple11/P11/SchurFkt11/Help/`;
  BROWSER_PATH:=`Mathematics/Algebra/`;
  ModuleName   :=`SchurFkt`;
```

```
Parent      :=`SchurFkt,Overview`;
GrandParent :=`Algebra`;
MapleVersion:="M11";
##############################################
##list of types defined in the module
##############################################
typesLIST:=["sfktmonom","sfktterm","sfktpolynom",
            "pfktmonom","pfktterm","pfktpolynom",
            "mfktmonom","mfktterm","mfktpolynom",
            "hfktmonom","hfktterm","hfktpolynom",
            "efktmonom","efktterm","efktpolynom",
            "ffktmonom","ffktterm","ffktpolynom"
            ];
##############################################
##list of types converts defined in the module
##############################################
convertsLIST:=[];
```

HDB_LIB_PATH := C:\Maple11/Cliffordlib

HELP_FILE_PATH := C:\Maple11/P11/SchurFkt11/Help/

BROWSER_PATH := Mathematics/Algebra/

ModuleName := SchurFkt

Parent := SchurFkt,Overview

GrandParent := Algebra

MapleVersion := "M11"

typesLIST := ["sfktmonom", "sfktterm", "sfktpolynom", "pfktmonom", "pfktterm", "pfktpolynom", "mfktmonom", "mfktterm", "mfktpolynom", "hfktmonom", "hfktterm", "hfktpolynom", "efktmonom", "efktterm", "efktpolynom", "ffktmonom", "ffktterm", "ffktpolynom"]

convertsLIST := [ ]

```
> L:=with(SchurFkt);
```

L := [*AlexComp*, *CharHook*, *CompNM*, *FLAT*, *Frob2part*, *GesselThetaP*, *GesselThetaS*, *KostkaPC*, *KostkaTable*, *LaplaceM*, *LaplaceM_mon*, *LaplaceTable*, *MLIN*, *MurNak*, *MurNak2*, *PartNM*, *Scalar*, *ScalarHM*, *ScalarMH*, *ScalarP*, *antipE*, *antipH*, *antipM*, *antipMC*, *antipP*, *antipS*, *branch*, *cinner*, *cinnerP*, *cmp2part*, *cmp2prtMult*, *concatM*, *conjpart*, *counitInnerP*, *counitInnerS*, *couter*, *couterE*, *couterH*, *couterM*, *couterON*, *couterP*, *cplethP*, *cplethS*, *dimSN*, *e_to_h*, *e_to_s*, *evalJacobiTrudiMatrix*, *getSfktSeries*, *grAlexComp*, *h_to_m*, *h_to_s*, *inner*, *innerH*, *innerP*, *isLattice*, *m_to_p*, *maxlengthSymFkt*, *mset2part*, *outer*, *outerE*, *outerH*, *outerM*, *outerON*, *outerP*, *outerS*, *p_to_m*, *p_to_s*, *part2Frob*, *part2mset*, *plethP*, *plethS*, *plethSnm*, *s_to_h*, *s_to_hJT*, *s_to_hmat*, *s_to_p*, *s_to_x*, *skew*, *sq_coeff*, *truncLEN*, *truncWT*, *x_to_s*, *zee*]

```
> #for m in L do
  #item:=convert(cat("Mathematics/Algebra/SchurFkt/",convert(m,strin
  g)),symbol);
  #INTERFACE_HELP('delete',browser=item,helpfile=HDB_LIB_PATH);
  #end do;
> #m:="type";
> #item:=convert(cat("Mathematics/Algebra/SchurFkt/",convert(m,strin
  g)),symbol);
  #INTERFACE_HELP('delete',browser=item,helpfile=HDB_LIB_PATH);
> #?SchurFkt
>
> ###From Bertfried
  #### types
  #
  #  type/sfktmonom`, `type/sfktterm`, `type/sfktpolynom`,
  #
  #
  [`SchurFkt,type,sfktpolynom`,Parent,["SchurFkt,type,sfktmonom","Sc
  hurFkt,type,sfktterm","SchurFkt,type,sfktpolynom"]],
  #
  [`SchurFkt,type`,Parent,["SchurFkt,type,pfktmonom","SchurFkt,type,
  pfktterm","SchurFkt,type,pfktpolynom"]],
  #
  [`SchurFkt,type`,Parent,["SchurFkt,type,mfktmonom","SchurFkt,type,
  mfktterm","SchurFkt,type,mfktpolynom"]],
  #
  [`SchurFkt,type`,Parent,["SchurFkt,type,hfktmonom","SchurFkt,type,
  hfktterm","SchurFkt,type,hfktpolynom"]],
  #
  [`SchurFkt,type`,Parent,["SchurFkt,type,efktmonom","SchurFkt,type,
  efktterm","SchurFkt,type,efktpolynom"]],
  #
  [`SchurFkt,type`,Parent,["SchurFkt,type,ffktmonom","SchurFkt,type,
  ffktterm","SchurFkt,type,ffktpolynom"]]
>
```

Step 1: Generating an automatic list from help page files:

```
> Lauto:=makeLIST(ModuleName,Parent,GrandParent,HELP_FILE_PATH,Maple
  Version);
```

Lauto := [[*SchurFkt,AlexComp*, *SchurFkt,Overview*, ["SchurFkt,AlexComp", "AlexComp"]],
[*SchurFkt,antipS*, *SchurFkt,Overview*, ["SchurFkt,antipS", "antipS"]],
[*SchurFkt,branch*, *SchurFkt,Overview*, ["SchurFkt,branch", "branch"]],

[*SchurFkt,CharHook*, *SchurFkt,Overview*, ["SchurFkt,CharHook", "CharHook"]],
[*SchurFkt,cinner*, *SchurFkt,Overview*, ["SchurFkt,cinner", "cinner"]],
[*SchurFkt,cmp2part*, *SchurFkt,Overview*, ["SchurFkt,cmp2part", "cmp2part"]],
[*SchurFkt,cmp2prtMult*, *SchurFkt,Overview*, ["SchurFkt,cmp2prtMult", "cmp2prtMult"]],
[*SchurFkt,CompNM*, *SchurFkt,Overview*, ["SchurFkt,CompNM", "CompNM"]],
[*SchurFkt,concatM*, *SchurFkt,Overview*, ["SchurFkt,concatM", "concatM"]],
[*SchurFkt,conjpart*, *SchurFkt,Overview*, ["SchurFkt,conjpart", "conjpart"]],
[*SchurFkt,couterE*, *SchurFkt,Overview*, ["SchurFkt,couterE", "couterE"]],
[*SchurFkt,couterH*, *SchurFkt,Overview*, ["SchurFkt,couterH", "couterH"]],
[*SchurFkt,couterM*, *SchurFkt,Overview*, ["SchurFkt,couterM", "couterM"]],
[*SchurFkt,couterON*, *SchurFkt,Overview*, ["SchurFkt,couterON", "couterON"]],
[*SchurFkt,couterP*, *SchurFkt,Overview*, ["SchurFkt,couterP", "couterP"]],
[*SchurFkt,couter*, *SchurFkt,Overview*, ["SchurFkt,couter", "couter"]],
[*SchurFkt,cplethP*, *SchurFkt,Overview*, ["SchurFkt,cplethP", "cplethP"]],
[*SchurFkt,cplethS*, *SchurFkt,Overview*, ["SchurFkt,cplethS", "cplethS"]],
[*SchurFkt,dimSN*, *SchurFkt,Overview*, ["SchurFkt,dimSN", "dimSN"]],
[*SchurFkt,Dummy*, *SchurFkt,Overview*, ["SchurFkt,Dummy", "Dummy"]],
[*SchurFkt,FLAT*, *SchurFkt,Overview*, ["SchurFkt,FLAT", "FLAT"]],
[*SchurFkt,Frob2part*, *SchurFkt,Overview*, ["SchurFkt,Frob2part", "Frob2part"]],
[*SchurFkt,GesselTheta*, *SchurFkt,Overview*, ["SchurFkt,GesselTheta", "GesselTheta"]],
[*SchurFkt,getSfktSeries*, *SchurFkt,Overview*, ["SchurFkt,getSfktSeries", "getSfktSeries"]],
[*SchurFkt,grAlexComp*, *SchurFkt,Overview*, ["SchurFkt,grAlexComp", "grAlexComp"]],
[*SchurFkt,h_to_s*, *SchurFkt,Overview*, ["SchurFkt,h_to_s", "h_to_s"]],
[*SchurFkt,innerP*, *SchurFkt,Overview*, ["SchurFkt,innerP", "innerP"]],
[*SchurFkt,inner*, *SchurFkt,Overview*, ["SchurFkt,inner", "inner"]],
[*SchurFkt,isLattice*, *SchurFkt,Overview*, ["SchurFkt,isLattice", "isLattice"]],
[*SchurFkt,KostkaPC*, *SchurFkt,Overview*, ["SchurFkt,KostkaPC", "KostkaPC"]],
[*SchurFkt,KostkaTable*, *SchurFkt,Overview*, ["SchurFkt,KostkaTable", "KostkaTable"]],
[*SchurFkt,LaplaceM*, *SchurFkt,Overview*, ["SchurFkt,LaplaceM", "LaplaceM"]],
[*SchurFkt,LaplaceM_mon*, *SchurFkt,Overview*, ["SchurFkt,LaplaceM_mon", "LaplaceM_mon"]]
, [*SchurFkt,LaplaceTable*, *SchurFkt,Overview*, ["SchurFkt,LaplaceTable", "LaplaceTable"]],
[*SchurFkt,MLIN*, *SchurFkt,Overview*, ["SchurFkt,MLIN", "MLIN"]],
[*SchurFkt,mset2part*, *SchurFkt,Overview*, ["SchurFkt,mset2part", "mset2part"]],
[*SchurFkt,MurNak2*, *SchurFkt,Overview*, ["SchurFkt,MurNak2", "MurNak2"]],
[*SchurFkt,MurNak*, *SchurFkt,Overview*, ["SchurFkt,MurNak", "MurNak"]],
[*SchurFkt,m_to_p*, *SchurFkt,Overview*, ["SchurFkt,m_to_p", "m_to_p"]],

[*SchurFkt,outerE*, *SchurFkt,Overview*, ["SchurFkt,outerE", "outerE"]],
[*SchurFkt,outerH*, *SchurFkt,Overview*, ["SchurFkt,outerH", "outerH"]],
[*SchurFkt,outerM*, *SchurFkt,Overview*, ["SchurFkt,outerM", "outerM"]],
[*SchurFkt,outerON*, *SchurFkt,Overview*, ["SchurFkt,outerON", "outerON"]],
[*SchurFkt,outerP*, *SchurFkt,Overview*, ["SchurFkt,outerP", "outerP"]],
[*SchurFkt,outerS*, *SchurFkt,Overview*, ["SchurFkt,outerS", "outerS"]],
[*SchurFkt,outer*, *SchurFkt,Overview*, ["SchurFkt,outer", "outer"]],
[*SchurFkt,Overview*, *Algebra*, ["SchurFkt,Overview", "Overview"]],
[*SchurFkt,part2Frob*, *SchurFkt,Overview*, ["SchurFkt,part2Frob", "part2Frob"]],
[*SchurFkt,part2mset*, *SchurFkt,Overview*, ["SchurFkt,part2mset", "part2mset"]],
[*SchurFkt,PartNM*, *SchurFkt,Overview*, ["SchurFkt,PartNM", "PartNM"]],
[*SchurFkt,plethP*, *SchurFkt,Overview*, ["SchurFkt,plethP", "plethP"]],
[*SchurFkt,plethSnm*, *SchurFkt,Overview*, ["SchurFkt,plethSnm", "plethSnm"]],
[*SchurFkt,plethS*, *SchurFkt,Overview*, ["SchurFkt,plethS", "plethS"]],
[*SchurFkt,p_to_m*, *SchurFkt,Overview*, ["SchurFkt,p_to_m", "p_to_m"]],
[*SchurFkt,p_to_s*, *SchurFkt,Overview*, ["SchurFkt,p_to_s", "p_to_s"]],
[*SchurFkt,ScalarP*, *SchurFkt,Overview*, ["SchurFkt,ScalarP", "ScalarP"]],
[*SchurFkt,Scalar*, *SchurFkt,Overview*, ["SchurFkt,Scalar", "Scalar"]], [
SchurFkt,type,sfktpolynom, *SchurFkt,Overview*,
["SchurFkt,type,sfktpolynom", "type,sfktpolynom"]],
[*SchurFkt,skew*, *SchurFkt,Overview*, ["SchurFkt,skew", "skew"]],
[*SchurFkt,sq_coeff*, *SchurFkt,Overview*, ["SchurFkt,sq_coeff", "sq_coeff"]],
[*SchurFkt,s_to_h*, *SchurFkt,Overview*, ["SchurFkt,s_to_h", "s_to_h"]],
[*SchurFkt,s_to_p*, *SchurFkt,Overview*, ["SchurFkt,s_to_p", "s_to_p"]],
[*SchurFkt,s_to_x*, *SchurFkt,Overview*, ["SchurFkt,s_to_x", "s_to_x"]],
[*SchurFkt,truncWT*, *SchurFkt,Overview*, ["SchurFkt,truncWT", "truncWT"]],
[*SchurFkt,x_to_s*, *SchurFkt,Overview*, ["SchurFkt,x_to_s", "x_to_s"]],
[*SchurFkt,zee*, *SchurFkt,Overview*, ["SchurFkt,zee", "zee"]]]

```
>
```

Step 2: Modifying, if needed, certain entries in Lauto list:

```
>
> ###From Bertfried
  ###Bertfried: Please do not insert Grandparent or parent into this
  list:
> L:=[[`SchurFkt,Overview`,[

  "SchurFkt","Schurfkt","schurfkt","help","SchurFkt,help","schurfkt,
  help",
```

```
"Schurfkt,Overview","SchurFkt,overview","schurfkt,overview",
         "Symmetric functions","Schur functions"]],
    [`SchurFkt,AlexComp`,["Anti Lexicographical Ordering"]],
    [`SchurFkt,CharHook`,["hook","hook Schur function"]],
    [`SchurFkt,CompNM`,["generate compositions"]],
    [`SchurFkt,FLAT`,["flattening","associativity"]],
    [`SchurFkt,Frob2part`,["partition","Frobenius partition"]],
    [`SchurFkt,GesselTheta`,["GesselThetaS","GesselThetaP"]],
    [`SchurFkt,KostkaPC`,["Kostka
matrix","partition","composition"]],
    [`SchurFkt,KostkaTable`,["Kostka
matrix","partition","composition"]],
    [`SchurFkt,LaplaceM`,["Laplace pairing","monomial symmetric
functions","cliffordization"]],
 #   [`SchurFkt,LaplaceM_mon`,["Laplace pairing on
monomials","monomial symmetric functions"]],
    [`SchurFkt,LaplaceTable`,["Laplace
matrix","cliffordization"]],
    [`SchurFkt,MLIN`,["multi linear","tensor product"]],
    [`SchurFkt,MurNak`,["Murnaghan
Nakayama","rule","character"]],
    [`SchurFkt,MurNak2`,["Murnaghan
Nakayama","rule","character"]],
    [`SchurFkt,PartNM`,["partition","generation of partitions"]],
    [`SchurFkt,Scalar`,["Redfield","Hall","Schur","Schur Hall
scalar product","Schur functions","symmetric functions"]],
#    [`SchurFkt,ScalarHM`,["Redfield","Hall","Schur","Schur Hall
scalar product","Schur functions","complete symmetric
functions","monomial symmetric functions"]],
#    [`SchurFkt,ScalarMH`,["Redfield","Hall","Schur","Schur Hall
scalar product","Schur functions","complete symmetric
functions","monomial symmetric functions"]],
    [`SchurFkt,ScalarP`,["Redfield","Hall","Schur","Schur Hall
scalar product","power sum symmetric functions"]],
    [`SchurFkt,antipS`,["antipode","outer Hopf algebra"]],
    [`SchurFkt,branch`,["group
branching","induction","subduction","reduction"]],
    [`SchurFkt,cinner`,["inner coproduct"]],
    [`SchurFkt,cmp2part`,["composition","composition projected to
partition"]],
    [`SchurFkt,cmp2prtMult`,["composition","composition as
multiset"]],
```

```
    [`SchurFkt,concatM`,["monomial concatenation
product","concatenation product","product"]],
    [`SchurFkt,conjpart`,["conjugate","conjugate
partition","partition"]],
    [`SchurFkt,couter`,["outer coproduct","Schur outer
coproduct"]],
    [`SchurFkt,couterE`,["outer coproduct","Schur outer
coproduct","elementary symmetric functions"]],
    [`SchurFkt,couterH`,["outer coproduct","Schur outer
coproduct","complete symmetric functions"]],
    [`SchurFkt,couterM`,["ouert coproduct","monomial
coproduct"]],
    [`SchurFkt,couterON`,["outer coproduct","Schur outer
coproduct","orthogonal Schur functions"]],
    [`SchurFkt,couterP`,["ouert coproduct","power sum
coproduct"]],
    [`SchurFkt,cplethS`,["plethysm coproduct Schur functions"]],
    [`SchurFkt,cplethP`,["plethysm coproduct power sum basis"]],
    [`SchurFkt,dimSN`,["SchurFkt,Overview","sfkt polynom"]],
    [`SchurFkt,getSfktSeries`,["S-function","Schur function
series"]],
    [`SchurFkt,grAlexComp`,["graded anti lexicographic
ordering"]],
    [`SchurFkt,h_to_s`,["homogeneous symmetric functions","Schur
functions"]],
    [`SchurFkt,inner`,["inner product","Schur function inner
product"]],
    [`SchurFkt,innerP`,["inner product","power sum inner
product"]],
    [`SchurFkt,isLattice`,["lattice permutation test"]],
    [`SchurFkt,m_to_p`,["monomial symmetric functions","power sum
symmetric functions"]],
    [`SchurFkt,mset2part`,["multiset partition","partition"]],
    [`SchurFkt,outer`,["outer product","Schur
functions","symmetric functions"]],
    [`SchurFkt,outerE`,["outer product","elementary symmetric
functions"]],
    [`SchurFkt,outerH`,["outer product","complete symmetric
functions"]],
    [`SchurFkt,outerM`,["outer product","monomial symmetric
functions"]],
    [`SchurFkt,outerON`,["outer product","orthogonal Schur
functions"]],
```

```
    [`SchurFkt,outerP`,["outer product","power sum symmetric
functions"]],
    [`SchurFkt,outerS`,["outer product","Schur
functions","symmetric functions"]],
    [`SchurFkt,p_to_m`,["power sum symmetric function","monomial
symmetric function"]],
    [`SchurFkt,p_to_s`,["power sum symmetric function","Schur
function","symmetric function"]],
    [`SchurFkt,part2Frob`,["partition","Frobenius notation"]],
    [`SchurFkt,part2mset`,["partition","partition in multiset
notation"]],
    [`SchurFkt,plethS`,["plethysm","Schur function plethysm"]],
    [`SchurFkt,plethP`,["plethysm","power sum plethysm"]],
    [`SchurFkt,s_to_h`,["Schur functions","symmetric
functions","complete symmetric functions"]],
    [`SchurFkt,s_to_p`,["Schur functions","symmetric
functions","power sum symmetric functions"]],
    [`SchurFkt,s_to_x`,["Schur functions","symmetric
functions","indeterminants","alphabet"]],
    [`SchurFkt,skew`,["skew","Foulkes derivative","Littlewood
Richardson rule"]],
    [`SchurFkt,sq_coeff`,["coeffcients","dimension"]],
    [`SchurFkt,truncWT`,["weight","truncate","Schur function
series"]],
    [`SchurFkt,x_to_s`,["alphabet","Schur functions","symmetric
functions","indeterminants"]],
    [`SchurFkt,zee`,["permutation","normalization","zee","z"]]
]:
> 
> 
> modsLIST:=[op(L)];
```

modsLIST := [[*SchurFkt,Overview*, ["SchurFkt", "Schurfkt", "schurfkt", "help", "SchurFkt,help", "schurfkt,help", "Schurfkt,Overview", "SchurFkt,overview", "schurfkt,overview", "Symmetric functions", "Schur functions"]],
[*SchurFkt,AlexComp*, ["Anti Lexicographical Ordering"]],
[*SchurFkt,CharHook*, ["hook", "hook Schur function"]],
[*SchurFkt,CompNM*, ["generate compositions"]],
[*SchurFkt,FLAT*, ["flattening", "associativity"]],
[*SchurFkt,Frob2part*, ["partition", "Frobenius partition"]],
[*SchurFkt,GesselTheta*, ["GesselThetaS", "GesselThetaP"]],
[*SchurFkt,KostkaPC*, ["Kostka matrix", "partition", "composition"]],

[*SchurFkt,KostkaTable*, ["Kostka matrix", "partition", "composition"]],
[*SchurFkt,LaplaceM*, ["Laplace pairing", "monomial symmetric functions", "cliffordization"]],
[*SchurFkt,LaplaceTable*, ["Laplace matrix", "cliffordization"]],
[*SchurFkt,MLIN*, ["multi linear", "tensor product"]],
[*SchurFkt,MurNak*, ["Murnaghan Nakayama", "rule", "character"]],
[*SchurFkt,MurNak2*, ["Murnaghan Nakayama", "rule", "character"]],
[*SchurFkt,PartNM*, ["partition", "generation of partitions"]], [*SchurFkt,Scalar*, [
"Redfield", "Hall", "Schur", "Schur Hall scalar product", "Schur functions", "symmetric functions"
]], [*SchurFkt,ScalarP*,
["Redfield", "Hall", "Schur", "Schur Hall scalar product", "power sum symmetric functions"]],
[*SchurFkt,antipS*, ["antipode", "outer Hopf algebra"]],
[*SchurFkt,branch*, ["group branching", "induction", "subduction", "reduction"]],
[*SchurFkt,cinner*, ["inner coproduct"]],
[*SchurFkt,cmp2part*, ["composition", "composition projected to partition"]],
[*SchurFkt,cmp2prtMult*, ["composition", "composition as multiset"]],
[*SchurFkt,concatM*, ["monomial concatenation product", "concatenation product", "product"]],
[*SchurFkt,conjpart*, ["conjugate", "conjugate partition", "partition"]],
[*SchurFkt,couter*, ["outer coproduct", "Schur outer coproduct"]], [*SchurFkt,couterE*,
["outer coproduct", "Schur outer coproduct", "elementary symmetric functions"]], [
SchurFkt,couterH, ["outer coproduct", "Schur outer coproduct", "complete symmetric functions"]
], [*SchurFkt,couterM*, ["ouert coproduct", "monomial coproduct"]], [
SchurFkt,couterON, ["outer coproduct", "Schur outer coproduct", "orthogonal Schur functions"]],
[*SchurFkt,couterP*, ["ouert coproduct", "power sum coproduct"]],
[*SchurFkt,cplethS*, ["plethysm coproduct Schur functions"]],
[*SchurFkt,cplethP*, ["plethysm coproduct power sum basis"]],
[*SchurFkt,dimSN*, ["SchurFkt,Overview", "sfkt polynom"]],
[*SchurFkt,getSfktSeries*, ["S-function", "Schur function series"]],
[*SchurFkt,grAlexComp*, ["graded anti lexicographic ordering"]],
[*SchurFkt,h_to_s*, ["homogeneous symmetric functions", "Schur functions"]],
[*SchurFkt,inner*, ["inner product", "Schur function inner product"]],
[*SchurFkt,innerP*, ["inner product", "power sum inner product"]],
[*SchurFkt,isLattice*, ["lattice permutation test"]],
[*SchurFkt,m_to_p*, ["monomial symmetric functions", "power sum symmetric functions"]],
[*SchurFkt,mset2part*, ["multiset partition", "partition"]],
[*SchurFkt,outer*, ["outer product", "Schur functions", "symmetric functions"]],
[*SchurFkt,outerE*, ["outer product", "elementary symmetric functions"]],

[*SchurFkt,outerH*, ["outer product", "complete symmetric functions"]],
[*SchurFkt,outerM*, ["outer product", "monomial symmetric functions"]],
[*SchurFkt,outerON*, ["outer product", "orthogonal Schur functions"]],
[*SchurFkt,outerP*, ["outer product", "power sum symmetric functions"]],
[*SchurFkt,outerS*, ["outer product", "Schur functions", "symmetric functions"]],
[*SchurFkt,p_to_m*, ["power sum symmetric function", "monomial symmetric function"]],
[*SchurFkt,p_to_s*, ["power sum symmetric function", "Schur function", "symmetric function"]],
[*SchurFkt,part2Frob*, ["partition", "Frobenius notation"]],
[*SchurFkt,part2mset*, ["partition", "partition in multiset notation"]],
[*SchurFkt,plethS*, ["plethysm", "Schur function plethysm"]],
[*SchurFkt,plethP*, ["plethysm", "power sum plethysm"]],
[*SchurFkt,s_to_h*, ["Schur functions", "symmetric functions", "complete symmetric functions"]],
[*SchurFkt,s_to_p*, ["Schur functions", "symmetric functions", "power sum symmetric functions"]
], [*SchurFkt,s_to_x*, ["Schur functions", "symmetric functions", "indeterminants", "alphabet"]],
[*SchurFkt,skew*, ["skew", "Foulkes derivative", "Littlewood Richardson rule"]],
[*SchurFkt,sq_coeff*, ["coeffcients", "dimension"]],
[*SchurFkt,truncWT*, ["weight", "truncate", "Schur function series"]],
[*SchurFkt,x_to_s*, ["alphabet", "Schur functions", "symmetric functions", "indeterminants"]],
[*SchurFkt,zee*, ["permutation", "normalization", "zee", "z"]]]

```
> for mem in modsLIST do
      Lauto:=modifyLIST(Lauto,op(mem))
  end do:
  Lauto;
```

[[*SchurFkt,AlexComp*, *SchurFkt,Overview*, ["Anti Lexicographical Ordering"]],
[*SchurFkt,antipS*, *SchurFkt,Overview*, ["antipode", "outer Hopf algebra"]], [*SchurFkt,branch*, *SchurFkt,Overview*, ["group branching", "induction", "subduction", "reduction"]],
[*SchurFkt,CharHook*, *SchurFkt,Overview*, ["hook", "hook Schur function"]],
[*SchurFkt,cinner*, *SchurFkt,Overview*, ["inner coproduct"]],
[*SchurFkt,cmp2part*, *SchurFkt,Overview*, ["composition", "composition projected to partition"]],
[*SchurFkt,cmp2prtMult*, *SchurFkt,Overview*, ["composition", "composition as multiset"]],
[*SchurFkt,CompNM*, *SchurFkt,Overview*, ["generate compositions"]], [*SchurFkt,concatM*, *SchurFkt,Overview*, ["monomial concatenation product", "concatenation product", "product"]],
[*SchurFkt,conjpart*, *SchurFkt,Overview*, ["conjugate", "conjugate partition", "partition"]], [*SchurFkt,couterE*, *SchurFkt,Overview*,
["outer coproduct", "Schur outer coproduct", "elementary symmetric functions"]], [*SchurFkt,couterH*, *SchurFkt,Overview*,
["outer coproduct", "Schur outer coproduct", "complete symmetric functions"]],

[*SchurFkt,couterM*, *SchurFkt,Overview*, ["ouert coproduct", "monomial coproduct"]], [*SchurFkt,couterON*, *SchurFkt,Overview*,
["outer coproduct", "Schur outer coproduct", "orthogonal Schur functions"]],
[*SchurFkt,couterP*, *SchurFkt,Overview*, ["ouert coproduct", "power sum coproduct"]],
[*SchurFkt,couter*, *SchurFkt,Overview*, ["outer coproduct", "Schur outer coproduct"]],
[*SchurFkt,cplethP*, *SchurFkt,Overview*, ["plethysm coproduct power sum basis"]],
[*SchurFkt,cplethS*, *SchurFkt,Overview*, ["plethysm coproduct Schur functions"]],
[*SchurFkt,dimSN*, *SchurFkt,Overview*, ["SchurFkt,Overview", "sfkt polynom"]],
[*SchurFkt,Dummy*, *SchurFkt,Overview*, ["SchurFkt,Dummy", "Dummy"]],
[*SchurFkt,FLAT*, *SchurFkt,Overview*, ["flattening", "associativity"]],
[*SchurFkt,Frob2part*, *SchurFkt,Overview*, ["partition", "Frobenius partition"]],
[*SchurFkt,GesselTheta*, *SchurFkt,Overview*, ["GesselThetaS", "GesselThetaP"]],
[*SchurFkt,getSfktSeries*, *SchurFkt,Overview*, ["S-function", "Schur function series"]],
[*SchurFkt,grAlexComp*, *SchurFkt,Overview*, ["graded anti lexicographic ordering"]], [*SchurFkt,h_to_s*, *SchurFkt,Overview*, ["homogeneous symmetric functions", "Schur functions"]],
[*SchurFkt,innerP*, *SchurFkt,Overview*, ["inner product", "power sum inner product"]],
[*SchurFkt,inner*, *SchurFkt,Overview*, ["inner product", "Schur function inner product"]],
[*SchurFkt,isLattice*, *SchurFkt,Overview*, ["lattice permutation test"]],
[*SchurFkt,KostkaPC*, *SchurFkt,Overview*, ["Kostka matrix", "partition", "composition"]],
[*SchurFkt,KostkaTable*, *SchurFkt,Overview*, ["Kostka matrix", "partition", "composition"]], [*SchurFkt,LaplaceM*, *SchurFkt,Overview*,
["Laplace pairing", "monomial symmetric functions", "cliffordization"]],
[*SchurFkt,LaplaceM_mon*, *SchurFkt,Overview*, ["SchurFkt,LaplaceM_mon", "LaplaceM_mon"]]
, [*SchurFkt,LaplaceTable*, *SchurFkt,Overview*, ["Laplace matrix", "cliffordization"]],
[*SchurFkt,MLIN*, *SchurFkt,Overview*, ["multi linear", "tensor product"]],
[*SchurFkt,mset2part*, *SchurFkt,Overview*, ["multiset partition", "partition"]],
[*SchurFkt,MurNak2*, *SchurFkt,Overview*, ["Murnaghan Nakayama", "rule", "character"]],
[*SchurFkt,MurNak*, *SchurFkt,Overview*, ["Murnaghan Nakayama", "rule", "character"]], [*SchurFkt,m_to_p*, *SchurFkt,Overview*,
["monomial symmetric functions", "power sum symmetric functions"]],
[*SchurFkt,outerE*, *SchurFkt,Overview*, ["outer product", "elementary symmetric functions"]],
[*SchurFkt,outerH*, *SchurFkt,Overview*, ["outer product", "complete symmetric functions"]],
[*SchurFkt,outerM*, *SchurFkt,Overview*, ["outer product", "monomial symmetric functions"]],
[*SchurFkt,outerON*, *SchurFkt,Overview*, ["outer product", "orthogonal Schur functions"]],
[*SchurFkt,outerP*, *SchurFkt,Overview*, ["outer product", "power sum symmetric functions"]], [*SchurFkt,outerS*, *SchurFkt,Overview*, ["outer product", "Schur functions", "symmetric functions"]

], [
SchurFkt,outer, *SchurFkt,Overview*, ["outer product", "Schur functions", "symmetric functions"]]
, [*SchurFkt,Overview*, *Algebra*, ["SchurFkt", "Schurfkt", "schurfkt", "help", "SchurFkt,help",
"schurfkt,help", "Schurfkt,Overview", "SchurFkt,overview", "schurfkt,overview",
"Symmetric functions", "Schur functions"]],
[*SchurFkt,part2Frob*, *SchurFkt,Overview*, ["partition", "Frobenius notation"]],
[*SchurFkt,part2mset*, *SchurFkt,Overview*, ["partition", "partition in multiset notation"]],
[*SchurFkt,PartNM*, *SchurFkt,Overview*, ["partition", "generation of partitions"]],
[*SchurFkt,plethP*, *SchurFkt,Overview*, ["plethysm", "power sum plethysm"]],
[*SchurFkt,plethSnm*, *SchurFkt,Overview*, ["SchurFkt,plethSnm", "plethSnm"]],
[*SchurFkt,plethS*, *SchurFkt,Overview*, ["plethysm", "Schur function plethysm"]], [
SchurFkt,p_to_m, *SchurFkt,Overview*,
["power sum symmetric function", "monomial symmetric function"]], [*SchurFkt,p_to_s*,
SchurFkt,Overview, ["power sum symmetric function", "Schur function", "symmetric function"]],
[*SchurFkt,ScalarP*, *SchurFkt,Overview*,
["Redfield", "Hall", "Schur", "Schur Hall scalar product", "power sum symmetric functions"]], [
SchurFkt,Scalar, *SchurFkt,Overview*, [
"Redfield", "Hall", "Schur", "Schur Hall scalar product", "Schur functions", "symmetric functions"
]], [*SchurFkt,type,sfktpolynom*, *SchurFkt,Overview*,
["SchurFkt,type,sfktpolynom", "type,sfktpolynom"]], [
SchurFkt,skew, *SchurFkt,Overview*, ["skew", "Foulkes derivative", "Littlewood Richardson rule"]
], [*SchurFkt,sq_coeff*, *SchurFkt,Overview*, ["coeffcients", "dimension"]], [*SchurFkt,s_to_h*,
SchurFkt,Overview, ["Schur functions", "symmetric functions", "complete symmetric functions"]
], [*SchurFkt,s_to_p*, *SchurFkt,Overview*,
["Schur functions", "symmetric functions", "power sum symmetric functions"]], [
SchurFkt,s_to_x, *SchurFkt,Overview*,
["Schur functions", "symmetric functions", "indeterminants", "alphabet"]],
[*SchurFkt,truncWT*, *SchurFkt,Overview*, ["weight", "truncate", "Schur function series"]], [
SchurFkt,x_to_s, *SchurFkt,Overview*,
["alphabet", "Schur functions", "symmetric functions", "indeterminants"]],
[*SchurFkt,zee*, *SchurFkt,Overview*, ["permutation", "normalization", "zee", "z"]]]

```
>
```

Step 3: Inserting all help pages into HDB and the browser using the last modified list:

```
> insert_helppages(Lauto,MapleVersion);
Trying to read file C:\Maple11/P11/SchurFkt11/Help/AlexComp_M11.mws...
Success... inserting topic SchurFkt,AlexComp from the file:
                    C:\Maple11/P11/SchurFkt11/Help/AlexComp_M11.mws
Trying to read file C:\Maple11/P11/SchurFkt11/Help/antipS_M11.mws...
```

```
Success... inserting topic SchurFkt,antipS from the file:
```

C:\Maple11/P11/SchurFkt11/Help/antipS_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/branch_M11.mws...
Success... inserting topic SchurFkt,branch from the file:
```

C:\Maple11/P11/SchurFkt11/Help/branch_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/CharHook_M11.mws...
Success... inserting topic SchurFkt,CharHook from the file:
```

C:\Maple11/P11/SchurFkt11/Help/CharHook_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/cinner_M11.mws...
Success... inserting topic SchurFkt,cinner from the file:
```

C:\Maple11/P11/SchurFkt11/Help/cinner_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/cmp2part_M11.mws...
Success... inserting topic SchurFkt,cmp2part from the file:
```

C:\Maple11/P11/SchurFkt11/Help/cmp2part_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/cmp2prtMult_M11.mws...
Success... inserting topic SchurFkt,cmp2prtMult from the file:
```

C:\Maple11/P11/SchurFkt11/Help/cmp2prtMult_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/CompNM_M11.mws...
Success... inserting topic SchurFkt,CompNM from the file:
```

C:\Maple11/P11/SchurFkt11/Help/CompNM_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/concatM_M11.mws...
Success... inserting topic SchurFkt,concatM from the file:
```

C:\Maple11/P11/SchurFkt11/Help/concatM_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/conjpart_M11.mws...
Success... inserting topic SchurFkt,conjpart from the file:
```

C:\Maple11/P11/SchurFkt11/Help/conjpart_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/couterE_M11.mws...
Success... inserting topic SchurFkt,couterE from the file:
```

C:\Maple11/P11/SchurFkt11/Help/couterE_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/couterH_M11.mws...
Success... inserting topic SchurFkt,couterH from the file:
```

C:\Maple11/P11/SchurFkt11/Help/couterH_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/couterM_M11.mws...
Success... inserting topic SchurFkt,couterM from the file:
```

C:\Maple11/P11/SchurFkt11/Help/couterM_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/couterON_M11.mws...
Success... inserting topic SchurFkt,couterON from the file:
```

C:\Maple11/P11/SchurFkt11/Help/couterON_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/couterP_M11.mws...
Success... inserting topic SchurFkt,couterP from the file:
```

C:\Maple11/P11/SchurFkt11/Help/couterP_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/couter_M11.mws...
Success... inserting topic SchurFkt,couter from the file:
```

C:\Maple11/P11/SchurFkt11/Help/couter_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/cplethP_M11.mws...
Success... inserting topic SchurFkt,cplethP from the file:
```

C:\Maple11/P11/SchurFkt11/Help/cplethP_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/cplethS_M11.mws...
Success... inserting topic SchurFkt,cplethS from the file:
```

C:\Maple11/P11/SchurFkt11/Help/cplethS_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/dimSN_M11.mws...
Success... inserting topic SchurFkt,dimSN from the file:
```

C:\Maple11/P11/SchurFkt11/Help/dimSN_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/Dummy_M11.mws...
Success... inserting topic SchurFkt,Dummy from the file:
```

C:\Maple11/P11/SchurFkt11/Help/Dummy_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/FLAT_M11.mws...
Success... inserting topic SchurFkt,FLAT from the file:
```

C:\Maple11/P11/SchurFkt11/Help/FLAT_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/Frob2part_M11.mws...
Success... inserting topic SchurFkt,Frob2part from the file:
```

C:\Maple11/P11/SchurFkt11/Help/Frob2part_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/GesselTheta_M11.mws...
Success... inserting topic SchurFkt,GesselTheta from the file:
```

C:\Maple11/P11/SchurFkt11/Help/GesselTheta_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/getSfktSeries_M11.mws...
Success... inserting topic SchurFkt,getSfktSeries from the file:
```

C:\Maple11/P11/SchurFkt11/Help/getSfktSeries_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/grAlexComp_M11.mws...
Success... inserting topic SchurFkt,grAlexComp from the file:
```

C:\Maple11/P11/SchurFkt11/Help/grAlexComp_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/h_to_s_M11.mws...
Success... inserting topic SchurFkt,h_to_s from the file:
```

C:\Maple11/P11/SchurFkt11/Help/h_to_s_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/innerP_M11.mws...
Success... inserting topic SchurFkt,innerP from the file:
```

C:\Maple11/P11/SchurFkt11/Help/innerP_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/inner_M11.mws...
Success... inserting topic SchurFkt,inner from the file:
```

C:\Maple11/P11/SchurFkt11/Help/inner_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/isLattice_M11.mws...
Success... inserting topic SchurFkt,isLattice from the file:
```

C:\Maple11/P11/SchurFkt11/Help/isLattice_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/KostkaPC_M11.mws...
Success... inserting topic SchurFkt,KostkaPC from the file:
```

C:\Maple11/P11/SchurFkt11/Help/KostkaPC_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/KostkaTable_M11.mws...
Success... inserting topic SchurFkt,KostkaTable from the file:
```

C:\Maple11/P11/SchurFkt11/Help/KostkaTable_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/LaplaceM_M11.mws...
Success... inserting topic SchurFkt,LaplaceM from the file:
```

C:\Maple11/P11/SchurFkt11/Help/LaplaceM_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/LaplaceM_mon_M11.mws...
Success... inserting topic SchurFkt,LaplaceM_mon from the file:
```

C:\Maple11/P11/SchurFkt11/Help/LaplaceM_mon_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/LaplaceTable_M11.mws...
Success... inserting topic SchurFkt,LaplaceTable from the file:
```

C:\Maple11/P11/SchurFkt11/Help/LaplaceTable_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/MLIN_M11.mws...
Success... inserting topic SchurFkt,MLIN from the file:
```

C:\Maple11/P11/SchurFkt11/Help/MLIN_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/mset2part_M11.mws...
Success... inserting topic SchurFkt,mset2part from the file:
```

C:\Maple11/P11/SchurFkt11/Help/mset2part_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/MurNak2_M11.mws...
Success... inserting topic SchurFkt,MurNak2 from the file:
```

C:\Maple11/P11/SchurFkt11/Help/MurNak2_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/MurNak_M11.mws...
Success... inserting topic SchurFkt,MurNak from the file:
```

C:\Maple11/P11/SchurFkt11/Help/MurNak_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/m_to_p_M11.mws...
Success... inserting topic SchurFkt,m_to_p from the file:
```

C:\Maple11/P11/SchurFkt11/Help/m_to_p_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/outerE_M11.mws...
Success... inserting topic SchurFkt,outerE from the file:
```

C:\Maple11/P11/SchurFkt11/Help/outerE_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/outerH_M11.mws...
Success... inserting topic SchurFkt,outerH from the file:
```

C:\Maple11/P11/SchurFkt11/Help/outerH_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/outerM_M11.mws...
Success... inserting topic SchurFkt,outerM from the file:
```

C:\Maple11/P11/SchurFkt11/Help/outerM_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/outerON_M11.mws...
Success... inserting topic SchurFkt,outerON from the file:
```

C:\Maple11/P11/SchurFkt11/Help/outerON_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/outerP_M11.mws...
Success... inserting topic SchurFkt,outerP from the file:
```

C:\Maple11/P11/SchurFkt11/Help/outerP_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/outerS_M11.mws...
Success... inserting topic SchurFkt,outerS from the file:
```

C:\Maple11/P11/SchurFkt11/Help/outerS_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/outer_M11.mws...
Success... inserting topic SchurFkt,outer from the file:
```

C:\Maple11/P11/SchurFkt11/Help/outer_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/Overview_M11.mws...
Success... inserting topic SchurFkt,Overview from the file:
```

C:\Maple11/P11/SchurFkt11/Help/Overview_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/part2Frob_M11.mws...
Success... inserting topic SchurFkt,part2Frob from the file:
```

C:\Maple11/P11/SchurFkt11/Help/part2Frob_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/part2mset_M11.mws...
Success... inserting topic SchurFkt,part2mset from the file:
```

C:\Maple11/P11/SchurFkt11/Help/part2mset_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/PartNM_M11.mws...
Success... inserting topic SchurFkt,PartNM from the file:
```

C:\Maple11/P11/SchurFkt11/Help/PartNM_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/plethP_M11.mws...
```

```
Success... inserting topic SchurFkt,plethP from the file:
```

C:\Maple11/P11/SchurFkt11/Help/plethP_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/plethSnm_M11.mws...
Success... inserting topic SchurFkt,plethSnm from the file:
```

C:\Maple11/P11/SchurFkt11/Help/plethSnm_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/plethS_M11.mws...
Success... inserting topic SchurFkt,plethS from the file:
```

C:\Maple11/P11/SchurFkt11/Help/plethS_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/p_to_m_M11.mws...
Success... inserting topic SchurFkt,p_to_m from the file:
```

C:\Maple11/P11/SchurFkt11/Help/p_to_m_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/p_to_s_M11.mws...
Success... inserting topic SchurFkt,p_to_s from the file:
```

C:\Maple11/P11/SchurFkt11/Help/p_to_s_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/ScalarP_M11.mws...
Success... inserting topic SchurFkt,ScalarP from the file:
```

C:\Maple11/P11/SchurFkt11/Help/ScalarP_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/Scalar_M11.mws...
Success... inserting topic SchurFkt,Scalar from the file:
```

C:\Maple11/P11/SchurFkt11/Help/Scalar_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/sfktpolynom_M11.mws...
Success... inserting topic SchurFkt,type,sfktpolynom from the file:
```

C:\Maple11/P11/SchurFkt11/Help/sfktpolynom_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/skew_M11.mws...
Success... inserting topic SchurFkt,skew from the file:
```

C:\Maple11/P11/SchurFkt11/Help/skew_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/sq_coeff_M11.mws...
Success... inserting topic SchurFkt,sq_coeff from the file:
```

C:\Maple11/P11/SchurFkt11/Help/sq_coeff_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/s_to_h_M11.mws...
Success... inserting topic SchurFkt,s_to_h from the file:
```

C:\Maple11/P11/SchurFkt11/Help/s_to_h_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/s_to_p_M11.mws...
Success... inserting topic SchurFkt,s_to_p from the file:
```

C:\Maple11/P11/SchurFkt11/Help/s_to_p_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/s_to_x_M11.mws...
Success... inserting topic SchurFkt,s_to_x from the file:
```

C:\Maple11/P11/SchurFkt11/Help/s_to_x_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/truncWT_M11.mws...
Success... inserting topic SchurFkt,truncWT from the file:
```

C:\Maple11/P11/SchurFkt11/Help/truncWT_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/x_to_s_M11.mws...
Success... inserting topic SchurFkt,x_to_s from the file:
```

C:\Maple11/P11/SchurFkt11/Help/x_to_s_M11.mws

```
Trying to read file C:\Maple11/P11/SchurFkt11/Help/zee_M11.mws...
Success... inserting topic SchurFkt,zee from the file:
```

C:\Maple11/P11/SchurFkt11/Help/zee_M11.mws

```
***********************
Finished inserting 66 file topics into the HDB and Browser
```

```
**********************
> #?SchurFkt
>
>
```

Inserting help pages for SINGULARPLURALlink:

```
> HDB_LIB_PATH:=convert(libname[1],name);
  HELP_FILE_PATH:=`C:\\Maple11/P11/SINGULARPLURALlink11/Help/`;
  BROWSER_PATH:=`Mathematics/Algebra/`;
  ModuleName  :=`SINGULARPLURALlink`;
  Parent      :=`SINGULARPLURALlink,Examples`;
  GrandParent :=`Algebra`;
  MapleVersion:="M11";
  ##############################################
  ##list of types defined in the module
  ##############################################
  typesLIST:=[];
  ##############################################
  ##list of types converts defined in the module
  ##############################################
  convertsLIST:=[];
```

$$HDB\_LIB\_PATH := C:\backslash Maple11/Cliffordlib$$
$$HELP\_FILE\_PATH := C:\backslash Maple11/P11/SINGULARPLURALlink11/Help/$$
$$BROWSER\_PATH := Mathematics/Algebra/$$
$$ModuleName := SINGULARPLURALlink$$
$$Parent := SINGULARPLURALlink,Examples$$
$$GrandParent := Algebra$$
$$MapleVersion := \text{"M11"}$$
$$typesLIST := [\ ]$$
$$convertsLIST := [\ ]$$

Step 1: Generating an automatic list from help page files:

```
> Lauto:=makeLIST(ModuleName,Parent,GrandParent,HELP_FILE_PATH,Maple
  Version);
```

$$Lauto := [[SINGULARPLURALlink,Examples, Algebra, [\text{"SINGULARPLURALlink,Examples"}, \text{"Examples"}]]]$$

```
>
```

Step 2: Modifying, if needed, certain entries in Lauto list:

```
> #modsLIST:=[
  #[`Octonion,omul`,["Octonion,omul","omul","&o","octonion","Octonio
```

```
n"]]
#]:
```

```
> #for mem in modsLIST do
  #    Lauto:=modifyLIST(Lauto,op(mem))
  #end do:
  Lauto;
```

[[*SINGULARPLURALlink,Examples*, *Algebra*,
["SINGULARPLURALlink,Examples", "Examples"]]]

```
>
```

Step 3: Inserting all help pages into HDB and the browser using the last modified list:

```
> insert_helppages(Lauto,MapleVersion);
Trying to read file C:\Maple11/P11/SINGULARPLURALlink11/Help/Examples_M11.mws...
Success... inserting topic SINGULARPLURALlink,Examples from the file:
```

C:\Maple11/P11/SINGULARPLURALlink11/Help/Examples_M11.mws

```
**********************
Finished inserting 1 file topics into the HDB and Browser
**********************
> #?SINGULARPLURALlink
>
```

Inserting help pages for SP:

```
> HDB_LIB_PATH:=convert(libname[1],name);
  HELP_FILE_PATH:=`C:\\Maple11/P11/SP11/Help/`;
  BROWSER_PATH:=`Mathematics/Algebra/`;
  ModuleName   :=`SP`;
  Parent       :=`SP,SPversion`;
  GrandParent  :=`Algebra`;
  MapleVersion:="M11";
  ##############################################
  ##list of types defined in the module
  ##############################################
  typesLIST:=[];
  ##############################################
  ##list of types converts defined in the module
  ##############################################
  convertsLIST:=[];
```

HDB_LIB_PATH := C:\Maple11/Cliffordlib

HELP_FILE_PATH := C:\Maple11/P11/SP11/Help/

BROWSER_PATH := Mathematics/Algebra/

```
Remember table of SymmetricGroup has been read and assigned
Remember table of AlternatingGroup has been read and assigned
Remember table of Reynolds has been read and assigned
Remember table of FiniteGroups has been read and assigned
```

```
Remember table of generateGinvariants has been read and assigned
```

$$ModuleName := SP$$

$$Parent := SP,SPversion$$

$$GrandParent := Algebra$$

$$MapleVersion := \text{"M11"}$$

$$typesLIST := [\ ]$$

$$convertsLIST := [\ ]$$

Step 1: Generating an automatic list from help page files:

```
> Lauto:=makeLIST(ModuleName,Parent,GrandParent,HELP_FILE_PATH,Maple
  Version);
```

Lauto := [[*SP,AlternatingGroup*, *SP,SPversion*, ["SP,AlternatingGroup", "AlternatingGroup"]],
[*SP,Dummy*, *SP,SPversion*, ["SP,Dummy", "Dummy"]],
[*SP,FiniteGroups*, *SP,SPversion*, ["SP,FiniteGroups", "FiniteGroups"]],
[*SP,generateGinvariants*, *SP,SPversion*, ["SP,generateGinvariants", "generateGinvariants"]],
[*SP,gpolynom*, *SP,SPversion*, ["SP,gpolynom", "gpolynom"]],
[*SP,hpolynom*, *SP,SPversion*, ["SP,hpolynom", "hpolynom"]],
[*SP,isContained*, *SP,SPversion*, ["SP,isContained", "isContained"]],
[*SP,isGinvariant*, *SP,SPversion*, ["SP,isGinvariant", "isGinvariant"]],
[*SP,isSymmetric*, *SP,SPversion*, ["SP,isSymmetric", "isSymmetric"]],
[*SP,MatrixAction*, *SP,SPversion*, ["SP,MatrixAction", "MatrixAction"]],
[*SP,powersum*, *SP,SPversion*, ["SP,powersum", "powersum"]],
[*SP,reduceGinvariants*, *SP,SPversion*, ["SP,reduceGinvariants", "reduceGinvariants"]],
[*SP,Reynolds*, *SP,SPversion*, ["SP,Reynolds", "Reynolds"]],
[*SP,Sigma*, *SP,SPversion*, ["SP,Sigma", "Sigma"]],
[*SP,sigma_to_powersum*, *SP,SPversion*, ["SP,sigma_to_powersum", "sigma_to_powersum"]],
[*SP,SPversion*, *Algebra*, ["SP,SPversion", "SPversion"]],
[*SP,SymmetricGroup*, *SP,SPversion*, ["SP,SymmetricGroup", "SymmetricGroup"]],
[*SP,SyzygyIdeal*, *SP,SPversion*, ["SP,SyzygyIdeal", "SyzygyIdeal"]]]

```
>
```

Step 2: Modifying, if needed, certain entries in Lauto list:

```
> #modsLIST:=[
  #[`Octonion,omul`,["Octonion,omul","omul","&o","octonion","Octonio
  n"]]
  #]:
> #for mem in modsLIST do
  #    Lauto:=modifyLIST(Lauto,op(mem))
  #end do:
```

```
> Lauto;
```

[[*SP,AlternatingGroup*, *SP,SPversion*, ["SP,AlternatingGroup", "AlternatingGroup"]],
[*SP,Dummy*, *SP,SPversion*, ["SP,Dummy", "Dummy"]],
[*SP,FiniteGroups*, *SP,SPversion*, ["SP,FiniteGroups", "FiniteGroups"]],
[*SP,generateGinvariants*, *SP,SPversion*, ["SP,generateGinvariants", "generateGinvariants"]],
[*SP,gpolynom*, *SP,SPversion*, ["SP,gpolynom", "gpolynom"]],
[*SP,hpolynom*, *SP,SPversion*, ["SP,hpolynom", "hpolynom"]],
[*SP,isContained*, *SP,SPversion*, ["SP,isContained", "isContained"]],
[*SP,isGinvariant*, *SP,SPversion*, ["SP,isGinvariant", "isGinvariant"]],
[*SP,isSymmetric*, *SP,SPversion*, ["SP,isSymmetric", "isSymmetric"]],
[*SP,MatrixAction*, *SP,SPversion*, ["SP,MatrixAction", "MatrixAction"]],
[*SP,powersum*, *SP,SPversion*, ["SP,powersum", "powersum"]],
[*SP,reduceGinvariants*, *SP,SPversion*, ["SP,reduceGinvariants", "reduceGinvariants"]],
[*SP,Reynolds*, *SP,SPversion*, ["SP,Reynolds", "Reynolds"]],
[*SP,Sigma*, *SP,SPversion*, ["SP,Sigma", "Sigma"]],
[*SP,sigma_to_powersum*, *SP,SPversion*, ["SP,sigma_to_powersum", "sigma_to_powersum"]],
[*SP,SPversion*, *Algebra*, ["SP,SPversion", "SPversion"]],
[*SP,SymmetricGroup*, *SP,SPversion*, ["SP,SymmetricGroup", "SymmetricGroup"]],
[*SP,SyzygyIdeal*, *SP,SPversion*, ["SP,SyzygyIdeal", "SyzygyIdeal"]]]

```
>
```

Step 3: Inserting all help pages into HDB and the browser using the last modified list:

```
> insert_helppages(Lauto,MapleVersion);
Trying to read file C:\Maple11/P11/SP11/Help/AlternatingGroup_M11.mws...
Success... inserting topic SP,AlternatingGroup from the file:
```

C:\Maple11/P11/SP11/Help/AlternatingGroup_M11.mws

```
Trying to read file C:\Maple11/P11/SP11/Help/Dummy_M11.mws...
Success... inserting topic SP,Dummy from the file:
```

C:\Maple11/P11/SP11/Help/Dummy_M11.mws

```
Trying to read file C:\Maple11/P11/SP11/Help/FiniteGroups_M11.mws...
Success... inserting topic SP,FiniteGroups from the file:
```

C:\Maple11/P11/SP11/Help/FiniteGroups_M11.mws

```
Trying to read file C:\Maple11/P11/SP11/Help/generateGinvariants_M11.mws...
Success... inserting topic SP,generateGinvariants from the file:
```

C:\Maple11/P11/SP11/Help/generateGinvariants_M11.mws

```
Trying to read file C:\Maple11/P11/SP11/Help/gpolynom_M11.mws...
Success... inserting topic SP,gpolynom from the file:
```

C:\Maple11/P11/SP11/Help/gpolynom_M11.mws

```
Trying to read file C:\Maple11/P11/SP11/Help/hpolynom_M11.mws...
Success... inserting topic SP,hpolynom from the file:
```

C:\Maple11/P11/SP11/Help/hpolynom_M11.mws

```
Trying to read file C:\Maple11/P11/SP11/Help/isContained_M11.mws...
Success... inserting topic SP,isContained from the file:
```

C:\Maple11/P11/SP11/Help/isContained_M11.mws

```
Trying to read file C:\Maple11/P11/SP11/Help/isGinvariant_M11.mws...
Success... inserting topic SP,isGinvariant from the file:
```

C:\Maple11/P11/SP11/Help/isGinvariant_M11.mws

```
Trying to read file C:\Maple11/P11/SP11/Help/isSymmetric_M11.mws...
Success... inserting topic SP,isSymmetric from the file:
```

C:\Maple11/P11/SP11/Help/isSymmetric_M11.mws

```
Trying to read file C:\Maple11/P11/SP11/Help/MatrixAction_M11.mws...
Success... inserting topic SP,MatrixAction from the file:
```

C:\Maple11/P11/SP11/Help/MatrixAction_M11.mws

```
Trying to read file C:\Maple11/P11/SP11/Help/powersum_M11.mws...
Success... inserting topic SP,powersum from the file:
```

C:\Maple11/P11/SP11/Help/powersum_M11.mws

```
Trying to read file C:\Maple11/P11/SP11/Help/reduceGinvariants_M11.mws...
Success... inserting topic SP,reduceGinvariants from the file:
```

C:\Maple11/P11/SP11/Help/reduceGinvariants_M11.mws

```
Trying to read file C:\Maple11/P11/SP11/Help/Reynolds_M11.mws...
Success... inserting topic SP,Reynolds from the file:
```

C:\Maple11/P11/SP11/Help/Reynolds_M11.mws

```
Trying to read file C:\Maple11/P11/SP11/Help/Sigma_M11.mws...
Success... inserting topic SP,Sigma from the file:
```

C:\Maple11/P11/SP11/Help/Sigma_M11.mws

```
Trying to read file C:\Maple11/P11/SP11/Help/sigma_to_powersum_M11.mws...
Success... inserting topic SP,sigma_to_powersum from the file:
```

C:\Maple11/P11/SP11/Help/sigma_to_powersum_M11.mws

```
Trying to read file C:\Maple11/P11/SP11/Help/SPversion_M11.mws...
Success... inserting topic SP,SPversion from the file:
```

C:\Maple11/P11/SP11/Help/SPversion_M11.mws

```
Trying to read file C:\Maple11/P11/SP11/Help/SymmetricGroup_M11.mws...
Success... inserting topic SP,SymmetricGroup from the file:
```

C:\Maple11/P11/SP11/Help/SymmetricGroup_M11.mws

```
Trying to read file C:\Maple11/P11/SP11/Help/SyzygyIdeal_M11.mws...
Success... inserting topic SP,SyzygyIdeal from the file:
```

C:\Maple11/P11/SP11/Help/SyzygyIdeal_M11.mws

```
***********************
Finished inserting 18 file topics into the HDB and Browser
***********************
```

```
> #?SP
>
>
```

Inserting help pages for code_support:

```
> HDB_LIB_PATH:=convert(libname[1],name);
  HELP_FILE_PATH:=`C:\\Maple11/P11/Code_support11/Help/`;
  BROWSER_PATH:=`Mathematics/Algebra/`;
  ModuleName   :=`code_support`;
  Parent       :=`code_support,code_support`;
```

```
GrandParent :=`Clifford,intro`;
MapleVersion:="M11";
typesLIST   :=[];
convertsLIST:=[];
```

$$HDB\_LIB\_PATH := C:\backslash Maple11/Cliffordlib$$
$$HELP\_FILE\_PATH := C:\backslash Maple11/P11/Code\_support11/Help/$$
$$BROWSER\_PATH := Mathematics/Algebra/$$
$$ModuleName := code\_support$$
$$Parent := code\_support,code\_support$$
$$GrandParent := Clifford,intro$$
$$MapleVersion := \text{"M11"}$$
$$typesLIST := [\ ]$$
$$convertsLIST := [\ ]$$

Step 1: Generating an automatic list from help page files:

```
> Lauto:=makeLIST(ModuleName,Parent,GrandParent,HELP_FILE_PATH,Maple
  Version);
```

Lauto := [
[*code_support,code_support*, *Clifford,intro*, ["code_support,code_support", "code_support"]],
[*code_support,examples*, *code_support,code_support*, ["code_support,examples", "examples"]],
[*code_support,INSERT_HELPPAGES*, *code_support,code_support*,
["code_support,INSERT_HELPPAGES", "INSERT_HELPPAGES"]]]

```
>
```

Step 2: Modifying, if needed, certain entries in Lauto list:

```
> #modsLIST:=[
  #[`Octonion,omul`,["Octonion,omul","omul","&o","octonion","Octonio
  n"]]
  #]:
> #for mem in modsLIST do
  #    Lauto:=modifyLIST(Lauto,op(mem))
  #end do:
  Lauto;
```

[[*code_support,code_support*, *Clifford,intro*, ["code_support,code_support", "code_support"]],
[*code_support,examples*, *code_support,code_support*, ["code_support,examples", "examples"]],
[*code_support,INSERT_HELPPAGES*, *code_support,code_support*,
["code_support,INSERT_HELPPAGES", "INSERT_HELPPAGES"]]]

```
>
```

Step 3: Inserting all help pages into HDB and the browser using the last modified list:

```
> insert_helppages(Lauto,MapleVersion);
Trying to read file C:\Maple11/P11/Code_support11/Help/code_support_M11.mws...
Success... inserting topic code_support,code_support from the file:
```

C:\Maple11/P11/Code_support11/Help/code_support_M11.mws

```
Trying to read file C:\Maple11/P11/Code_support11/Help/examples_M11.mws...
Success... inserting topic code_support,examples from the file:
```

C:\Maple11/P11/Code_support11/Help/examples_M11.mws

```
Trying to read file C:\Maple11/P11/Code_support11/Help/INSERT_HELPPAGES_M11.mws...
Success... inserting topic code_support,INSERT_HELPPAGES from the file:
```

C:\Maple11/P11/Code_support11/Help/INSERT_HELPPAGES_M11.mws

```
***********************
Finished inserting 3 file topics into the HDB and Browser
***********************
> ?code_support
>
>
>
>
> ########################################
>
```

See Also: code_support, examples, code_support, code_support

Last modified: June 19, 2008, RA/BF

"code_support' for CLIFFORD, Bigebra, Octonion, GTP, Cliplus, Octonion in Maple 11

(*) Department of Mathematics, Box 5054
Tennessee Technological University, Cookeville, TN 38505
Tel: USA (931) 372-3569, fax: USA (931) 372-6353
E-mail: rablamowicz@tntech.edu
URL: http://math.tntech.edu/rafal/

($) Universit"at Konstanz, Fachbereich Physik, Fach M678
78457 Konstanz, Germany
E-mail: Bertfried.Fauser@uni-konstanz.de
URL: http://kaluza.physik.uni-konstanz.de/~fauser/

```
> restart:
  with(code_support);
```

Module code_cupport ver. 1.03 for CLIFFORD et al. for Maple 11

Last revised: March 10, 2007

[*NamesInLibrary*, *change_helpfiles*, *change_name*, *copy_file*, *get_TEXT*, *get_dir*, *insert_helppages*, *makeLIST*, *modifyLIST*, *replace_in_file*, *split*]

```
>
```

This help page shows various commands and their code of the supplementary package "code_support" that can be used to manipulate help pages for various packages, for example, CLIFFORD.

This package enables one to:

- copy Maple worksheets from one directory to another and make certain string replacements using **change_helpfiles**,
- change worksheet names using **change_name**,
- copy files from one directory to another using **copy_file**,
- read a Maple worksheet as text using **get_TEXT**,
- read and return names of Maple worksheets located in a specified directory using **get_dir**,
- take a list of file names with help topics and insert them automatically into Maple browser and HDB database entries using **insert_helppages**,
- make an automatic list of data entries needed by **insert_helppages** using **makeLIST**,
- modify a list of data entries created by **makeLIST** using **modifyLIST**,
- replace certain strings in Maple worksheets using **replace_in_file**,

- split file names into a sequence of strings using **split**,
- convert directory names listed as symbols to strings using **`convert/symbol_dir_to_string_dir`**,
- convert back directory name from a string format to a symbol format using **`convert/string_dir_to_symbol_dir`**.

For examples how these commands can be used see examples of code_support.

Cookeville, June 19, 2008

```
> ################################################################
  ################
  ### It is assumed that directories can be entered either as
  strings:
  ### (a) In Windows:   "C:\\Maple11\\Clifford\\Help_11_New\\"
  ### (b) In Linux:
  "/home/fauser/BIG_ALL/P11/Clifford/Help_11_New/"
  ###
  ### or as symbols:
  ### (c) In Windows:   `D:\\Bigebra/Help_Bigebra/`
  ### (d) In Linux:     `/home/fauser/BIG_ALL/Help_Bigebra/`
  ################################################################
  ################
```

```
> restart:
```

```
> code_support:=module()
  export
  split,copy_file,get_dir,change_name,replace_in_file,change_helpfil
  es,get_TEXT,
         makeLIST,modifyLIST,insert_helppages,NamesInLibrary;
  local setup;
  option package, load=setup:
  ############################
```

1. Procedure **NamesInLibrary** gives names of programs stored in the library specified as the argument. It gives names as strings.

```
> NamesInLibrary:=proc(lib) local e,L;
  ##############################################
  options `Copyright (c) 2002-2008 by Rafal Ablamowicz and Bertfried
  Fauser. All rights reserved.`;
  description `Last revised: March 10, 2007`;
  ##############################################
    L:=march('list',lib):
```

```
    for e in L do
      if SearchText(":",e[1],1..1)=0 then print(e[1]);
    end if;
    end do:
  end proc:
>
```

2. Procedure **split** takes character pattern "pat" and splits a string into a sequence of substrings remained after removing "pat" from the string. If the pattern does not match, the string is returned back.

```
> split:=proc(pat,str)
    local a_seq,a1,x,pos,len,n;
  ################################################
  options `Copyright (c) 2002-2008 by Rafal Ablamowicz and Bertfried
  Fauser. All rights reserved.`;
  description `Last revised: March 10, 2007`;
  ################################################
    a_seq:=NULL;
    a1:=str;
    len:=length(a1);
    pos:=1;
    while(SearchText(pat,a1,pos..len) <>0) do
      n:=SearchText(pat,a1,pos..len);
      x:=substring(a1,pos..n-1);
      a_seq:=a_seq,x;
      a1:=substring(a1,n+length(pat)..len);
      len:=length(a1);
    od:
    a_seq:=a_seq,a1;
  end:
```

3. Procedure **copy_file** copies a single file named 'in_file' and located in the directory 'path' as another file called 'out_file' to the same directory.
The path can be specified as a symbol, e.g., as `D:\\Bigebra/Help_Bigebra/` or as a "string", e.g., "D:\\Bigebra\\Help_Bigebra\\".

```
> copy_file:=proc(path::{string,symbol},in_file::{string,symbol},out
  _file::{string,symbol})
    local OUT,line,n_lines,Text,path1,in_file1,out_file1;
  ################################################
    options `Copyright (c) 2002-2008 by Rafal Ablamowicz and
  Bertfried Fauser. All rights reserved.`;
    description `Last revised: March 10, 2007`;
```

```
##############################################
if type(path,symbol) then
   path1:=path;
else
   path1:=convert(path,string_dir_to_symbol_dir);
end if;
##############################################
if type(in_file,symbol) then
   in_file1:=in_file;
else
   in_file1:=convert(in_file,name);
end if:
##############################################
if type(out_file,symbol) then
   out_file1:=out_file;
else
   out_file1:=convert(out_file,name);
end if:
##############################################
  n_lines:=0;
  Text:=[];
  OUT:=fopen(cat(path1,out_file1),WRITE,TEXT);
  while true do
    line:=readline(cat(path1,in_file1));
    if line = 0 then break end if:
    Text:=[op(Text),line];
    writeline(OUT,line);
    n_lines:=n_lines+1;
  end do:
  close(OUT);
printf("file %s containing %d lines has been copied as file %s in
the directory %s\n",in_file1,n_lines,out_file1,path1);
  return;
end proc:
```

4. Procedure **get_dir** fetches the *.mws files from a directory given as an argument and returns file names in a list of strings. Note that the directory can be specified as a "string", for example, "C:\\Maple10\\P10\\test1\\", or as `symbol`, for example, `C:\\Maple10/P10/test1`.

```
> get_dir:=proc(dir::{string,symbol})
  local wc_line,out_list,dir1;
##############################################
  options `Copyright (c) 2002-2008 by Rafal Ablamowicz and
```

```
Bertfried Fauser. All rights reserved.`;
  description `Last revised: March 10, 2007`;
##############################################
if type(dir,symbol) then
   dir1:=convert(dir,symbol_dir_to_string_dir):
else
   dir1:=dir:
end if;
##############################################
out_list:=ssystem(cat("dir ",dir1));
if out_list[1] <> 0 then error "Could not read the directory" end
if:
#####################################
wc_line:=proc(line::string)
  local c,nw,out,List,item:
  nw := 0;
  out := true;
  List:=[]:
  item:="":
  for c in line do
    if c = " " or c = "\t" or c = "\n" then
      out := true;
      if SearchText(`.mws`,item)<>0 then
        List:=[op(List),item];
      end if:
      item:="";
    elif out then
      out := false;
      nw:=nw+1;
      item:=eval(cat(item,c));
    else
      item:=eval(cat(item,c));
    end if:
  end do:
  if length(item) <> 0 and SearchText(`.mws`,item)<> 0 then
    List:=[op(List),item];
  end if:
  return List
end proc:
##########################################
return wc_line(out_list[2])
end proc:
```

5. Procedure **change_name** changes a name of a file "filename" specified as a string by replacing a substring "substrout" with a new substring "substrin". It returns the name of the new file as a string. If the string "substrout" is not found in the "filename" then it is appended to the name of the file before ".mws" extension. Note that "substrout" could be an empty string "".

```
> change_name:=proc(filename::string,substrout::string,substrin::str
  ing)
                local N,fileext,filenamenew,n,nout,nb;
  ##############################################
  options `Copyright (c) 2002-2008 by Rafal Ablamowicz and Bertfried
  Fauser. All rights reserved.`;
  description `Last revised: March 10, 2007`;
  ##############################################
  N:=length(filename);                  ###length of the whole string
  nout:=length(substrout);              ###length of the string to be
  removed
  n:=SearchText(substrout,filename);  ###location of the string to
  be removed
  #return n,nout,N;
  if n+nout-1<N then
     if n=0 then
        fileext:=substring(filename,-4..-1):

  filenamenew:=cat(substring(filename,1..(N-4)),substrin,fileext);
     else
        nb:=N-(n+nout-1);
        fileext:=substring(filename,-nb..-1):

  filenamenew:=cat(substring(filename,1..n-1),substrin,substring(fil
  ename,n+nout..N-nb),fileext);
     end if;
  elif n+nout-1=N then
     filenamenew:=cat(substring(filename,1..n-1),substrin);
  else
     error "wrong string lengths"
  end if;
  return filenamenew;
  end proc:
```

6. Procedure **replace_in_file** replaces strings specified in a list 'pat_list' with strings specified in a list 'rep_list' in a single file 'in_file' and writes a new file file 'out_file'. File name needs to be of type string, e.g., "adfmatrix.mws" with extensions while lists contain strings, for example, ["November 1","2002"], etc. Procedure **replace** permits lists of replacements to be empty. In that case, it just writes

a new file but no replacements are made. Note that "in_file" and "out_file" must contain full paths to the in directory and to the out directory respectively.

```
> replace_in_file:=proc(in_file::{string,name},
                        out_file::{string,name},
                        pat_list::list(string),
                        rep_list::list(string))
  local
  N1,Np,OUT,i,j,pattern,replace_string,pos,str1,str2,line,n_repl,Tex
  t,repflag;
  ##############################################
  options `Copyright (c) 2002-2008 by Rafal Ablamowicz and Bertfried
  Fauser. All rights reserved.`;
  description `Last revised: March 10, 2007`;
  ##############################################
  if nops(pat_list) <> nops(rep_list) then
      error "Need two lists of equal length, or two blank lists if
  no replacements are to be made!"
  end if:
  ##############################################
  if member("",pat_list) then
     error "first list cannot contain an empty string"
  end if;
  ##############################################
  if evalb(nops(pat_list)>0 and pat_list<>rep_list) then
       n_repl:=[seq(0,i=1..nops(pat_list))];
       repflag:=true;
  else
       n_repl:=[];
       repflag:=false;
  end if;
  ##############################
  ##Reading file in first
  ##############################
  Text:=[];
  while true do
    line:=readline(`in_file`);
    if line = 0 then break end if:
    Text:=[op(Text),line];
  end do:
  ##############################
  ##Make replacements if needed:
  ##############################
```

```
if repflag then
  for i from 1 to nops(Text)-1 do
    for j from 1 to nops(pat_list) do
      pattern:=pat_list[j];
      replace_string:=rep_list[j];
      if not evalb(pattern=replace_string) then
         Np:=length(pattern);
         if abs(Np - length(replace_string)) > 5 then
            error "Strings %1 and %2 ought to be of the same
length or within 5 characters", pattern,replace_string
         end if:
         while true do
           N1:=length(Text[i]);
           line:=cat(Text[i],Text[i+1]);
           pos:=SearchText(pattern,line,1..length(line));
           #
           if pos = 0 then break end if:
           #
           n_repl:=subsop(j=n_repl[j]+1,n_repl);
           str1:=line[1..pos-1];
           if pos+Np <= length(line) then
              str2:=line[pos+Np..-1];
           else
              str2:="";
           end if:
           line:=cat(str1,replace_string,str2);
           Text:=subsop(i=line[1..N1],Text);
           Text:=subsop(i+1=line[N1+1..-1],Text);
         end do:
      end if;
    end do:
  end do:
end if;
#####################
##Write new file with or without replacements:
#####################
OUT:=fopen(out_file,WRITE,TEXT);
for i from 1 to nops(Text) do
    writeline(OUT,Text[i]);
end do:
close(OUT);
return n_repl;
end proc:
```

7. Procedure **change_helpfiles** automatically copies all Maple help pages *.mws from "dir_in" specified as string, for example, "C:\\Maple10\\P10\\test1\\", to a different directory "dir_out" also specified as a string, for example, "C:\\Maple10\\P10\\test2\\". It can replace string patterns specified as a list 'pat_list' of strings with strings listed in a list 'rep_list'. Both lists must be of equal length, or they can be empty. If the lists are empty, no replacements are made. This procedure uses optional 5th and 6th argument, each of type "string". When used, the substring specified as the 5th argument in the name of the file currently processed is replaced with string entered as the last 6th argument. This way, replacements can be made at the same time that the file names are changed.

```
> change_helpfiles:=proc(dir_in::{string,name},
                         dir_out::{string,name},
                         pat_list::list({string,name}),
                         rep_list::list({string,name}))
  local
  dir1,dir2,N,dir_list,fileold,filenew,changenameflag,makerepflag,fi
  le:
  ##############################################
  options `Copyright (c) 2002-2008 by Rafal Ablamowicz and Bertfried
  Fauser. All rights reserved.`;
  description `Last revised: March 10, 2007`;
  ##############################################
  dir1,dir2:=dir_in,dir_out;
  ########################################
  if nops(pat_list)<>nops(rep_list) then
     error "pattern and replacement lists must be of equal length"
  end if;
  ########################################
  if nops(pat_list)>0 then makerepflag:=true else makerepflag:=false
  end if:
  ########################################
  changenameflag:=false:
  if nargs=6 then
       if not type(args[5],string) or not type(args[6],string) then
          error "arguments 5 and 6, when used, must be strings,
  e.g., %1 and %2","_M5","_M6"
       end if;
  changenameflag:=true:
  end if:
  ########################################
  dir_list:=get_dir(dir1);
  ########################################
  for file in dir_list do
```

```
fileold:=file:
if changenameflag then
filenew:=change_name(fileold,args[5],args[6])
                        else filenew:=file
end if;
N:=replace_in_file(cat(dir1,fileold),cat(dir2,filenew),pat_list,re
p_list):
makerepflag:=evalb(pat_list<>rep_list):
if makerepflag and changenameflag then
   printf("processing file %s renamed as %s with replacement(s):
%a\n",fileold,filenew,N)
elif makerepflag and not changenameflag then
   printf("processing file %s without renaming but with
replacement(s): %a\n",fileold,N);
elif not makerepflag and changenameflag then
   printf("processing file %s renamed as %s without
replacements\n",fileold,filenew)
else
printf("processing file %s without renaming and without
replacements - just copying\n",fileold,filenew)
end if;
end do:
printf("************************\n");
printf("finished processing %d files\n",nops(dir_list));
printf("************************\n");
NULL
end proc:
```

8. Procedure **get_TEXT** reads a file "FILE" and puts it into a TEXT(line1,line2,...) format needed by INTERFACE_HELP.

```
> get_TEXT:=proc(FILE)
    local textobject,line;
  ##############################################
  options `Copyright (c) 2002-2008 by Rafal Ablamowicz and Bertfried
  Fauser. All rights reserved.`;
  description `Last revised: March 10, 2007`;
  ##############################################
    textobject := NULL;
    line := readline(FILE);
    while line <> 0 do
      if length(line) <= 10 then
        line := ``||line||`    `:
```

```
      fi;
      textobject := textobject, line:
      line := readline(FILE):
    od;
    textobject := TEXT(textobject):
  end:
```

9. Procedure **makeLIST** makes a list of data entries needed by **insert_helppages** to insert help pages into the HDB and the browser. In particular, it automatically creates a list of aliases that later can be modified by the procedure **modifyLIST**. It takes four arguments:

modulename - name of the module as a `symbol`
parent - the name of the Parent as a `symbol`
grandparent - the name of the GrandParent as `symbol`
LDIR - it is either a list of strings with topic entries such as, for example, ["&cco","&cco_d"], or a directory where *._M8.mws files are stored
version - it is a string that gives the current version of Maple, e.g., "M10". It is expected that the file names *._M10.mws will contain the same string. If not, error message is returned and the process is stopped

```
> makeLIST:=proc(modname::{symbol,name},
                 parent::{symbol,name},
                 grandparent::{symbol,name},
                 LDIR::{list(string),name,string},
                 version::string)
                 local
  modulename,mainmodulepage,i,N,L,datastring,dataentry,Toplevel,stri
  nglist,mem;
                 global typesLIST,convertsLIST;
  ##############################################
  options `Copyright (c) 2002-2008 by Rafal Ablamowicz and Bertfried
  Fauser. All rights reserved.`;
  description `Last revised: March 10, 2007`;
  ##############################################
  if not assigned(typesLIST) then error "global variable `typesLIST`
  has not been assigned a list" end if;
  if not assigned(convertsLIST) then error "global variable
  `convertsLIST` has not been assigned a list" end if;
  if nops(typesLIST)>0 then
     if not type(typesLIST,list(string)) then
        error "global list `typesLIST` must be either empty or it
  must contain strings"
     end if
```

```
end if;
if nops(convertsLIST)>0 then
   if not type(convertsLIST,list(string)) then
      error "global list `convertsLIST` must be either empty or it
must contain strings"
   end if
end if;
##############################################
if type(LDIR,{name,string}) then
   stringlist:=get_dir(LDIR);
#############################################
   for datastring in stringlist do
       if evalb(SearchText(version,datastring)=0) then
          error "filename %1 in the directory %2 does not match
Maple version %3 - process is
aborted!!!",datastring,LDIR,version;
       end if;
   end do:
##############################################

stringlist:=map(change_name,stringlist,cat("_",version,".mws"),"")
;
##############################################
   ###########modify by adding "type"
   if nops(typesLIST)>0 then
      for i from 1 to nops(stringlist) do
          mem:=stringlist[i]:
          if member(mem,typesLIST) then

stringlist:=subsop(i=cat("type",",",mem),stringlist);
          end if;
      end do;
   end if;
   ###########modify by adding "convert"
   if nops(convertsLIST)>0 then
      for i from 1 to nops(stringlist) do
          mem:=stringlist[i]:
          if member(mem,convertsLIST) then

stringlist:=subsop(i=cat("convert",",",mem),stringlist);
          end if;
      end do;
   end if;
```

```
##############################################
else
   stringlist:=LDIR;
end if:
##############################################
modulename,mainmodulepage:=split(`,`,Parent);
modulename:=modname;
##############################################
L:=[]:
for datastring in stringlist do
    if convert(datastring,name)=mainmodulepage then
       Toplevel:=grandparent
    else
       Toplevel:=parent
    end if;
    dataentry:=convert(datastring,name);
    L:=[op(L),[cat(modulename,`,`,convert(dataentry,name)),
               Toplevel,

[cat(convert(modulename,string),",",datastring),datastring]]]
end do;
return L;
end proc:
```

10. Procedure **modifyLIST** can be used to replace "aliases" that are automatically generated by the procedure **makeLIST** for each help topic to be inserted into the browser. In the data entry for each help topic, the third entry is a list of aliases ["alias1","alias2",...]. This list is then replaced with the third argument of type 'list(string)' to **modifyLIST** and the procedure returns a new complete list. The original list remains unchanged.

```
> modifyLIST:=proc(Lp::listlist,topic::symbol,newaliases::list(strin
g))
            local item,flag,i,newentryp,L,newitem;
##############################################
options `Copyright (c) 2002-2008 by Rafal Ablamowicz and Bertfried
Fauser. All rights reserved.`;
description `Last revised: March 10, 2007`;
##############################################
L:=Lp:
flag:=false:
for i from 1 to nops(L) while not flag do
    flag:=evalb(op(1,L[i])=topic);
end do;
if flag=false then
```

```
    error "topic %1 has not been found in the entered list",topic;
  end if;
  newitem:=subsop(3=newaliases,L[i-1]);
  return subsop(i-1=newitem,L);
  end proc:
```

11. Procedure **insert_help pages** takes a list of file names with help topics and inserts them into Maple browser and HDB Database entries.

- This procedure puts the below listed help pages into a maple.hdb help-page-database. These are the official help pages for the CLIFFORD, Bigebra, Cliplus, GTP, Octonion modules. It uses the following global path names:

- HDB_LIB_PATH : The path to a directory in the Maple search path (not the path of the Maple.hdb database in .../lib This pathname is here taken from libname, and has to be added to libname in any case to be able to use the help pages.
- HELP_FILE_PATH : The location of the help pages.
- BROWSER_PATH: The location in the Maple Help Page Browser at which the Package should be 'mounted'.
- Parent: The Package name or any help topic
- GrandParent: The Clifford package, (location in the Help Browser)

```
> insert_helppages:=proc(LIST)
    local Text,delim,x,lst,name,topic,browser_path,File,MODULE;
    global HDB_LIB_PATH, HELP_FILE_PATH, BROWSER_PATH;
  ##############################################
  options `Copyright (c) 2002-2008 by Rafal Ablamowicz and Bertfried
  Fauser. All rights reserved.`;
  description `Last revised: March 10, 2007`;
  ##############################################
  for x in LIST do
  #printf("Inserting topic %s and reading file:",x[1]);
      if (SearchText(`/`,x[1],1..length(x[1])) <> 0 ) then
        lst:=[split(`/`,x[1])];
        name :=lst[-1..-1];
        topic := ``;
        delim := ``;
        while(lst <> []) do
          topic := cat(topic,cat(delim,lst[1]));
          lst:=lst[2..-1];
```

```
        delim:=`,`;
      od;
      browser_path:=cat(BROWSER_PATH,x[1]);
    elif (SearchText(`,`,x[1],1..length(x[1])) <> 0 ) then
      lst:=[split(`,`,x[1])];
      name :=lst[-1..-1];
      topic:=x[1];
      browser_path:=BROWSER_PATH;
      delim:=``;
      while(lst <> []) do
        browser_path:=cat(browser_path,cat(delim,lst[1]));
        lst:=lst[2..-1];
        delim:=`/`;
      od;
    else
      topic:=x[1];
      name :=x[1];
      browser_path:=cat(BROWSER_PATH,topic);
    fi;
    name:=op(name);
##############################################
if nargs=1 then
   File:=cat(HELP_FILE_PATH,cat(name,".mws")):
elif nargs=2 then
   if not type(args[2],string) then
      error "second optional argument must be a string giving
Maple version included in file names, e.g. %1","M6"
   else
   File:=cat(HELP_FILE_PATH,cat(name,"_",args[2],".mws")):
   end if;
end if;
##############################################
###HELP_FILE   = cat(HELP_FILE_PATH,cat(name,".mws")),
##############################################
    MODULE[x[1]]:=table( [
      NAME   = name,
      PARENT = x[2],
      TOPIC  = topic,
      HELP_FILE   = File,
      ALIASES= x[3],
      ACTIVE = `true`,
      BROWSER_LOC = browser_path,
      LIB         = HDB_LIB_PATH
```

```
    ]);
    ##
    ## get TEXT
    printf("Trying to read file %s...\n",File); ###for testing
    Text:=get_TEXT(File);
    ##
    ## put into DB
  ##################################################
  printf("Success... inserting topic %s from the file:",x[1]);
  print(MODULE[x[1]][HELP_FILE]);
  print(INTERFACE_HELP('insert','topic'=topic,'text'=Text,helpfile=H
  DB_LIB_PATH,'aliases'=x[3],'active'=false) );
  print(INTERFACE_HELP('insert','topic'=topic,'browser'=browser_path
  ,helpfile=HDB_LIB_PATH) );
  od:
  printf("***********************\n");
  printf("Finished inserting %d file topics into the HDB and
  Browser\n",nops(LIST));
  printf("***********************\n");
  NULL

  end:
```

1. Procedure **setup** for the module **code_support** defines two conversion functions that convert directory names from "string" form to `name` (or `symbol`) form, and vice versa.

```
> setup:=proc()
  global
  `convert/string_dir_to_symbol_dir`,`convert/symbol_dir_to_string_d
  ir`;
  ##############################################
  options `Copyright (c) 2002-2008 by Rafal Ablamowicz and Bertfried
  Fauser. All rights reserved.`;
  description `Last revised: March 10, 2007`;
  ##############################################
  `convert/symbol_dir_to_string_dir`:=proc(dir::{symbol,string})
  local T,i,dir1,linuxflag;
  if type(dir,string) then return dir end if;
  linuxflag:=evalb(SearchText(`\\`,dir)=0);
  if linuxflag then return convert(dir,string) end if;
  T:=remove(member,[split("/",convert(dir,string))],{""});
  dir1:=cat(T[1],"\\"):
  if nops(T) >1 then
```

```
      for i from 2 to nops(T) do dir1:=cat(dir1,T[i],"\\") end
do:
end if;
return dir1;
end proc:
##############################################
`convert/string_dir_to_symbol_dir`:=proc(dir::{symbol,string})
local T,i,dir1,linuxflag;
if type(dir,symbol) then return dir end if;
linuxflag:=evalb(SearchText(`\\`,dir)=0);
if linuxflag then return convert(dir,symbol) end if;
T:=remove(member,[split("\\",convert(dir,string))],{""});
dir1:=cat(T[1],"\\",T[2],"\\"):
if nops(T)>2 then
      for i from 3 to nops(T) do dir1:=cat(dir1,T[i],"/") end do:
end if;
return convert(dir1,name);
end proc:
##############################################
print(``);
print(`Module code_cupport ver. 1.03 for CLIFFORD et al. for Maple
11`);
print(`Copyright (c) 2002-2008 by Rafal Ablamowicz and Bertfried
Fauser. All rights reserved.`);
print(`Last revised: March 10, 2007`);
print(``);
##############################################
end proc:

end module:
savelib(code_support);
```

Module code_cupport ver. 1.03 for CLIFFORD et al. for Maple 11
Copyright (c) 2002-2008 by Rafal Ablamowicz and Bertfried Fauser. All rights reserved.
Last revised: March 10, 2007

```
> restart:with(code_support);
```

Module code_cupport ver. 1.03 for CLIFFORD et al. for Maple 11
Copyright (c) 2002-2008 by Rafal Ablamowicz and Bertfried Fauser. All rights reserved.
Last revised: March 10, 2007

[*NamesInLibrary*, *change_helpfiles*, *change_name*, *copy_file*, *get_TEXT*, *get_dir*, *insert_helppages*, *makeLIST*, *modifyLIST*, *replace_in_file*, *split*]

```
> NamesInLibrary(libname[1]);
```

"matrealL.m"
"Clifford.m"
"_AlternatingGroup_rem_table.m"
"_SymmetricGroup_rem_table.m"
"Define.m"
"SINGULARPLURALlink.m"
"GTP.m"
"_implicitbezierpolynomial.m"
"Bigebra.m"
"matquatL.m"
"_Reynolds_rem_table.m"
"Cliplus.m"
"SchurFkt.m"
"_FiniteGroups_rem_table.m"
"_generateGinvariants_rem_table.m"
"SP.m"
"matcompR.m"
"TNB.m"
"code_support.m"
"matrealR.m"
"RJgrobner.m"
"Octonion.m"
"matcompL.m"
"GfG.m"
"matquatR.m"

```
>
>
```

See Also: code_support, examples

Last modified: June 19, 2008, RA/BF.

Function: code_support-NAME where NAME is one of the following: change_helpfiles, change_name, copy_file, get_TEXT, get_dir, insert_helppages, makeLIST, modifyLIST, replace_in_file, split

Calling Sequence:

See examples below

Parameters:

See examples below

Description:

- code_support is a package that provides various functions to copy, rename, modify, save, etc. Maple help worksheets.
- Examples below can be of course re-executed on one's own machine provided that directories and files used in these examples exist on a local system.

Examples:

```
> restart:with(code_support);
```

Module code_cupport ver. 1.03 for CLIFFORD et al. for Maple 11

Copyright (c) 2002-2008 by Rafal Ablamowicz and Bertfried Fauser. All rights reserved.

Last revised: March 10, 2007

[*NamesInLibrary*, *change_helpfiles*, *change_name*, *copy_file*, *get_TEXT*, *get_dir*, *insert_helppages*, *makeLIST*, *modifyLIST*, *replace_in_file*, *split*]

```
> DIR1:=`D:\\Bigebra/Help_Bigebra/`;  ### Directory specified as a symbol in Windows
  DIR2:="C:\\Maple11\\Clifford\\Help_11_New\\";  ###Directory specified as a string in Windows
  DIR3:=`/home/Bigebra/Help_Bigebra/`;  ### Directory specified as a symbol in Linux
  DIR4:="/home/Bigebra/Help_Bigebra/";  ###Directory specified as a string in Linux
```

DIR1 := *D:\Bigebra/Help_Bigebra/*

DIR2 := "C:\Maple11\Clifford\Help_11_New\"

DIR3 := */home/Bigebra/Help_Bigebra/*

DIR4 := "/home/Bigebra/Help_Bigebra/"

```
>
```

Example 1: Procedure **split** removes a string specified as the first argument from another string specified as the second argument:

```
> split("a","adfmatrix.mws");
```

"", "dfm", "trix.mws"

```
> split("b","adfmatrix.mws");
```

"adfmatrix.mws"

```
>
```

Example 2: Procedure **`convert/symbol_dir_to_string_dir`** converts a directory name listed as a symbol to a strings while **`convert/string_dir_to_symbol_dir`** converts back directory name from a string format to a symbol format.

```
> convert(DIR1,symbol_dir_to_string_dir);
  convert(DIR2,symbol_dir_to_string_dir);
  convert(DIR3,symbol_dir_to_string_dir);
  convert(DIR4,symbol_dir_to_string_dir);
```

"D:\Bigebra\Help_Bigebra\"

"C:\Maple11\Clifford\Help_11_New\"

"/home/Bigebra/Help_Bigebra/"

"/home/Bigebra/Help_Bigebra/"

```
> convert(DIR1,string_dir_to_symbol_dir);
  convert(DIR2,string_dir_to_symbol_dir);
  convert(DIR3,string_dir_to_symbol_dir);
  convert(DIR4,string_dir_to_symbol_dir);
```

D:\Bigebra/Help_Bigebra/

C:\Maple11\Clifford/Help_11_New/

/home/Bigebra/Help_Bigebra/

/home/Bigebra/Help_Bigebra/

```
>
```

Example 3: Procedure **copy_file** copies and renames Maple worksheets in one directory. For example, file `all_sigs_M10.mws` will be copied and saved as a new file `all_sigs_M10_copy1.mws` in the directory `C:\\Maple10/P10/test1/`. The directory name can also be specified as a string "C:\\Maple10\\P10\\test1\\".

```
> TEST1:=`C:\\Maple10/P10/test1/`;
  TEST2:="C:\\Maple10\\P10\\test1\\";
  filein:=`all_sigs_M10.mws`;
  fileout:=`all_sigs_M10_copy1.mws`;
```

TEST1 := *C:\Maple10/P10/test1/*

TEST2 := "C:\Maple10\P10\test1\"

filein := *all_sigs_M10.mws*

$$fileout := all\_sigs\_M10\_copy1.mws$$

```
> copy_file(TEST1,filein,fileout);
  copy_file(TEST2,filein,fileout);
file all_sigs_M10.mws containing 117 lines has been copied as file all_sigs_M10_co
py1.mws in the directory C:\Maple10/P10/test1/
file all_sigs_M10.mws containing 117 lines has been copied as file all_sigs_M10_co
py1.mws in the directory C:\Maple10\P10/test1/
>
```

Example 4: Procedure **get_dir returns** a list of strings which are names of files located in the specified directory.

```
> dir1:="C:\\Maple10\\P10\\test1\\";
  dir2:=`C:\\Maple10/P10/test1/`;
```

$$dir1 := \text{"C:\Maple10\P10\test1\"}$$

$$dir2 := C:\backslash Maple10/P10/test1/$$

```
> S:=get_dir(dir1);
  S:=get_dir(dir2);
```

$S :=$ ["all_sigs_M10.mws", "all_sigs_M10_copy1.mws", "examples723.mws", "Groebner.for.Grassmann_M10.23vii07.mws", "SchurFkt-code-22vii07.mws"]

$S :=$ ["all_sigs_M10.mws", "all_sigs_M10_copy1.mws", "examples723.mws", "Groebner.for.Grassmann_M10.23vii07.mws", "SchurFkt-code-22vii07.mws"]

```
>
```

Example 5: Procedure **change_name** changes names of Maple worksheets specified as strings. Various cases are shown below.

```
> X:="adfmatrix_M8.mws";
```

$X :=$ "adfmatrix_M8.mws"

```
> X1:=change_name(X,"_M8","_M10");
```

$X1 :=$ "adfmatrix_M10.mws"

```
> X2:=change_name(X,"_M8","");
```

$X2 :=$ "adfmatrix.mws"

```
> X3:=change_name(X1,"_M10","_M11");
```

$X3 :=$ "adfmatrix_M11.mws"

```
> change_name(X,"_M8.mws","");
```

"adfmatrix"

```
> `S`:=S;
```

$S :=$ ["all_sigs_M10.mws", "all_sigs_M10_copy1.mws", "examples723.mws", "Groebner.for.Grassmann_M10.23vii07.mws", "SchurFkt-code-22vii07.mws"]

```
> S:=map(change_name,S,"_M10","_M11");
```

$S :=$ ["all_sigs_M11.mws", "all_sigs_M11_copy1.mws", "examples723_M11.mws", "Groebner.for.Grassmann_M11.23vii07.mws", "SchurFkt-code-22vii07_M11.mws"]

```
> S:=map(change_name,S,"_M11","_M10");
```

S := ["all_sigs_M10.mws", "all_sigs_M10_copy1.mws", "examples723_M10.mws", "Groebner.for.Grassmann_M10.23vii07.mws", "SchurFkt-code-22vii07_M10.mws"]

```
> map(change_name,S,"_M8","");
```

["all_sigs_M10.mws", "all_sigs_M10_copy1.mws", "examples723_M10.mws", "Groebner.for.Grassmann_M10.23vii07.mws", "SchurFkt-code-22vii07_M10.mws"]

```
>
```

Example 6: Procedure **replace_in_file** reads a Maple worksheet from directory DIR1 and then saves is in a new directory DIR2 under the same name after replacements have been made in the text of the file. The replacements are specified as two lists of strings: the first list contains strings to be replaced while the second list contains their string replacements. It returns a list that shows the number of replacements for each entry in the first list. If no replacements have been made, 0's are returned.

```
> DIR1:="C:\\Maple8\\P8\\test1\\";
  DIR2:="C:\\Maple10\\P10\\test2\\";
```

$DIR1$:= "C:\Maple8\P8\test1\"

$DIR2$:= "C:\Maple10\P10\test2\"

```
> infile:="&c_M8.mws";
```

$infile$:= "&c_M8.mws"

```
> cat(DIR1,infile);
```

"C:\Maple8\P8\test1\&c_M8.mws"

```
> get_dir(DIR1);
```

["&c_M8.mws", "adfmatrix_M8.mws", "all_sigs_M8.mws", "all_sigs_M8_copy1.mws"]

```
> replace_in_file(cat(DIR1,"&c_M8.mws"),cat(DIR2,"&c_M10.mws"),
  ["Cliff8","Clifford[","`]","November
  16","2002","Cli8plus","Cliplus["],
  ["Clifford","Clifford:-","`","January
  4","2003","Cliplus","Cliplus:-"]);
```

[0, 0, 0, 0, 1, 0, 0]

```
> replace_in_file(cat(DIR1,"&c_M8.mws"),cat(DIR2,"&c_M10.mws"),["Nov
  ember 3","2003"],["January 5","2003"]);
```

[1, 0]

```
>
```

Example 7: Procedure **change_helpfiles** automatically copies all Maple worksheets from one directory to another after making appropriate replacements as in **replace_in_file** except that **replace_in_file** makes such changes only in one file. It provides feedback to the user by listing all files that have been copied and the number of required replacements. It can also change file names when saving new files.

```
> TEST1:="C:\\Maple10\\P10\\test1\\";
  TEST2:="C:\\Maple10\\P10\\test2\\";
```

```
TEST11:=`C:\\Maple10/P10/test1/`;
TEST22:=`C:\\Maple10/P10/test2/`;
```

$$TEST1 := \text{"C:\Maple10\P10\test1\"}$$

$$TEST2 := \text{"C:\Maple10\P10\test2\"}$$

$$TEST11 := C:\backslash Maple10/P10/test1/$$

$$TEST22 := C:\backslash Maple10/P10/test2/$$

```
> change_helpfiles(TEST1,TEST2,["Cliff8","October 8, 1995"],["Clifford"," 1995-2006"]); #copying with replacements
processing file all_sigs_M10.mws without renaming but with replacement(s): [8, 1]
processing file all_sigs_M10_copy1.mws without renaming but with replacement(s): [
8, 1]
processing file examples723.mws without renaming but with replacement(s): [0, 0]
processing file Groebner.for.Grassmann_M10.23vii07.mws without renaming but with r
eplacement(s): [0, 0]
processing file SchurFkt-code-22vii07.mws without renaming but with replacement(s)
: [0, 0]
***********************
finished processing 5 files
***********************
```

```
> change_helpfiles(TEST1,TEST2,["Cliff8"],["Clifford"],"_M10","_M11"); #copying with name changes
processing file all_sigs_M10.mws renamed as all_sigs_M11.mws with replacement(s):
[8]
processing file all_sigs_M10_copy1.mws renamed as all_sigs_M11_copy1.mws with repl
acement(s): [8]
processing file examples723.mws renamed as examples723_M11.mws with replacement(s)
: [0]
processing file Groebner.for.Grassmann_M10.23vii07.mws renamed as Groebner.for.Gra
ssmann_M11.23vii07.mws with replacement(s): [0]
processing file SchurFkt-code-22vii07.mws renamed as SchurFkt-code-22vii07_M11.mws
with replacement(s): [0]
***********************
finished processing 5 files
***********************
```

```
> change_helpfiles(TEST1,TEST2,["Cliff8"],["Clifford"]); #copying without name changes
processing file all_sigs_M10.mws without renaming but with replacement(s): [8]
processing file all_sigs_M10_copy1.mws without renaming but with replacement(s): [
8]
processing file examples723.mws without renaming but with replacement(s): [0]
processing file Groebner.for.Grassmann_M10.23vii07.mws without renaming but with r
eplacement(s): [0]
processing file SchurFkt-code-22vii07.mws without renaming but with replacement(s)
: [0]
***********************
finished processing 5 files
***********************
```

```
> change_helpfiles(TEST1,TEST2,["Cliff8"],["Cliff8"]); #copying without name changes or replacements
processing file all_sigs_M10.mws without renaming and without replacements - just
copying
processing file all_sigs_M10_copy1.mws without renaming and without replacements -
just copying
```

```
processing file examples723.mws without renaming and without replacements - just c
opying
processing file Groebner.for.Grassmann_M10.23vii07.mws without renaming and withou
t replacements - just copying
processing file SchurFkt-code-22vii07.mws without renaming and without replacement
s - just copying
**********************
finished processing 5 files
**********************
```

The following error message shows that the list of strings to be replaced cannot contain empty strings:

```
> change_helpfiles(TEST1,TEST2,[""],[""]); ###<<<Intended error
  message
Error, (in code_support:-replace_in_file) first list cannot contain an empty
string
```

```
> change_helpfiles(TEST1,TEST2,["November 5",""],["January 5",""]);
  ###<<<Intended error message
Error, (in code_support:-replace_in_file) first list cannot contain an empty
string
```

Example 8: Procedure **get_TEXT** whose output we won't show reads in a Maple worksheet as a text file. This is needed in order to make replacements in the text of the worksheet.

```
> TEST1:="C:\\Maple10\\P10\\test1\\";
  TEST11:=`C:\\Maple10/P10/test1/`;
> L1:=get_dir(TEST1);
  L11:=get_dir(TEST11);
```

TEST1 := "C:\Maple10\P10\test1\"

TEST11 := *C:\Maple10/P10/test1/*

L1 := ["all_sigs_M10.mws", "all_sigs_M10_copy1.mws", "examples723.mws", "Groebner.for.Grassmann_M10.23vii07.mws", "SchurFkt-code-22vii07.mws"]

L11 := ["all_sigs_M10.mws", "all_sigs_M10_copy1.mws", "examples723.mws", "Groebner.for.Grassmann_M10.23vii07.mws", "SchurFkt-code-22vii07.mws"]

```
> filename1:=cat(TEST1,L1[1]);
  filename11:=cat(TEST11,L11[1]);
```

filename1 := "C:\Maple10\P10\test1\all_sigs_M10.mws"

filename11 := *C:\Maple10/P10/test1/all_sigs_M10.mws*

```
> get_TEXT(filename1):
>
```

Example 9: Procedure **makeLIST** reads in file names of Maple worksheets and automatically creates, on the basis of additional input, a list of help topics for ModuleName, with Parent and GrandParent specified ahead of time, from the directory HELP_FILE_PATH. There is a need for an additional input for types and converts which are listed differently in the library. This procedure automatically creates lists of aliases for each help topic. This list can be modified later with the procedure **modifyLIST**.

```
> libname[1];
```

"C:\Maple11/Cliffordlib"

```
> HDB_LIB_PATH:=convert(libname[1],name);
  HELP_FILE_PATH:=`C:\\Maple10/P10/Cliff10/Help/`; ##Directory where
  *_M10.mws help files are located
  BROWSER_PATH:=`Mathematics/Algebra/`;
  ModuleName  :=`Clifford`;
  Parent      :=`Clifford,intro`;
  GrandParent :=`Clifford,intro`;
  MapleVersion:="M10"; ### substring of file names that gives Maple
  version
  typesLIST:=["antisymmatrix","clibasmon","climatrix","climon","clip
  olynom",

  "cliprod","cliscalar","diagmatrix","dfmatrix","evenelement",

  "fieldelement","gencomplex","genquatbasis","genquaternion",

  "idempotent","nilpotent","oddelement","primitiveidemp","purequatba
  sis",
              "quaternion","symmatrix","tensorprod"];
  convertsLIST:=["mlist","str_to_int"];
```

HDB_LIB_PATH := *C:\Maple11/Cliffordlib*

HELP_FILE_PATH := *C:\Maple10/P10/Cliff10/Help/*

BROWSER_PATH := *Mathematics/Algebra/*

ModuleName := *Clifford*

Parent := *Clifford,intro*

GrandParent := *Clifford,intro*

MapleVersion := "M10"

typesLIST := ["antisymmatrix", "clibasmon", "climatrix", "climon", "clipolynom", "cliprod", "cliscalar", "diagmatrix", "dfmatrix", "evenelement", "fieldelement", "gencomplex", "genquatbasis", "genquaternion", "idempotent", "nilpotent", "oddelement", "primitiveidemp", "purequatbasis", "quaternion", "symmatrix", "tensorprod"]

convertsLIST := ["mlist", "str_to_int"]

```
> Lauto:=makeLIST(ModuleName,Parent,GrandParent,HELP_FILE_PATH,Maple
  Version);
```

Lauto := [[*Clifford,&c*, *Clifford,intro*, ["Clifford,&c", "&c"]], [*Clifford,adfmatrix*, *Clifford,intro*, ["Clifford,adfmatrix", "adfmatrix"]], [*Clifford,all_sigs*, *Clifford,intro*, ["Clifford,all_sigs", "all_sigs"]], [*Clifford,type,antisymmatrix*, *Clifford,intro*, ["Clifford,type,antisymmatrix", "type,antisymmatrix"]], [*Clifford,beta_minus*, *Clifford,intro*, ["Clifford,beta_minus", "beta_minus"]],

[*Clifford,beta_plus*, *Clifford,intro*, ["Clifford,beta_plus", "beta_plus"]],
[*Clifford,Bsignature*, *Clifford,intro*, ["Clifford,Bsignature", "Bsignature"]],
[*Clifford,buildm*, *Clifford,intro*, ["Clifford,buildm", "buildm"]],
[*Clifford,bygrade*, *Clifford,intro*, ["Clifford,bygrade", "bygrade"]],
[*Clifford,cbasis*, *Clifford,intro*, ["Clifford,cbasis", "cbasis"]],
[*Clifford,cdfmatrix*, *Clifford,intro*, ["Clifford,cdfmatrix", "cdfmatrix"]],
[*Clifford,cexpQ*, *Clifford,intro*, ["Clifford,cexpQ", "cexpQ"]],
[*Clifford,cexp*, *Clifford,intro*, ["Clifford,cexp", "cexp"]],
[*Clifford,cinv*, *Clifford,intro*, ["Clifford,cinv", "cinv"]],
[*Clifford,type,clibasmon*, *Clifford,intro*, ["Clifford,type,clibasmon", "type,clibasmon"]],
[*Clifford,clibilinear*, *Clifford,intro*, ["Clifford,clibilinear", "clibilinear"]],
[*Clifford,clicollect*, *Clifford,intro*, ["Clifford,clicollect", "clicollect"]],
[*Clifford,clidata*, *Clifford,intro*, ["Clifford,clidata", "clidata"]],
[*Clifford,CLIFFORD_ENV*, *Clifford,intro*, ["Clifford,CLIFFORD_ENV", "CLIFFORD_ENV"]],
[*Clifford,clilinear*, *Clifford,intro*, ["Clifford,clilinear", "clilinear"]],
[*Clifford,type,climatrix*, *Clifford,intro*, ["Clifford,type,climatrix", "type,climatrix"]],
[*Clifford,climinpoly*, *Clifford,intro*, ["Clifford,climinpoly", "climinpoly"]],
[*Clifford,type,climon*, *Clifford,intro*, ["Clifford,type,climon", "type,climon"]],
[*Clifford,cliparse*, *Clifford,intro*, ["Clifford,cliparse", "cliparse"]],
[*Clifford,type,clipolynom*, *Clifford,intro*, ["Clifford,type,clipolynom", "type,clipolynom"]],
[*Clifford,type,cliprod*, *Clifford,intro*, ["Clifford,type,cliprod", "type,cliprod"]],
[*Clifford,cliremove*, *Clifford,intro*, ["Clifford,cliremove", "cliremove"]],
[*Clifford,type,cliscalar*, *Clifford,intro*, ["Clifford,type,cliscalar", "type,cliscalar"]],
[*Clifford,clisolve*, *Clifford,intro*, ["Clifford,clisolve", "clisolve"]],
[*Clifford,clisort*, *Clifford,intro*, ["Clifford,clisort", "clisort"]],
[*Clifford,cliterms*, *Clifford,intro*, ["Clifford,cliterms", "cliterms"]],
[*Clifford,cmulgen*, *Clifford,intro*, ["Clifford,cmulgen", "cmulgen"]],
[*Clifford,cmulNUM*, *Clifford,intro*, ["Clifford,cmulNUM", "cmulNUM"]],
[*Clifford,cmulQ*, *Clifford,intro*, ["Clifford,cmulQ", "cmulQ"]],
[*Clifford,cmulRS*, *Clifford,intro*, ["Clifford,cmulRS", "cmulRS"]],
[*Clifford,cmul*, *Clifford,intro*, ["Clifford,cmul", "cmul"]], [
Clifford,cmul_user_defined, *Clifford,intro*, ["Clifford,cmul_user_defined", "cmul_user_defined"]
], [*Clifford,cocycle*, *Clifford,intro*, ["Clifford,cocycle", "cocycle"]], [
Clifford,commutingelements, *Clifford,intro*,
["Clifford,commutingelements", "commutingelements"]],
[*Clifford,conjugation*, *Clifford,intro*, ["Clifford,conjugation", "conjugation"]],

[*Clifford,c_conjug*, *Clifford,intro*, ["Clifford,c_conjug", "c_conjug"]],
[*Clifford,ddfmatrix*, *Clifford,intro*, ["Clifford,ddfmatrix", "ddfmatrix"]],
[*Clifford,type,dfmatrix*, *Clifford,intro*, ["Clifford,type,dfmatrix", "type,dfmatrix"]],
[*Clifford,type,diagmatrix*, *Clifford,intro*, ["Clifford,type,diagmatrix", "type,diagmatrix"]],
[*Clifford,diagonalize*, *Clifford,intro*, ["Clifford,diagonalize", "diagonalize"]],
[*Clifford,displayid*, *Clifford,intro*, ["Clifford,displayid", "displayid"]],
[*Clifford,type,evenelement*, *Clifford,intro*, ["Clifford,type,evenelement", "type,evenelement"]],
[*Clifford,extract*, *Clifford,intro*, ["Clifford,extract", "extract"]],
[*Clifford,factoridempotent*, *Clifford,intro*, ["Clifford,factoridempotent", "factoridempotent"]],
[*Clifford,type,fieldelement*, *Clifford,intro*, ["Clifford,type,fieldelement", "type,fieldelement"]],
[*Clifford,find1str*, *Clifford,intro*, ["Clifford,find1str", "find1str"]],
[*Clifford,findbasis*, *Clifford,intro*, ["Clifford,findbasis", "findbasis"]],
[*Clifford,type,gencomplex*, *Clifford,intro*, ["Clifford,type,gencomplex", "type,gencomplex"]],
[*Clifford,type,genquatbasis*, *Clifford,intro*, ["Clifford,type,genquatbasis", "type,genquatbasis"]], [
Clifford,type,genquaternion, *Clifford,intro*, ["Clifford,type,genquaternion", "type,genquaternion"]
], [*Clifford,gradeinv*, *Clifford,intro*, ["Clifford,gradeinv", "gradeinv"]],
[*Clifford,type,idempotent*, *Clifford,intro*, ["Clifford,type,idempotent", "type,idempotent"]],
[*Clifford,intro*, *Clifford,intro*, ["Clifford,intro", "intro"]],
[*Clifford,isproduct*, *Clifford,intro*, ["Clifford,isproduct", "isproduct"]],
[*Clifford,isVahlenmatrix*, *Clifford,intro*, ["Clifford,isVahlenmatrix", "isVahlenmatrix"]],
[*Clifford,Kfield*, *Clifford,intro*, ["Clifford,Kfield", "Kfield"]],
[*Clifford,LCQ*, *Clifford,intro*, ["Clifford,LCQ", "LCQ"]],
[*Clifford,LC*, *Clifford,intro*, ["Clifford,LC", "LC"]],
[*Clifford,makealiases*, *Clifford,intro*, ["Clifford,makealiases", "makealiases"]],
[*Clifford,makeclibasmon*, *Clifford,intro*, ["Clifford,makeclibasmon", "makeclibasmon"]],
[*Clifford,matKrepr*, *Clifford,intro*, ["Clifford,matKrepr", "matKrepr"]],
[*Clifford,maxgrade*, *Clifford,intro*, ["Clifford,maxgrade", "maxgrade"]],
[*Clifford,maxindex*, *Clifford,intro*, ["Clifford,maxindex", "maxindex"]],
[*Clifford,mdfmatrix*, *Clifford,intro*, ["Clifford,mdfmatrix", "mdfmatrix"]],
[*Clifford,minimalideal*, *Clifford,intro*, ["Clifford,minimalideal", "minimalideal"]],
[*Clifford,convert,mlist*, *Clifford,intro*, ["Clifford,convert,mlist", "convert,mlist"]],
[*Clifford,type,nilpotent*, *Clifford,intro*, ["Clifford,type,nilpotent", "type,nilpotent"]],
[*Clifford,type,oddelement*, *Clifford,intro*, ["Clifford,type,oddelement", "type,oddelement"]],
[*Clifford,ord*, *Clifford,intro*, ["Clifford,ord", "ord"]],
[*Clifford,permsign*, *Clifford,intro*, ["Clifford,permsign", "permsign"]], [
Clifford,type,primitiveidemp, *Clifford,intro*,

["Clifford,type,primitiveidemp", "type,primitiveidemp"]],
[*Clifford,pseudodet*, *Clifford,intro*, ["Clifford,pseudodet", "pseudodet"]], [
Clifford,type,purequatbasis, *Clifford,intro*, ["Clifford,type,purequatbasis", "type,purequatbasis"]],
[*Clifford,qdisplay*, *Clifford,intro*, ["Clifford,qdisplay", "qdisplay"]],
[*Clifford,qinv*, *Clifford,intro*, ["Clifford,qinv", "qinv"]],
[*Clifford,qmul*, *Clifford,intro*, ["Clifford,qmul", "qmul"]],
[*Clifford,qnorm*, *Clifford,intro*, ["Clifford,qnorm", "qnorm"]],
[*Clifford,type,quaternion*, *Clifford,intro*, ["Clifford,type,quaternion", "type,quaternion"]],
[*Clifford,q_conjug*, *Clifford,intro*, ["Clifford,q_conjug", "q_conjug"]],
[*Clifford,RCQ*, *Clifford,intro*, ["Clifford,RCQ", "RCQ"]],
[*Clifford,RC*, *Clifford,intro*, ["Clifford,RC", "RC"]],
[*Clifford,rd_clibasmon*, *Clifford,intro*, ["Clifford,rd_clibasmon", "rd_clibasmon"]],
[*Clifford,rd_climon*, *Clifford,intro*, ["Clifford,rd_climon", "rd_climon"]],
[*Clifford,rd_clipolynom*, *Clifford,intro*, ["Clifford,rd_clipolynom", "rd_clipolynom"]],
[*Clifford,reorder*, *Clifford,intro*, ["Clifford,reorder", "reorder"]],
[*Clifford,reversion*, *Clifford,intro*, ["Clifford,reversion", "reversion"]],
[*Clifford,RHnumber*, *Clifford,intro*, ["Clifford,RHnumber", "RHnumber"]],
[*Clifford,rmulm*, *Clifford,intro*, ["Clifford,rmulm", "rmulm"]],
[*Clifford,rot3d*, *Clifford,intro*, ["Clifford,rot3d", "rot3d"]],
[*Clifford,scalarpart*, *Clifford,intro*, ["Clifford,scalarpart", "scalarpart"]],
[*Clifford,setup*, *Clifford,intro*, ["Clifford,setup", "setup"]],
[*Clifford,sexp*, *Clifford,intro*, ["Clifford,sexp", "sexp"]],
[*Clifford,specify_constants*, *Clifford,intro*, ["Clifford,specify_constants", "specify_constants"]],
[*Clifford,spinorKbasis*, *Clifford,intro*, ["Clifford,spinorKbasis", "spinorKbasis"]],
[*Clifford,spinorKrepr*, *Clifford,intro*, ["Clifford,spinorKrepr", "spinorKrepr"]],
[*Clifford,squaremodf*, *Clifford,intro*, ["Clifford,squaremodf", "squaremodf"]],
[*Clifford,convert,str_to_int*, *Clifford,intro*, ["Clifford,convert,str_to_int", "convert,str_to_int"]],
[*Clifford,subs_clipolynom*, *Clifford,intro*, ["Clifford,subs_clipolynom", "subs_clipolynom"]],
[*Clifford,type,symmatrix*, *Clifford,intro*, ["Clifford,type,symmatrix", "type,symmatrix"]],
[*Clifford,type,tensorprod*, *Clifford,intro*, ["Clifford,type,tensorprod", "type,tensorprod"]],
[*Clifford,useproduct*, *Clifford,intro*, ["Clifford,useproduct", "useproduct"]],
[*Clifford,vectorpart*, *Clifford,intro*, ["Clifford,vectorpart", "vectorpart"]],
[*Clifford,version*, *Clifford,intro*, ["Clifford,version", "version"]],
[*Clifford,wedge*, *Clifford,intro*, ["Clifford,wedge", "wedge"]],
[*Clifford,wexp*, *Clifford,intro*, ["Clifford,wexp", "wexp"]]]

>

Example 10: Procedure **modifyLIST** can be used to modify a list of help topic entries that has been created automatically by the procedure **makeLIST**. Desired changes need to be entered as two lists of strings: the first list contains strings that need to be replaced while the second lists contains the replacement strings. The original list remains unchanged. This procedure needs to be used separately for each help topic that we want to modify. For example, in the list Lauto we have the following entry:

[*Clifford,wedge*, *Clifford,intro*, ["Clifford,wedge", "wedge"]]

which we would like to modify this entry by adding an additional alias `&w` to the list of aliases:

```
> Lmodified:=modifyLIST(Lauto,`Clifford,wedge`,["Clifford,wedge","we
  dge","`&w`"]);
```

Lmodified := [[*Clifford,&c*, *Clifford,intro*, ["Clifford,&c", "&c"]],
[*Clifford,adfmatrix*, *Clifford,intro*, ["Clifford,adfmatrix", "adfmatrix"]],
[*Clifford,all_sigs*, *Clifford,intro*, ["Clifford,all_sigs", "all_sigs"]], [
Clifford,type,antisymmatrix, *Clifford,intro*, ["Clifford,type,antisymmatrix", "type,antisymmatrix"]
], [*Clifford,beta_minus*, *Clifford,intro*, ["Clifford,beta_minus", "beta_minus"]],
[*Clifford,beta_plus*, *Clifford,intro*, ["Clifford,beta_plus", "beta_plus"]],
[*Clifford,Bsignature*, *Clifford,intro*, ["Clifford,Bsignature", "Bsignature"]],
[*Clifford,buildm*, *Clifford,intro*, ["Clifford,buildm", "buildm"]],
[*Clifford,bygrade*, *Clifford,intro*, ["Clifford,bygrade", "bygrade"]],
[*Clifford,cbasis*, *Clifford,intro*, ["Clifford,cbasis", "cbasis"]],
[*Clifford,cdfmatrix*, *Clifford,intro*, ["Clifford,cdfmatrix", "cdfmatrix"]],
[*Clifford,cexpQ*, *Clifford,intro*, ["Clifford,cexpQ", "cexpQ"]],
[*Clifford,cexp*, *Clifford,intro*, ["Clifford,cexp", "cexp"]],
[*Clifford,cinv*, *Clifford,intro*, ["Clifford,cinv", "cinv"]],
[*Clifford,type,clibasmon*, *Clifford,intro*, ["Clifford,type,clibasmon", "type,clibasmon"]],
[*Clifford,clibilinear*, *Clifford,intro*, ["Clifford,clibilinear", "clibilinear"]],
[*Clifford,clicollect*, *Clifford,intro*, ["Clifford,clicollect", "clicollect"]],
[*Clifford,clidata*, *Clifford,intro*, ["Clifford,clidata", "clidata"]],
[*Clifford,CLIFFORD_ENV*, *Clifford,intro*, ["Clifford,CLIFFORD_ENV", "CLIFFORD_ENV"]],
[*Clifford,clilinear*, *Clifford,intro*, ["Clifford,clilinear", "clilinear"]],
[*Clifford,type,climatrix*, *Clifford,intro*, ["Clifford,type,climatrix", "type,climatrix"]],
[*Clifford,climinpoly*, *Clifford,intro*, ["Clifford,climinpoly", "climinpoly"]],
[*Clifford,type,climon*, *Clifford,intro*, ["Clifford,type,climon", "type,climon"]],
[*Clifford,cliparse*, *Clifford,intro*, ["Clifford,cliparse", "cliparse"]],
[*Clifford,type,clipolynom*, *Clifford,intro*, ["Clifford,type,clipolynom", "type,clipolynom"]],

[*Clifford,type,cliprod*, *Clifford,intro*, ["Clifford,type,cliprod", "type,cliprod"]],
[*Clifford,cliremove*, *Clifford,intro*, ["Clifford,cliremove", "cliremove"]],
[*Clifford,type,cliscalar*, *Clifford,intro*, ["Clifford,type,cliscalar", "type,cliscalar"]],
[*Clifford,clisolve*, *Clifford,intro*, ["Clifford,clisolve", "clisolve"]],
[*Clifford,clisort*, *Clifford,intro*, ["Clifford,clisort", "clisort"]],
[*Clifford,cliterms*, *Clifford,intro*, ["Clifford,cliterms", "cliterms"]],
[*Clifford,cmulgen*, *Clifford,intro*, ["Clifford,cmulgen", "cmulgen"]],
[*Clifford,cmulNUM*, *Clifford,intro*, ["Clifford,cmulNUM", "cmulNUM"]],
[*Clifford,cmulQ*, *Clifford,intro*, ["Clifford,cmulQ", "cmulQ"]],
[*Clifford,cmulRS*, *Clifford,intro*, ["Clifford,cmulRS", "cmulRS"]],
[*Clifford,cmul*, *Clifford,intro*, ["Clifford,cmul", "cmul"]], [
Clifford,cmul_user_defined, *Clifford,intro*, ["Clifford,cmul_user_defined", "cmul_user_defined"]
], [*Clifford,cocycle*, *Clifford,intro*, ["Clifford,cocycle", "cocycle"]], [
Clifford,commutingelements, *Clifford,intro*,
["Clifford,commutingelements", "commutingelements"]],
[*Clifford,conjugation*, *Clifford,intro*, ["Clifford,conjugation", "conjugation"]],
[*Clifford,c_conjug*, *Clifford,intro*, ["Clifford,c_conjug", "c_conjug"]],
[*Clifford,ddfmatrix*, *Clifford,intro*, ["Clifford,ddfmatrix", "ddfmatrix"]],
[*Clifford,type,dfmatrix*, *Clifford,intro*, ["Clifford,type,dfmatrix", "type,dfmatrix"]],
[*Clifford,type,diagmatrix*, *Clifford,intro*, ["Clifford,type,diagmatrix", "type,diagmatrix"]],
[*Clifford,diagonalize*, *Clifford,intro*, ["Clifford,diagonalize", "diagonalize"]],
[*Clifford,displayid*, *Clifford,intro*, ["Clifford,displayid", "displayid"]],
[*Clifford,type,evenelement*, *Clifford,intro*, ["Clifford,type,evenelement", "type,evenelement"]],
[*Clifford,extract*, *Clifford,intro*, ["Clifford,extract", "extract"]],
[*Clifford,factoridempotent*, *Clifford,intro*, ["Clifford,factoridempotent", "factoridempotent"]],
[*Clifford,type,fieldelement*, *Clifford,intro*, ["Clifford,type,fieldelement", "type,fieldelement"]],
[*Clifford,find1str*, *Clifford,intro*, ["Clifford,find1str", "find1str"]],
[*Clifford,findbasis*, *Clifford,intro*, ["Clifford,findbasis", "findbasis"]],
[*Clifford,type,gencomplex*, *Clifford,intro*, ["Clifford,type,gencomplex", "type,gencomplex"]],
[*Clifford,type,genquatbasis*, *Clifford,intro*, ["Clifford,type,genquatbasis", "type,genquatbasis"]], [
Clifford,type,genquaternion, *Clifford,intro*, ["Clifford,type,genquaternion", "type,genquaternion"]
], [*Clifford,gradeinv*, *Clifford,intro*, ["Clifford,gradeinv", "gradeinv"]],
[*Clifford,type,idempotent*, *Clifford,intro*, ["Clifford,type,idempotent", "type,idempotent"]],
[*Clifford,intro*, *Clifford,intro*, ["Clifford,intro", "intro"]],
[*Clifford,isproduct*, *Clifford,intro*, ["Clifford,isproduct", "isproduct"]],
[*Clifford,isVahlenmatrix*, *Clifford,intro*, ["Clifford,isVahlenmatrix", "isVahlenmatrix"]],

[*Clifford,Kfield*, *Clifford,intro*, ["Clifford,Kfield", "Kfield"]],
[*Clifford,LCQ*, *Clifford,intro*, ["Clifford,LCQ", "LCQ"]],
[*Clifford,LC*, *Clifford,intro*, ["Clifford,LC", "LC"]],
[*Clifford,makealiases*, *Clifford,intro*, ["Clifford,makealiases", "makealiases"]],
[*Clifford,makeclibasmon*, *Clifford,intro*, ["Clifford,makeclibasmon", "makeclibasmon"]],
[*Clifford,matKrepr*, *Clifford,intro*, ["Clifford,matKrepr", "matKrepr"]],
[*Clifford,maxgrade*, *Clifford,intro*, ["Clifford,maxgrade", "maxgrade"]],
[*Clifford,maxindex*, *Clifford,intro*, ["Clifford,maxindex", "maxindex"]],
[*Clifford,mdfmatrix*, *Clifford,intro*, ["Clifford,mdfmatrix", "mdfmatrix"]],
[*Clifford,minimalideal*, *Clifford,intro*, ["Clifford,minimalideal", "minimalideal"]],
[*Clifford,convert,mlist*, *Clifford,intro*, ["Clifford,convert,mlist", "convert,mlist"]],
[*Clifford,type,nilpotent*, *Clifford,intro*, ["Clifford,type,nilpotent", "type,nilpotent"]],
[*Clifford,type,oddelement*, *Clifford,intro*, ["Clifford,type,oddelement", "type,oddelement"]],
[*Clifford,ord*, *Clifford,intro*, ["Clifford,ord", "ord"]],
[*Clifford,permsign*, *Clifford,intro*, ["Clifford,permsign", "permsign"]], [
Clifford,type,primitiveidemp, *Clifford,intro*,
["Clifford,type,primitiveidemp", "type,primitiveidemp"]],
[*Clifford,pseudodet*, *Clifford,intro*, ["Clifford,pseudodet", "pseudodet"]], [
Clifford,type,purequatbasis, *Clifford,intro*, ["Clifford,type,purequatbasis", "type,purequatbasis"]],
[*Clifford,qdisplay*, *Clifford,intro*, ["Clifford,qdisplay", "qdisplay"]],
[*Clifford,qinv*, *Clifford,intro*, ["Clifford,qinv", "qinv"]],
[*Clifford,qmul*, *Clifford,intro*, ["Clifford,qmul", "qmul"]],
[*Clifford,qnorm*, *Clifford,intro*, ["Clifford,qnorm", "qnorm"]],
[*Clifford,type,quaternion*, *Clifford,intro*, ["Clifford,type,quaternion", "type,quaternion"]],
[*Clifford,q_conjug*, *Clifford,intro*, ["Clifford,q_conjug", "q_conjug"]],
[*Clifford,RCQ*, *Clifford,intro*, ["Clifford,RCQ", "RCQ"]],
[*Clifford,RC*, *Clifford,intro*, ["Clifford,RC", "RC"]],
[*Clifford,rd_clibasmon*, *Clifford,intro*, ["Clifford,rd_clibasmon", "rd_clibasmon"]],
[*Clifford,rd_climon*, *Clifford,intro*, ["Clifford,rd_climon", "rd_climon"]],
[*Clifford,rd_clipolynom*, *Clifford,intro*, ["Clifford,rd_clipolynom", "rd_clipolynom"]],
[*Clifford,reorder*, *Clifford,intro*, ["Clifford,reorder", "reorder"]],
[*Clifford,reversion*, *Clifford,intro*, ["Clifford,reversion", "reversion"]],
[*Clifford,RHnumber*, *Clifford,intro*, ["Clifford,RHnumber", "RHnumber"]],
[*Clifford,rmulm*, *Clifford,intro*, ["Clifford,rmulm", "rmulm"]],
[*Clifford,rot3d*, *Clifford,intro*, ["Clifford,rot3d", "rot3d"]],
[*Clifford,scalarpart*, *Clifford,intro*, ["Clifford,scalarpart", "scalarpart"]],

[*Clifford,setup*, *Clifford,intro*, ["Clifford,setup", "setup"]],
[*Clifford,sexp*, *Clifford,intro*, ["Clifford,sexp", "sexp"]],
[*Clifford,specify_constants*, *Clifford,intro*, ["Clifford,specify_constants", "specify_constants"]],
[*Clifford,spinorKbasis*, *Clifford,intro*, ["Clifford,spinorKbasis", "spinorKbasis"]],
[*Clifford,spinorKrepr*, *Clifford,intro*, ["Clifford,spinorKrepr", "spinorKrepr"]],
[*Clifford,squaremodf*, *Clifford,intro*, ["Clifford,squaremodf", "squaremodf"]],
[*Clifford,convert,str_to_int*, *Clifford,intro*, ["Clifford,convert,str_to_int", "convert,str_to_int"]],
[*Clifford,subs_clipolynom*, *Clifford,intro*, ["Clifford,subs_clipolynom", "subs_clipolynom"]],
[*Clifford,type,symmatrix*, *Clifford,intro*, ["Clifford,type,symmatrix", "type,symmatrix"]],
[*Clifford,type,tensorprod*, *Clifford,intro*, ["Clifford,type,tensorprod", "type,tensorprod"]],
[*Clifford,useproduct*, *Clifford,intro*, ["Clifford,useproduct", "useproduct"]],
[*Clifford,vectorpart*, *Clifford,intro*, ["Clifford,vectorpart", "vectorpart"]],
[*Clifford,version*, *Clifford,intro*, ["Clifford,version", "version"]],
[*Clifford,wedge*, *Clifford,intro*, ["Clifford,wedge", "wedge", "`&w`"]],
[*Clifford,wexp*, *Clifford,intro*, ["Clifford,wexp", "wexp"]]]

Notice that this entry now looks as follows:

[*Clifford,wedge*, *Clifford,intro*, ["Clifford,wedge", "wedge", "`&w`"]]

that is, the alias `&w` has been added. To modify another topic, we need to execute this procedure once more. For example, the last topic has this entry:

[*Clifford,wexp*, *Clifford,intro*, ["Clifford,wexp", "wexp"]]

If wanted to modify it by removing, for example, the alias "wexp", we would need to do the following:

```
> Lmodified2:=modifyLIST(Lmodified,`Clifford,wexp`,["Clifford,wexp"]
  );
```

Lmodified2 := [[*Clifford,&c*, *Clifford,intro*, ["Clifford,&c", "&c"]],
[*Clifford,adfmatrix*, *Clifford,intro*, ["Clifford,adfmatrix", "adfmatrix"]],
[*Clifford,all_sigs*, *Clifford,intro*, ["Clifford,all_sigs", "all_sigs"]], [
Clifford,type,antisymmatrix, *Clifford,intro*, ["Clifford,type,antisymmatrix", "type,antisymmatrix"]
], [*Clifford,beta_minus*, *Clifford,intro*, ["Clifford,beta_minus", "beta_minus"]],
[*Clifford,beta_plus*, *Clifford,intro*, ["Clifford,beta_plus", "beta_plus"]],
[*Clifford,Bsignature*, *Clifford,intro*, ["Clifford,Bsignature", "Bsignature"]],
[*Clifford,buildm*, *Clifford,intro*, ["Clifford,buildm", "buildm"]],
[*Clifford,bygrade*, *Clifford,intro*, ["Clifford,bygrade", "bygrade"]],

[*Clifford,cbasis*, *Clifford,intro*, ["Clifford,cbasis", "cbasis"]],
[*Clifford,cdfmatrix*, *Clifford,intro*, ["Clifford,cdfmatrix", "cdfmatrix"]],
[*Clifford,cexpQ*, *Clifford,intro*, ["Clifford,cexpQ", "cexpQ"]],
[*Clifford,cexp*, *Clifford,intro*, ["Clifford,cexp", "cexp"]],
[*Clifford,cinv*, *Clifford,intro*, ["Clifford,cinv", "cinv"]],
[*Clifford,type,clibasmon*, *Clifford,intro*, ["Clifford,type,clibasmon", "type,clibasmon"]],
[*Clifford,clibilinear*, *Clifford,intro*, ["Clifford,clibilinear", "clibilinear"]],
[*Clifford,clicollect*, *Clifford,intro*, ["Clifford,clicollect", "clicollect"]],
[*Clifford,clidata*, *Clifford,intro*, ["Clifford,clidata", "clidata"]],
[*Clifford,CLIFFORD_ENV*, *Clifford,intro*, ["Clifford,CLIFFORD_ENV", "CLIFFORD_ENV"]],
[*Clifford,clilinear*, *Clifford,intro*, ["Clifford,clilinear", "clilinear"]],
[*Clifford,type,climatrix*, *Clifford,intro*, ["Clifford,type,climatrix", "type,climatrix"]],
[*Clifford,climinpoly*, *Clifford,intro*, ["Clifford,climinpoly", "climinpoly"]],
[*Clifford,type,climon*, *Clifford,intro*, ["Clifford,type,climon", "type,climon"]],
[*Clifford,cliparse*, *Clifford,intro*, ["Clifford,cliparse", "cliparse"]],
[*Clifford,type,clipolynom*, *Clifford,intro*, ["Clifford,type,clipolynom", "type,clipolynom"]],
[*Clifford,type,cliprod*, *Clifford,intro*, ["Clifford,type,cliprod", "type,cliprod"]],
[*Clifford,cliremove*, *Clifford,intro*, ["Clifford,cliremove", "cliremove"]],
[*Clifford,type,cliscalar*, *Clifford,intro*, ["Clifford,type,cliscalar", "type,cliscalar"]],
[*Clifford,clisolve*, *Clifford,intro*, ["Clifford,clisolve", "clisolve"]],
[*Clifford,clisort*, *Clifford,intro*, ["Clifford,clisort", "clisort"]],
[*Clifford,cliterms*, *Clifford,intro*, ["Clifford,cliterms", "cliterms"]],
[*Clifford,cmulgen*, *Clifford,intro*, ["Clifford,cmulgen", "cmulgen"]],
[*Clifford,cmulNUM*, *Clifford,intro*, ["Clifford,cmulNUM", "cmulNUM"]],
[*Clifford,cmulQ*, *Clifford,intro*, ["Clifford,cmulQ", "cmulQ"]],
[*Clifford,cmulRS*, *Clifford,intro*, ["Clifford,cmulRS", "cmulRS"]],
[*Clifford,cmul*, *Clifford,intro*, ["Clifford,cmul", "cmul"]], [
Clifford,cmul_user_defined, *Clifford,intro*, ["Clifford,cmul_user_defined", "cmul_user_defined"]
], [*Clifford,cocycle*, *Clifford,intro*, ["Clifford,cocycle", "cocycle"]], [
Clifford,commutingelements, *Clifford,intro*,
["Clifford,commutingelements", "commutingelements"]],
[*Clifford,conjugation*, *Clifford,intro*, ["Clifford,conjugation", "conjugation"]],
[*Clifford,c_conjug*, *Clifford,intro*, ["Clifford,c_conjug", "c_conjug"]],
[*Clifford,ddfmatrix*, *Clifford,intro*, ["Clifford,ddfmatrix", "ddfmatrix"]],
[*Clifford,type,dfmatrix*, *Clifford,intro*, ["Clifford,type,dfmatrix", "type,dfmatrix"]],
[*Clifford,type,diagmatrix*, *Clifford,intro*, ["Clifford,type,diagmatrix", "type,diagmatrix"]],

[*Clifford,diagonalize*, *Clifford,intro*, ["Clifford,diagonalize", "diagonalize"]],
[*Clifford,displayid*, *Clifford,intro*, ["Clifford,displayid", "displayid"]],
[*Clifford,type,evenelement*, *Clifford,intro*, ["Clifford,type,evenelement", "type,evenelement"]],
[*Clifford,extract*, *Clifford,intro*, ["Clifford,extract", "extract"]],
[*Clifford,factoridempotent*, *Clifford,intro*, ["Clifford,factoridempotent", "factoridempotent"]],
[*Clifford,type,fieldelement*, *Clifford,intro*, ["Clifford,type,fieldelement", "type,fieldelement"]],
[*Clifford,find1str*, *Clifford,intro*, ["Clifford,find1str", "find1str"]],
[*Clifford,findbasis*, *Clifford,intro*, ["Clifford,findbasis", "findbasis"]],
[*Clifford,type,gencomplex*, *Clifford,intro*, ["Clifford,type,gencomplex", "type,gencomplex"]],
[*Clifford,type,genquatbasis*, *Clifford,intro*, ["Clifford,type,genquatbasis", "type,genquatbasis"]], [
Clifford,type,genquaternion, *Clifford,intro*, ["Clifford,type,genquaternion", "type,genquaternion"]
], [*Clifford,gradeinv*, *Clifford,intro*, ["Clifford,gradeinv", "gradeinv"]],
[*Clifford,type,idempotent*, *Clifford,intro*, ["Clifford,type,idempotent", "type,idempotent"]],
[*Clifford,intro*, *Clifford,intro*, ["Clifford,intro", "intro"]],
[*Clifford,isproduct*, *Clifford,intro*, ["Clifford,isproduct", "isproduct"]],
[*Clifford,isVahlenmatrix*, *Clifford,intro*, ["Clifford,isVahlenmatrix", "isVahlenmatrix"]],
[*Clifford,Kfield*, *Clifford,intro*, ["Clifford,Kfield", "Kfield"]],
[*Clifford,LCQ*, *Clifford,intro*, ["Clifford,LCQ", "LCQ"]],
[*Clifford,LC*, *Clifford,intro*, ["Clifford,LC", "LC"]],
[*Clifford,makealiases*, *Clifford,intro*, ["Clifford,makealiases", "makealiases"]],
[*Clifford,makeclibasmon*, *Clifford,intro*, ["Clifford,makeclibasmon", "makeclibasmon"]],
[*Clifford,matKrepr*, *Clifford,intro*, ["Clifford,matKrepr", "matKrepr"]],
[*Clifford,maxgrade*, *Clifford,intro*, ["Clifford,maxgrade", "maxgrade"]],
[*Clifford,maxindex*, *Clifford,intro*, ["Clifford,maxindex", "maxindex"]],
[*Clifford,mdfmatrix*, *Clifford,intro*, ["Clifford,mdfmatrix", "mdfmatrix"]],
[*Clifford,minimalideal*, *Clifford,intro*, ["Clifford,minimalideal", "minimalideal"]],
[*Clifford,convert,mlist*, *Clifford,intro*, ["Clifford,convert,mlist", "convert,mlist"]],
[*Clifford,type,nilpotent*, *Clifford,intro*, ["Clifford,type,nilpotent", "type,nilpotent"]],
[*Clifford,type,oddelement*, *Clifford,intro*, ["Clifford,type,oddelement", "type,oddelement"]],
[*Clifford,ord*, *Clifford,intro*, ["Clifford,ord", "ord"]],
[*Clifford,permsign*, *Clifford,intro*, ["Clifford,permsign", "permsign"]], [
Clifford,type,primitiveidemp, *Clifford,intro*,
["Clifford,type,primitiveidemp", "type,primitiveidemp"]],
[*Clifford,pseudodet*, *Clifford,intro*, ["Clifford,pseudodet", "pseudodet"]], [
Clifford,type,purequatbasis, *Clifford,intro*, ["Clifford,type,purequatbasis", "type,purequatbasis"]],
[*Clifford,qdisplay*, *Clifford,intro*, ["Clifford,qdisplay", "qdisplay"]],

[*Clifford,qinv*, *Clifford,intro*, ["Clifford,qinv", "qinv"]],
[*Clifford,qmul*, *Clifford,intro*, ["Clifford,qmul", "qmul"]],
[*Clifford,qnorm*, *Clifford,intro*, ["Clifford,qnorm", "qnorm"]],
[*Clifford,type,quaternion*, *Clifford,intro*, ["Clifford,type,quaternion", "type,quaternion"]],
[*Clifford,q_conjug*, *Clifford,intro*, ["Clifford,q_conjug", "q_conjug"]],
[*Clifford,RCQ*, *Clifford,intro*, ["Clifford,RCQ", "RCQ"]],
[*Clifford,RC*, *Clifford,intro*, ["Clifford,RC", "RC"]],
[*Clifford,rd_clibasmon*, *Clifford,intro*, ["Clifford,rd_clibasmon", "rd_clibasmon"]],
[*Clifford,rd_climon*, *Clifford,intro*, ["Clifford,rd_climon", "rd_climon"]],
[*Clifford,rd_clipolynom*, *Clifford,intro*, ["Clifford,rd_clipolynom", "rd_clipolynom"]],
[*Clifford,reorder*, *Clifford,intro*, ["Clifford,reorder", "reorder"]],
[*Clifford,reversion*, *Clifford,intro*, ["Clifford,reversion", "reversion"]],
[*Clifford,RHnumber*, *Clifford,intro*, ["Clifford,RHnumber", "RHnumber"]],
[*Clifford,rmulm*, *Clifford,intro*, ["Clifford,rmulm", "rmulm"]],
[*Clifford,rot3d*, *Clifford,intro*, ["Clifford,rot3d", "rot3d"]],
[*Clifford,scalarpart*, *Clifford,intro*, ["Clifford,scalarpart", "scalarpart"]],
[*Clifford,setup*, *Clifford,intro*, ["Clifford,setup", "setup"]],
[*Clifford,sexp*, *Clifford,intro*, ["Clifford,sexp", "sexp"]],
[*Clifford,specify_constants*, *Clifford,intro*, ["Clifford,specify_constants", "specify_constants"]],
[*Clifford,spinorKbasis*, *Clifford,intro*, ["Clifford,spinorKbasis", "spinorKbasis"]],
[*Clifford,spinorKrepr*, *Clifford,intro*, ["Clifford,spinorKrepr", "spinorKrepr"]],
[*Clifford,squaremodf*, *Clifford,intro*, ["Clifford,squaremodf", "squaremodf"]],
[*Clifford,convert,str_to_int*, *Clifford,intro*, ["Clifford,convert,str_to_int", "convert,str_to_int"]],
[*Clifford,subs_clipolynom*, *Clifford,intro*, ["Clifford,subs_clipolynom", "subs_clipolynom"]],
[*Clifford,type,symmatrix*, *Clifford,intro*, ["Clifford,type,symmatrix", "type,symmatrix"]],
[*Clifford,type,tensorprod*, *Clifford,intro*, ["Clifford,type,tensorprod", "type,tensorprod"]],
[*Clifford,useproduct*, *Clifford,intro*, ["Clifford,useproduct", "useproduct"]],
[*Clifford,vectorpart*, *Clifford,intro*, ["Clifford,vectorpart", "vectorpart"]],
[*Clifford,version*, *Clifford,intro*, ["Clifford,version", "version"]],
[*Clifford,wedge*, *Clifford,intro*, ["Clifford,wedge", "wedge", "`&w`"]],
[*Clifford,wexp*, *Clifford,intro*, ["Clifford,wexp"]]]

Observe that the last entry now reads:

[*Clifford,wexp*, *Clifford,intro*, ["Clifford,wexp"]]

>

Example 11:

Procedure **NamesInLibrary** shows what programs have been added to the library specified as the single argument. For example, we can see what programs or files have been added so far do the library. For a complete information, see **march** command.

```
> libname;
```

"C:\Maple11/Cliffordlib", "C:\Maple11/lib", "C:\Brachey.Troy/TNB", "C:\Maple11/SINGULARPLURALlinklib"

```
> NamesInLibrary(libname[1]);
```

"matrealL.m"
"Clifford.m"
"_AlternatingGroup_rem_table.m"
"_SymmetricGroup_rem_table.m"
"Define.m"
"SINGULARPLURALlink.m"
"GTP.m"
"_implicitbezierpolynomial.m"
"Bigebra.m"
"matquatL.m"
"_Reynolds_rem_table.m"
"Cliplus.m"
"SchurFkt.m"
"_FiniteGroups_rem_table.m"
"_generateGinvariants_rem_table.m"
"SP.m"
"matcompR.m"
"TNB.m"
"code_support.m"
"matrealR.m"
"RJgrobner.m"
"Octonion.m"
"matcompL.m"
"GfG.m"
"matquatR.m"

```
>
```

Example 12:

To see examples how **insert_helppages** works, see the help page INSERT_HELPPAGES.

See Also: code_support, code_support, code_support, INSERT_HELPPAGES

Last modified: June 19, 2008/RA/BF.

```
> restart:
  with(GfG);
```

```
GfG - Groebner for Grassmann 0.6 beta (June 22, 2008) says Hello...
===>WARNING: This is purely experimental package at this time, i.e., it is likely
to produce wrong results.
===>If you find this packages useful, please let us know about your derived work.
===>You can contact us at http://math.tntech.edu/rafal/ or http://clifford.physik.
uni-konstanz.de/~fauser/
Clifford package with 84 functions loaded...
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help
]
Bigebra package with 33 functions loaded...
```

[*CanForm*, *CommonFactor*, *Deg*, *GDivide*, *GGbasis*, *GLCM*, *GReduce*, *GSpoly*, *InvDeg*, *InvLex*, *InvRevLex*, *LCoeff*, *LMon*, *LTerm*, *LeftGDivide*, *LeftGSpoly*, *LeftNormalForm*, *Lex*, *OrderIdealO*, *RevLex*, *RightGDivide*, *RightGSpoly*, *RightNormalForm*, *SemiGroupIdealT*, *SemiGroupS*, *chooseg1*, *chooseg2*, *isadmissible*, *leftidealmember*, *leftmost*, *make_rnd_list*, *make_rnd_polynomial_in_ideal*, *makealpha*, *minGGbasis*, *multideg*, *rightidealmember*, *rightmost*, *tilde*, *version*]

```
> with(linalg):
  B:=diag(0$9):
> eval(makealiases(9,'ordered')):
> version();
```

```
                    ++++++++++++++++++++++++++++++++++++++++++++++
```

GfG - Groebner for Grassmann - A Maple 11 Package for Groebner Bases for Grassmann Algebr\ as

Last revised: June 22, 2008 (Source file: Groebner.for.Grassmann_22vi08_M11.mws)

with contributions from Troy Brachey (*)

(*) Department of Mathematics, Box 5054
Tennessee Technological University, Cookeville, TN 38505
tel: USA (931) 372-3662, fax: USA (931) 372-6353
rablamowicz@tntech.edu
http://math.tntech.edu/rafal/

($) Universit"at Konstanz, Fachbereich Physik, Fach M678
78457 Konstanz, Germany
Bertfried.Fauser@uni-konstanz.de
http://clifford.physik.uni-konstanz.de/~fauser/
http://math.tntech.edu/rafal/

++++++++++++This is GfG - Groebner for Grassmann for Maple 11 version 0.6+++++++++
++++

```
>
```

Example 1:

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

$$cbas := [Id, e1, e2, e3, e12, e13, e23, e123]$$

```
> map(makealpha,cbas);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now inclu
de &C and &C[K]. Type ?cliprod for help.
```

$$[[0, 0, 0, 0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0, 0, 0], [1, 1, 0, 0, 0, 0, 0, 0, 0], [1, 0, 1, 0, 0, 0, 0, 0, 0], [0, 1, 1, 0, 0, 0, 0, 0, 0], [1, 1, 1, 0, 0, 0, 0, 0, 0]]$$

```
>
```

Example 2: Grassmann monomials sorted from the greatest to the smallest according to each order.

Definition: We say that an order T on Grassmann monomials is admissible if

1. m > 1 for every monomial m in the Grassmann basis
2. If m_2 > m_1 then m_l m_2 m_r > m_l m_1 m_r for every m_1, m_2, m_l, and m_r in the Grassmann basis

In the following we sort Grassmann monomials in the basis for /\ R^3 according to each monomial order. Notice that:

1. RevLex, InvRevLex, and InvDeg[Lex] orders are NOT admissible because they violate the first condition that m > 1 for every m in the Grassmann basis.

NOTE: Not all of these orders below are admissible, for example, RevLex, InvRevLex, and InvDeg[Lex] orders are NOT admissible because they violate the first condition that m > 1 for every m in the Grassmann basis. The remaining orders are admissible. Check procedure 'isadmissible'

Let a monomial m = e^alpha = (e1^alpha1 &w e2^alpha2) &w (...) &w (e9^alpha9) where alpha = [alpha1, alpha2, ..., alpha9]. Of course, each alphai = 0 or 1 as ei^alphai = &w(ei,ei,...,ei) = 0 if alphai >=2. We also agree that ei^0 = Id in the Grassmann algebra /\. We will refer to the list alpha also as a vector.

In the following let m1 = e^alpha and m2 = e^beta.

'Lex(m1,m2)' returns true if either the difference vector alpha - beta is the zero vector or the leftmost non-zero entry in alpha - beta is positive, in which case we say m1 >= m2. Otherwise it returns false, in which case we say that m1 < m2. For example, it gives a lexicographic order on generators e1>e2>...>e9 (NOT ADMISSIBLE in general but admissible in Grassmann algebra).

'InvLex(m1,m2)' returns true if either the difference vector alpha - beta is the zero vector or the rightmost non-zero entry in alpha - beta is positive, in which case we say m1 >= m2. Otherwise it returns false, in which case we say that m1 < m2. For example, it gives the inverse lexicographic order on generators e9>e8>...>e1 (NOT ADMISSIBLE in general and not admissible in Grassmann algebra)

'RevLex(m1,m2)' returns true if either the difference vector alpha - beta is the zero vector or the rightmost non-zero entry in alpha - beta is negative, in which case we say m1 >= m2. Otherwise it returns false, in which case we say that m1 < m2. For example, it gives the reverse lexicographic order on generators e1>e2>...>e9 (NOT ADMISSIBLE in general and not admissible in Grassmann algebra)

'InvRevLex(m1,m2)' returns true if either the difference vector alpha - beta is the zero vector or the leftmost non-zero entry in alpha - beta is negative, in which case we say m1 >= m2. Otherwise it returns false, in which case we say that m1 < m2. For example, it gives the reverse lexicographic order on generators e1>e2>...>e9 (NOT ADMISSIBLE in general but admissible in Grassmann algebra)

'Deg' procedure takes as index (parameter) any of the above monomial orders T and uses that order to resolve total degree ties: It first computes a degree (grade) of two monomial terms m1, m2, and it returns true if the total degree |m1| > |m2|. When |m1| < |m2| it returns false, and when the degrees are equal |m1| = |m2|, it uses the order T to determine the order. (NOT ADMISSIBLE in general but admissible in Grassmann algebra)

'InvDeg' procedure takes as index (parameter) any of the above monomial orders T and uses that order to resolve total degree ties: It first computes a degree (grade) of two monomial terms m1, m2, and it returns false if the total degree |m1| > |m2|. When |m1| < |m2| it returns true, and when the degrees are equal |m1| = |m2|, it uses the order T to determine the order. (NOT ADMISSIBLE in general but admissible in Grassmann algebra for all orders T except InvDeg[Lex] is not admissible)

```
> cbas:=cbasis(3);
  "Lex order gives"=sort(cbas,Lex);
  "InvLex order gives"=sort(cbas,InvLex);
  "RevLex order gives"=sort(cbas,RevLex);            #Not admissible
  because 1 > m for every m instead of m > 1 for every m
  "InvRevLex order gives"=sort(cbas,InvRevLex);      #Not admissible
  because 1 > m for every m instead of m > 1 for every m
  "Deg[Lex] order gives"=sort(cbas,Deg[Lex]);
  "Deg[InvLex] order gives"=sort(cbas,Deg[InvLex]);
  "InvDeg[Lex] order gives"=sort(cbas,InvDeg[Lex]);  #Not admissible
  because 1 > m for every m instead of m > 1 for every m
```

$$cbas := [Id, e1, e2, e3, e12, e13, e23, e123]$$

$$\text{"Lex order gives"} = [e123, e12, e13, e1, e23, e2, e3, Id]$$

$$\text{"InvLex order gives"} = [e123, e23, e13, e3, e12, e2, e1, Id]$$

$$\text{"RevLex order gives"} = [Id, e1, e2, e12, e3, e13, e23, e123]$$

$$\text{"InvRevLex order gives"} = [Id, e3, e2, e23, e1, e13, e12, e123]$$

$$\text{"Deg[Lex] order gives"} = [e123, e12, e13, e23, e1, e2, e3, Id]$$

$$\text{"Deg[InvLex] order gives"} = [e123, e23, e13, e12, e3, e2, e1, Id]$$

$$\text{"InvDeg[Lex] order gives"} = [Id, e1, e2, e3, e12, e13, e23, e123]$$

```
>
```

Example 2: Procedure 'isadmissible' checks whether the given order is admissible. It takes as input a list of Grassmann monomials (in CLIFFORD, they are of type 'clibasmon', and one of the monomial orders. It returns true of the order as admissible and false if it is not admissible. Note, that this procedure has a remember table to speed up computations but this remember table is forgotten when Maple is closed. A local function in this procedure called 'Fwedge' is essentially the same as Clifford:-wedge except that it has a remember table and remembers products of Grassmann monomials. Its remember table is also erased when Maple is closed.

```
> cbas:=cbasis(4):
  T:=[Lex,InvLex,RevLex,InvRevLex,Deg[Lex],Deg[InvLex],InvDeg[Lex]]:
  nops(T);
```

$$7$$

```
> for morder in T do
  flag:=isadmissible(cbas,morder):
  morder1:=convert(morder,string):
  printf("Monomial order %s is admissible: %s\n",morder1,flag);
  end do;
```

$$flag := true$$

$$morder1 := \text{"Lex"}$$

```
Monomial order Lex is admissible: true
```

$$flag := true$$

$$morder1 := \text{"InvLex"}$$

```
Monomial order InvLex is admissible: true
```

$$flag := false$$

$$morder1 := \text{"RevLex"}$$

```
Monomial order RevLex is admissible: false
```

$$flag := false$$

$$morder1 := \text{"InvRevLex"}$$

```
Monomial order InvRevLex is admissible: false
```

$$flag := true$$

$$morder1 := \text{"GfG:-Deg[GfG:-Lex]"}$$

```
Monomial order GfG:-Deg[GfG:-Lex] is admissible: true
```

$$flag := true$$

$$morder1 := \text{"GfG:-Deg[GfG:-InvLex]"}$$

```
Monomial order GfG:-Deg[GfG:-InvLex] is admissible: true
```

$$flag := false$$

$$morder1 := \text{"GfG:-InvDeg[GfG:-Lex]"}$$

```
Monomial order GfG:-InvDeg[GfG:-Lex] is admissible: false
```

```
>
> p1:=5*e18+5*e6we8we9-5*e2we3we8;
  L:=convert(cliterms(p1),list);
  sort(L,Deg[Lex]);
  cbas:=cbasis(3):
  sort(cbas,Deg[Lex]);
```

$$p1 := 5\ e18 + 5\ e689 - 5\ e238$$

$$L := [e18, e238, e689]$$

$$[e238, e689, e18]$$

$$[e123, e12, e13, e23, e1, e2, e3, Id]$$

```
>
```

Example 3: Procedures LTerm, LMon, and LCoeff return the leading term, the leading monomial, and the leading coefficient for a Grassmann polynomial for a stated monomial order.

```
> p:=Id+e4+e6we7-3*e3we9+e1+e2;
```

$$p := Id + e4 + e67 - 3\ e39 + e1 + e2$$

```
> L:=convert(cliterms(p),list);
```

$$L := [Id, e2, e1, e39, e67, e4]$$

```
> sort(L,Lex);
  LTerm(p,Lex),LMon(p,Lex),LCoeff(p,Lex);
```

$$[e1, e2, e39, e4, e67, Id]$$

$$e1, e1, 1$$

```
> sort(L,Deg[Lex]);
  LTerm(p,Deg[Lex]),LMon(p,Deg[Lex]),LCoeff(p,Deg[Lex]);
```

$$[e39, e67, e1, e2, e4, Id]$$

$$-3\ e39, e39, -3$$

```
>
```

Example 4: Procedure 'CommonFactor' finds a common Grassmann factor in two Grassmann monomials x and y. If the overlap between the two monomials is empty, the procedure returns the identity element Id. Note that common factor of monomials x and y may differ in sign from the common factor of y and x.

'CommonFactor' is Grassmann's original 'regressive' product, which he computed in his first extension

theory A1 using the 'rule of the common factor'. Here 'CommonFactor makes use of two aspects of a Graded Hopf algebra, we can compute the meet (dual of join=wedge) in a Grassmann algebra using the volume element. Thereby depending on the dimension. Common factor looks into the smallest Grassmann algebra which contains the monomials x and y and extracts the coefficient of the highest grade element (volume form). In this way, the common factor function becomes independent of an 'ambient dimension' and loses its weak dependence on the (weight of) the metric.

```
> CommonFactor(e1we2,e1we3);
  CommonFactor(e1we3,e1we2);
```

$$-e1$$

$$e1$$

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

$$cbas := [Id, e1, e2, e3, e12, e13, e23, e123]$$

```
> f:=(m1,m2)->CommonFactor(m1,m2);
```

$$f := (m1, m2) \rightarrow GfG{:-}CommonFactor(m1, m2)$$

```
> for m1 in cbas do
      for m2 in cbas do
  printf("The common factor of %a and %a is %a \n",m1,m2,f(m1,m2));
  end do end do:
The common factor of Id and Id is Id
The common factor of Id and e1 is Id
The common factor of Id and e2 is Id
The common factor of Id and e3 is Id
The common factor of Id and e12 is Id
The common factor of Id and e13 is Id
The common factor of Id and e23 is Id
The common factor of Id and e123 is Id
The common factor of e1 and Id is Id
The common factor of e1 and e1 is e1
The common factor of e1 and e2 is -Id
The common factor of e1 and e3 is -Id
The common factor of e1 and e12 is e1
The common factor of e1 and e13 is e1
The common factor of e1 and e23 is Id
The common factor of e1 and e123 is e1
The common factor of e2 and Id is Id
The common factor of e2 and e1 is Id
The common factor of e2 and e2 is e2
The common factor of e2 and e3 is -Id
The common factor of e2 and e12 is e2
The common factor of e2 and e13 is -Id
The common factor of e2 and e23 is e2
The common factor of e2 and e123 is e2
The common factor of e3 and Id is Id
The common factor of e3 and e1 is Id
The common factor of e3 and e2 is Id
The common factor of e3 and e3 is e3
The common factor of e3 and e12 is Id
The common factor of e3 and e13 is e3
The common factor of e3 and e23 is e3
The common factor of e3 and e123 is e3
```

```
The common factor of e12 and Id is Id
The common factor of e12 and e1 is e1
The common factor of e12 and e2 is e2
The common factor of e12 and e3 is Id
The common factor of e12 and e12 is e12
The common factor of e12 and e13 is -e1
The common factor of e12 and e23 is -e2
The common factor of e12 and e123 is e12
The common factor of e13 and Id is Id
The common factor of e13 and e1 is e1
The common factor of e13 and e2 is -Id
The common factor of e13 and e3 is e3
The common factor of e13 and e12 is e1
The common factor of e13 and e13 is e13
The common factor of e13 and e23 is -e3
The common factor of e13 and e123 is e13
The common factor of e23 and Id is Id
The common factor of e23 and e1 is Id
The common factor of e23 and e2 is e2
The common factor of e23 and e3 is e3
The common factor of e23 and e12 is e2
The common factor of e23 and e13 is e3
The common factor of e23 and e23 is e23
The common factor of e23 and e123 is e23
The common factor of e123 and Id is Id
The common factor of e123 and e1 is e1
The common factor of e123 and e2 is e2
The common factor of e123 and e3 is e3
The common factor of e123 and e12 is e12
The common factor of e123 and e13 is e13
The common factor of e123 and e23 is e23
The common factor of e123 and e123 is e123
```

```
>
>
>
```

Example 5: Procedure GDivide divides a Grassmann polynomial p1 by another Grassmann polynomial p2 in the admissible order 'MonOrder' entered as a second argument. The user needs to know first whether this monomial order is admissible (use procedure 'isadmissible' first). This procedure returns the remainder r of the division of p1 by p2 that is, a polynomial r such that p1 = wedge(q,p2) + r where r is a polynomial such that GDivide(r,p2) = r, that is, r cannot be further reduced by dividing it by p2.

Note that GDivide and GLCM (defined below) need the reversion. The reversion is a map into the opposite algebra and introduces signs due to the generation of the algebra by left multiplication turned into right multiplication. The natural order of the opposite algebra would possibly be a decreasing order on the indices like e32, e421, e321 etc. One should keep in mind that ordering may be reversed when we use a contravariant functor like /\ --F--> /\^op.

When extending Groebner for Grassmann to Clifford algebras, all GDivide, GLCM, etc., functions will then need the correct (possibly containing antisymmetric) parts either !

This procedure may be used with an optional fourth argument of type 'symbol': If used that way, it

returns the remainedr r and the quotient q as a sequence r,q (that is, r first and q second).

GDivide:=proc(p1,p2,MonOrder)
divide p2 into p1 [divide p1 by p2], or, write p1 = q &w p2 + r, or q &w p2 = p1 - r

```
> p1:=2*e1we2-3*e4+5*e1we2we3;
```

$$p1 := 2\ e12 - 3\ e4 + 5\ e123$$

```
> p2:=e2-e1+7;
```

$$p2 := e2 - e1 + 7$$

```
> r:=GDivide(p1,p2,Deg[Lex]);
  r = GDivide(r,p2,Deg[Lex]);
```

$$r := -3\ e4 + 35\ e23 - 14\ e2$$

$$-3\ e4 + 35\ e23 - 14\ e2 = -3\ e4 + 35\ e23 - 14\ e2$$

Note that it is possible to find a Grassmann polynomial q (the quotient) in the Grassmann algebra /\R^n, where n = max(maxindex(p1),maxindex(p2)), such that q &w p2 = p1 - r, as follows:

```
> r,q:=GDivide(p1,p2,Deg[Lex],'q');
  p1 = Clifford:-wedge(q,p2)+r;
```

$$r, q := -3\ e4 + 35\ e23 - 14\ e2, 2\ e2 - 5\ e23$$

$$2\ e12 - 3\ e4 + 5\ e123 = 2\ e12 - 3\ e4 + 5\ e123$$

Let's change the order:

```
> r:=GDivide(p1,p2,InvLex);
  r =GDivide(r,p2,InvLex);
```

$$r := -3\ e4 + 35\ e13 - 14\ e1$$

$$-3\ e4 + 35\ e13 - 14\ e1 = -3\ e4 + 35\ e13 - 14\ e1$$

Note that it is possible to find a Grassmann polynomial q (the quotient) in the Grassmann algebra /\R^n, where n = max(maxindex(p1),maxindex(p2)), such that q &w p2 = p1 - r, as follows:

```
> r,q:=GDivide(p1,p2,InvLex,'q');
  p1 = Clifford:-wedge(q,p2)+r;
```

$$r, q := -3\ e4 + 35\ e13 - 14\ e1, 2\ e1 - 5\ e13$$

$$2\ e12 - 3\ e4 + 5\ e123 = 2\ e12 - 3\ e4 + 5\ e123$$

```
>
```

Example 6: Procedure GLCM is the Grassmann LCM of two Grassmann monomials of type 'climon' or 'clibasmon'. It uses the fact that AP = Clifford:-RC(A,reversion(C)), where Clifford:-RC is the right contraction of A by C: It contracts out the right factor C, that is, A = AP &w C. Similarly, it extracts in the same way the C factor from the lft in B, that is, B = C &w BP, where C is the common factor (overlap of indices) of A and B. It returns a list [AP, C, BP]. In the special case of no overlap, it returns C = Id, that is, [A,Id,B].

Note: We temporarily set B = diag(1$9) so that each basis vector e_i would contact to 1, that is, LC(e_i,e_j) = RC(e_i,e_j) = 1, when i = j, and 0, when i <> j.

Warning: GLCM returns signed factors of the least common multiple of monomials m1 and m2

Actually a Grassmann algebra is a graded commutative algebra! Hence a bimodule is graded commutative isomorphic to a left or right module. In this sense, a Grassmann algebra behaves like a (graded) commutative algebra. The sign accounts for the right action written as left action!

```
> cbas;
  p1:=7*e2+5*e2we3;
  # right actions
  ra:=map((x)->wedge(p1,x),cbas);
  # left action written as right action
  la:=map((x)->drop_t(&map(gswitch(&t(x,p1),1),1,wedge)),cbas);
  diff=1/2*(ra-la);
  summ=1/2*(ra+la);
```

$$[Id, e1, e2, e3, e12, e13, e23, e123]$$

$$p1 := 7\,e2 + 5\,e23$$

$$ra := [7\,e2 + 5\,e23, -7\,e12 + 5\,e123, 0, 7\,e23, 0, -7\,e123, 0, 0]$$

$$la := [7\,e2 + 5\,e23, 7\,e12 + 5\,e123, 0, -7\,e23, 0, -7\,e123, 0, 0]$$

$$diff = [0, -7\,e12, 0, 7\,e23, 0, 0, 0, 0]$$

$$summ = [7\,e2 + 5\,e23, 5\,e123, 0, 0, 0, -7\,e123, 0, 0]$$

In tangle notation the first computation is done using the wedge (a product tangle 2 inputs 1 output)

The right action written as left action picks up some signs due to the graded commutativity. We need this signs to be able to write all actions as left actions only. This is also the source of the fact that we need to concentrate only on one overlap, since a Grassmann monom cannot have two identical parts due to the graded commutativity. A Grassmann (and later a Clifford) algebra is hence much more commutative than noncommutative (algebra of words)

```
> m1:=e1we2we3we5;m2:=e4we3we2;
  GLCM(m1,m2);
  wedge(op(%));
  GLCM(m2,m1);
  wedge(op(%));
```

$$m1 := e1235$$

$$m2 := e4we3we2$$

$$[e15, e23, -e4]$$

$$e12345$$

$$[-e4, -e3we2, e15]$$

$$e12345$$

```
>
```

Example 7: GSpoly computes an S-polynomial for two Grassmann polynomials p1 and p2 for the stated monomial order MonOrder as follows.

cf1,lm1:=LCoeff(p1,MonOrder),LMon(p1,MonOrder);
cf2,lm2:=LCoeff(p2,MonOrder),LMon(p2,MonOrder);
lst:=GLCM(lm1,lm2);

Then, the S-polynomial is Clifford:-wedge(p1,lst[3])/cf1-Clifford:-wedge(lst[1],p2)/cf2;

```
> p1:=rd_clipolynom();
  p2:=rd_clipolynom();
```

$$p1 := 3\ e3568 + 3\ e16$$
$$p2 := 11\ Id + 11\ e3678$$

```
> MonOrder:=Deg[Lex]:
  'p1'=p1,'p2'=p2;
  Spoly:=GSpoly(p1,p2,MonOrder);
  ##############################
  'p1'=p1,'p2'=p2;
  MonOrder:=InvLex:
  Spoly:=GSpoly(p1,p2,MonOrder);
```

$$p1 = 3\ e3568 + 3\ e16, p2 = 11\ Id + 11\ e3678$$
$$Spoly := e167 - e5$$
$$p1 = 3\ e3568 + 3\ e16, p2 = 11\ Id + 11\ e3678$$
$$Spoly := e167 - e5$$

```
>
```

Example 8: Procedure GReduce is obsolete and wrong. It was supposed to reduce a Grassmann polynomial p with respect to a list [p1,...,pn] of n Grassmann monomials for the given monomial order entered as the third argument. Use instead CanForm, LeftNormalForm, or Right NormalForm.

Procedure 'CanForm' decomposes a polynomial f into two components phi and h such that phi belongs to an ideal I (left, right) generated by a list plst while h belongs to Span_k(O(I)) (the span of the order ideal O(I) = S \ T(I) where S is a free semigroup (not quite) and T(I) is the ideal generated by the leading terms (with respect to the monomial order MonOrder). Here S is generated by e1,e2,...e_N where N is the larger of the maximum indices occuring in f and plst. It follows algorithm in [Mora] on page 3.

- When used with three arguments, ideal I is assumed to be a left ideal (default).

- When used with the fourth optional argument 'left' or left, ideal I is assumed to be a left ideal. It calls procedure 'chooseg1' with the optional fourth argument.
- When used with the fourth optional argument 'right' or right, ideal I is assumed to be a right ideal. It calls procedure 'chooseg1' with the optional fourth argument.

The procedure 'LeftNormalForm' computes left normal form of a polynomial f with respect to finite set G that generates a left ideal I and is a counterpart of incomplete Gaussian reduction [Mora]. See his definition of normal form and Proposition 2.1 in [Mora] on page 6.

Definition: Let G subset of k<S> (free algebra - in our case it will be Grassmann algebra) and let I be the ideal (left, right) generated by G, and let f be a polynomial in k<S>. A normal form of f w.r.t. G si an element h in k<S> such that:

(a) f - h belongs to I;
(b) either h = 0 or T(h) does not belong to T(G), that is, the leading term of h does not belong to the ideal T(G) in k<S> that is being generated by the leading terms of the set G.

Proposition 2.1 [Mora]
G is a Groebner basis of I if and only if for every f in k<S>, and for every h, the normal form of f:

- if h = 0 then f belongs to I;
- if h <> 0 then f does not belong to I.

Procedure RightNormalForm computes right normal form of a polynomial f with respect to finite set G that generates a right ideal I and is a counterpart of incomplete Gaussian reduction [Mora]. See his definition of normal form and Proposition 2.1 in [Mora] on page 6.

```
> p1:=2*e1-3*e1we3;
  LTerm(p1,Deg[Lex]);
```

$$p1 := 2\,e1 - 3\,e13$$

$$-3\,e13$$

```
> p2:=Id+e3;
  LTerm(p2,Deg[Lex]);
```

$$p2 := Id + e3$$

$$e3$$

```
> p3:=2*e1;
  LTerm(p3,Deg[Lex]);
```

$$p3 := 2\,e1$$

$$2\,e1$$

```
> p:=-2*e1we2+e2we3;
  LTerm(p,Deg[Lex]);
```

$$p := -2\,e12 + e23$$

$$-2\,e12$$

```
> r:=GReduce(p,[p1,p2,p3],Deg[Lex]);
Warning, procedure GReduce is obsolete and produces wrong results. Use
GfG:-CanForm, GfG:-LeftNormalForm, or GfG:-RightNormalForm
```

$$r := -e2$$

```
> r:=CanForm(p,[p1,p2,p3],Deg[Lex]);
  r:=LeftNormalForm(p,[p1,p2,p3],Deg[Lex]);
  r:=RightNormalForm(p,[p1,p2,p3],Deg[Lex]);
Warning, procedure CanForm may produce wrong results. It uses Mora's definition
and algorithm in [Mora] on page 3.
```

$$r := -2\,e12 + e23, 0$$

```
Warning, procedure LeftNormalForm may produce wrong results. It uses Mora's
definition and algorithm in [Mora] on page 6 modified by R.A.
```

$$r := -e2$$

```
Warning, procedure RightNormalForm may produce wrong results. It uses Mora's
definition and algorithm in [Mora] on page 6 modified by R.A.
```

$$r := -e2$$

```
> r=GDivide(r,p1,Deg[Lex]);
  r=GDivide(r,p2,Deg[Lex]);
  r=GDivide(r,p3,Deg[Lex]);
```

$$-e2 = -e2$$

$$-e2 = -e2$$

$$-e2 = -e2$$

Change the order:

```
> r:=GReduce(p,[p1,p2,p3],InvLex);
Warning, procedure GReduce is obsolete and produces wrong results. Use
GfG:-CanForm, GfG:-LeftNormalForm, or GfG:-RightNormalForm
```

$$r := -e2$$

```
> r:=CanForm(p,[p1,p2,p3],InvLex);
  r:=LeftNormalForm(p,[p1,p2,p3],InvLex);
  r:=RightNormalForm(p,[p1,p2,p3],InvLex);
Warning, procedure CanForm may produce wrong results. It uses Mora's definition
and algorithm in [Mora] on page 3.
```

$$r := e23 - 2\,e12, 0$$

```
Warning, procedure LeftNormalForm may produce wrong results. It uses Mora's
definition and algorithm in [Mora] on page 6 modified by R.A.
```

$$r := -e2$$

```
Warning, procedure RightNormalForm may produce wrong results. It uses Mora's
definition and algorithm in [Mora] on page 6 modified by R.A.
```

$$r := -e2$$

```
> r=GDivide(r,p1,InvLex);
  r=GDivide(r,p2,InvLex);
  r=GDivide(r,p3,InvLex);
```

$$-e2 = -e2$$

$$-e2 = -e2$$

$$-e2 = -e2$$

```
>
```

Example 9: Procedure 'GGbasis' computes a Groebner basis for the ideal generated by a list of Grassmann polynomials F = [p1,p2,p3] with respect to the monomial order MonOrder that needs to be admissible. It seems (this needs to be checked) that this basis can be used for solving the ideal membership problem.

```
> F:=[p1,p2,p3];
  n:=max(op(map(Clifford:-maxindex,F)));
```

$$F := [2\,e1 - 3\,e13,\ e3 + Id,\ 2\,e1]$$

$$n := 3$$

```
> c1,c2,c3:=rd_clipolynom(n),rd_clipolynom(n),rd_clipolynom(n);
```

$$c1, c2, c3 := 5\,Id + 5\,e13 - 5\,e2,\ 6\,Id - 4\,e23,\ -5\,Id$$

```
> p_in:=wedge(c1,p1)+wedge(c2,p2)+wedge(c3,p3);
  p_out:=p_in+2*Id+3*e2;
  GReduce(p_out,F,Deg[Lex]);
```

$$p\_in := 10\,e12 - 15\,e13 - 15\,e123 + 6\,e3 + 6\,Id - 4\,e23$$

$$p\_out := 10\,e12 - 15\,e13 - 15\,e123 + 6\,e3 + 8\,Id - 4\,e23 + 3\,e2$$

```
Warning, procedure GReduce is obsolete and produces wrong results. Use
GfG:-CanForm, GfG:-LeftNormalForm, or GfG:-RightNormalForm
```

$$2\,Id + 7\,e2$$

```
> G:=GGbasis(F,Deg[Lex]);
Warning, procedure GReduce is obsolete and produces wrong results. Use
GfG:-CanForm, GfG:-LeftNormalForm, or GfG:-RightNormalForm

Warning, procedure GReduce is obsolete and produces wrong results. Use
GfG:-CanForm, GfG:-LeftNormalForm, or GfG:-RightNormalForm
```

$$G := \left[-\frac{2\,e1}{3} + e13,\ e3 + Id,\ e1\right]$$

```
> rem1:=GReduce(p_in,G,Deg[Lex]);
  rem2:=GReduce(p_out,G,Deg[Lex]);
Warning, procedure GReduce is obsolete and produces wrong results. Use
GfG:-CanForm, GfG:-LeftNormalForm, or GfG:-RightNormalForm
```

$$rem1 := 4\,e2$$

```
Warning, procedure GReduce is obsolete and produces wrong results. Use
GfG:-CanForm, GfG:-LeftNormalForm, or GfG:-RightNormalForm
```

$$rem2 := 2\,Id + 7\,e2$$

```
>
```

Example 10: This procedure computes the minimal Groebner basis from a Groebner basis for a monomial order MonOrder.

```
> minG:=minGGbasis(G,Deg[Lex]);

Warning, procedure GReduce is obsolete and produces wrong results. Use
GfG:-CanForm, GfG:-LeftNormalForm, or GfG:-RightNormalForm

Warning, procedure GReduce is obsolete and produces wrong results. Use
GfG:-CanForm, GfG:-LeftNormalForm, or GfG:-RightNormalForm

Warning, procedure GReduce is obsolete and produces wrong results. Use
GfG:-CanForm, GfG:-LeftNormalForm, or GfG:-RightNormalForm
```

$$minG := [\,e3 + Id, e1\,]$$

```
> GReduce(G[1],minG,Deg[Lex]);
Warning, procedure GReduce is obsolete and produces wrong results. Use
GfG:-CanForm, GfG:-LeftNormalForm, or GfG:-RightNormalForm
```

$$0$$

```
>
```

.

We check if the minG Groebner basis does also reduce the polynomial p:

```
> 'rem0' = GReduce(p,[p1,p2,p3],Deg[Lex]); #remainder after reducing
  p w.r.t. the original polynomials p1,p2,p3
  'rem1' = GReduce(p,G,Deg[Lex]);          #remainder after reducing
  p w.r.t. the Groebner basis G
  'rem2' = GReduce(p,minG,Deg[Lex]);       #remainder after reducing
  p w.r.t. the minimal Groebner basis minG

Warning, procedure GReduce is obsolete and produces wrong results. Use
GfG:-CanForm, GfG:-LeftNormalForm, or GfG:-RightNormalForm
```

$$rem0 = -e2$$

```
Warning, procedure GReduce is obsolete and produces wrong results. Use
GfG:-CanForm, GfG:-LeftNormalForm, or GfG:-RightNormalForm
```

$$rem1 = -e2$$

```
Warning, procedure GReduce is obsolete and produces wrong results. Use
GfG:-CanForm, GfG:-LeftNormalForm, or GfG:-RightNormalForm
```

$$rem2 = -e2$$

Further check, that the minG Groebner basis reduces the G Groebner basis to zero, showing that all

polynomials in G are in the ideal generated by minG too:

```
> map((x)->GReduce(x,minG,Deg[Lex]),G);
Warning, procedure GReduce is obsolete and produces wrong results. Use
GfG:-CanForm, GfG:-LeftNormalForm, or GfG:-RightNormalForm

Warning, procedure GReduce is obsolete and produces wrong results. Use
GfG:-CanForm, GfG:-LeftNormalForm, or GfG:-RightNormalForm

Warning, procedure GReduce is obsolete and produces wrong results. Use
GfG:-CanForm, GfG:-LeftNormalForm, or GfG:-RightNormalForm
```

$$[0, 0, 0]$$

NOTE: Further checking is needed to make sure that procedures GGbasis and minGGbasis compute what we want.

```
>
>
```

Example 11: Procedure 'multideg' computes the multidegree of a Grassmann polynomial with respect to a monomial order entered as a second argument. It returns a vector with entries 1 and 0 that represents the leading monomial of the polynomial with respect to the chosen order.

```
> f := -3*Id-3*e36-3*e2389+e1568+e349+e1256;
```

$$f := -3\,Id - 3\,e36 - 3\,e2389 + e1568 + e349 + e1256$$

```
> multideg(f,Deg[Lex]);
```

$$[1, 1, 0, 0, 1, 1, 0, 0, 0]$$

```
> multideg(f,InvLex);
```

$$[0, 1, 1, 0, 0, 0, 0, 1, 1]$$

```
>
```

Cookeville/Konstanz, June 22, 2008.

Function: GTP:-tensorrank - find the rank of a tensor or an elements of the type 'gradedpolynom'

Calling Sequence:

tensorrank(p);

Parameters:

p - graded polynomials of type GTP:-`type/gradedpolynom`

Description:

- Procedure 'tensorrank' finds the rank of a graded polynomial, that is, an element of the graded tensor product Cl(B1) &t Cl(B2) &t ... &t Cl(Br) of some Clifford algebras Cl(B1), Cl(B2), ... , Cl(Br).
- This procedure is needed to determine if the tensors entered in GTP:-gprod or GTP:-gradedprod are of the same rank.
- If, by mistake, tensors of different ranks are detected in p, an error message is returned.

Examples:

```
> restart:with(Clifford):with(GTP):
> tensorrank(e1 &t e2);
```

$$2$$

```
> tensorrank(&t((e1 + e2),e2,Id,e1+e2,Id));

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

$$5$$

```
> p:=a*(Id &t Id &t e2)-3*(e1we2 &t e2 &t e2we1 )+b*4*(Id &t e2
  &t e1)-
       Id &t e1we2 &t e1we2;
```

$$p := a\,((Id \,\&t\, Id) \,\&t\, e2) - 3\,((e1we2 \,\&t\, e2) \,\&t\, e2we1) + 4\,b\,((Id \,\&t\, e2) \,\&t\, e1) - ((Id \,\&t\, e1we2) \,\&t\, e1we2)$$

```
> tensorrank(p);
```

$$3$$

```
> p:=&t(e1,e2,e3)-&t(e1,e2);
```

$$p := ((e1 \,\&t\, e2) \,\&t\, e3) - (e1 \,\&t\, e2)$$

```
> tensorrank(p); #testing an error message
Error, (in GTP:-tensorrank) tensors of mixed ranks are found in
&t(&t(e1,e2),e3)-&t(e1,e2)

>
```

See Also: GTP:-`type/gradedmonom`, GTP:-gbasis, GTP:-`type/gradedodd`, GTP:-grade, GTP:-`&t`, Clifford:-`type/tensorprod`, GTP:-gradedprod, GTP:-`type/gradedeven`

Last revised: December 20, 2007 /RA/BF

Function: GTP:-`&t` - graded tensor product

Calling Sequence:

&t(p1,p2,...,ps);
p1 &t p2 &t ... &t ps

Parameters:

p1 - expression of type GTP:-`type/tensorprod` or `type/clipolynom`.
p2, ..., ps - polynomials of type `type/clipolynom`.

Description:

- Procedure '`&t`' is a place-holder for the basis elements in the graded tensor product Cl(B1) &t Cl(B2) &t ... &t Cl(Br) of r Clifford algebras Cl(B1), Cl(B2), ... , Cl(Br). In that case each pi is of 'tensorrank' r (see GTP:-tensorrank).
- This procedure is multi-linear and it is left-associative.
- To simplify notation, one may enter '1' instead of 'Id' to denote the identity element in any Clifford algebra.

Examples:

```
> restart:with(Clifford):with(GTP):
> B:=linalg[diag](1,1,1,1,1):eval(makealiases(5)):
> &t(e1,e2,e3);type(%,tensorprod);
```

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

$$true$$

```
> 2*&t(e1,e2,e3);type(%,tensorprod);
```

$$2\,((e1 \,\&t\, e2) \,\&t\, e3)$$

$$false$$

```
> 2*&t(e1,e2,e3);type(%,gradedmonom);
```

$$2\,((e1 \,\&t\, e2) \,\&t\, e3)$$

$$true$$

```
> e1 &t e2 &t e3;&t(e1,e2,e3);
```

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

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

```
> p1:=e1-2*e2we3+a*e1we2we3;p2:=1+e2+2*e1we2;
```

$$p1 := e1 - 2\,e23 + a\,e123$$

$$p2 := 1 + e2 + 2\,e12$$

```
> &t(p1,p2);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$(e1 \,\&t\, 1) + (e1 \,\&t\, e2) + 2\,(e1 \,\&t\, e12) - 2\,(e23 \,\&t\, 1) - 2\,(e23 \,\&t\, e2) - 4\,(e23 \,\&t\, e12)$$

$$+ a\,(e123\ \&t\ 1) + a\,(e123\ \&t\ e2) + 2\,a\,(e123\ \&t\ e12)$$

```
> &t(p1,p2,p1);
```

$$a^2\,((e123\ \&t\ 1)\ \&t\ e123) + a^2\,((e123\ \&t\ e2)\ \&t\ e123) + ((e1\ \&t\ 1)\ \&t\ e1) - 2\,((e1\ \&t\ 1)\ \&t\ e23) + a\,((e1\ \&t\ 1)\ \&t\ e123) + ((e1\ \&t\ e2)\ \&t\ e1) + a\,((e1\ \&t\ e2)\ \&t\ e123) - 2\,((e1\ \&t\ e2)\ \&t\ e23) + 2\,((e1\ \&t\ e12)\ \&t\ e1) - 4\,((e1\ \&t\ e12)\ \&t\ e23) + 2\,a\,((e1\ \&t\ e12)\ \&t\ e123) - 2\,((e23\ \&t\ 1)\ \&t\ e1) + 4\,((e23\ \&t\ 1)\ \&t\ e23) - 2\,a\,((e23\ \&t\ 1)\ \&t\ e123) - 2\,((e23\ \&t\ e2)\ \&t\ e1) + 4\,((e23\ \&t\ e2)\ \&t\ e23) - 2\,a\,((e23\ \&t\ e2)\ \&t\ e123) - 4\,((e23\ \&t\ e12)\ \&t\ e1) + 8\,((e23\ \&t\ e12)\ \&t\ e23) - 4\,a\,((e23\ \&t\ e12)\ \&t\ e123) + a\,((e123\ \&t\ 1)\ \&t\ e1) - 2\,a\,((e123\ \&t\ 1)\ \&t\ e23) + a\,((e123\ \&t\ e2)\ \&t\ e1) - 2\,a\,((e123\ \&t\ e2)\ \&t\ e23) + 2\,a\,((e123\ \&t\ e12)\ \&t\ e1) - 4\,a\,((e123\ \&t\ e12)\ \&t\ e23) + 2\,a^2\,((e123\ \&t\ e12)\ \&t\ e123)$$

```
> p:=&t(1,1,1,1);
```

$$p := ((1\ \&t\ 1)\ \&t\ 1)\ \&t\ 1$$

```
> gprod(p,p);
```

$$((Id\ \&t\ Id)\ \&t\ Id)\ \&t\ Id$$

```
> K:=cbasis(2):L:=gbasis(K$2);
```

$$L := [Id\ \&t\ Id, Id\ \&t\ e1, Id\ \&t\ e2, Id\ \&t\ e12, e1\ \&t\ Id, e1\ \&t\ e1, e1\ \&t\ e2, e1\ \&t\ e12, e2\ \&t\ Id, e2\ \&t\ e1, e2\ \&t\ e2, e2\ \&t\ e12, e12\ \&t\ Id, e12\ \&t\ e1, e12\ \&t\ e2, e12\ \&t\ e12]$$

```
>
```

See Also: GTP:-`type/gradedmonom`, GTP:-gbasis, GTP:-`type/gradedodd`, GTP:-grades, Clifford:-`type/tensorprod`, GTP:-gradedprods, GTP:-gprods, GTP:-`type/gradedeven`

Last revised: December 20, 2007 /RA/BF

Function: GTP:-cmulB - Clifford product in the Clifford algebra Cl(B1)

Calling Sequence:

cmulB(p1,p2,B1);

Parameters:

p1, p2 - expressions of the type 'cliscalar' or 'clipolynom'
B1 - matrix of a bilinear form B1

Description:

- Procedure 'cmulB' gives the Clifford product in the Clifford algebra Cl(B1) where B1 an arbitrary bilinear form.
- It differs from cmul in that a user may enter any form B1 as the third argument and the product of p1 and p2 will be computed in Cl(B1) regardless of the current value of B. Recall that B, when defined, is a global variable storing a bilinear form B used as a default to compute the Clifford product in Cl(B). Thus, 'cmul' by default computes the Clifford product with the form B while 'cmulB' computes the same product with the user-supplied form B1.
- The bilinear form B1 is totally arbitrary. It may be symbolic, undefined, symmetric, diagonal, with or without an antisymmetric part, numeric. When used as the third argument, it does not overwrite B.
- No infix form for 'cmulB' is available since it is intended to be used with the third argument. If the third argument is not entered, an error message is displayed.
- Calls to 'cmulB' with different values of B1 are made when computing products of tensors with GTP:-gradedprod or GTP:-gprod represented as polynomials of type GTP:-`type/gradedpolynom`, GTP:-`type/gradedmonom`, or GTP:-`type/tensorprod`. In that case, B1 is expected to be diagonal. See examples below.

Examples:

```
> restart:with(Clifford):with(GTP):
> B:=linalg[diag](1,1):B1:=linalg[diag](1,-1):B2:=linalg[matrix](
  2,2,[1,-1,1,-1]):
> p1:=e1-2*e2+e1we2:p2:=-e1we2+3*e2:
> cmul(p1,p2);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$-e2 + e1 - 5\,Id + 3\,e1we2$$

```
> cmulB(p1,p2); #testing an error message
Error, invalid input: GTP:-cmulB uses a 3rd argument, B1 (of type matrix),
which is missing
> cmulB(p1,p2,B); #product of p1 and p2 in Cl(B)
```

$$-e2 + e1 - 5\,Id + 3\,e1we2$$

```
> cmulB(p1,p2,B1); #product of p1 and p2 in Cl(B1)
```

$$-e2 - e1 + 5\,Id + 3\,e1we2$$

```
> cmulB(p1,p2,B2); #product of p1 and p2 in Cl(B1)
```

$$4\,e2 - 2\,e1 + e1we2 + 3\,Id$$

The above products can also be accomplished using procedure Clifford:-cmul with optional index. For example,

```
> cmul[B](p1,p2); #product of p1 and p2 in Cl(B)
```

$$-5\,Id + 3\,e1we2 + e1 - e2$$

```
> cmul[B1](p1,p2); #product of p1 and p2 in Cl(B1)
```

$$5\,Id + 3\,e1we2 - e1 - e2$$

```
> cmul[B2](p1,p2); #product of p1 and p2 in Cl(B2)
```

$$3\,Id + e1we2 - 2\,e1 + 4\,e2$$

```
> B3:=matrix(2,2,[]): #matrix B3 may be symbolic
> cmulB(p1,p2,B3);
```

$$(-B3_{2,1}\,B3_{1,2} + B3_{2,2}\,B3_{1,1} - 6\,B3_{2,2} + 3\,B3_{1,2})\,Id + (B3_{1,2} + 3 - B3_{2,1})\,e1we2 + (B3_{2,2} + B3_{1,2})\,e1 - (B3_{1,1} - 2\,B3_{2,1} + 3\,B3_{1,2})\,e2$$

Here is an example where 'cmulB' will be called by GTP:-gradedprod, GTP:-gprod when three different arguments B, B1, and B2 are used. Remember that in that case all three forms B, B1, and B2 must be diagonal.

```
> gprod(3*e1 &t e2 &t e2, 2*e1 &t e2 &t e2,B,B1,B1);
```

$$-6\,((Id\,\&t\,Id)\,\&t\,Id)$$

```
> gradedprod((2/3)*(e1 &t e2)-(5/a)*(e2we1 &t e2we1),3*(e1 &t
  e2)+Pi*(e1we2 &t e2we1),B1,B);
```

$$-2\,(Id\,\&t\,Id) + \frac{2}{3}\,\pi\,(e2\,\&t\,e1) + \frac{15\,(e2\,\&t\,e1)}{a} - \frac{5\,\pi\,(Id\,\&t\,Id)}{a}$$

```
> gcollect(%);
```

$$-\frac{(2\,a + 5\,\pi)\,(Id\,\&t\,Id)}{a} + \frac{1}{3}\,\frac{(2\,\pi\,a + 45)\,(e2\,\&t\,e1)}{a}$$

```
> evalm(B); #B is not overwritten
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

```
>
```

See Also: Clifford:-cmul, GTP:-`type/gradedmonom`, GTP:-gbasis, GTP:-`type/gradedodd`, GTP:-grade, GTP:-`&t`, Clifford:-`type/tensorprod`, GTP:-gradedprod, GTP:-gprod, GTP:-`type/gradedeven`

Last revised: December 20, 2007 /RA/BF

Function: GTP:-gbasis - define a standard basis in the graded tensor product of n Clifford algebras Cl(B1), Cl(B2),..., Cl(Bn)

Calling Sequence:

gbasis(L1,L2,..,Ln);
gbasis(L$n);

Parameters:

L1, L2,...,Ln - lists of basis elements in Cl(B1), Cl(B2),..., Cl(Bn)
L - one list of basis elements in Cl(B) replicated n times in the call gbasis(L$n)
n - rank of the graded tensor product of Cl(B) &t Cl(B) &t ... &t Cl(B), n>=2

Description:

- Procedure 'gbasis' writes a basis for a graded tensor product GTP:-`&t` of n Clifford algebras Cl(B1), Cl(B2),..., Cl(Bn) where each Clifford algebra Cl(Bi) is over a real vector space Vi endowed with a bilinear form Bi.
- The input may consist of a sequence of n lists L1, L2,...,Ln where each list Li, 1<=i<=n, contains a canonical basis for Cl(Bi) obtained with a help of Clifford:-cbasis. Note that n must be at least 2.
- If all lists are the same, then a shorter input is possible of the form 'L$n' where L is a basis for Cl(B) and n is the rank of the desired graded tensor product.
- It returns a list of elements of type GTP:-`type/tensorprod`.
- For more information, see also makealiases.

Examples:

```
> restart:with(Clifford):with(GTP):eval(makealiases(4)):
> L:=cbasis(2);L1:=cbasis(2);L2:=cbasis(3);
```

$$L := [Id, e1, e2, e12]$$

$$L1 := [Id, e1, e2, e12]$$

$$L2 := [Id, e1, e2, e3, e12, e13, e23, e123]$$

```
> gbasis(L$2);nops(%);
```

$$[Id \,\&t\, Id, Id \,\&t\, e1, Id \,\&t\, e2, Id \,\&t\, e12, e1 \,\&t\, Id, e1 \,\&t\, e1, e1 \,\&t\, e2, e1 \,\&t\, e12, e2 \,\&t\, Id, e2 \,\&t\, e1, e2 \,\&t\, e2, e2 \,\&t\, e12, e12 \,\&t\, Id, e12 \,\&t\, e1, e12 \,\&t\, e2, e12 \,\&t\, e12]$$

16

```
> gbasis(L1,L2);nops(%);
```

$$[Id \,\&t\, Id, Id \,\&t\, e1, Id \,\&t\, e2, Id \,\&t\, e3, Id \,\&t\, e12, Id \,\&t\, e13, Id \,\&t\, e23, Id \,\&t\, e123, e1 \,\&t\, Id, e1 \,\&t\, e1, e1 \,\&t\, e2, e1 \,\&t\, e3, e1 \,\&t\, e12, e1 \,\&t\, e13, e1 \,\&t\, e23, e1 \,\&t\, e123, e2 \,\&t\, Id, e2 \,\&t\, e1, e2 \,\&t\, e2, e2 \,\&t\, e3, e2 \,\&t\, e12, e2 \,\&t\, e13, e2 \,\&t\, e23, e2 \,\&t\, e123, e12 \,\&t\, Id, e12 \,\&t\, e1, e12 \,\&t\, e2, e12 \,\&t\, e3, e12 \,\&t\, e12, e12 \,\&t\, e13, e12 \,\&t\, e23, e12 \,\&t\, e123]$$

$$32$$

```
> gbasis(L); #testing an error message
Error, (in GTP:-gbasis) at least two lists with elements of type 'clibasmon' or
'tensorprod' are needed as input
```

```
> gbasis(L$3);nops(%);
```

$$[(Id \,\&t\, Id) \,\&t\, Id, (Id \,\&t\, Id) \,\&t\, e1, (Id \,\&t\, Id) \,\&t\, e2, (Id \,\&t\, Id) \,\&t\, e12, (Id \,\&t\, e1) \,\&t\, Id, (Id \,\&t\, e1) \,\&t\, e1, (Id \,\&t\, e1) \,\&t\, e2, (Id \,\&t\, e1) \,\&t\, e12, (Id \,\&t\, e2) \,\&t\, Id, (Id \,\&t\, e2) \,\&t\, e1, (Id \,\&t\, e2) \,\&t\, e2, (Id \,\&t\, e2) \,\&t\, e12, (Id \,\&t\, e12) \,\&t\, Id, (Id \,\&t\, e12) \,\&t\, e1, (Id \,\&t\, e12) \,\&t\, e2, (Id \,\&t\, e12) \,\&t\, e12, (e1 \,\&t\, Id) \,\&t\, Id, (e1 \,\&t\, Id) \,\&t\, e1, (e1 \,\&t\, Id) \,\&t\, e2, (e1 \,\&t\, Id) \,\&t\, e12, (e1 \,\&t\, e1) \,\&t\, Id, (e1 \,\&t\, e1) \,\&t\, e1, (e1 \,\&t\, e1) \,\&t\, e2, (e1 \,\&t\, e1) \,\&t\, e12, (e1 \,\&t\, e2) \,\&t\, Id, (e1 \,\&t\, e2) \,\&t\, e1, (e1 \,\&t\, e2) \,\&t\, e2, (e1 \,\&t\, e2) \,\&t\, e12, (e1 \,\&t\, e12) \,\&t\, Id, (e1 \,\&t\, e12) \,\&t\, e1, (e1 \,\&t\, e12) \,\&t\, e2, (e1 \,\&t\, e12) \,\&t\, e12, (e2 \,\&t\, Id) \,\&t\, Id, (e2 \,\&t\, Id) \,\&t\, e1, (e2 \,\&t\, Id) \,\&t\, e2, (e2 \,\&t\, Id) \,\&t\, e12, (e2 \,\&t\, e1) \,\&t\, Id, (e2 \,\&t\, e1) \,\&t\, e1, (e2 \,\&t\, e1) \,\&t\, e2, (e2 \,\&t\, e1) \,\&t\, e12, (e2 \,\&t\, e2) \,\&t\, Id, (e2 \,\&t\, e2) \,\&t\, e1, (e2 \,\&t\, e2) \,\&t\, e2, (e2 \,\&t\, e2) \,\&t\, e12, (e2 \,\&t\, e12) \,\&t\, Id, (e2 \,\&t\, e12) \,\&t\, e1, (e2 \,\&t\, e12) \,\&t\, e2, (e2 \,\&t\, e12) \,\&t\, e12, (e12 \,\&t\, Id) \,\&t\, Id, (e12 \,\&t\, Id) \,\&t\, e1, (e12 \,\&t\, Id) \,\&t\, e2, (e12 \,\&t\, Id) \,\&t\, e12, (e12 \,\&t\, e1) \,\&t\, Id, (e12 \,\&t\, e1) \,\&t\, e1, (e12 \,\&t\, e1) \,\&t\, e2, (e12 \,\&t\, e1) \,\&t\, e12, (e12 \,\&t\, e2) \,\&t\, Id, (e12 \,\&t\, e2) \,\&t\, e1, (e12 \,\&t\, e2) \,\&t\, e2, (e12 \,\&t\, e2) \,\&t\, e12, (e12 \,\&t\, e12) \,\&t\, Id, (e12 \,\&t\, e12) \,\&t\, e1, (e12 \,\&t\, e12) \,\&t\, e2, (e12 \,\&t\, e12) \,\&t\, e12]$$

$$64$$

```
>
```

See Also: Clifford:-cbasis, GTP:-`type/gradedmonom`, GTP:-gbasis, GTP:-`&t`, Clifford:-`type/tensorprod`

Last revised: December 20, 2007 /RA/BF

Function: GTP:-gcollect - collect graded tensors with respect to the basis elements

Calling Sequence:

gcollect(p);
gcollect(p,s);

Parameters:

p - polynomial of type GTP:-`type/gradedpolynom`
s - (optional) string which could be one of simplify, normal, etc.,

Description:

- Procedure 'gcollect' collects terms in a graded polynomial with respect to its basis elements of Clifford:-`type/tensorprod`.
- When the optional string is used, appropriate operation is performed on the coefficients. See collect for more help.
- By default, coefficients are factored.

Examples:

```
> restart:with(Clifford):with(GTP):
> p:=(a/b-a^2/b^2)*(e1 &t e2) +
  2*(&t(e1,e2));gcollect(p);gcollect(p,simplify);
```

$$p := \left(\frac{a}{b} - \frac{a^2}{b^2}\right)(e1 \,\&t\, e2) + 2\,(e1 \,\&t\, e2) - \frac{(a+b)\,(a-2\,b)\,(e1 \,\&t\, e2)}{b^2} - \frac{(-a\,b + a^2 - 2\,b^2)\,(e1 \,\&t\, e2)}{b^2}$$

```
> p:=-2*a^3*&t(Id,Id,Id)+32*a*b^2*&t(Id,Id,Id);
```

$$p := -2\,a^3\,((Id \,\&t\, Id) \,\&t\, Id) + 32\,a\,b^2\,((Id \,\&t\, Id) \,\&t\, Id)$$

```
> gcollect(p);gcollect(p,normal);gcollect(p,simplify);
```

$$-2\,a\,(a-4\,b)\,(a+4\,b)\,((Id \,\&t\, Id) \,\&t\, Id)$$

$$(-2\,a^3 + 32\,a\,b^2)\,((Id \,\&t\, Id) \,\&t\, Id)$$

$$(-2\,a^3 + 32\,a\,b^2)\,((Id \,\&t\, Id) \,\&t\, Id)$$

```
>
```

See Also: Clifford:-`type/tensorprod`, GTP:-`type/gradedmonom`, GTP:-`type/gradedpolynom`, GTP:-`&t`

Last revised: December 20, 2007 /RA/BF

Function: GTP:-gprod - compute a graded product of elements of the type 'tensorprod' or 'gradedmonom'

Calling Sequence:

gprod(p1,p2);
gprod(p1,p2,B1,B2,...,Br);

Parameters:

p1, p2 - graded monomials of type GTP:-`type/gradedmonom` or Clifford:-`type/tensorprod` of rank r
B1,B2,...,Br - (optional) sequence of r diagonal forms Bi, 1 <= i <= r, where r is the tensor rank of p1 and p2

Description:

- Procedure 'gprod' computes a graded product of two monomials in the graded tensor product Cl(B1) &t Cl(B2) &t ... &t Cl(Br) of r Clifford algebras Cl(B1), Cl(B2), ... , Cl(Br).
- When the optional sequence is used, Clifford products are computed component-wise in Cl(B1), Cl(B2), ... , Cl(Br) with a help of the procedure GTP:-cmulB. However, the Z2-gradation is taken into consideration in order to assure that elements of the type e1 &t 1 and 1 &t e1 belonging to Cl(B1) &t Cl(B2) anticommute.
- Note that the product of two homogeneous elements a &t b and c &t d of Cl(B1) &t Cl(B2) is computed as follows:

  gprod(a &t b, c &t d) = (-1)^(grade(b)*grade(c))*(cmulB(a,c,B1) &t cmulB(b,d,B2)

  where a and c are homogeneous elements that belong to Cl(B1), b and d are homogeneous elements that belong to Cl(B2), and the grades of b and c modulo 2 may be computed with GTP:-grade.
- Since b and c elements are homogeneous elements, they are either even or odd. If b is even, then grade(b) = 0 mod 2 , otherwise grade(b) = 1 mod 2.
- The graded multiplication defined above on homogeneous tensors of rank 2 may be extended to homogeneous tensors of higher ranks and then to non-homogeneous tensors by means of linearity. In that latter case, use procedure GTP:-gradedprod.
- When the optional sequence is not used, the default bilinear form B is applied. Thus, in this case, the r products will be computed in r different copies of Cl(B).
- The ranks of p1 and p2 must be the same. They can be found with GTP:-`type/tensorrank`.

Examples:

```
> restart:with(Clifford):with(GTP):eval(makealiases(5)):_prolevel
  :=true:
```

Example 1:

```
> B:=linalg[diag](1,-1,-1,-1):B1:=linalg[diag](-1,1,-1,1):B2:=lin
  alg[diag](1,1,-1,1):
> type(e1 &t e2,tensorprod),
  type(2*(e1 &t e3),gradedmonom),
  type(2*&t(e1,e2,e3),gradedmonom);
```

$$true, true, true$$

```
> gprod(1 &t e1,e2 &t 1);gprod(e2 &t 1,1 &t e1);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$-(e2 \,\&t\, e1)$$

$$e2 \,\&t\, e1$$

```
> gprod(Id &t e1 &t e2,e1 &t e2 &t Id);tensorrank(%);
```

$$-((e1 \,\&t\, e12) \,\&t\, e2)$$

$$3$$

```
> gprod(e1 &t e2, e3 &t e1,B1,B2);
```

$$e13 \,\&t\, e12$$

```
> gprod(Id &t e2 &t e2,e1 &t Id &t e1);tensorrank(%);
```

$$-((e1 \,\&t\, e2) \,\&t\, e12)$$

$$3$$

```
> gprod(3*(Id &t e2),2*(e1 &t Id));tensorrank(%);
```

$$-6\,(e1 \,\&t\, e2)$$

$$2$$

```
> gprod(&t(1,e1,e2),&t(1,e1,e2));
```

$$(Id \,\&t\, Id) \,\&t\, Id$$

Example 2: Consider a graded tensor product of three copies of the Clifford algebra Cl(B) where B is a quadratic form (1,-1,-1,-1). We will show that the above definition of the product assures non-commutativity of the basis elements.

```
> B:=linalg[diag](1,-1,-1,-1):
> for i from 1 to 4 do E[i]:=&t(e||i,1,1) od; #basis 1-vectors in
  one copy of Cl(B)
```

$$E_1 := (e1 \,\&t\, 1) \,\&t\, 1$$

$$E_2 := (e2 \,\&t\, 1) \,\&t\, 1$$

$$E_3 := (e3 \,\&t\, 1) \,\&t\, 1$$

$$E_4 := (e4 \,\&t\, 1) \,\&t\, 1$$

```
> for i from 1 to 4 do F[i]:=&t(1,e||i,1) od; #basis 1-vectors in
  another copy of Cl(B)
```

$$F_1 := (1 \,\&t\, e1) \,\&t\, 1$$

$$F_2 := (1 \,\&t\, e2) \,\&t\, 1$$

$$F_3 := (1 \,\&t\, e3) \,\&t\, 1$$

$$F_4 := (1 \,\&t\, e4) \,\&t\, 1$$

```
> for i from 1 to 4 do G[i]:=&t(1,1,e||i) od; #basis 1-vectors in
  another copy of Cl(B)
```

$$G_1 := (1 \,\&t\, 1) \,\&t\, e1$$

$$G_2 := (1 \,\&t\, 1) \,\&t\, e2$$

$$G_3 := (1 \,\&t\, 1) \,\&t\, e3$$

$$G_4 := (1 \,\&t\, 1) \,\&t\, e4$$

We will combine now all basis rank 1-tensors into a list A. These rank 1-tensors generate the graded tensor product of rank three of the three copies of Cl(B). This new algebra is Z2-isomorphic with Cl(3B) where 3B is a quadratic form of signature (3,9). Below we explicitly show that the generators anticommute.

```
> printlevel:=2:A:=[seq(E[i],i=1..4),seq(F[i],i=1..4),seq(G[i],i=
  1..4)];
```

$$A := [(e1 \,\&t\, 1) \,\&t\, 1, (e2 \,\&t\, 1) \,\&t\, 1, (e3 \,\&t\, 1) \,\&t\, 1, (e4 \,\&t\, 1) \,\&t\, 1, (1 \,\&t\, e1) \,\&t\, 1, (1 \,\&t\, e2) \,\&t\, 1, (1 \,\&t\, e3) \,\&t\, 1, (1 \,\&t\, e4) \,\&t\, 1, (1 \,\&t\, 1) \,\&t\, e1, (1 \,\&t\, 1) \,\&t\, e2, (1 \,\&t\, 1) \,\&t\, e3, (1 \,\&t\, 1) \,\&t\, e4]$$

```
> for i from 1 to nops(A) do
  for j from i to nops(A) do
  1/2*('gprod'(A[i],A[j])+'gprod'(A[j],A[i])) =
  1/2*(gprod(A[i],A[j])+gprod(A[j],A[i]));
  od od;
```

$$GTP{:-}gprod((e1 \,\&t\, 1) \,\&t\, 1, (e1 \,\&t\, 1) \,\&t\, 1) = (Id \,\&t\, Id) \,\&t\, Id$$

$$\frac{1}{2} GTP{:-}gprod((e1 \,\&t\, 1) \,\&t\, 1, (e2 \,\&t\, 1) \,\&t\, 1) + \frac{1}{2} GTP{:-}gprod((e2 \,\&t\, 1) \,\&t\, 1, (e1 \,\&t\, 1) \,\&t\, 1) = 0$$

$$\frac{1}{2} GTP{:-}gprod((e1 \,\&t\, 1) \,\&t\, 1, (e3 \,\&t\, 1) \,\&t\, 1) + \frac{1}{2} GTP{:-}gprod((e3 \,\&t\, 1) \,\&t\, 1, (e1 \,\&t\, 1) \,\&t\, 1) = 0$$

$$\frac{1}{2} GTP{:-}gprod((e1 \,\&t\, 1) \,\&t\, 1, (e4 \,\&t\, 1) \,\&t\, 1) + \frac{1}{2} GTP{:-}gprod((e4 \,\&t\, 1) \,\&t\, 1, (e1 \,\&t\, 1) \,\&t\, 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e1\ \&t\ 1)\ \&t\ 1, (1\ \&t\ e1)\ \&t\ 1)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e1)\ \&t\ 1, (e1\ \&t\ 1)\ \&t\ 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e1\ \&t\ 1)\ \&t\ 1, (1\ \&t\ e2)\ \&t\ 1)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e2)\ \&t\ 1, (e1\ \&t\ 1)\ \&t\ 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e1\ \&t\ 1)\ \&t\ 1, (1\ \&t\ e3)\ \&t\ 1)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e3)\ \&t\ 1, (e1\ \&t\ 1)\ \&t\ 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e1\ \&t\ 1)\ \&t\ 1, (1\ \&t\ e4)\ \&t\ 1)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e4)\ \&t\ 1, (e1\ \&t\ 1)\ \&t\ 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e1\ \&t\ 1)\ \&t\ 1, (1\ \&t\ 1)\ \&t\ e1)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e1, (e1\ \&t\ 1)\ \&t\ 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e1\ \&t\ 1)\ \&t\ 1, (1\ \&t\ 1)\ \&t\ e2)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e2, (e1\ \&t\ 1)\ \&t\ 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e1\ \&t\ 1)\ \&t\ 1, (1\ \&t\ 1)\ \&t\ e3)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e3, (e1\ \&t\ 1)\ \&t\ 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e1\ \&t\ 1)\ \&t\ 1, (1\ \&t\ 1)\ \&t\ e4)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e4, (e1\ \&t\ 1)\ \&t\ 1) = 0$$

$$GTP{:}\text{-}gprod((e2\ \&t\ 1)\ \&t\ 1, (e2\ \&t\ 1)\ \&t\ 1) = -((Id\ \&t\ Id)\ \&t\ Id)$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e2\ \&t\ 1)\ \&t\ 1, (e3\ \&t\ 1)\ \&t\ 1)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((e3\ \&t\ 1)\ \&t\ 1, (e2\ \&t\ 1)\ \&t\ 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e2\ \&t\ 1)\ \&t\ 1, (e4\ \&t\ 1)\ \&t\ 1)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((e4\ \&t\ 1)\ \&t\ 1, (e2\ \&t\ 1)\ \&t\ 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e2\ \&t\ 1)\ \&t\ 1, (1\ \&t\ e1)\ \&t\ 1)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e1)\ \&t\ 1, (e2\ \&t\ 1)\ \&t\ 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e2\ \&t\ 1)\ \&t\ 1, (1\ \&t\ e2)\ \&t\ 1)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e2)\ \&t\ 1, (e2\ \&t\ 1)\ \&t\ 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e2\ \&t\ 1)\ \&t\ 1, (1\ \&t\ e3)\ \&t\ 1)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e3)\ \&t\ 1, (e2\ \&t\ 1)\ \&t\ 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e2\ \&t\ 1)\ \&t\ 1, (1\ \&t\ e4)\ \&t\ 1)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e4)\ \&t\ 1, (e2\ \&t\ 1)\ \&t\ 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e2\ \&t\ 1)\ \&t\ 1, (1\ \&t\ 1)\ \&t\ e1)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e1, (e2\ \&t\ 1)\ \&t\ 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e2\ \&t\ 1)\ \&t\ 1, (1\ \&t\ 1)\ \&t\ e2)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e2, (e2\ \&t\ 1)\ \&t\ 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e2\ \&t\ 1)\ \&t\ 1, (1\ \&t\ 1)\ \&t\ e3)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e3, (e2\ \&t\ 1)\ \&t\ 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e2\ \&t\ 1)\ \&t\ 1, (1\ \&t\ 1)\ \&t\ e4)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e4,(e2\ \&t\ 1)\ \&t\ 1)=0$$

$$GTP{:}\text{-}gprod((e3\ \&t\ 1)\ \&t\ 1,(e3\ \&t\ 1)\ \&t\ 1)=-((Id\ \&t\ Id)\ \&t\ Id)$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e3\ \&t\ 1)\ \&t\ 1,(e4\ \&t\ 1)\ \&t\ 1)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((e4\ \&t\ 1)\ \&t\ 1,(e3\ \&t\ 1)\ \&t\ 1)=0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e3\ \&t\ 1)\ \&t\ 1,(1\ \&t\ e1)\ \&t\ 1)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e1)\ \&t\ 1,(e3\ \&t\ 1)\ \&t\ 1)=0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e3\ \&t\ 1)\ \&t\ 1,(1\ \&t\ e2)\ \&t\ 1)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e2)\ \&t\ 1,(e3\ \&t\ 1)\ \&t\ 1)=0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e3\ \&t\ 1)\ \&t\ 1,(1\ \&t\ e3)\ \&t\ 1)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e3)\ \&t\ 1,(e3\ \&t\ 1)\ \&t\ 1)=0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e3\ \&t\ 1)\ \&t\ 1,(1\ \&t\ e4)\ \&t\ 1)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e4)\ \&t\ 1,(e3\ \&t\ 1)\ \&t\ 1)=0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e3\ \&t\ 1)\ \&t\ 1,(1\ \&t\ 1)\ \&t\ e1)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e1,(e3\ \&t\ 1)\ \&t\ 1)=0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e3\ \&t\ 1)\ \&t\ 1,(1\ \&t\ 1)\ \&t\ e2)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e2,(e3\ \&t\ 1)\ \&t\ 1)=0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((e3\ \&t\ 1)\ \&t\ 1,(1\ \&t\ 1)\ \&t\ e3)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e3,(e3\ \&t\ 1)\ \&t\ 1)=0$$

$$\frac{1}{2}\, GTP{:}\text{-}gprod((e3\ \&t\ 1)\ \&t\ 1, (1\ \&t\ 1)\ \&t\ e4)$$

$$+\frac{1}{2}\, GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e4, (e3\ \&t\ 1)\ \&t\ 1) = 0$$

$$GTP{:}\text{-}gprod((e4\ \&t\ 1)\ \&t\ 1, (e4\ \&t\ 1)\ \&t\ 1) = -((Id\ \&t\ Id)\ \&t\ Id)$$

$$\frac{1}{2}\, GTP{:}\text{-}gprod((e4\ \&t\ 1)\ \&t\ 1, (1\ \&t\ e1)\ \&t\ 1)$$

$$+\frac{1}{2}\, GTP{:}\text{-}gprod((1\ \&t\ e1)\ \&t\ 1, (e4\ \&t\ 1)\ \&t\ 1) = 0$$

$$\frac{1}{2}\, GTP{:}\text{-}gprod((e4\ \&t\ 1)\ \&t\ 1, (1\ \&t\ e2)\ \&t\ 1)$$

$$+\frac{1}{2}\, GTP{:}\text{-}gprod((1\ \&t\ e2)\ \&t\ 1, (e4\ \&t\ 1)\ \&t\ 1) = 0$$

$$\frac{1}{2}\, GTP{:}\text{-}gprod((e4\ \&t\ 1)\ \&t\ 1, (1\ \&t\ e3)\ \&t\ 1)$$

$$+\frac{1}{2}\, GTP{:}\text{-}gprod((1\ \&t\ e3)\ \&t\ 1, (e4\ \&t\ 1)\ \&t\ 1) = 0$$

$$\frac{1}{2}\, GTP{:}\text{-}gprod((e4\ \&t\ 1)\ \&t\ 1, (1\ \&t\ e4)\ \&t\ 1)$$

$$+\frac{1}{2}\, GTP{:}\text{-}gprod((1\ \&t\ e4)\ \&t\ 1, (e4\ \&t\ 1)\ \&t\ 1) = 0$$

$$\frac{1}{2}\, GTP{:}\text{-}gprod((e4\ \&t\ 1)\ \&t\ 1, (1\ \&t\ 1)\ \&t\ e1)$$

$$+\frac{1}{2}\, GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e1, (e4\ \&t\ 1)\ \&t\ 1) = 0$$

$$\frac{1}{2}\, GTP{:}\text{-}gprod((e4\ \&t\ 1)\ \&t\ 1, (1\ \&t\ 1)\ \&t\ e2)$$

$$+\frac{1}{2}\, GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e2, (e4\ \&t\ 1)\ \&t\ 1) = 0$$

$$\frac{1}{2}\, GTP{:}\text{-}gprod((e4\ \&t\ 1)\ \&t\ 1, (1\ \&t\ 1)\ \&t\ e3)$$

$$+\frac{1}{2}\, GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e3, (e4\ \&t\ 1)\ \&t\ 1) = 0$$

$$\frac{1}{2}\, GTP{:}\text{-}gprod((e4\ \&t\ 1)\ \&t\ 1, (1\ \&t\ 1)\ \&t\ e4)$$

$$+\frac{1}{2}\,GTP\text{:-}gprod((1\ \&t\ 1)\ \&t\ e4,(e4\ \&t\ 1)\ \&t\ 1)=0$$

$$GTP\text{:-}gprod((1\ \&t\ e1)\ \&t\ 1,(1\ \&t\ e1)\ \&t\ 1)=(Id\ \&t\ Id)\ \&t\ Id$$

$$\frac{1}{2}\,GTP\text{:-}gprod((1\ \&t\ e1)\ \&t\ 1,(1\ \&t\ e2)\ \&t\ 1)$$

$$+\frac{1}{2}\,GTP\text{:-}gprod((1\ \&t\ e2)\ \&t\ 1,(1\ \&t\ e1)\ \&t\ 1)=0$$

$$\frac{1}{2}\,GTP\text{:-}gprod((1\ \&t\ e1)\ \&t\ 1,(1\ \&t\ e3)\ \&t\ 1)$$

$$+\frac{1}{2}\,GTP\text{:-}gprod((1\ \&t\ e3)\ \&t\ 1,(1\ \&t\ e1)\ \&t\ 1)=0$$

$$\frac{1}{2}\,GTP\text{:-}gprod((1\ \&t\ e1)\ \&t\ 1,(1\ \&t\ e4)\ \&t\ 1)$$

$$+\frac{1}{2}\,GTP\text{:-}gprod((1\ \&t\ e4)\ \&t\ 1,(1\ \&t\ e1)\ \&t\ 1)=0$$

$$\frac{1}{2}\,GTP\text{:-}gprod((1\ \&t\ e1)\ \&t\ 1,(1\ \&t\ 1)\ \&t\ e1)$$

$$+\frac{1}{2}\,GTP\text{:-}gprod((1\ \&t\ 1)\ \&t\ e1,(1\ \&t\ e1)\ \&t\ 1)=0$$

$$\frac{1}{2}\,GTP\text{:-}gprod((1\ \&t\ e1)\ \&t\ 1,(1\ \&t\ 1)\ \&t\ e2)$$

$$+\frac{1}{2}\,GTP\text{:-}gprod((1\ \&t\ 1)\ \&t\ e2,(1\ \&t\ e1)\ \&t\ 1)=0$$

$$\frac{1}{2}\,GTP\text{:-}gprod((1\ \&t\ e1)\ \&t\ 1,(1\ \&t\ 1)\ \&t\ e3)$$

$$+\frac{1}{2}\,GTP\text{:-}gprod((1\ \&t\ 1)\ \&t\ e3,(1\ \&t\ e1)\ \&t\ 1)=0$$

$$\frac{1}{2}\,GTP\text{:-}gprod((1\ \&t\ e1)\ \&t\ 1,(1\ \&t\ 1)\ \&t\ e4)$$

$$+\frac{1}{2}\,GTP\text{:-}gprod((1\ \&t\ 1)\ \&t\ e4,(1\ \&t\ e1)\ \&t\ 1)=0$$

$$GTP\text{:-}gprod((1\ \&t\ e2)\ \&t\ 1,(1\ \&t\ e2)\ \&t\ 1)=-((Id\ \&t\ Id)\ \&t\ Id)$$

$$\frac{1}{2}\,GTP\text{:-}gprod((1\ \&t\ e2)\ \&t\ 1,(1\ \&t\ e3)\ \&t\ 1)$$

$$+\frac{1}{2}\,GTP\text{:-}gprod((1\ \&t\ e3)\ \&t\ 1,(1\ \&t\ e2)\ \&t\ 1)=0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e2)\ \&t\ 1, (1\ \&t\ e4)\ \&t\ 1)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e4)\ \&t\ 1, (1\ \&t\ e2)\ \&t\ 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e2)\ \&t\ 1, (1\ \&t\ 1)\ \&t\ e1)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e1, (1\ \&t\ e2)\ \&t\ 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e2)\ \&t\ 1, (1\ \&t\ 1)\ \&t\ e2)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e2, (1\ \&t\ e2)\ \&t\ 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e2)\ \&t\ 1, (1\ \&t\ 1)\ \&t\ e3)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e3, (1\ \&t\ e2)\ \&t\ 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e2)\ \&t\ 1, (1\ \&t\ 1)\ \&t\ e4)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e4, (1\ \&t\ e2)\ \&t\ 1) = 0$$

$$GTP{:}\text{-}gprod((1\ \&t\ e3)\ \&t\ 1, (1\ \&t\ e3)\ \&t\ 1) = -((Id\ \&t\ Id)\ \&t\ Id)$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e3)\ \&t\ 1, (1\ \&t\ e4)\ \&t\ 1)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e4)\ \&t\ 1, (1\ \&t\ e3)\ \&t\ 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e3)\ \&t\ 1, (1\ \&t\ 1)\ \&t\ e1)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e1, (1\ \&t\ e3)\ \&t\ 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e3)\ \&t\ 1, (1\ \&t\ 1)\ \&t\ e2)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e2, (1\ \&t\ e3)\ \&t\ 1) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e3)\ \&t\ 1, (1\ \&t\ 1)\ \&t\ e3)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e3,(1\ \&t\ e3)\ \&t\ 1)=0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e3)\ \&t\ 1,(1\ \&t\ 1)\ \&t\ e4)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e4,(1\ \&t\ e3)\ \&t\ 1)=0$$

$$GTP{:}\text{-}gprod((1\ \&t\ e4)\ \&t\ 1,(1\ \&t\ e4)\ \&t\ 1)=-((Id\ \&t\ Id)\ \&t\ Id)$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e4)\ \&t\ 1,(1\ \&t\ 1)\ \&t\ e1)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e1,(1\ \&t\ e4)\ \&t\ 1)=0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e4)\ \&t\ 1,(1\ \&t\ 1)\ \&t\ e2)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e2,(1\ \&t\ e4)\ \&t\ 1)=0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e4)\ \&t\ 1,(1\ \&t\ 1)\ \&t\ e3)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e3,(1\ \&t\ e4)\ \&t\ 1)=0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ e4)\ \&t\ 1,(1\ \&t\ 1)\ \&t\ e4)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e4,(1\ \&t\ e4)\ \&t\ 1)=0$$

$$GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e1,(1\ \&t\ 1)\ \&t\ e1)=(Id\ \&t\ Id)\ \&t\ Id$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e1,(1\ \&t\ 1)\ \&t\ e2)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e2,(1\ \&t\ 1)\ \&t\ e1)=0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e1,(1\ \&t\ 1)\ \&t\ e3)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e3,(1\ \&t\ 1)\ \&t\ e1)=0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e1,(1\ \&t\ 1)\ \&t\ e4)$$

$$+\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e4,(1\ \&t\ 1)\ \&t\ e1)=0$$

$$GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e2, (1\ \&t\ 1)\ \&t\ e2) = -((Id\ \&t\ Id)\ \&t\ Id)$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e2, (1\ \&t\ 1)\ \&t\ e3) + \frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e3, (1\ \&t\ 1)\ \&t\ e2) = 0$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e2, (1\ \&t\ 1)\ \&t\ e4) + \frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e4, (1\ \&t\ 1)\ \&t\ e2) = 0$$

$$GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e3, (1\ \&t\ 1)\ \&t\ e3) = -((Id\ \&t\ Id)\ \&t\ Id)$$

$$\frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e3, (1\ \&t\ 1)\ \&t\ e4) + \frac{1}{2}\,GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e4, (1\ \&t\ 1)\ \&t\ e3) = 0$$

$$GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e4, (1\ \&t\ 1)\ \&t\ e4) = -((Id\ \&t\ Id)\ \&t\ Id)$$

Thus, we have

(1/2)*(gprod(A[i],A[j]) + gprod(A[j],A[i]) = 0, 1 <= i,j <= 12, i <> j

and

gprod(A[i],A[i]) = ((Id &t Id) &t Id) or gprod(A[i],A[i]) = -((Id &t Id) &t Id) , i= 1..12.

```
> for i from 1 to nops(A) do 'gprod'(A[i],A[i])=gprod(A[i],A[i])
  od;
```

$$GTP{:}\text{-}gprod((e1\ \&t\ 1)\ \&t\ 1, (e1\ \&t\ 1)\ \&t\ 1) = (Id\ \&t\ Id)\ \&t\ Id$$
$$GTP{:}\text{-}gprod((e2\ \&t\ 1)\ \&t\ 1, (e2\ \&t\ 1)\ \&t\ 1) = -((Id\ \&t\ Id)\ \&t\ Id)$$
$$GTP{:}\text{-}gprod((e3\ \&t\ 1)\ \&t\ 1, (e3\ \&t\ 1)\ \&t\ 1) = -((Id\ \&t\ Id)\ \&t\ Id)$$
$$GTP{:}\text{-}gprod((e4\ \&t\ 1)\ \&t\ 1, (e4\ \&t\ 1)\ \&t\ 1) = -((Id\ \&t\ Id)\ \&t\ Id)$$
$$GTP{:}\text{-}gprod((1\ \&t\ e1)\ \&t\ 1, (1\ \&t\ e1)\ \&t\ 1) = (Id\ \&t\ Id)\ \&t\ Id$$
$$GTP{:}\text{-}gprod((1\ \&t\ e2)\ \&t\ 1, (1\ \&t\ e2)\ \&t\ 1) = -((Id\ \&t\ Id)\ \&t\ Id)$$
$$GTP{:}\text{-}gprod((1\ \&t\ e3)\ \&t\ 1, (1\ \&t\ e3)\ \&t\ 1) = -((Id\ \&t\ Id)\ \&t\ Id)$$
$$GTP{:}\text{-}gprod((1\ \&t\ e4)\ \&t\ 1, (1\ \&t\ e4)\ \&t\ 1) = -((Id\ \&t\ Id)\ \&t\ Id)$$
$$GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e1, (1\ \&t\ 1)\ \&t\ e1) = (Id\ \&t\ Id)\ \&t\ Id$$
$$GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e2, (1\ \&t\ 1)\ \&t\ e2) = -((Id\ \&t\ Id)\ \&t\ Id)$$
$$GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e3, (1\ \&t\ 1)\ \&t\ e3) = -((Id\ \&t\ Id)\ \&t\ Id)$$
$$GTP{:}\text{-}gprod((1\ \&t\ 1)\ \&t\ e4, (1\ \&t\ 1)\ \&t\ e4) = -((Id\ \&t\ Id)\ \&t\ Id)$$

We have explicitly shown that the generators A[1], A[5] and A[9] have squares equal to +1 while the remaining generators have squares equal to -1.

```
> map(grade,A); #all elements of A are of grade 1
```

$$[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$$

```
> map(tensorrank,A); #all elements of A are tensors of rank 3
```

$$[3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]$$

```
>
```

See Also: GTP:-`type/gradedmonom`, GTP:-gbasis, GTP:-`type/gradedodd`, GTP:-grade, GTP:-`&t`, Clifford:-`type/tensorprod`, GTP:-gradedprod, GTP:-`type/gradedeven`

Last revised: December 20, 2007 /RA/BF

Function: GTP:-grade - find the grade of any element of type 'cliscalar', 'climon', 'tensorprod', 'gradedmonom'

Calling Sequence:

grade(p);

Parameters:

p - element of one of these types: Clifford:-`type/cliscalar`, Clifford:-`type/climon`, Clifford:-`type/tensorprod`, GTP:-`type/gradedmonom`.

Description:

- Procedure 'grade' returns 0 of p is even and 1 if p is odd.
- When applied to homogeneous elements of the type GTP:-`type/tensorprod` or GTP:-`type/gradedmonom`, the procedure adds the grades of individual entries. For example, the grade of e1 &t e1 is 0 since e1 and e2 are odd elements, the grade of e1we2 &t e1 &t e2 is 0, while the grade of e1 &t Id is 1.
- Check also GTP:-`type/gradedodd` and GTP:-`type/gradedeven`.

Examples:

```
> restart:with(Clifford):with(GTP):
> grade(e1 &t e1),grade(e1we2 &t e1 &t e2),grade(e1 &t Id);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$0, 0, 1$$

```
> grade(2*e1we2);
```

$$0$$

```
> grade(1);
```

$$0$$

```
> grade(e1we2we3);
```

$$1$$

```
> grade(e1 &t e1we2 &t e3);
```

$$0$$

```
> grade(&t(e1,e2,e3,e4));
```

$$0$$

```
>
>
```

See Also: GTP:-`type/gradedmonom`, GTP:-gbasis, GTP:-`type/gradedodd`, GTP:-grade, GTP:-`&t`, Clifford:-`type/tensorprod`, GTP:-gradedprod, GTP:-`type/tensorrank`, GTP:-`type/gradedeven`

Last revised: December 20, 2007 /RA/BF

Function: GTP:-`type/gradedeven` - define type 'gradedeven'

Calling Sequence:

type(p,gradedeven);

Parameters:

p - element of one of these types: GTP:-`type/gradedpolynom`, GTP:-`type/gradedmonom`, Clifford:-`type/tensorprod`

Description:

- Polynomial elements in the graded tensor product Cl(B1) &t Cl(B2) &t ... &t &t Cl(Br) of r Clifford algebras Cl(Bi), where Bi are quadratic forms, 1<=i<=r, whose all monomial terms are of grade 0 are by definition of type *'gradedeven'*. Otherwise, if at least one monomial term is of grade 1, the entire polynomial is of type 'gradedodd'.
- Polynomials of type 'gradedeven' form a subalgebra in the graded tensor algebra Cl(B1) &t Cl(B2) &t ... &t &t Cl(Br) which is isomorphic with the even subalgebra of a Clifford algebra Cl(B) where B is an orthogonal sum of the quadratic forms B1, B2, ..., Br.
- See also GTP:-`type/gradedodd` and GTP:-grade.

Examples:

```
> restart:with(Clifford):with(GTP):eval(makealiases(5)):_prolevel
  :=true:
```

Example 1: Some type checking.

```
> p:=e1 &t e1 + 2* (e1we2 &t e2);
```

$$p := (e1 \ \&t \ e1) + 2\,(e12 \ \&t \ e2)$$

```
> type(p,gradedeven);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$false$$

```
> B:=linalg[diag](1,1,1):
  p1:=2*&t(e1we2,e2we3)-Pi*&t(e1,e2);p2:=2*(e1we2 &t
  (Id-e1we3+e2we3));
```

$$p1 := 2\,(e12 \ \&t \ e23) - \pi\,(e1 \ \&t \ e2)$$

$$p2 := 2\,(e12 \ \&t \ Id) - 2\,(e12 \ \&t \ e13) + 2\,(e12 \ \&t \ e23)$$

```
> type(p1,gradedeven),type(p2,gradedeven);
```

$$true, true$$

```
> type(gradedprod(p1,p2),gradedeven);
```

$$true$$

Example 2: Even basis monomials form a basis for an even subalgebra of Cl(B1) &t Cl(B2) &t ... &t &t Cl(Br), while the odd basis monomials give an even basis monomial when multiplied out.

```
> dim:=2:K:=cbasis(2):L:=gbasis(K$2);
```

$L := [Id \,\&t\, Id, Id \,\&t\, e1, Id \,\&t\, e2, Id \,\&t\, e12, e1 \,\&t\, Id, e1 \,\&t\, e1, e1 \,\&t\, e2, e1 \,\&t\, e12, e2 \,\&t\, Id, e2 \,\&t\, e1, e2 \,\&t\, e2, e2 \,\&t\, e12, e12 \,\&t\, Id, e12 \,\&t\, e1, e12 \,\&t\, e2, e12 \,\&t\, e12]$

```
> evenmonomials:=select(type,L,gradedeven);
```

$evenmonomials := [Id \,\&t\, Id, Id \,\&t\, e12, e1 \,\&t\, e1, e1 \,\&t\, e2, e2 \,\&t\, e1, e2 \,\&t\, e2, e12 \,\&t\, Id, e12 \,\&t\, e12]$

```
> oddmonomials:=select(type,L,gradedodd);
```

$oddmonomials := [Id \,\&t\, e1, Id \,\&t\, e2, e1 \,\&t\, Id, e1 \,\&t\, e12, e2 \,\&t\, Id, e2 \,\&t\, e12, e12 \,\&t\, e1, e12 \,\&t\, e2]$

We now compute all possible products of even and odd monomials, and store them in Se and So respectively. Then we check that the elements of Se and So are plus or minus the elements from 'evenmonomials'. In Seo we will store all products of even and odd elements. These elements should be either plus or minus the elements from 'oddmonomials':

```
> Se:={}:So:={}:Seo:={}:
  for i from 1 to nops(evenmonomials) do
  for j from 1 to nops(evenmonomials) do
      Se:=Se union {gprod(evenmonomials[i],evenmonomials[j])}:
      So:=So union {gprod(oddmonomials[i],oddmonomials[j])}:
      Seo:=Seo union {gprod(oddmonomials[i],evenmonomials[j])}:
      Seo:=Seo union {gprod(evenmonomials[i],oddmonomials[j])}:
  end do end do:
```

```
> Se,nops(Se);
```

$\{e2 \,\&t\, e2, e1 \,\&t\, e1, e1 \,\&t\, e2, e12 \,\&t\, Id, e12 \,\&t\, e12, -(e2 \,\&t\, e1), Id \,\&t\, e12, Id \,\&t\, Id, e2 \,\&t\, e1, -(e12 \,\&t\, Id), -(e2 \,\&t\, e2), -(Id \,\&t\, Id), -(e1 \,\&t\, e1), -(e1 \,\&t\, e2), -(Id \,\&t\, e12), -(e12 \,\&t\, e12)\}, 16$

```
> So,nops(So);
```

$\{e2 \,\&t\, e2, e1 \,\&t\, e1, e1 \,\&t\, e2, e12 \,\&t\, Id, e12 \,\&t\, e12, -(e2 \,\&t\, e1), Id \,\&t\, e12, Id \,\&t\, Id, e2 \,\&t\, e1, -(e12 \,\&t\, Id), -(e2 \,\&t\, e2), -(Id \,\&t\, Id), -(e1 \,\&t\, e1), -(e1 \,\&t\, e2), -(Id \,\&t\, e12), -(e12 \,\&t\, e12)\}, 16$

```
> Seo,nops(Seo);
```

$\{-(e2 \,\&t\, e12), e12 \,\&t\, e2, -(Id \,\&t\, e2), -(e1 \,\&t\, Id), Id \,\&t\, e1, Id \,\&t\, e2, e1 \,\&t\, Id, e12 \,\&t\, e1, -(e12 \,\&t\, e2), -(Id \,\&t\, e1), e1 \,\&t\, e12, e2 \,\&t\, Id, e2 \,\&t\, e12, -(e2 \,\&t\, Id), -(e12 \,\&t\, e1), -(e1 \,\&t\, e12)\}, 16$

Now we verify that elements in Se and So are of type 'gradedeven' while the elements in Seo are of type 'gradedodd':

```
> map(type,Se,gradedeven);
```

$$\{true\}$$

```
> map(type,So,gradedeven);
```

$$\{true\}$$

```
> map(type,Seo,gradedodd);
```

$$\{true\}$$

```
> 
> 
```

See Also: GTP:-`type/gradedmonom`, GTP:-gbasis, GTP:-`type/gradedodd`, GTP:-grade, GTP:-`&t`, Clifford:-`type/tensorprod`

Last revised: December 20, 2007 /RA/BF

Function: GTP:-`type/gradedmonom` - define a type 'gradedmonom'

Calling Sequence:

type(p,gradedmonom);

Parameters:

p - element of one of these types: `*`, function, algebraic.

Description:

- Monomial (homogeneous) elements in the graded tensor product Cl(B1) &t Cl(B2) &t ... &t Cl(Br) of r Clifford algebras Cl(Bi), where Bi are quadratic forms, 1<=i<=r, are by definition of type *'gradedmonom'*. Thus, they are either of type Clifford:-`type/tensorprod` or they are products of two elements, one of type Clifford:-`type/tensorprod` and one of type Clifford:-`type/cliscalar`.
- See also GTP:-`type/gradedpolynom`.

Examples:

```
> restart:with(Clifford):with(GTP):
> type(e1 &t e1,gradedmonom),type(Pi*(e1we2 &t e1 &t
  e2),gradedmonom);
```

true, *true*

```
> type(2*&t(e1,e2,e3),gradedmonom);
```

true

```
> type(2*&t(e1,e2,e3)+e2we3 &t e2we1,gradedmonom);
```

false

```
> type(2*&t(e1,e2,e3)+e2we3 &t e2we1,gradedpolynom);
```

true

```
> dim:=2:K:=cbasis(2):L:=gbasis(K$2);
```

$L := [Id \,\&t\, Id, Id \,\&t\, e1, Id \,\&t\, e2, Id \,\&t\, e1we2, e1 \,\&t\, Id, e1 \,\&t\, e1, e1 \,\&t\, e2, e1 \,\&t\, e1we2, e2 \,\&t\, Id, e2 \,\&t\, e1, e2 \,\&t\, e2, e2 \,\&t\, e1we2, e1we2 \,\&t\, Id, e1we2 \,\&t\, e1, e1we2 \,\&t\, e2, e1we2 \,\&t\, e1we2]$

```
> map(type,L,gradedmonom);
```

[*true*, *true*, *true*, *true*, *true*, *true*, *true*, *true*, *true*, *true*, *true*, *true*, *true*, *true*, *true*, *true*]

```
>
>
```

See Also: GTP:-gbasis, GTP:-`type/gradedodd`, GTP:-grade, GTP:-`&t`, Clifford:-`type/tensorprod`, GTP:-gradedprod, GTP:-gprod, GTP:-`type/gradedeven`

Last revised: December 20, 2007 /RA/BF

Function: GTP:-`type/gradedodd` - define type 'gradedodd'

Calling Sequence:

type(p,gradedodd);

Parameters:

p - element of one of these types: GTP:-`type/gradedpolynom`, GTP:-`type/gradedmonom`, Clifford:-`type/tensorprod`

Description:

- Polynomial elements in the graded tensor product Cl(B1) &t Cl(B2) &t ... &t &t Cl(Br) of r Clifford algebras Cl(Bi), where Bi are quadratic forms, 1<=i<=r, which contain all monomials of grade 1 are of type *'gradedodd'*.
- Polynomials of type 'gradedodd' do not form a subalgebra in the graded tensor algebra Cl(B1) &t Cl(B2) &t ... &t Cl(Br). However, a product of two odd elements is even.
- See also GTP:-`type/gradedeven` and GTP:-grade.

Examples:

```
> restart:with(Clifford):with(GTP):
> p1:=e1 &t e1 &t e2+ 2* (e1 &t e2 &t e2);
```

$$p1 := ((e1 \,\&t\, e1) \,\&t\, e2) + 2\,((e1 \,\&t\, e2) \,\&t\, e2)$$

```
> type(p1,gradedeven),type(p1,gradedodd);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$false, true$$

```
> p2:=2*&t(e1,e2,e3);type(p2,gradedodd);
```

$$p2 := 2\,((e1 \,\&t\, e2) \,\&t\, e3)$$

$$true$$

```
> B:=linalg[diag](1,1,1):type(gradedprod(p1,p2),gradedeven);
```

$$true$$

```
> dim:=2:K:=cbasis(2):L:=gbasis(K$2);
```

$$L := [Id \,\&t\, Id, Id \,\&t\, e1, Id \,\&t\, e2, Id \,\&t\, e1we2, e1 \,\&t\, Id, e1 \,\&t\, e1, e1 \,\&t\, e2, e1 \,\&t\, e1we2, e2 \,\&t\, Id, e2 \,\&t\, e1, e2 \,\&t\, e2, e2 \,\&t\, e1we2, e1we2 \,\&t\, Id, e1we2 \,\&t\, e1, e1we2 \,\&t\, e2, e1we2 \,\&t\, e1we2]$$

```
> map(type,L,gradedodd);
```

$$[false, true, true, false, true, false, false, true, true, false, false, true, false, true, true, false]$$

```
> map(type,L,gradedeven);
```

$$[true, false, false, true, false, true, true, false, false, true, true, false, true, false, false, true]$$

```
>
>
```

See Also: GTP:-`type/gradedmonom`, GTP:-gbasis, GTP:-grade, GTP:-`&t`, Clifford:-`type/tensorprod`, GTP:-gradedprod, GTP:-gprod, GTP:-`type/gradedeven`

Last revised: December 20, 2007 /RA/BF

Function: GTP:-`type/gradedpolynom` - define a type 'gradedpolynom'

Calling Sequence:

type(p,gradedpolynom);

Parameters:

p - element of one of these types: `*`, `+`, 'function', or 'algebraic'.

Description:

- Polynomial elements in the graded tensor product Cl(B1) &t Cl(B2) &t ... &t &t Cl(Br) of r Clifford algebras Cl(Bi), where Bi are quadratic forms, 1<=i<=r, are by definition of type *'gradedpolynom'*. Thus, they are linear combinations of the basis elements of the type Clifford:-`type/tensorprod` while their coefficients are of the type Clifford:-`type/cliscalar`.
- See also GTP:-`type/gradedmonom`. Elements of type 'gradedmonom' are also of type 'gradedpolynom'.
- Elements of the type 'gradedpolynom' are multivariate polynomials used by the procedures GTP:-gprod and GTP:-gradedprod.

Examples:

```
> restart:with(Clifford):with(GTP):
> type(e1 &t e1 + 2* (e1we2 &t e2),gradedpolynom),
  type(Pi*(e1we2 &t e1 &t e2),gradedpolynom);
```

$$true, true$$

```
> type(2*&t(e1,e2,e3),gradedpolynom);
```

$$true$$

```
> dim:=2:K:=cbasis(2):L:=gbasis(K$2);
```

$$L := [Id \,\&t\, Id, Id \,\&t\, e1, Id \,\&t\, e2, Id \,\&t\, e1we2, e1 \,\&t\, Id, e1 \,\&t\, e1, e1 \,\&t\, e2, e1 \,\&t\, e1we2, e2 \,\&t\, Id, e2 \,\&t\, e1, e2 \,\&t\, e2, e2 \,\&t\, e1we2, e1we2 \,\&t\, Id, e1we2 \,\&t\, e1, e1we2 \,\&t\, e2, e1we2 \,\&t\, e1we2]$$

```
> map(type,L,gradedpolynom);
```

$$[true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true]$$

```
>
>
```

See Also: GTP:-`type/gradedmonom`, GTP:-gbasis, GTP:-`type/gradedodd`, GTP:-grade, GTP:-`&t`, Clifford:-`type/tensorprod`, GTP:-gradedprod, GTP:-gprod, GTP:-`type/gradedeven`

Last revised: December 20, 2007 /RA/BF

Function: GTP:-gradedprod - compute a graded product of elements of the type 'tensorprod', 'gradedmonom', or 'gradedpolynom'

Calling Sequence:

gradedprod(p1,p2);
gradedprod(p1,p2,B1,B2,...,Br);

Parameters:

p1, p2 - graded polynomials of type GTP:-`type/gradedpolynom` of rank r
B1,B2,...,Br - (optional) sequence of r diagonal forms Bi, 1<=i<=r, where r is the tensor rank of p1 and p2

Description:

- Procedure 'gradedprod' is an extension of GTP:-prod and computes a product of two graded polynomials in the graded tensor product Cl(B1) &t Cl(B2) &t ... &t Cl(Br) of r Clifford algebras Cl(B1), Cl(B2), ... , Cl(Br). In particular, it can handle also monomials.
- When the optional sequence is used, Clifford products are computed component-wise on homogeneous elements in Cl(B1), Cl(B2), ... , Cl(Br) with a help of the procedure GTP:-cmulB. However, the Z2-gradation is taken into consideration in order to assure, for example, that elements of the type e1 &t 1 and 1 &t e1 belonging to Cl(B1) &t Cl(B2) anticommute.
- For more information how multiplication is defined on homogeneous elements in the graded tensor product Cl(B1) &t Cl(B2) of two Clifford algebras Cl(B1) and Cl(B2) see GTP:-gprod. This definition is extended to tensors of higher ranks and then to non-homogeneous tensors by linearity.
- When the optional sequence is not used, the default bilinear form B is applied. Thus, in this case, the r products will be computed in r different copies of Cl(B).
- The ranks of p1 and p2 must be the same. They can be found with GTP:-tensorrank.

Examples:

```
> restart:with(Clifford):with(GTP):eval(makealiases(5)):_prolevel
  :=true:
```

Example 1:

```
> B:=linalg[diag](1,-1,-1,-1):
> type(e1 &t e2,tensorprod),
  type(2*(e1 &t e3),gradedmonom),
  type(2*&t(e1,e2,e3),gradedmonom);
  type((e1 &t e2) + b*(e1 &t e2) +e2 &t e3,gradedpolynom);
```

true, true, true

true

```
> gradedprod(e1 &t e2 &t e2,e1 &t e2 &t e2);tensorrank(%);
```

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

$$-((Id \,\&t\, Id) \,\&t\, Id)$$

$$3$$

```
> gradedprod((e1 + e2) &t e2 &t Id,e1 &t e2 &t Id);tensorrank(%);
```

$$((Id \,\&t\, Id) \,\&t\, Id) - ((e12 \,\&t\, Id) \,\&t\, Id)$$

$$3$$

```
> p:=a*(Id &t Id &t e2)-3*(e1we2 &t e2 &t e2we1 )+b*4*(Id &t e2
  &t e1)-
      Id &t e1we2 &t e1we2;
```

$$p := a\,((Id \,\&t\, Id) \,\&t\, e2) - 3\,((e12 \,\&t\, e2) \,\&t\, e21) + 4\,b\,((Id \,\&t\, e2) \,\&t\, e1) - ((Id \,\&t\, e12) \,\&t\, e12)$$

```
> type(p,gradedpolynom);tensorrank(p);
```

$$true$$

$$3$$

```
> r1:=gradedprod(-a*p,2*p);
```

$$r1 := 2\,a^3\,((Id \,\&t\, Id) \,\&t\, Id) + 12\,a^2\,((e12 \,\&t\, e2) \,\&t\, e1) - 16\,a^2\,b\,((Id \,\&t\, e2) \,\&t\, e12) + 16\,a\,((Id \,\&t\, Id) \,\&t\, Id) - 48\,a\,b\,((e12 \,\&t\, Id) \,\&t\, e2) - 32\,a\,b^2\,((Id \,\&t\, Id) \,\&t\, Id) + 16\,a\,b\,((Id \,\&t\, e1) \,\&t\, e2)$$

```
> tensorrank(r1);
```

$$3$$

The results from 'gradedprod' may be collected using GTP:-gcollect:

```
> gcollect(r1);
```

$$2\,a\,(8 + a^2 - 16\,b^2)\,((Id \,\&t\, Id) \,\&t\, Id) - 48\,a\,b\,((e12 \,\&t\, Id) \,\&t\, e2) + 12\,a^2\,((e12 \,\&t\, e2) \,\&t\, e1) - 16\,a^2\,b\,((Id \,\&t\, e2) \,\&t\, e12) + 16\,a\,b\,((Id \,\&t\, e1) \,\&t\, e2)$$

Example 2: In Example 1, r1 is an element of C(B) &t Cl(B) &t Cl(B) which is Z2-isomorphic with Cl(3B) where 3B is a quadratic form of signature (3,9). Now use different quadratic forms in each component. For example, consider p defined above as an element of the algebra Cl(B) &t Cl(B1) &t Cl(B2) where B1 and B2 are defined below.

```
> B1:=linalg[diag](-1,1,-1,1):B2:=linalg[diag](1,1,-1,1):
```

Now, we compute the graded product of p with p. Notice, that the result r2 is different than r1:

```
> r2:=gradedprod(p,p,B,B1,B2);
```

$$r2 := a^2\,((Id \,\&t\, Id) \,\&t\, Id) + 6\,a\,((e12 \,\&t\, e2) \,\&t\, e1) + 8\,a\,b\,((Id \,\&t\, e2) \,\&t\, e12) - 10\,((Id \,\&t\, Id) \,\&t\, Id) - 24\,b\,((e12 \,\&t\, Id) \,\&t\, e2) - 16\,b^2\,((Id \,\&t\, Id) \,\&t\, Id) + 8\,b\,((Id \,\&t\, e1) \,\&t\, e2)$$

```
> gcollect(r2);
```

$$(-10 + a^2 - 16\,b^2)\,((Id \,\&t\, Id) \,\&t\, Id) - 24\,b\,((e12 \,\&t\, Id) \,\&t\, e2) + 6\,a\,((e12 \,\&t\, e2) \,\&t\, e1) + 8\,a\,b\,((Id \,\&t\, e2) \,\&t\, e12) + 8\,b\,((Id \,\&t\, e1) \,\&t\, e2)$$

Example 3: Some more computations.

```
> p:=gradedprod(e1 &t e2,e2 &t e1we2-2*Pi*cos(alpha)*e1we2 &t Id
  -(a+b)/(a-b)*e2we1 &t e1);
```

$$p := -(e12 \,\&t\, e1) - 2\,\pi\cos(\alpha)\,(e2 \,\&t\, e2) - \frac{a\,(e2 \,\&t\, e12)}{a-b} - \frac{b\,(e2 \,\&t\, e12)}{a-b}$$

```
> gcollect(p);
```

$$-(e12 \,\&t\, e1) - 2\,\pi\cos(\alpha)\,(e2 \,\&t\, e2) - \frac{(a+b)\,(e2 \,\&t\, e12)}{a-b}$$

```
> p1:=&t(-Pi*cos(alpha)*e1,e2-e2we1,Id,2*e1);
  p2:=&t(Id,e1-3*(a-b)/(a+c)*e1we2,e2we1,1);
```

$$p1 := -2\,\pi\cos(\alpha)\,(((e1 \,\&t\, e2) \,\&t\, Id) \,\&t\, e1) + 2\,\pi\cos(\alpha)\,(((e1 \,\&t\, e21) \,\&t\, Id) \,\&t\, e1)$$

$$p2 := (((Id \,\&t\, e1) \,\&t\, e21) \,\&t\, 1) - \frac{3\,(a-b)\,(((Id \,\&t\, e12) \,\&t\, e21) \,\&t\, 1)}{a+c}$$

```
> tensorrank(p1),tensorrank(p2);
```

$$4, 4$$

```
> r3:=gradedprod(p1,p2,B2,B1,B,B2);
```

$$r3 := 2\,\pi\cos(\alpha)\,(((e1 \,\&t\, e12) \,\&t\, e12) \,\&t\, e1) + \frac{6\,\pi\cos(\alpha)\,a\,(((e1 \,\&t\, e1) \,\&t\, e12) \,\&t\, e1)}{a+c} - \frac{6\,\pi\cos(\alpha)\,b\,(((e1 \,\&t\, e1) \,\&t\, e12) \,\&t\, e1)}{a+c} - 2\,\pi\cos(\alpha)\,(((e1 \,\&t\, e2) \,\&t\, e12) \,\&t\, e1) - \frac{6\,\pi\cos(\alpha)\,a\,(((e1 \,\&t\, Id) \,\&t\, e12) \,\&t\, e1)}{a+c} + \frac{6\,\pi\cos(\alpha)\,b\,(((e1 \,\&t\, Id) \,\&t\, e12) \,\&t\, e1)}{a+c}$$

```
> gcollect(r3);
```

$$-2\,\pi\cos(\alpha)\,(((e1 \,\&t\, e2) \,\&t\, e12) \,\&t\, e1) - \frac{6\,\pi\cos(\alpha)\,(a-b)\,(((e1 \,\&t\, Id) \,\&t\, e12) \,\&t\, e1)}{a+c} + \frac{6\,\pi\cos(\alpha)\,(a-b)\,(((e1 \,\&t\, e1) \,\&t\, e12) \,\&t\, e1)}{a+c} + 2\,\pi\cos(\alpha)\,(((e1 \,\&t\, e12) \,\&t\, e12) \,\&t\, e1)$$

Thus, element r3 belongs to an algebra Z2-isomorphic with Cl(B2,B1,B,B2), that is, Cl(B3) where B3 is a quadratic form of signature (9,7).

```
>
>
```

See Also: GTP:-`type/gradedmonom`, GTP:-gbasis, GTP:-`type/gradedodd`, GTP:-grade, GTP:-`&t`, Clifford:-`type/tensorprod`, GTP:-`type/tensorrank`, GTP:-`type/gradedeven`

Last revised: December 20, 2007 /RA/BF

Function: Octonion:-oversion - display information about the current version of the 'Octonion' package

Calling Sequence:

oversion();

Parameters:

no parameters needed

Description:

- Procedure 'oversion' displays information about the current version of the 'Octonion' package.
- The 'Octonion' package must be loaded after the 'CLIFFORD' package has been loaded. Therefore, in order to avoid confusion with the procedure Clifford:-version, this procedure is called 'oversion'.
- To display 'CLIFFORD' and 'Octonion' environmental variables, use procedure Clifford:-CLIFFORD_ENV.
- To multiply octonionic matrices, see Clifford:-rmulm.

Examples:

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

[Φ, *associator*, *commutator*, *def_omultable*, *o_conjug*, *oinv*, *omul*, *omultable*, *onorm*, *oversion*, *purevectorpart*, *realpart*]

```
> version(); #current version of CLIFFORD
```

++

CLIFFORD - A Maple 11 Package for Clifford Algebras with "Bigebra"

(Version 10 with environmental variables given by CLIFFORD_ENV())

Last revised: December 20, 2007 (Source file: clifford_M11_08.mws)

Copyright 1995-2008 by Rafal Ablamowicz (*) and Bertfried Fauser ($)

(*) Department of Mathematics, Box 5054

Tennessee Technological University, Cookeville, TN 38505

tel: USA (931) 372-3569, fax: USA (931) 372-6353

rablamowicz@tntech.edu

http://math.tntech.edu/rafal/Cliff8/

($) Universit"at Konstanz, Fachbereich Physik, Fach M678

78457 Konstanz, Germany

Bertfried.Fauser@uni-konstanz.de

http://kaluza.physik.uni-konstanz.de/~fauser/

If you are a Clifford algebra pro, assign 'true' to '_prolevel' and see
how much faster your computations will be! But watch your syntax!
Use 'useproduct' to change value of _default_Clifford_product in Cl(B) from
cmulRS when B is symbolic to cmulNUM when B is numeric. Type ?cmul for help.
Type CLIFFORD_ENV() to see current values of environmental variables.
++++++++++++This is CLIFFORD version 11++++++++++++

```
> oversion(); #current version of Octonion
```

++
'Octonion' - A Maple 11 Package for Computations with Octonions (version 11)
Last revised: December 20, 2007
Copyright 1995-2008, by Rafal Ablamowicz, Tennessee Technological University
Department of Mathematics, Box 5054
Tennessee Technological University, Cookeville, TN 38505
phone: USA (931) 372-3569, fax: USA (931) 372-6353
e-mail: rablamowicz@tntech.edu
http://math.tntech.edu/rafal/
++

```
>
>
```

See Also: omul

Last modified: December 20, 2007, RA/BF.

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

Calling Sequence:

none

Parameters:

none

Description:

- Procedure 'setup' is the initialization procedure for the 'Octonion' package. It is executed automatically when the package is loaded.
- At the time of loading, the following are defined:

  - `&o` - infix form for omul, the octonionic multiplication
  - _octbasis = [Id, e1, e2, e3, e4, e5, e6, e7] - standard octonion basis as Maple global variable in Cl(0,7)
  - _pureoctbasis = [e1, e2, e3, e4, e5, e6, e7] - pure octonion basis as Maple global variable in Cl(0,7)
  - _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 Fano triples that define octonionic multiplication
  - _default_squares = [-Id, -Id, -Id, -Id, -Id, -Id, -Id] - default squares of the pure octonionic basis

- To see all environmental variables that are defined and used by 'CLIFFORD', use procedure Clifford:-CLIFFORD_ENV.
- All procedures and types in 'Octonion' are protected.

Examples:

```
> restart:with(Clifford):with(Octonion):
> CLIFFORD_ENV();
```

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

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


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


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



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



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


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

>

>

See Also: `type/Fano_triples`, omultable, omul

Last modified: December 20, 2007, RA/BF.

Function: Octonion:-associator - returns the associator value of three octonions,
Octonion:-commutator - returns the commutator value of two octonions
Octonion:-Phi - associative 3-form of three octonions

Calling Sequence:

associator(p1,p2,p3);
commutator(p1,p2);
Phi(p1,p2,p2);

Parameters:

p1, p2, p3 - polynomials of type 'octonion'

Description:

- The associator of three octonions p1, p2, and p3 is defined as:

  associator(p1,p2,p3) = (p1 &o p2) &o p3 - p1 &o (p2 &o p3).

- The commutator of two octonions p1 and p2 is defined as:

  commutator(p1,p2) = p1 &o p2 - p2 &o p1.

- The associative 3-form Phi of three octonions is defined as:

  Phi(p1,p2,p3) = (1/2)*realpart(p1 &o (p2_bar &o p3) - p3 &o (p2_bar &o p1))

  where p2_bar = o_conjug(p2).

- For information about type 'octonion' see `type/octonion`.

Examples:

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

$[\Phi, associator, commutator, def\_omultable, o\_conjug, oinv, omul, omultable, onorm, oversion, purevectorpart, realpart]$

```
> p1:=1-2*e1+e4+3*e6-e7;p2:=2-e1+e3+2*e6-e7;p3:=2*e2+e3+3*e5-e6;
```

$$p1 := 1 - 2\,e1 + e4 + 3\,e6 - e7$$
$$p2 := 2 - e1 + e3 + 2\,e6 - e7$$
$$p3 := 2\,e2 + e3 + 3\,e5 - e6$$

```
> type(p1,octonion);type(p2,octonion);type(p3,octonion);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$true$$

$$true$$

$$true$$

Octonion multiplication is not associative:

```
> associator(p1,p2,p3);
```

$$-8\,e1 - 2\,e2 + 20\,e3 + 14\,e4 - 6\,e5 - 2\,e6 + 24\,e7$$

However, when p1, p2, and p3 are considered as elements in the Clifford algebra Cl(0,7), which is associative, we get:

```
> (p1 &c p2) &c p3 - p1 &c (p2 &c p3);
```

$$0$$

```
> commutator(p1,p2);
```

$$2\,e1 - 4\,e2 + 2\,e3 + 6\,e4 + 4\,e5 - 2\,e6 - 4\,e7$$

```
> Phi(p1,p2,p3);
```

$$4$$

```
>
```

See Also: Clifford:-`&c`, def_omultable, omultable, omul

Last modified: December 20, 2007, RA/BF.

Function: Octonion:-associator - returns the associator value of three octonions,
Octonion:-commutator - returns the commutator value of two octonions
Octonion:-Phi - associative 3-form of three octonions

Calling Sequence:

associator(p1,p2,p3);
commutator(p1,p2);
Phi(p1,p2,p2);

Parameters:

p1, p2, p3 - polynomials of type 'octonion'

Description:

- The associator of three octonions p1, p2, and p3 is defined as:

  associator(p1,p2,p3) = (p1 &o p2) &o p3 - p1 &o (p2 &o p3).

- The commutator of two octonions p1 and p2 is defined as:

  commutator(p1,p2) = p1 &o p2 - p2 &o p1.

- The associative 3-form Phi of three octonions is defined as:

  Phi(p1,p2,p3) = (1/2)*realpart(p1 &o (p2_bar &o p3) - p3 &o (p2_bar &o p1))

  where p2_bar = o_conjug(p2).

- For information about type 'octonion' see `type/octonion`.

Examples:

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

$[\Phi, associator, commutator, def\_omultable, o\_conjug, oinv, omul, omultable, onorm, oversion, purevectorpart, realpart]$

```
> p1:=1-2*e1+e4+3*e6-e7;p2:=2-e1+e3+2*e6-e7;p3:=2*e2+e3+3*e5-e6;
```

$$p1 := 1 - 2\,e1 + e4 + 3\,e6 - e7$$
$$p2 := 2 - e1 + e3 + 2\,e6 - e7$$
$$p3 := 2\,e2 + e3 + 3\,e5 - e6$$

```
> type(p1,octonion);type(p2,octonion);type(p3,octonion);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$true$$

$$true$$

$$true$$

Octonion multiplication is not associative:

```
> associator(p1,p2,p3);
```

$$-8\,e1 - 2\,e2 + 20\,e3 + 14\,e4 - 6\,e5 - 2\,e6 + 24\,e7$$

However, when p1, p2, and p3 are considered as elements in the Clifford algebra Cl(0,7), which is associative, we get:

```
> (p1 &c p2) &c p3 - p1 &c (p2 &c p3);
```

$$0$$

```
> commutator(p1,p2);
```

$$2\,e1 - 4\,e2 + 2\,e3 + 6\,e4 + 4\,e5 - 2\,e6 - 4\,e7$$

```
> Phi(p1,p2,p3);
```

$$4$$

```
>
>
```

See Also: Clifford:-`&c`, def_omultable, omultable, omul

Last modified: December 20, 2007, RA/BF.

Function: Octonion:-def_omultable - define octonionic multiplication table

Calling Sequence:

def_omultable(F);

Parameters:

F - a list of type `Fano_triples`

Description:

- Procedure 'def_omultable' allows user to define an octonionic multiplication table which could be different than the default one.
- The default multiplication table is initialized at the time when the 'OCTONION' package is being loaded. It can also be re-defined by issuing the following command:

  ```
  >def_omultable(_default_Fano_triples);
  ```

  where _default_Fano_triples is a global list with default Fano triples. See `type/Fano_triples` for more information.
- Use omultable to display currently defined multiplication table.
- Use Clifford:-CLIFFORD_ENV to display current environmental variables used by 'CLIFFORD' and 'Octonion'.

Examples:

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

$$[\Phi, associator, commutator, def\_omultable, o\_conjug, oinv, omul, omultable, onorm, oversion, purevectorpart, realpart]$$

```
> omultable(); #default multiplication table
```

$$\begin{bmatrix} -Id & e4 & e7 & -e2 & e6 & -e5 & -e3 \\ -e4 & -Id & e5 & e1 & -e3 & e7 & -e6 \\ -e7 & -e5 & -Id & e6 & e2 & -e4 & e1 \\ e2 & -e1 & -e6 & -Id & e7 & e3 & -e5 \\ -e6 & e3 & -e2 & -e7 & -Id & e1 & e4 \\ e5 & -e7 & e4 & -e3 & -e1 & -Id & e2 \\ e3 & e6 & -e1 & e5 & -e4 & -e2 & -Id \end{bmatrix}$$

For example, we get the first row as follows:

```
> seq(e1 &o e||i,i=1..7);
```

$$-Id, e4, e7, -e2, e6, -e5, -e3$$

The second row we get as follows:

```
> seq(e2 &o e||i,i=1..7);
```

$$-e4, -Id, e5, e1, -e3, e7, -e6$$

and so on.

Multiplication table can be erased as follows:

```
> subsop(4=NULL,eval(omul)):
> omultable();
Octonion multiplication table is not currently defined. Use 'def_omultable' to
define a new table.
```

Finally, we re-initialize the table using the default Fano triples:

```
> _default_Fano_triples;
```

$$[[1, 3, 7], [1, 2, 4], [1, 5, 6], [2, 3, 5], [2, 6, 7], [3, 4, 6], [4, 5, 7]]$$

```
> def_omultable(_default_Fano_triples);
> omultable();
```

$$\begin{bmatrix} -Id & e4 & e7 & -e2 & e6 & -e5 & -e3 \\ -e4 & -Id & e5 & e1 & -e3 & e7 & -e6 \\ -e7 & -e5 & -Id & e6 & e2 & -e4 & e1 \\ e2 & -e1 & -e6 & -Id & e7 & e3 & -e5 \\ -e6 & e3 & -e2 & -e7 & -Id & e1 & e4 \\ e5 & -e7 & e4 & -e3 & -e1 & -Id & e2 \\ e3 & e6 & -e1 & e5 & -e4 & -e2 & -Id \end{bmatrix}$$

However, the following is another valid list of Fano triples:

```
> new_Fano_triples:=[[6,2,5],[6,3,4],[6,7,1],[2,3,7],[3,1,5],[2,4
,1],[4,5,7]];
```

$$new\_Fano\_triples := [[6, 2, 5], [6, 3, 4], [6, 7, 1], [2, 3, 7], [3, 1, 5], [2, 4, 1], [4, 5, 7]]$$

```
> type(new_Fano_triples,Fano_triples);
```

$$true$$

```
> def_omultable(new_Fano_triples);
> omultable();
```

$$\begin{bmatrix} -Id & e4 & -e5 & -e2 & e3 & e7 & -e6 \\ -e4 & -Id & e7 & e1 & e6 & -e5 & -e3 \\ e5 & -e7 & -Id & e6 & -e1 & -e4 & e2 \\ e2 & -e1 & -e6 & -Id & e7 & e3 & -e5 \\ -e3 & -e6 & e1 & -e7 & -Id & e2 & e4 \\ -e7 & e5 & e4 & -e3 & -e2 & -Id & e1 \\ e6 & e3 & -e2 & e5 & -e4 & -e1 & -Id \end{bmatrix}$$

which is a different multiplication table than before.

```
>
```

See Also: `type/Fano_triples`, omultable, omul

Last modified: December 20, 2007, RA/BF.

Function: Octonion:-`type/Fano_triples` - a list of lists used to define octonionic multiplication table

Calling Sequence:

type(F,Fano_triples);

Parameters:

F - a list of lists

Description:

- A list of lists F is of type 'Fano_triples' if:
  (1) the list F contains seven lists F1, F2, F3, F4, F5, F6, and F7;
  (2) each of the seven lists F1, ... , F7 contains three integers from the set {1,2,3,4,5,6,7};
  (3) each of the seven integers {1,2,3,4,5,6,7} appears in exactly three of the seven lists F1, ... , F7.
- A default list of Fano triples is stored in a global list _default_Fano_triples.
- A valid list of seven Fano triples may be used to label seven points and seven lines in the Fano plane F_2.
- The set of integers {1,2,3,4,5,6,7} is used because we use {e1,e2,e3,e4,e5,e6,e7} for the pure octonion basis.
- If [i,j,k] is one of the seven valid Fano triples F1, ... , F7, then:
  (1) omul(ei,ej) = ek, omul(ej,ek) = ei, omul(ek,ei) = ej;
  (2) omul(ej,ei) = -ek, omul(ek,ej) = -ei, omul(ei,ek) = -ej;
- The default multiplication table is initialized at the time when the 'Octonion' package is being loaded. It can also be re-defined by issuing the following command:

  ```
  >def_omultable(_default_Fano_triples);
  ```

  where _default_Fano_triples is a global list with default Fano triples. See `type/Fano_triples` for more information.
- Use omultable to display currently defined multiplication table.
- See omul for octonionic multiplication.
- To display all environmental variables used by 'CLIFFORD' and 'Octonion' packages, use Clifford:-CLIFFORD_ENV.

Examples:

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

[Φ, *associator*, *commutator*, *def_omultable*, *o_conjug*, *oinv*, *omul*, *omultable*, *onorm*, *oversion*, *purevectorpart*, *realpart*]

```
> _default_Fano_triples;#default Fano triples
```

$$[[1, 3, 7], [1, 2, 4], [1, 5, 6], [2, 3, 5], [2, 6, 7], [3, 4, 6], [4, 5, 7]]$$

For example, the first list implies the following about {e1,e3,e7}:

```
> 'omul(e1,e3)'=omul(e1,e3);'omul(e3,e7)'=omul(e3,e7);'omul(e7,e1
  )'=omul(e7,e1);
```

$$Octonion{:}\text{-}omul(e1, e3) = e7$$
$$Octonion{:}\text{-}omul(e3, e7) = e1$$
$$Octonion{:}\text{-}omul(e7, e1) = e3$$

and

```
> 'omul(e3,e1)'=omul(e3,e1);'omul(e7,e3)'=omul(e7,e3);'omul(e1,e7
  )'=omul(e1,e7);
```

$$Octonion{:}\text{-}omul(e3, e1) = -e7$$
$$Octonion{:}\text{-}omul(e7, e3) = -e1$$
$$Octonion{:}\text{-}omul(e1, e7) = -e3$$

and so on.

However, the following is another valid list of Fano triples:

```
> new_Fano_triples:=[[6,2,5],[6,3,4],[6,7,1],[2,3,7],[3,1,5],[2,4
  ,1],[4,5,7]];
```

$$new\_Fano\_triples := [[6, 2, 5], [6, 3, 4], [6, 7, 1], [2, 3, 7], [3, 1, 5], [2, 4, 1], [4, 5, 7]]$$

```
> type(new_Fano_triples,Fano_triples);
```

$$true$$

while the following is not:

```
> another_Fano_triples:=
  [[4,2,5],[6,3,4],[6,7,1],[2,3,7],[3,1,5],[2,4,1],[4,5,7]];
```

$$another\_Fano\_triples := [[4, 2, 5], [6, 3, 4], [6, 7, 1], [2, 3, 7], [3, 1, 5], [2, 4, 1], [4, 5, 7]]$$

```
> type(another_Fano_triples,Fano_triples);
```

$$false$$

The reason is that '4' appears in four lists.

```
>
```

See Also: def_omultable, omultable, omul

Last modified: December 20, 2007, RA/BF.

Function: Octonion:-o_conjug - octonionic conjugation in the octonionic algebra

Calling Sequence:

o_conjug(o);

Parameters:

o - expression of the type 'octonion'

Description:

- Procedure 'o_conjug' computes octonionic conjugation in the octonionic algebra:

$$\text{o\_conjug}(x0 + \mathbf{x}) = x0 - \mathbf{x}$$

- where x0 is a real number and **x** = x1*e1+x2*e2+x3*e3+x4*e4+x5*e5+x6*e6+x7*e7.
- Conjugation is an anti-automorphism of the octonionic algebra. This means that o_conjug(o1 &o o2) = o_conjug(o2) &o o_conjug(o1).
- For information about type 'octonion' see `type/octonion`.

Examples:

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

[Φ, *associator*, *commutator*, *def_omultable*, *o_conjug*, *oinv*, *omul*, *omultable*, *onorm*, *oversion*, *purevectorpart*, *realpart*]

```
> o1:=x0+add(x||i*e||i,i=1..7);
```

$$o1 := x0 + x1\ e1 + x2\ e2 + x3\ e3 + x4\ e4 + x5\ e5 + x6\ e6 + x7\ e7$$

```
> o2:=y0+add(y||i*e||i,i=1..7);
```

$$o2 := y0 + y1\ e1 + y2\ e2 + y3\ e3 + y4\ e4 + y5\ e5 + y6\ e6 + y7\ e7$$

```
> L:=o_conjug(omul(o1,o2)):
```

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

```
> R:=omul(o_conjug(o2),o_conjug(o1)):
> simplify(L-R);
```

$$0$$

Any octonion o1 times its conjugate is a scalar:

```
> o1inv:=o_conjug(o1);
```

$$o1inv := x0 - x1\ e1 - x2\ e2 - x3\ e3 - x4\ e4 - x5\ e5 - x6\ e6 - x7\ e7$$

```
> o1 &o o1inv;
```

$$x1^2\ Id + x2^2\ Id + x3^2\ Id + x4^2\ Id + x5^2\ Id + x6^2\ Id + x7^2\ Id + x0^2\ Id$$

```
> realpart(%);
```

$$x1^2 + x2^2 + x3^2 + x4^2 + x5^2 + x6^2 + x7^2 + x0^2$$

Since octonions are treated as paravectors in the Clifford algebra Cl(0,7), octonionic conjugate of any octonion can be obtained also by taking grade involution:

```
> gradeinv(o1);
```

$$x0\ Id - x1\ e1 - x2\ e2 - x3\ e3 - x4\ e4 - x5\ e5 - x6\ e6 - x7\ e7$$

However, grade involution in Cl(0,7) is not an antiautomorphism of Cl(0,7): it is an automorphism of Cl(0,7). Note: in the above output, the unit element in Cl(0,7) is denoted as 'Id'.

```
>
```

See Also: omul, oinv, Clifford:-q_conjug, Clifford:-conjugation, Clifford:-gradeinv, realpart

Last modified: December 20, 2007, RA/BF.

Function: Octonion:-`type/octonion` - type octonion

Calling Sequence:

type(p,octonion);

Parameters:

p - an expression of type 'algebraic'

Description:

- Any polynomial p expressible as follows

  p = x0+x1*e1+x2*e2+x3*e3+x4*e4+x5*e5+x6*e6+x7*e7 or p = x0*Id+x1*e1+x2*e2+x3*e3+x4*e4+x5*e5+x6*e6+x7*e7

  is of type 'octonion'.
- The unit element in the octonion algebra may be entered as 1 or as 'Id'. In some computations 'Id' will be returned.
- Use omultable to display currently defined multiplication table.
- See omul for octonionic multiplication.

Examples:

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

$$[\Phi, associator, commutator, def\_omultable, o\_conjug, oinv, omul, omultable, onorm, oversion, purevectorpart, realpart]$$

```
> p1:=1-2*e2+e4+e5+4*e6-e7;
```

$$p1 := 1 - 2\,e2 + e4 + e5 + 4\,e6 - e7$$

```
> type(p1,octonion);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$true$$

```
> p2:=p1+e8;
```

$$p2 := 1 - 2\,e2 + e4 + e5 + 4\,e6 - e7 + e8$$

```
> type(p2,octonion);
```

$$false$$

```
>
```

See Also: def_omultable, omultable, omul

Last modified: December 20, 2007, RA/BF.

Function: Octonion:-oinv - symbolic inverse in the octonionic division ring

Calling Sequence:

oinv(o);

Parameters:

o - expression of the type 'octonion'

Description:

- Procedure 'oinv' calculates a symbolic inverse of any non-zero octonion. Recall that octonions form a non-associative, non-commutative division ring.
- For information about type 'octonion' see `type/octonion`.
- Note that any of the following is an illegal entry: 1/e1, e1^(-1), etc.
- Recall that octonionic product can be computed with the procedure omul.

Examples:

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

$$[\Phi, associator, commutator, def\_omultable, o\_conjug, oinv, omul, omultable, onorm, oversion, purevectorpart, realpart]$$

```
> o1:=1-2*e1+3*e3+e4-e6+e7;
```

$$o1 := 1 - 2\,e1 + 3\,e3 + e4 - e6 + e7$$

```
> p:=e1+e2;pinv:=oinv(p);
```

$$p := e1 + e2$$

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

$$pinv := -\frac{e1}{2} - \frac{e2}{2}$$

```
> omul(p,pinv);
```

$$Id$$

```
> o1inv:=oinv(o1); #inverse of o1
```

$$o1inv := \frac{1}{17} + \frac{2\,e1}{17} - \frac{3\,e3}{17} - \frac{e4}{17} + \frac{e6}{17} - \frac{e7}{17}$$

```
> omul(o1inv,o1); #checking that o1inv is the inverse of o1
```

$$Id$$

```
> o2:=x0+add(x||i*e||i,i=1..7);
```

$$o2 := x0 + x1\,e1 + x2\,e2 + x3\,e3 + x4\,e4 + x5\,e5 + x6\,e6 + x7\,e7$$

```
> o2inv:=oinv(o2); #symbolic inverse of o2
```

$$o2inv := \frac{x0}{x0^2 + x1^2 + x2^2 + x3^2 + x4^2 + x5^2 + x6^2 + x7^2}$$

$$-\frac{x1\ e1}{x0^2+x1^2+x2^2+x3^2+x4^2+x5^2+x6^2+x7^2}$$
$$-\frac{x2\ e2}{x0^2+x1^2+x2^2+x3^2+x4^2+x5^2+x6^2+x7^2}$$
$$-\frac{x3\ e3}{x0^2+x1^2+x2^2+x3^2+x4^2+x5^2+x6^2+x7^2}$$
$$-\frac{x4\ e4}{x0^2+x1^2+x2^2+x3^2+x4^2+x5^2+x6^2+x7^2}$$
$$-\frac{x5\ e5}{x0^2+x1^2+x2^2+x3^2+x4^2+x5^2+x6^2+x7^2}$$
$$-\frac{x6\ e6}{x0^2+x1^2+x2^2+x3^2+x4^2+x5^2+x6^2+x7^2}$$
$$-\frac{x7\ e7}{x0^2+x1^2+x2^2+x3^2+x4^2+x5^2+x6^2+x7^2}$$

```
> omul(o2,o2inv);
```

$$Id$$

```
>
```

See Also: onorm, omul, def_omultable, omultable

Last modified: December 20, 2007, RA/BF.

Function: Octonion:-omul - octonion product in the octonion non-associative division ring and its infix form '&o'

Calling Sequence:

omul(o1,o2,...on);

o1 &o o2 &o ... &o on;

Parameters:

o1, o2, ..., on - expressions of the type 'octonion'

Description:

- Procedure 'omul' and its infix form '&o' give the octonion product in the non-associative division ring of octonions.
- Octonions are considered here as para-vectors in the Clifford algebra Cl(0,7), that is, any expression of the form

  x0 + x1*e1 + x2*e2 + x3*e3 + x4*e4 + x5*e5 + x6*e6 + x7*e7

  where x0, x1, ..., x7 are real numbers, is of type 'octonion'. See `type/octonion` for more information.
- The basis elements for the octonion algebra are {1,e1,e2,e3,e4,e5,e6,e7} (sometimes 'Id' is returned instead of '1'). They are collected in a global variable '_octbasis'. The basis elements {e1,e2,e3,e4,e5,e6,e7} give pure octonions and are collected in a global variable _pureoctbasis.
- To display environmental variables from CLIFFORD and Octonion, use Clifford:-CLIFFORD_ENV.
- The infix form is given by `&o`, e.g., omul(e1,e2) = e1 &o e2. Remember that 'omul' is non-associative!
- Octonionic inverse is computed with oinv.
- To speed up computations, set the global variable _prolevel to 'true'. To find out more, see help page on Clifford:-cliparse.
- To see the default multiplication table try omultable and to define your own octonionic multiplication see def_omultable.

Examples:

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

[Φ, *associator*, *commutator*, *def_omultable*, *o_conjug*, *oinv*, *omul*, *omultable*, *onorm*, *oversion*, *purevectorpart*, *realpart*]

The following is the default octonionic multiplication table:

```
> omultable();
```

$$\begin{bmatrix} -Id & e4 & e7 & -e2 & e6 & -e5 & -e3 \\ -e4 & -Id & e5 & e1 & -e3 & e7 & -e6 \\ -e7 & -e5 & -Id & e6 & e2 & -e4 & e1 \\ e2 & -e1 & -e6 & -Id & e7 & e3 & -e5 \\ -e6 & e3 & -e2 & -e7 & -Id & e1 & e4 \\ e5 & -e7 & e4 & -e3 & -e1 & -Id & e2 \\ e3 & e6 & -e1 & e5 & -e4 & -e2 & -Id \end{bmatrix}$$

```
> o1:=1-2*e1+3*e3+e4-e6+e7;
```

$$o1 := 1 - 2\,e1 + 3\,e3 + e4 - e6 + e7$$

```
> o2:=2+e3-4*e6+e7;
```

$$o2 := 2 + e3 - 4\,e6 + e7$$

```
> type(o1,octonion),type(o2,octonion);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$true, true$$

```
> omul(o1,o2);
```

$$-6\,Id - 2\,e1 + 5\,e3 + 13\,e4 - 7\,e6 + e7 - 9\,e5 + 3\,e2$$

Octonionic multiplication is not commutative:

```
> o1 &o o2;
```

$$-6\,Id - 2\,e1 + 5\,e3 + 13\,e4 - 7\,e6 + e7 - 9\,e5 + 3\,e2$$

```
> o2 &o o1;
```

$$-6\,Id + 9\,e3 - 5\,e6 + 5\,e7 - 6\,e1 + 9\,e5 - 9\,e4 - 3\,e2$$

We show now that it is not associative either:

```
> (e1 &o e2) &o e3;
```

$$-e6$$

```
> e1 &o (e2 &o e3);
```

$$e6$$

```
> o3:=2-3*e1+e5-e7;
```

$$o3 := 2 - 3\,e1 + e5 - e7$$

```
> (o1 &o o2) &o o3;
```

$$-8\,Id + 16\,e1 - 21\,e2 + 2\,e3 + 43\,e4 + 10\,e5 - 40\,e6 + 36\,e7$$

```
> o1 &o (o2 &o o3);
```

$$-8\,Id + 6\,e1 + 47\,e2 + 8\,e3 + 5\,e4 - 18\,e5 - 38\,e6 + 38\,e7$$

The difference between (o1 &o o2) &o o3 and o1 &o (o2 &o o3) is measured by an associator, or see associator:

```
> associator(o1,o2,o3);
```

$$10\,e1 - 68\,e2 - 6\,e3 + 38\,e4 + 28\,e5 - 2\,e6 - 2\,e7$$

The difference between o1 &o o2 and o2 &o o1 is measured by a commutator, or see commutator:

```
> commutator(o1,o2);
```

$$4\,e1 - 4\,e3 + 22\,e4 - 2\,e6 - 4\,e7 - 18\,e5 + 6\,e2$$

```
>
```

See Also: Clifford:-version, oinv, def_omultable, omultable

Last modified: December 20, 2007, RA/BF.

Function: Octonion:-omultable - display current octonionic multiplication table

Calling Sequence:

omultable();

Parameters:

no parameters needed

Description:

- Procedure 'omultable' displays current octonionic multiplication table or returns a message informing that the table has not been defined.
- When the Octonion package is loaded, the default multiplication table is initialized. This default table is defined by a default list of Fano triples (see `type/Fano_triples`) which are stored in a global variable _default_Fano_triples.
- To see environmental variables used in 'CLIFFORD' and 'Octonion', see procedure Clifford:-CLIFFORD_ENV.
- The multiplication table is displayed in a form of a 7 by 7 matrix such that its (i,j)-entry, i,j=1,...,7, gives the octonion product of ei and ej, that is, the product ei &o ej.
- Recall that the elements of the pure octonion basis {e1,e2,e3,e4,e5,e6,e7} are stored in a global list _pureoctbasis.
- To speed up computations, procedure 'omul', which gives the octonionic product (see omul) has a remember table. This remember table can be erased using the command subsop(4=NULL,eval(omul)).
- Octonionic multiplication can be re-defined by the user using the procedure def_omultable.

Examples:

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

[Φ, *associator*, *commutator*, *def_omultable*, *o_conjug*, *oinv*, *omul*, *omultable*, *onorm*, *oversion*, *purevectorpart*, *realpart*]

```
> oversion();
```

++

'Octonion' - A Maple 11 Package for Computations with Octonions (version 11)

Last revised: December 20, 2007

Copyright 1995-2008, by Rafal Ablamowicz, Tennessee Technological University

Department of Mathematics, Box 5054

Tennessee Technological University, Cookeville, TN 38505

phone: USA (931) 372-3569, fax: USA (931) 372-6353

e-mail: rablamowicz@tntech.edu

http://math.tntech.edu/rafal/

++

```
> omultable();
```

$$\begin{bmatrix} -Id & e4 & e7 & -e2 & e6 & -e5 & -e3 \\ -e4 & -Id & e5 & e1 & -e3 & e7 & -e6 \\ -e7 & -e5 & -Id & e6 & e2 & -e4 & e1 \\ e2 & -e1 & -e6 & -Id & e7 & e3 & -e5 \\ -e6 & e3 & -e2 & -e7 & -Id & e1 & e4 \\ e5 & -e7 & e4 & -e3 & -e1 & -Id & e2 \\ e3 & e6 & -e1 & e5 & -e4 & -e2 & -Id \end{bmatrix}$$

For example, we get the first row as follows:

```
> seq(e1 &o e||i,i=1..7);
```

$$-Id, e4, e7, -e2, e6, -e5, -e3$$

The second row we get as follows:

```
> seq(e2 &o e||i,i=1..7);
```

$$-e4, -Id, e5, e1, -e3, e7, -e6$$

and so on.

Multiplication table can be erased as follows:

```
> subsop(4=NULL,eval(omul)):
> omultable();
Octonion multiplication table is not currently defined. Use 'def_omultable' to
define a new table.
```

When the multiplication table has been erased, computations still can be performed using the approach described in Pertti Lounesto's 'Clical'. In fact, the default multiplication table is the one used by Lounesto's. However, they will take longer to accomplish. For example:

```
> f:=(x,y)->x &o y;
```

$$f := (x, y) \rightarrow x \,\&o\, y$$

```
> L:=[e1,e2,e3,e4,e5,e6,e7];
```

$$L := [e1, e2, e3, e4, e5, e6, e7]$$

```
> map2(f,e1,L);
  map2(f,e2,L);
  map2(f,e3,L);
  map2(f,e4,L);
  map2(f,e5,L);
  map2(f,e6,L);
  map2(f,e7,L);
```

$$[-Id, e4, e7, -e2, e6, -e5, -e3]$$

$$[-e4, -Id, e5, e1, -e3, e7, -e6]$$

$$[-e7, -e5, -Id, e6, e2, -e4, e1]$$

$$[e2, -e1, -e6, -Id, e7, e3, -e5]$$
$$[-e6, e3, -e2, -e7, -Id, e1, e4]$$
$$[e5, -e7, e4, -e3, -e1, -Id, e2]$$
$$[e3, e6, -e1, e5, -e4, -e2, -Id]$$

```
>
```

which yields the same result as before.

Finally, we re-initialize the table using the default Fano triples:

```
> _default_Fano_triples;
```

$$[[1, 3, 7], [1, 2, 4], [1, 5, 6], [2, 3, 5], [2, 6, 7], [3, 4, 6], [4, 5, 7]]$$

```
> def_omultable(_default_Fano_triples);
> omultable();
```

$$\begin{bmatrix} -Id & e4 & e7 & -e2 & e6 & -e5 & -e3 \\ -e4 & -Id & e5 & e1 & -e3 & e7 & -e6 \\ -e7 & -e5 & -Id & e6 & e2 & -e4 & e1 \\ e2 & -e1 & -e6 & -Id & e7 & e3 & -e5 \\ -e6 & e3 & -e2 & -e7 & -Id & e1 & e4 \\ e5 & -e7 & e4 & -e3 & -e1 & -Id & e2 \\ e3 & e6 & -e1 & e5 & -e4 & -e2 & -Id \end{bmatrix}$$

```
>
```

Thus, the table has been re-initialized.

See Also: `type/Fano_triples`,def_omultable, omul

Last modified: December 20, 2007, RA/BF.

Function: Octonion:-onorm - norm of an octonion

Calling Sequence:

onorm(o);

Parameters:

o - expression of the type 'octonion'

Description:

- Procedure 'onorm' calculates norm of an octonion o. It is defined as follows:

  onorm(o) = sqrt(o &o o_conjug(o)) = sqrt(x0^2+x1^2+ x2^2+x3^2+x4^2+x5^2+x6^2+x7^2)

  where o = x0+x1*e1+x2*e2+x3*e3+x4*e4+x5*e5+x6*e6+x7*e7, and x0,x1,...,x7, are real parameters.

- Recall that octonionic product can be computed with the procedure omul or with its infix form `&o`.

- For information about type 'octonion' see `type/octonion`.

Examples:

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

$[\Phi, associator, commutator, def\_omultable, o\_conjug, oinv, omul, omultable, onorm, oversion, purevectorpart, realpart]$

```
> o1:=1-2*e1+3*e3+e4-e6+e7;
```

$$o1 := 1 - 2\,e1 + 3\,e3 + e4 - e6 + e7$$

```
> onorm(o1); #norm of o1
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$\sqrt{17}$$

```
> o2:=2-3*e4+e5+4*e6-e7;
```

$$o2 := 2 - 3\,e4 + e5 + 4\,e6 - e7$$

```
> onorm(o2);
```

$$\sqrt{31}$$

Theorem [The Eight-Square Identity]

The norm in the octonion algebra is a ring homomorphism.

```
> o1:=x0+x1*e1+x2*e2+x3*e3+x4*e4+x5*e5+x6*e6+x7*e7;
```

$$o1 := x0 + x1\,e1 + x2\,e2 + x3\,e3 + x4\,e4 + x5\,e5 + x6\,e6 + x7\,e7$$

```
> o2:=y0+y1*e1+y2*e2+y3*e3+y4*e4+y5*e5+y6*e6+y7*e7;
```

$$o2 := y0 + y1\,e1 + y2\,e2 + y3\,e3 + y4\,e4 + y5\,e5 + y6\,e6 + y7\,e7$$

We will now verify that

*onorm(o1 &o o2) = onorm(o1) * onorm(o2).*

```
> factor(onorm(o1 &o o2));
```

$$\sqrt{(y4^2+y5^2+y7^2+y0^2+y6^2+y3^2+y2^2+y1^2)\,(x0^2+x3^2+x2^2+x4^2+x6^2+x1^2+x5^2+x7^2)}$$

```
> onorm(o1)*onorm(o2);
```

$$\sqrt{x0^2+x3^2+x2^2+x4^2+x6^2+x1^2+x5^2+x7^2}\,\sqrt{y4^2+y5^2+y7^2+y0^2+y6^2+y3^2+y2^2+y1^2}$$

```
>
```

See Also: oversion, omul, oinv, def_omultable, omultable

Last modified: December 20, 2007, RA/BF.

Function: Octonion:-associator - returns the associator value of three octonions,
Octonion:-commutator - returns the commutator value of two octonions
Octonion:-Phi - associative 3-form of three octonions

Calling Sequence:

associator(p1,p2,p3);
commutator(p1,p2);
Phi(p1,p2,p2);

Parameters:

p1, p2, p3 - polynomials of type 'octonion'

Description:

- The associator of three octonions p1, p2, and p3 is defined as:

  associator(p1,p2,p3) = (p1 &o p2) &o p3 - p1 &o (p2 &o p3).

- The commutator of two octonions p1 and p2 is defined as:

  commutator(p1,p2) = p1 &o p2 - p2 &o p1.

- The associative 3-form Phi of three octonions is defined as:

  Phi(p1,p2,p3) = (1/2)*realpart(p1 &o (p2_bar &o p3) - p3 &o (p2_bar &o p1))

  where p2_bar = o_conjug(p2).

- For information about type 'octonion' see `type/octonion`.

Examples:

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

$[\Phi, associator, commutator, def\_omultable, o\_conjug, oinv, omul, omultable, onorm, oversion, purevectorpart, realpart]$

```
> p1:=1-2*e1+e4+3*e6-e7;p2:=2-e1+e3+2*e6-e7;p3:=2*e2+e3+3*e5-e6;
```

$$p1 := 1 - 2\,e1 + e4 + 3\,e6 - e7$$
$$p2 := 2 - e1 + e3 + 2\,e6 - e7$$
$$p3 := 2\,e2 + e3 + 3\,e5 - e6$$

```
> type(p1,octonion);type(p2,octonion);type(p3,octonion);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$true$$

$$true$$

$$true$$

Octonion multiplication is not associative:

```
> associator(p1,p2,p3);
```

$$-8\,e1 - 2\,e2 + 20\,e3 + 14\,e4 - 6\,e5 - 2\,e6 + 24\,e7$$

However, when p1, p2, and p3 are considered as elements in the Clifford algebra Cl(0,7), which is associative, we get:

```
> (p1 &c p2) &c p3 - p1 &c (p2 &c p3);
```

$$0$$

```
> commutator(p1,p2);
```

$$2\,e1 - 4\,e2 + 2\,e3 + 6\,e4 + 4\,e5 - 2\,e6 - 4\,e7$$

```
> Phi(p1,p2,p3);
```

$$4$$

```
>
```

See Also: Clifford:-`&c`, def_omultable, omultable, omul

Last modified: December 20, 2007, RA/BF.

Function: Octonion:-realpart - returns real part of any octonion,
Octonion:-purevectorpart - returns pure vector part of any octonion

Calling Sequence:

realpart(p);
purevectorpart(p);

Parameters:

p - a polynomial of type 'octonion'

Description:

- Any octonion p is expressible as

  p= x0+x1*e1+x2*e2+x3*e3+x4*e4+x5*e5+x6*e6+x7*e7 or p = x0*Id+x1*e1+x2*e2+x3*e3+x4*e4+x5*e5+x6*e6+x7*e7

  where x0, x1,..., x7, are real parameters.
- For information about type 'octonion' see `type/octonion`.
- The part x1*e1+x2*e2+x3*e3+x4*e4+x5*e5+x6*e6+x7*e7 of p is referred to as the 'pure vector part' of p.
- The coefficient x0 is referred to as the 'real part' of p.
- Procedure 'realpart' is similar to Clifford:-scalarpart.

Examples:

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

$$[\Phi, associator, commutator, def\_omultable, o\_conjug, oinv, omul, omultable, onorm, oversion, purevectorpart, realpart]$$

```
> p1:=1-2*e1+e4+3*e6-e7;
```

$$p1 := 1 - 2\,e1 + e4 + 3\,e6 - e7$$

```
> type(p1,octonion);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$true$$

```
> realpart(p1);
```

$$1$$

```
> scalarpart(p1);
```

$$1$$

```
> purevectorpart(p1);
```

$$-2\,e1 + e4 + 3\,e6 - e7$$

```
>
```

See Also: def_omultable, omultable, omul

Last modified: December 20, 2007, RA/BF.

Function: Octonion:-realpart - returns real part of any octonion,
Octonion:-purevectorpart - returns pure vector part of any octonion

Calling Sequence:

realpart(p);
purevectorpart(p);

Parameters:

p - a polynomial of type 'octonion'

Description:

- Any octonion p is expressible as

  p= x0+x1*e1+x2*e2+x3*e3+x4*e4+x5*e5+x6*e6+x7*e7 or p = x0*Id+x1*e1+x2*e2+x3*e3+x4*e4+x5*e5+x6*e6+x7*e7

  where x0, x1,..., x7, are real parameters.
- For information about type 'octonion' see `type/octonion`.
- The part x1*e1+x2*e2+x3*e3+x4*e4+x5*e5+x6*e6+x7*e7 of p is referred to as the 'pure vector part' of p.
- The coefficient x0 is referred to as the 'real part' of p.
- Procedure 'realpart' is similar to Clifford:-scalarpart.

Examples:

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

$$[\Phi, associator, commutator, def\_omultable, o\_conjug, oinv, omul, omultable, onorm, oversion, purevectorpart, realpart]$$

```
> p1:=1-2*e1+e4+3*e6-e7;
```

$$p1 := 1 - 2\,e1 + e4 + 3\,e6 - e7$$

```
> type(p1,octonion);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$true$$

```
> realpart(p1);
```

$$1$$

```
> scalarpart(p1);
```

$$1$$

```
> purevectorpart(p1);
```

$$-2\,e1 + e4 + 3\,e6 - e7$$

```
>
```

See Also: def_omultable, omultable, omul

Last modified: December 20, 2007, RA/BF.

Examples of using package RJgrobner.

Procedures encoded in this package represent a 'naive' programming effort to understand various Groebner theory algorithms presented in the book "Ideals, Varieties, and Algorithms," by David Cox, John Little, and Donal O'Shea, Second Edition, Springer-Verlag, New York, 1997.

```
> restart:
> with(RJgrobner);
RJgrobner Version 7 (19 vi 2008) at your service
(c) 2003-2008 RA&JL, sorry, no warranty for anything!
Load SINGULARPLURALlink for interface with Singular:Plural
(c) 2003-2008 RA&BF
```

[*Beziercubic*, *Gbasis*, *GbasisL*, *GreatestCommonDivisor*, *IntersectionOfIdeals*, *LeastCommonMultiple*, *ProductOfIdeals*, *QuotientOfIdeals*, *RJversion*, *RadicalMembership*, *Spoly*, *SumOfIdeals*, *completelyreducedGbasis*, *condition*, *condition2*, *condition3*, *curvature*, *homogenize*, *implicitBeziercubic*, *maxdegree*, *maxmindegree*, *maxtotaldegree*, *minimalGbasis*, *reducedGbasis*, *reducepol*, *reduction*]

```
>
```

0: Procedure RJversion gives version of the package.

```
> RJversion();
```

++

RJgrobner - A Maple 11 Small Package for Grobner Bases

Last revised: June 19, 2008 (Source file: RJgrobner_M11_07.mws)

(*) Department of Mathematics, Box 5054

Tennessee Technological University, Cookeville, TN 38505

tel: USA (931) 372-3622, fax: USA (931) 372-6353

rablamowicz@tntech.edu

http://math.tntech.edu/rafal/

($) Department of Civil and Environmental Engineering

Tennessee Technological University, Cookeville, TN 38505

tel: USA (931) 372-3256, fax: USA 372-6239

jliu@tntech.edu

http://math.tntech.edu/rafal/

++++++++++++*This is RJgrobner for Maple 11 version 7*++++++++++++

```
>
```

1. Procedure 'reducepol' reduces the given polynomial by dividing it by its content, that is, it returns

the polynomial divided by the greatest common divisor of its coefficients:

Usage:
reducepol(f);

```
> f:=2*x^3-6*x^2-10;
```

$$f := 2\,x^3 - 6\,x^2 - 10$$

```
> reducepol(f);
```

$$x^3 - 3\,x^2 - 5$$

```
>
```

2. Procedure Spoly computes an S-polynomial for any two monomials/polynomials specified as the i-th entry F[i] and the j-th entry F[j] in the list F, a basis of some ideal I, for the given monomial order T entered as the third argument, for example, plex(t,z,y,x) or lexdeg([x],[y,z]) (see below). It computes the remainder of the division of the S-polynomial with respect to the basis F of I specified as the second argument. The procedure normally returns an updated basis for the ideal. If used with a fourth optional parameter 'spoly', it returns the S-polynomial only.

Recall that Maple uses these term orders:

- **tdeg(v_1,...,v_p)** to denote a total degree order
- **wdeg([w_1,...,w_p],[v_1,...,v_p])** to denote a weighted degree order
- **plex(v_1,...,v_p)** to denote a pure lexicographic order
- **lexdeg([v_1,...,v_p],[w_1,...,w_q])** to denote an elimination order
- **'matrix'(mat,[v_1,...,v_p])** to denote a matrix-defined term order
- **user(P, L)** or **user(P, Q, L)** to denote a user-defined term order.

Usage:
Spoly(1,2,F,plex(t,z,y,x),'spoly');
Spoly(1,2,F,lexdeg([t],[z,y,x]));

```
> F:=[2*x^3*y-x^2*y^3+7,3*x^4*y-x^2*y^4+5,2*x^2*y^4-x+7*y,x^2-x*y+2]
  ;
  nops(F);
```

$$F := [2\,x^3\,y - x^2\,y^3 + 7,\ 3\,x^4\,y - x^2\,y^4 + 5,\ 2\,x^2\,y^4 - x + 7\,y,\ x^2 - x\,y + 2]$$

$$4$$

```
> Spoly(1,2,F,plex(x,y),'spoly');
  reducepol(%);
```

$$-3\,x^3\,y^3 + 21\,x + 2\,x^2\,y^4 - 10$$

$$-3\,x^3\,y^3 + 21\,x + 2\,x^2\,y^4 - 10$$

```
> Spoly(1,2,F,plex(x,y));
```

$$[2\,x^3\,y - x^2\,y^3 + 7,\ 3\,x^4\,y - x^2\,y^4 + 5,\ 2\,x^2\,y^4 - x + 7\,y,\ x^2 - x\,y + 2,\ 88\,x - 28\,y - 3\,x\,y + 63\,y^2 - 40\ ]$$

```
> nops(%);
```

$$5$$

```
>
```

3. Procedure "Gbasis" computes a Groebner basis for the given ideal I generated by some polynomials [f1,f2,...,fr] entered as the first argument F with respect to some monomial order T entered as the second argument. It uses procedure Spoly to compute the S-polynomials S(fi,fj), 1 <= i < j, and then the remainder of the division of S(fi,fj) with respect to the list F and the selected monomial order. If the reminder is not zero, the list F gets updated by the reminder. If the reminder is zero, the list is not changed. This way, the S-polynomials are computed and then reduced w.r.t. to F until every S-polynomial for any two polynomials in S produces a zero remainder when reduced. Then, the procedure returns a Groebner basis for the ideal I for the given monomial order.

Usage:
Gbasis(F,plex(t,z,y,x));
Gbasis(F,lexdeg([x],[y,z]));

```
> Gbasis(F,plex(x,y));
```

$$[2\,x^3\,y - x^2\,y^3 + 7,\ 3\,x^4\,y - x^2\,y^4 + 5,\ 2\,x^2\,y^4 - x + 7\,y,\ x^2 - x\,y + 2,\ 88\,x - 28\,y - 3\,x\,y + 63\,y^2 - 40,$$
$$-2728\,x + 898\,y - 1932\,y^2 + 1231,$$
$$865560\,y + 706456\,y^2 + 637069 - 4622162\,y^3 - 157542\,y^4 - 5859\,y^5,$$
$$112128\,y + 88887\,y^2 + 80020 - 583730\,y^3 - 19530\,y^4,$$
$$6656082\,y + 726523\,y^2 + 5284040 - 22540875\,y^3,$$
$$47540260193012\,y - 276085314055082\,y^2 + 8530431030265,$$
$$-236570502466204574\,y - 84218989678651503,\ 1\ ]$$

```
> G:=Gbasis(F,lexdeg([x],[y]));
```

$$G := [2\,x^3\,y - x^2\,y^3 + 7,\ 3\,x^4\,y - x^2\,y^4 + 5,\ 2\,x^2\,y^4 - x + 7\,y,\ x^2 - x\,y + 2,$$
$$88\,x - 28\,y - 3\,x\,y + 63\,y^2 - 40,\ -2728\,x + 898\,y - 1932\,y^2 + 1231,$$
$$865560\,y + 706456\,y^2 + 637069 - 4622162\,y^3 - 157542\,y^4 - 5859\,y^5,$$
$$112128\,y + 88887\,y^2 + 80020 - 583730\,y^3 - 19530\,y^4,$$
$$6656082\,y + 726523\,y^2 + 5284040 - 22540875\,y^3,$$
$$47540260193012\,y - 276085314055082\,y^2 + 8530431030265,$$
$$-236570502466204574\,y - 84218989678651503,\ 1\ ]$$

```
>
```

4. Procedure GbasisL eliminates from a Groebner basis G entered as a first argument those

polynomials which contain variables specified in a list L entered as the second argument. Thus, GbasisL returns a Groebner basis for the i-th elimination ideal by returning an intersection of any Groebner basis in n variables entered as the first argument with a ring of polynomials in the remaining {(i+1),...n} variables.

Usage:
GbasisL(G,[t]);

```
> GbasisL(G,[x]);
```

$$[865560\,y + 706456\,y^2 + 637069 - 4622162\,y^3 - 157542\,y^4 - 5859\,y^5,$$
$$112128\,y + 88887\,y^2 + 80020 - 583730\,y^3 - 19530\,y^4,$$
$$6656082\,y + 726523\,y^2 + 5284040 - 22540875\,y^3,$$
$$47540260193012\,y - 276085314055082\,y^2 + 8530431030265,$$
$$-236570502466204574\,y - 84218989678651503,\ 1]$$

```
>
```

5. Procedure minimalGbasis computes a minimal Groebner basis from the given Groebner basis G (first argument) and for the given monomial order T (second argument).

Usage:
minimalGbasis(Gx,plex(z,y,x));
minimalGbasis(Gx,lexdeg([z],[y,x]));

```
> G:=Gbasis(F,plex(x,y));
  minimalGbasis(G,plex(x,y));
```

$$G := [2\,x^3\,y - x^2\,y^3 + 7,\ 3\,x^4\,y - x^2\,y^4 + 5,\ 2\,x^2\,y^4 - x + 7\,y,\ x^2 - x\,y + 2,$$
$$88\,x - 28\,y - 3\,x\,y + 63\,y^2 - 40,\ -2728\,x + 898\,y - 1932\,y^2 + 1231,$$
$$865560\,y + 706456\,y^2 + 637069 - 4622162\,y^3 - 157542\,y^4 - 5859\,y^5,$$
$$112128\,y + 88887\,y^2 + 80020 - 583730\,y^3 - 19530\,y^4,$$
$$6656082\,y + 726523\,y^2 + 5284040 - 22540875\,y^3,$$
$$47540260193012\,y - 276085314055082\,y^2 + 8530431030265,$$
$$-236570502466204574\,y - 84218989678651503,\ 1]$$

$$[1]$$

```
>
```

6. Procedure reducedGbasis computes a reduced Groebner basis from the given minimal Groebner basis Gmin (first argument) and for the given monomial order T (second argument).

Usage:
reducedGbasis(G,plex(z,y,x));

```
> G:=Gbasis(F,plex(x,y));
  reducedGbasis(G,plex(x,y));
```

$$G := [2x^3y - x^2y^3 + 7, 3x^4y - x^2y^4 + 5, 2x^2y^4 - x + 7y, x^2 - xy + 2,$$
$$88x - 28y - 3xy + 63y^2 - 40, -2728x + 898y - 1932y^2 + 1231,$$
$$865560y + 706456y^2 + 637069 - 4622162y^3 - 157542y^4 - 5859y^5,$$
$$112128y + 88887y^2 + 80020 - 583730y^3 - 19530y^4,$$
$$6656082y + 726523y^2 + 5284040 - 22540875y^3,$$
$$47540260193012y - 276085314055082y^2 + 8530431030265,$$
$$-236570502466204574y - 84218989678651503, 1]$$

$$[1]$$

7. Procedure completelyreducedGbasis computes a completely reduced Groebner basis from the given reduced Groebner basis Gred (first argument) and for the lex order T (second argument).

Usage:
completelyreducedGbasis(reducedGx,plex(z,y,x));

```
> G:=Gbasis(F,plex(x,y));
  completelyreducedGbasis(G,plex(x,y));
```

$$G := [2x^3y - x^2y^3 + 7, 3x^4y - x^2y^4 + 5, 2x^2y^4 - x + 7y, x^2 - xy + 2,$$
$$88x - 28y - 3xy + 63y^2 - 40, -2728x + 898y - 1932y^2 + 1231,$$
$$865560y + 706456y^2 + 637069 - 4622162y^3 - 157542y^4 - 5859y^5,$$
$$112128y + 88887y^2 + 80020 - 583730y^3 - 19530y^4,$$
$$6656082y + 726523y^2 + 5284040 - 22540875y^3,$$
$$47540260193012y - 276085314055082y^2 + 8530431030265,$$
$$-236570502466204574y - 84218989678651503, 1]$$

$$[1]$$

```
>
```

8. Procedure 'condition' computes a polynomial that gives a normal line to a planar curve defined implictly as f1 = f1(x0,y0) = 0 at a point (x0,y0).

Usage:

f3:=condition(f1);

```
> f:=x0^2+y0^2-2;
  condition(f);
```

$$f := x0^2 + y0^2 - 2$$
$$x\,y0 - x0\,y$$

```
> f:=x0^2*b^2+y0^2*a^2-a^2*b^2;
  condition(f);
```

$$f := x0^2\, b^2 + y0^2\, a^2 - a^2\, b^2$$

$$x\, y0\, a^2 - x0\, y0\, a^2 - x0\, b^2\, y + x0\, b^2\, y0$$

```
>
```

9. Procedure 'condition2' computes a polynomial that gives a normal line to a planar curve defined parametrically X = X(t) and Y = Y(t) via two polynomials of t.

Usage:
For example, we can define Bezier cubic parametrically as

X:=simplify(((1-t)^3*x1+3*t*(1-t)^2*x2+3*t^2*(1-t)*x3+t^3*x4));
Y:=simplify(((1-t)^3*y1+3*t*(1-t)^2*y2+3*t^2*(1-t)*y3+t^3*y4));

and an equation of a circle of radius r located at a point (X,Y) on the cubic as

f3:=(y-Y)^2+(x-X)^2-r^2;

where (x,y) represent coordinates of a point on the circle. Then

f4:= condition2(X,Y);

will be a polynomial that gives the equation of a normal line to the cubic. Polynomial f4 will contain variables x,y and t.

```
> X:=simplify(((1-t)^3*x1+3*t*(1-t)^2*x2+3*t^2*(1-t)*x3+t^3*x4));
  Y:=simplify(((1-t)^3*y1+3*t*(1-t)^2*y2+3*t^2*(1-t)*y3+t^3*y4));
```

$$X := x1 - 3\, x1\, t + 3\, x1\, t^2 - x1\, t^3 + 3\, t\, x2 - 6\, t^2\, x2 + 3\, t^3\, x2 + 3\, t^2\, x3 - 3\, t^3\, x3 + t^3\, x4$$

$$Y := y1 - 3\, y1\, t + 3\, y1\, t^2 - y1\, t^3 + 3\, t\, y2 - 6\, t^2\, y2 + 3\, t^3\, y2 + 3\, t^2\, y3 - 3\, t^3\, y3 + t^3\, y4$$

```
> f4:=condition2(X,Y);
```

$$f4 := (x - x1 + 3\, x1\, t - 3\, x1\, t^2 + x1\, t^3 - 3\, t\, x2 + 6\, t^2\, x2 - 3\, t^3\, x2 - 3\, t^2\, x3 + 3\, t^3\, x3 - t^3\, x4)$$
$$(-3\, x1 + 6\, x1\, t - 3\, x1\, t^2 + 3\, x2 - 12\, t\, x2 + 9\, t^2\, x2 + 6\, t\, x3 - 9\, t^2\, x3 + 3\, t^2\, x4) +$$
$$(y - y1 + 3\, y1\, t - 3\, y1\, t^2 + y1\, t^3 - 3\, t\, y2 + 6\, t^2\, y2 - 3\, t^3\, y2 - 3\, t^2\, y3 + 3\, t^3\, y3 - t^3\, y4)$$
$$(-3\, y1 + 6\, y1\, t - 3\, y1\, t^2 + 3\, y2 - 12\, t\, y2 + 9\, t^2\, y2 + 6\, t\, y3 - 9\, t^2\, y3 + 3\, t^2\, y4)$$

```
>
```

10. Procedure 'condition3' computes coordinates x,y,z, of a point located along a line normal to the surface defined parametrically as X=x(s,t), Y=y(s,t), Z=z(s,t) with a running parameter u along the line.
Usage:

```
X:=2*sin(s)*cos(t);
Y:=2*sin(s)*sin(t);
Z:=2*cos(s);

condition3(X,Y,Z);
```

```
> X:=2*sin(s)*cos(t);
  Y:=2*sin(s)*sin(t);
  Z:=2*cos(s);
  condition3(X,Y,Z);
```

$$X := 2\sin(s)\cos(t)$$
$$Y := 2\sin(s)\sin(t)$$
$$Z := 2\cos(s)$$
$$x - 4\,u\sin(s)^2\cos(t) - 2\sin(s)\cos(t),\ y - 4\,u\sin(s)^2\sin(t) - 2\sin(s)\sin(t),$$
$$z - 4\,u\cos(s)\cos(t)^2\sin(s) - 4\,u\cos(s)\sin(t)^2\sin(s) - 2\cos(s)$$

```
>
```

11. Procedure 'maxdegree' computes a total degree of the given multivariate polynomial f for the given monomial order T.

Usage:

```
f:=2*x*y+z^2-w^5;
maxdegree(f,plex(x,y,z,w));
```

```
> f:=2*x*y+z^2-w^5;
  maxdegree(f,plex(x,y,z,w));
  maxdegree(f,plex(w,z,y,x));
  maxdegree(f,plex(z,y,x,w));
```

$$f := 2\,x\,y + z^2 - w^5$$
$$2$$
$$5$$
$$2$$

```
>
```

12. Procedure 'maxtotaldegree' computes a maximum total degree of the given multivariate polynomial f. It uses any monomial order T just to sort monomials in f. That is, this procedure returns a maximum total degree of all monomials. This procedure is by 'homogenize'.

Usage:

f:=2*x*y+z^2-w^5;
maxtotaldegree(f,plex(x,y,z,w));

```
> f:=2*x*y+z^2-w^5;
  maxtotaldegree(f,plex(x,y,z,w));
  maxtotaldegree(f,plex(w,y,z,x));
  maxtotaldegree(f,plex(x,z,y,w));
  maxtotaldegree(f,plex(w,y,z,x));
```

$$f := 2\,x\,y + z^2 - w^5$$

$$5$$

$$5$$

$$5$$

$$5$$

```
>
```

13. Procedure 'maxmindegree' returns the highest and the lowest degree of all monomials appearing in a multivariate polynomial f. It uses any monomial order T to just sort monomials in f. That is, this procedure returns the maximum and the monimum total degree of all monomials in f a list.

Usage:

f:=2*x*y+z^2-w^5;
maxmindegree(f,plex(x,y,z,w));

```
> f:=2*x*y+z^2-w^5;
  maxmindegree(f,plex(x,y,z,w));
  maxmindegree(f,plex(w,z,y,x));
  maxmindegree(f,plex(x,z,y,w));
  maxmindegree(f,plex(x,y,w,z));
```

$$f := 2\,x\,y + z^2 - w^5$$

$$[5, 2]$$

$$[5, 2]$$

$$[5, 2]$$

$$[5, 2]$$

```
>
```

14. Procedure 'homogenize' homogenizes polynomial f for the given order T using a new indeterminate 'h' entered as the last argument. Of course, order here is not important: Any T will yield the same homogeneous polynomial.
Usage:

f:=2*x*y+z^2-w^5;

homogenize(f,plex(x,y,z,w),h);

```
> f:=2*x*y+z^2-w^5;
```

$$f := 2\,x\,y + z^2 - w^5$$

```
> homogenize(f,plex(x,y,z,w),h);
  homogenize(f,plex(w,z,y,x),h);
```

$$2\,x\,y\,h^3 + z^2\,h^3 - w^5$$

$$2\,x\,y\,h^3 + z^2\,h^3 - w^5$$

```
>
```

15. Radical membership algorithm (page 176 in Cox et al.)

Given:
1. Polynomial f in k[x1,x2,...,xn]
2. Polynomials f1,f2,...,fs that generate ideal I will be entered as a list F = [f1,f2,...,fs]

Algorithm:

1. Compute reduced Groebner basis GB for the ideal J = <f1,f2,...,fs, 1-y*f> with respect to any ordering, ex., plex(t,y, x1,x2,...xn).
2. Check whether GB = {1} or not. If GB = {1} then f belongs to radical of I; otherwise f does not belong to radical of I.

Usage:

f1:=x*y^2+2*y^2;
f2:=x^4-2*x^2+1;
F:=[f1,f2];
f:=y-x^2+1;
RadicalMembership(f,F);

```
> f1:=x*y^2+2*y^2;
  f2:=x^4-2*x^2+1;
  F:=[f1,f2];
  f:=y-x^2+1;
  RadicalMembership(f,F);
```

$$f1 := y^2\,x + 2\,y^2$$

$$f2 := x^4 - 2\,x^2 + 1$$

$$F := [\,y^2\,x + 2\,y^2, x^4 - 2\,x^2 + 1\,]$$

$$f := y - x^2 + 1$$

$$true$$

>

16. Computation of f_red, or, the reduction of f per Proposition 12 on page 179 in Cox et al..

Given:

1. A field k that contains the rational numbers, e.g., R, C.
2. Polynomial f in the ring k[x1,x2,x3] that generates a principal ideal I = <f>.

Algorithm (formula on page 179):

1. Compute GCD(f, f1, f2,..., fn) where fi is the derivative of f w.r.t. xi, i=1,...,n.
Recall that GCD(f, f1, f2,..., fn) can be computed as follows:

GCD(f, f1, f2,..., fn) = GCD(GCD(GCD(GCD(f,f1),f2),f3),.....fn)

NOTE: Here we use Maple's Gcd command instead of defined below GreatestCommonDivisor.

2. Divide f/GCD(f, f1, f2,..., fn).

Usage:

f:=(x+2*y)^3*(2*x-y+z)^6*(x+z^2)^3;
reduction(f);

>

```
> f:=(x+2*y)^3*(2*x-y+z)^6*(x+z^2)^3;
  reduction(f);
```

$$f := (x+2y)^3 (2x-y+z)^6 (x+z^2)^3$$

$$2x^3 + 2x^2 z^2 + 3yx^2 + x^2 z + 3xyz^2 + xz^3 + 2xyz - 2y^2 x + 2yz^3 - 2y^2 z^2$$

>

17. Computation of a sum I + J of ideals I = <f1,f2,...,fr> and J = <g1,g2,...,gs> as in Cox et al.

Given:

1. Polynomials f1,f2,...,fr entered as a list F that generate ideal I = <f1,f2,...,fr> and polynomials g1,g2,...,gs entered as a list G that generate ideal J = <g1,g2,...,gs>.

Algorithm:

1. Return a list [f1,f2,...,fr, g1,g2,...,gs] that generates I + J = <f1,f2,...,fr, g1,g2,...,gs> per Proposition 2 on page 181.

f1:=2*x^3+y^2*z;

```
f2:=2*x^2*y^6+y^2*z*x;
g1:=-x*y*z+y^2*z*x^5;
g2:=-y^3*z^3+y^2*z^3*x;
F:=[seq(f||i,i=1..2)];
G:=[seq(g||j,j=1..2)];
SumOfIdeals(F,G);
```

```
>
> f1:=2*x^3+y^2*z;
  f2:=2*x^2*y^6+y^2*z*x;
  g1:=-x*y*z+y^2*z*x^5;
  g2:=-y^3*z^3+y^2*z^3*x;
  F:=[seq(f||i,i=1..2)];
  G:=[seq(g||j,j=1..2)];
  SumOfIdeals(F,G);
```

$$f1 := 2\,x^3 + y^2\,z$$
$$f2 := 2\,x^2\,y^6 + x\,y^2\,z$$
$$g1 := -x\,y\,z + x^5\,y^2\,z$$
$$g2 := -y^3\,z^3 + x\,y^2\,z^3$$
$$F := [\,2\,x^3 + y^2\,z,\ 2\,x^2\,y^6 + x\,y^2\,z\,]$$
$$G := [\,-x\,y\,z + x^5\,y^2\,z,\ -y^3\,z^3 + x\,y^2\,z^3\,]$$
$$[\,2\,x^3 + y^2\,z,\ 2\,x^2\,y^6 + x\,y^2\,z,\ -x\,y\,z + x^5\,y^2\,z,\ -y^3\,z^3 + x\,y^2\,z^3\,]$$

```
>
```

18. Computation of a product I . J of ideals I = <f1,f2,...,fr> and J = <g1,g2,...,gs> as in Cox et al.

Given:

1. Polynomials f1,f2,...,fr entered as a list F that generate ideal I = <f1,f2,...,fr> and polynomials g1,g2,...,gs entered as a list G that generate ideal J = <g1,g2,...,gs>.

Algorithm:

1. Return a list [F[i] * G[j]], i=1..r, j=1..s, that generates I . J per Proposition 6 on page 183 in Cox et al..

Usage:

```
f1:=2*x^3+y^2*z;
f2:=2*x^2*y^6+y^2*z*x;
g1:=-x*y^3*z+y^2*z*x^5;
g2:=-y*z^3+y^2*z^3*x;
F:=[seq(f||i,i=1..2)];
```

G:=[seq(g||j,j=1..2)];
ProductOfIdeals(F,G);

```
> f1:=2*x^3+y^2*z;
  f2:=2*x^2*y^6+y^2*z*x;
  f3:=z-x^6+y^4;
  g1:=-x*y^3*z+y^2*z*x^5;
  g2:=-y*z^3+y^2*z^3*x;
  F:=[seq(f||i,i=1..3)];
  G:=[seq(g||j,j=1..2)];
  ProductOfIdeals(F,G);
```

$$f1 := 2\,x^3 + y^2\,z$$

$$f2 := 2\,x^2\,y^6 + x\,y^2\,z$$

$$f3 := z - x^6 + y^4$$

$$g1 := -x\,y^3\,z + x^5\,y^2\,z$$

$$g2 := -y\,z^3 + x\,y^2\,z^3$$

$$F := [\,2\,x^3 + y^2\,z,\ 2\,x^2\,y^6 + x\,y^2\,z,\ z - x^6 + y^4\,]$$

$$G := [\,-x\,y^3\,z + x^5\,y^2\,z,\ -y\,z^3 + x\,y^2\,z^3\,]$$

$$[\,-2\,x^4\,y^3\,z + 2\,x^8\,y^2\,z - x\,y^5\,z^2 + x^5\,y^4\,z^2,\ -2\,x^3\,y\,z^3 + 2\,x^4\,y^2\,z^3 - y^3\,z^4 + y^4\,x\,z^4,$$
$$-2\,x^3\,y^9\,z + 2\,x^7\,y^8\,z - y^5\,x^2\,z^2 + x^6\,y^4\,z^2,\ -2\,y^7\,z^3\,x^2 + 2\,x^3\,y^8\,z^3 - x\,y^3\,z^4 + x^2\,y^4\,z^4,$$
$$-x\,y^3\,z^2 + x^5\,y^2\,z^2 + x^7\,y^3\,z - x^{11}\,y^2\,z - y^7\,x\,z + z\,x^5\,y^6,$$
$$-y\,z^4 + x\,y^2\,z^4 + x^6\,y\,z^3 - x^7\,y^2\,z^3 - y^5\,z^3 + x\,y^6\,z^3\,]$$

```
>
```

19. Computation of an interesection I cap J of two ideals I = <f1,f2,...,fr> and J = <g1,g2,...,gs> as in Cox et al.

Given:

1. Polynomials f1,f2,...,fr entered as a list F that generate ideal I = <f1,f2,...,fr> and polynomials g1,g2,...,gs entered as a list G that generate ideal J = <g1,g2,...,gs>.

Algorithm (see page 186):

1. Compute a Groebner basis GB for the ideal <t*f1,...,t*fr,(1-t)*g1,...,(1-t)*gs> in k[x1,x2,...,xn,t] with respect to a lex order in which t > x1 > x2 > ... >xn.
2. Return only those polynomials from the list GB which do not contain t, that is, belong to the ring k[x1,x2,...,xn,t].

f1:=x^2*y;

```
g1:=x*y^2;
F:=[f1];
G:=[g1];
IntersectionOfIdeals(F,G);
```

```
> f1:=x^2*y;
  g1:=x*y^2;
  F:=[f1];
  G:=[g1];
  IntersectionOfIdeals(F,G);
```

$$f1 := y\,x^2$$

$$g1 := y^2\,x$$

$$F := [y\,x^2]$$

$$G := [y^2\,x]$$

$$[x^2\,y^2]$$

```
> f1,f2,f3:=x^2*y+2*x*y^6,-y+2*x,x-y^6+z*y*x;
  g1,g2:=2*x*y^4-z*y^3,z-x^6+z*y;
  F,G:=[f1,f2,f3],[g1,g2];
  IntersectionOfIdeals(F,G);
```

$$f1, f2, f3 := y\,x^2 + 2\,y^6\,x,\ 2\,x - y,\ x - y^6 + x\,y\,z$$

$$g1, g2 := 2\,y^4\,x - y^3\,z,\ z - x^6 + y\,z$$

$$F, G := [y\,x^2 + 2\,y^6\,x,\ 2\,x - y,\ x - y^6 + x\,y\,z],\ [2\,y^4\,x - y^3\,z,\ z - x^6 + y\,z]$$

$$[-32\,z^9\,y^3 + 2048\,z^4\,y^9 + 2048\,z^4\,y^{10} - 16\,y^3\,z^{10} + 1024\,y^9\,z^5 + 4160\,z\,y^{10} + 1024\,z^5\,y^{10} - 65\,z^6\,y^3$$
$$+ 4160\,z\,y^9 - 8\,z^7\,y^3 + 512\,z^2\,y^9 + 512\,z^2\,y^{10} + 1536\,z^3\,y^{10} - 24\,z^8\,y^3 + 1536\,z^3\,y^9,\ 8\,z^9\,y^3$$
$$- 512\,z^4\,y^9 - 512\,z^4\,y^{10} + 1984\,z\,y^{10} + 2048\,z\,y^{11} + z^6\,y^3 - 64\,z\,y^9 + 6\,z^7\,y^3 - 384\,z^2\,y^9 - 32\,z^6\,y^4$$
$$- 384\,z^2\,y^{10} - 768\,z^3\,y^{10} + 12\,z^8\,y^3 - 768\,z^3\,y^9,\ 854768756768\,y^3\,x\,z^5 - 27352600216576\,y^8\,z$$
$$- 34870853624\,z^9\,y^3 + 2231734631936\,z^4\,y^9 + 2231734631936\,z^4\,y^{10} - 9748605128768\,z\,y^{10}$$
$$+ 152321955137\,z^6\,y^3 - 37101205345344\,z\,y^9 + 240970008198\,z^7\,y^3 - 15422080524672\,z^2\,y^9$$
$$- 15422080524672\,z^2\,y^{10} - 824797152000\,z^3\,y^{10} + 12887455500\,z^8\,y^3 - 824797152000\,z^3\,y^9,$$
$$10598131136\,z^9\,y^3 - 678280392704\,z^4\,y^9 - 678280392704\,z^4\,y^{10} - 1736249037185\,y^3\,z$$
$$- 12592168716032\,z\,y^{10} + 3472498074370\,y^4\,x + 196752636188\,z^6\,y^3 - 12592168716032\,z\,y^9$$
$$+ 175084938144\,z^7\,y^3 - 11205436041216\,z^2\,y^9 - 11205436041216\,z^2\,y^{10} - 5831822691328\,z^3\,y^{10}$$
$$+ 91122229552\,z^8\,y^3 - 5831822691328\,z^3\,y^9,\ 427384378384\,y^3\,x^2\,z^4 - 6838150054144\,y^8\,z$$
$$- 28778902512\,z^9\,y^3 + 1841849760768\,z^4\,y^9 + 1841849760768\,z^4\,y^{10} + 4251469221696\,z\,y^{10}$$
$$- 6838150054144\,y^7\,z - 66429206589\,z^6\,y^3 + 4251469221696\,z\,y^9 - 78868179184\,z^7\,y^3$$
$$+ 5047563467776\,z^2\,y^9 + 5047563467776\,z^2\,y^{10} + 6235349547264\,z^3\,y^{10} - 97427336676\,z^8\,y^3$$

$$+ 6235349547264\, z^3 y^9,\ 106846094596\, x^3 y^3 z^3 + 6850790808\, z^9 y^3 - 438450611712\, z^4 y^9$$
$$- 438450611712\, z^4 y^{10} + 556676946624\, z\, y^{10} - 854768756768\, y^7 z - 8698077291\, z^6 y^3$$
$$+ 556676946624\, z\, y^9 - 854768756768\, y^6 z + 12492991033\, z^7 y^3 - 799551426112\, z^2 y^9$$
$$- 799551426112\, z^2 y^{10} - 658297977472\, z^3 y^{10} + 10285905898\, z^8 y^3 - 658297977472\, z^3 y^9,$$
$$106846094596\, x^4 y^3 z^2 - 427384378384\, y^5 z + 139448480\, z^9 y^3 - 8924702720\, z^4 y^9$$
$$- 8924702720\, z^4 y^{10} - 1050949617856\, z\, y^{10} + 16421087779\, z^6 y^3 - 1050949617856\, z\, y^9$$
$$- 427384378384\, y^6 z + 3338243240\, z^7 y^3 - 213647567360\, z^2 y^9 - 213647567360\, z^2 y^{10}$$
$$+ 207760734720\, z^3 y^{10} - 3246261480\, z^8 y^3 + 207760734720\, z^3 y^9,\ 3419075027072\, x^5 y^3 z$$
$$- 6838150054144\, y^5 z + 56122313208\, z^9 y^3 - 3591828045312\, z^4 y^9 - 3591828045312\, z^4 y^{10}$$
$$+ 8128391397440\, z\, y^{10} - 6838150054144\, y^4 z - 127006115585\, z^6 y^3 + 8128391397440\, z\, y^9$$
$$- 288055318022\, z^7 y^3 + 18435540353408\, z^2 y^9 + 18435540353408\, z^2 y^{10} + 1378480128768\, z^3 y^{10}$$
$$- 21538752012\, z^8 y^3 + 1378480128768\, z^3 y^9,\ 1736249037185\, x^6 y - 1532473104\, z^9 y^3$$
$$+ 98078278656\, z^4 y^9 - 1736249037185\, y^2 z + 98078278656\, z^4 y^{10} + 17051500950208\, z\, y^{10}$$
$$- 266429702347\, z^6 y^3 + 17051500950208\, z\, y^9 - 1736249037185\, y\, z - 51869080116\, z^7 y^3$$
$$+ 3319621127424\, z^2 y^9 + 3319621127424\, z^2 y^{10} - 3468166451008\, z^3 y^{10} + 54190100797\, z^8 y^3$$
$$- 3468166451008\, z^3 y^9,\ -266429702347\, z^6 y^3 - 51869080116\, z^7 y^3 + 54190100797\, z^8 y^3$$
$$- 1532473104\, z^9 y^3 + 17051500950208\, z\, y^9 + 3319621127424\, z^2 y^9 - 3468166451008\, z^3 y^9$$
$$+ 98078278656\, z^4 y^9 + 17051500950208\, z\, y^{10} + 3319621127424\, z^2 y^{10} - 3468166451008\, z^3 y^{10}$$
$$+ 98078278656\, z^4 y^{10} - 3472498074370\, x\, z - 3472498074370\, x\, y\, z + 3472498074370\, x^7]$$

```
>
```

20. Computation of a LCM of two multivariate polynomials f and g by computing interesection I cap J of two ideals I = <f> and J = <g>. Then, we know from Proposition 13 on page 187 in Cox et al. that I cap J is generated by the LCM(f,g).

Given:

1. Polynomials f and g whose LCM is to be computed in some polynomial ring k[x1,x2,...,xn].

Algorithm (see page 187):

1. Let I = <f> and J = <g>. Compute a Groebner basis GB for the intersection I cap J, which is a principal ideal generated by the LCM(f,g) using our procedure IntersectionOfIdeals.
2. Return the only generator for the interesection I cap J which is the LCM of f and g.

Usage:

```
f:=x^2*y;
```

g:=x*y^2;
LeastCommonMultiple(f,g);

```
> f:=factor(x^2*y+2*x-y);
  g:=factor(x*y^2+x^5-y^3*x);
  LCMfg:=factor(LeastCommonMultiple(f,g));
```

$$f := y\,x^2 + 2\,x - y$$

$$g := x\,(y^2 + x^4 - y^3)$$

$$LCMfg := x\,(y\,x^2 + 2\,x - y)\,(y^2 + x^4 - y^3)$$

```
>
```

21. Computation of a GCD of two multivariate polynomials f and g by using formula (2) on page 187 rather than built into Maple procedure Gcd.

Given:

1. Polynomials f and g whose GCD is to be computed in some polynomial ring k[x1,x2,...,xn].

Algorithm (see formula (2) on page 187):

1. Let I = <f> and J = <g>. Compute LCM(f,g) using procedure LeastCommonMultiple defined above.
2. Return the ration (f*g)/LCM(f,g).

Usage:

f:=x^2*y;
g:=x*y^2;
GreatestCommonDivisor(f,g);

```
> f:=factor((x^2*y+2*x-y)*(x^2-y^2));
  g:=factor((x*y^2+x^5-y^3*x)*(x-y));
  LCMfg:=factor(LeastCommonMultiple(f,g));
  GCDfg:=GreatestCommonDivisor(f,g);
  simplify(f*g - LCMfg*GCDfg);
```

$$f := (y\,x^2 + 2\,x - y)\,(x - y)\,(x + y)$$

$$g := x\,(y^2 + x^4 - y^3)\,(x - y)$$

$$LCMfg := x\,(y\,x^2 + 2\,x - y)\,(x - y)\,(x + y)\,(y^2 + x^4 - y^3)$$

$$GCDfg := x - y$$

$$0$$

```
>
```

22. Computation of the ideal quotient I : J of two ideals I = <f1,f2,...,fr> and J = <g1,g2,...,gs> using

an algortihm described on page 194 as in Cox et al..

Given:

1. Polynomials f1,f2,...,fr entered as a list F and polynomials g1,g2,...,gs entered as a list G.

Algorithm is as follows:

1. Compute a basis for I : <gi> for each i = 1,..., s, using Theorem 11. Thus,
   STEP 1, we first compute a Groebner basis H[i]i = {h1,...,hp} for I cap <gi>, i=1,..., s, using IntersectionOfIdeals procedure defined above.
   STEP 2 Then, a basis for I : <gi> is Hg[i] = {h1/gi,...,hp/gi}, i=1,...,s.

2. STEP 3: Compute a basis for I : J by applying formula (5) on page 193 s - 1 times as explained on page 194.

```
f1:=2*x^3+y^2*z;
f2:=2*x^2*y^6+y^2*z*x;
g1:=-x*y^3*z+y^2*z*x^5;
g2:=-y*z^3+y^2*z^3*x;
g3:=x*y*z-z^3*x;
F:=[seq(f||i,i=1..2)];
G:=[seq(g||j,j=1..1)];
QuotientOfIdeals(F,G);
```

```
> f1:=2*x^3+y^2*z;
  f2:=2*x^2*y^6+y^2*z*x;
  g1:=-x*y^3*z+y^2*z*x^5;
  g2:=-y*z^3+y^2*z^3*x;
  g3:=x*y*z-z^3*x;
  F:=[seq(f||i,i=1..2)];
  G:=[seq(g||j,j=1..1)];
  QuotientOfIdeals(F,G);
```

$$f1 := 2\,x^3 + y^2\,z$$

$$f2 := 2\,x^2\,y^6 + y^2\,z\,x$$

$$g1 := -x\,y^3\,z + y^2\,z\,x^5$$

$$g2 := -y\,z^3 + y^2\,z^3\,x$$

$$g3 := x\,y\,z - z^3\,x$$

$$F := [\,2\,x^3 + y^2\,z,\ 2\,x^2\,y^6 + y^2\,z\,x\,]$$

$$G := [\,-x\,y^3\,z + y^2\,z\,x^5\,]$$

$$[4\,y^{14} - z^2, x\,z + 2\,y^{10}, 2\,y^4\,x + z, -y^6 + x^2]$$

>

23. Procedure 'Beziercubic' defines a random Bezier cubic in parametric form [X = X(t), Y = Y(t)]. It returns functions X(t) and Y(t) as expressions followed by a list of coordinates of four control points of type list(list), in that order. It can be used to create a random Bezier cubic by calling it as

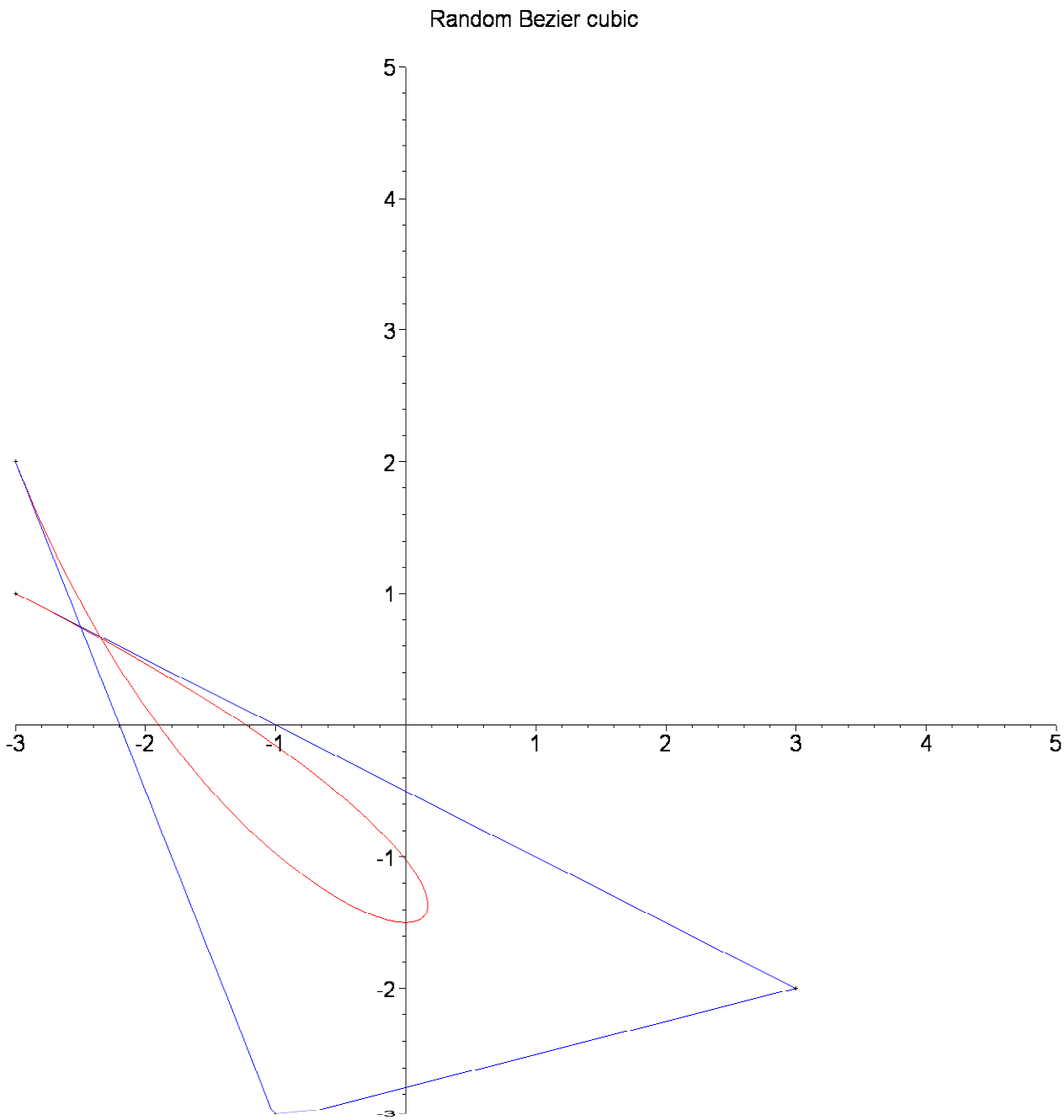
Beziercubic(r)

where r is a range, e.g., -3..5, in which random integer coordinates of four control points [x1,y1], [x2,y2], [x3,y3], [x4,y5] must be contained. If r is a llist of four lists as in [[x1,y1], [x2,y2], [x3,y3], [x4,y5]], then these coordinates are used for the control points. In either case, the list [[x1,y1], [x2,y2], [x3,y3], [x4,y5]] is returned as the third item in the output with X and Y being, respectively, the first and the second item in the output.

```
> X,Y,pp:=Beziercubic(-3..5);
  for i from 1 to 3 do
        PL||i:=plot([pp[i],pp[i+1]],style=line,color=blue):
  end do:
  PP:=plot(pp,x=-3..5,y=-3..5,style=point,color=black,symbol=cross):

  BC:=plot([X,Y,t=0..1]):
  plots:-display({PP,BC,PL1,PL2,PL3},title=`Random Bezier cubic`);
```

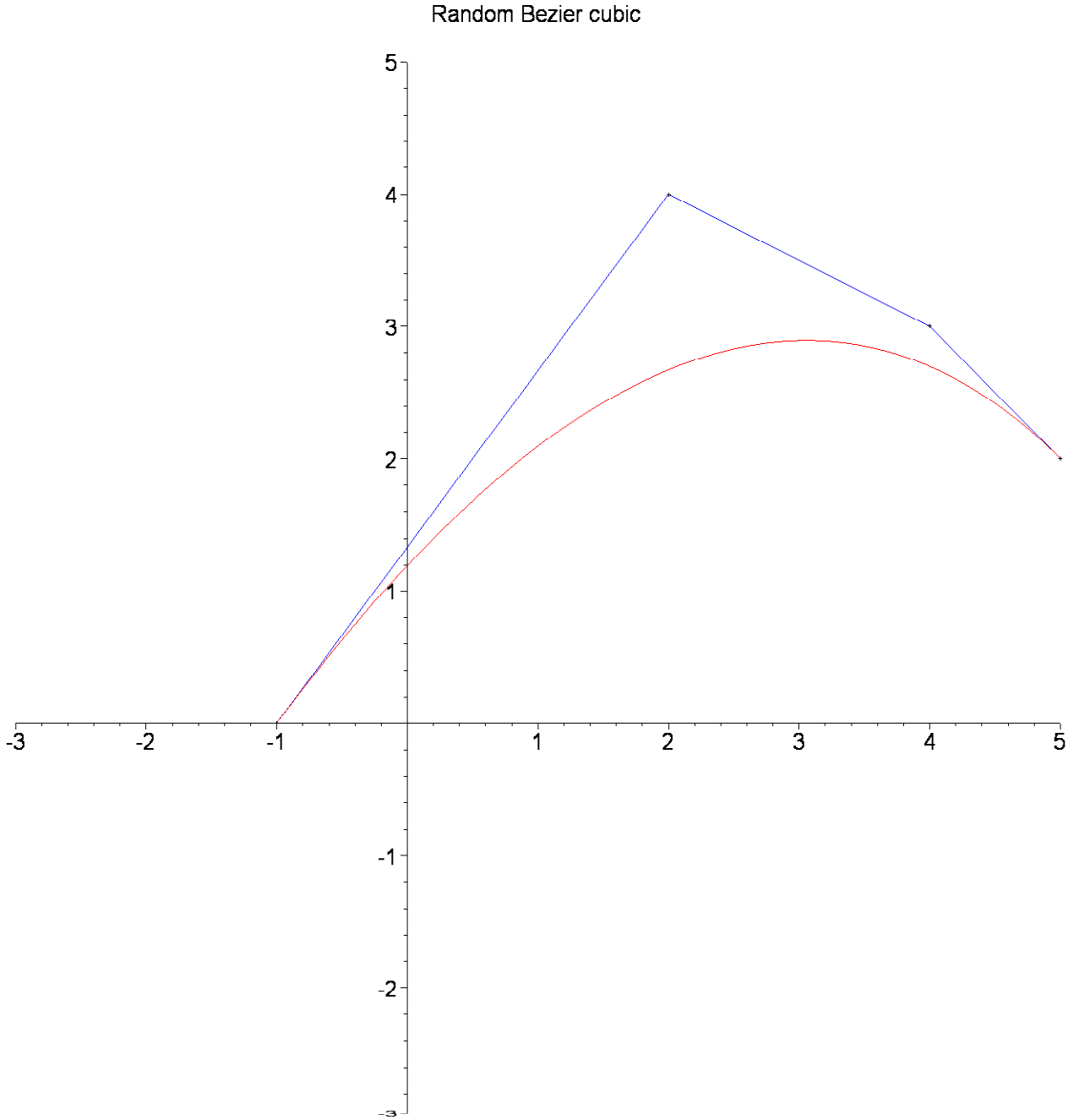
$$X, Y, pp := -3 + 6\,t + 6\,t^2 - 12\,t^3, 2 - 15\,t + 18\,t^2 - 4\,t^3, [[-3, 2], [-1, -3], [3, -2], [-3, 1]]$$




```
> X,Y,pp:=Beziercubic(-3..5);
  for i from 1 to 3 do
        PL||i:=plot([pp[i],pp[i+1]],style=line,color=blue):
  end do:
  PP:=plot(pp,x=-3..5,y=-3..5,style=point,color=black,symbol=cross):

  BC:=plot([X,Y,t=0..1]):
  plots:-display({PP,BC,PL1,PL2,PL3},title=`Random Bezier cubic`);
```

$$X, Y, pp := -1 + 9\,t - 3\,t^2,\ 12\,t - 15\,t^2 + 5\,t^3,\ [[-1, 0], [2, 4], [4, 3], [5, 2]]$$



>

24. Procedure 'implicitBeziercubic' reads in from the database a polynomial in variables x1,y1,x2,y2,x3,y3,x4,y4, and x,y which gives a Bezier cubic in implicit form. Points [x1,y1], [x2,y2], [x3,y3], [x4,y4] are four control points of the cubic. This procedure can be used without any argument, like this:

implicitBeziercubic();

and then it returns the implicit polynomial. It can also be used with one argument L of the type list(list) where L is a list of four lists [x1,y1], [x2,y2], [x3,y3], [x4,y4] that give the control point:

implicitBeziercubic([[x1,y1], [x2,y2], [x3,y3], [x4,y4]]);

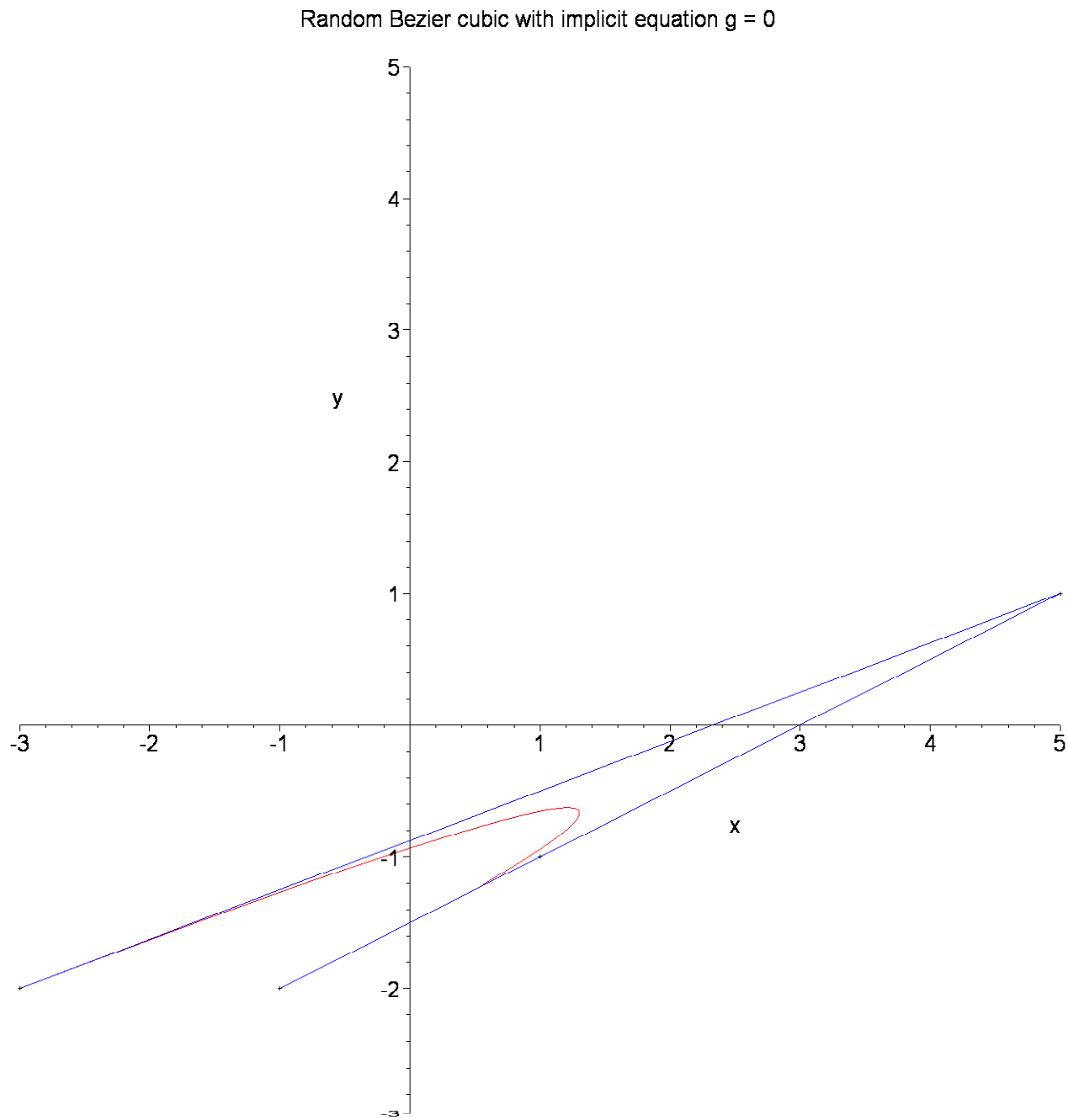
```
> X,Y,pp:=Beziercubic(-3..5);
  g:=implicitBeziercubic(pp);
```

$$X, Y, pp := -3 + 24\,t - 42\,t^2 + 22\,t^3, -2 + 9\,t - 18\,t^2 + 10\,t^3, [[-3, -2], [5, 1], [-1, -2], [1, -1]]$$

$$g := -22724 + 23484\,x - 54408\,y - 6600\,y\,x^2 + 37488\,x\,y + 1000\,x^3 - 8304\,x^2 - 10648\,y^3 - 42144\,y^2 + 14520\,y^2\,x$$

```
> for i from 1 to 3 do
        PL||i:=plot([pp[i],pp[i+1]],style=line,color=blue):
  end do:
  PP:=plot(pp,x=-3..5,y=-3..5,style=point,color=black,symbol=cross):

  BC:=plot([X,Y,t=0..1]):
  plots:-display({PP,BC,PL1,PL2,PL3},title=`Random Bezier cubic with
  implicit equation g = 0`);
```




```
>
```

25. Procedure 'curvature' computes curvature of a paramerized curve in R^3, that is, a curve given as [X,Y,Z] where X,Y,Z are expressions in t that give points (x,y,z) on the curve.
When X,Y,Z, are functions of t, they maust be entered as X(t),Y(t), and Z(t):

curvature(X,Y,Z);
curvature(X(t),Y(t),Z(t));

```
> X,Y,pp:=Beziercubic(-3..5);
  curvature(X,Y,0);
```

$$X, Y, pp := 5 - 12\,t + 6\,t^2 + 6\,t^3, 4 - 12\,t + 9\,t^2 - 2\,t^3, [[5, 4], [1, 0], [-1, -1], [5, -1]]$$

$$\frac{\sqrt{(2-16\,t+11\,t^2)^2}}{6\,(8-20\,t+5\,t^2+6\,t^3+10\,t^4)^{(3/2)}}$$

```
> X,Y,Z:=r*cos(t),r*sin(t),a*t;
  curvature(X,Y,Z);
```

$$X, Y, Z := r\cos(t), r\sin(t), a\,t$$

$$\frac{\sqrt{a^2\,r^2+r^4}}{(a^2+r^2)^{(3/2)}}$$

```
>
```

26. Last procedure is the 'setup' procedure. It is automatically executed when RJgrobner is loaded. It prints welcome message and defines one convert function `convert/set_to_pts`.

Procedure `convert/set_to_pts` converts Maple output from the procedure 'solve' to a list of points that can be plotted either in the plane or in the space.

Usage:

S:=remove(has,map(allvalues,{solve(sys,vars)}),I);
PTS:=map(convert,S,set_to_pts);

```
> sys:={x^2+y^2-2,y-x^2};
  vars:={x,y};
```

$$sys := \{y-x^2, x^2+y^2-2\}$$

$$vars := \{x, y\}$$

```
> S:=remove(has,map(allvalues,{solve(sys,vars)}),I);
  PTS:=map(convert,S,set_to_pts);
```

$$S := \{\{x=1, y=1\}, \{x=-1, y=1\}\}$$

$$PTS := \{[1, 1], [-1, 1]\}$$

```
>
```

27. Note: In the database, a polynomial is stored under the name _implicitbezierpolynomial that gives implicit form of a Bezier cubic. This polynomial is read by the procedure implicitBeziercubic. To see it displayed, type

implicitBeziercubic();

without any argument. This polynomial belogs to R[x,y,x1,x2,x3,x4,y2,y2,y3,y4] where [x1,y1], [x2,y2], [x3,y3], [x4,y4] are coordinates of four points that defineBezier cubic.

```
> g:=implicitBeziercubic();
```

$$g := -54\,x1\,x3\,y^2\,y3\,x - 27\,x1\,x3\,y3\,x4\,y^2 - 63\,x1\,y1\,x3\,x4\,y^2 + 243\,y\,x2\,x3\,y2\,y4\,x$$

$$
\begin{aligned}
&- 9\, y\, x2\, y1\, x4\, x1\, y4 - 27\, y\, x1^2\, y2\, y3\, x - 6\, y\, y1\, x4\, x1^2\, y4 + 27\, y\, x4\, y2\, x1^2\, y4 - 63\, y\, x1^2\, y3\, y4\, x \\
&+ 9\, y\, x2\, x1\, y3\, y1\, x + 126\, y\, x2\, x1\, y3\, y4\, x - 18\, y\, x2\, x1^2\, y3\, y4 + 81\, y\, x1\, x3\, y3\, y4\, x \\
&- 27\, y\, x1\, x4\, y2^2\, x + 108\, y\, x1\, y1\, x3\, x4\, y3 + 54\, y\, x1^2\, y3\, y4\, x3 - 36\, y\, x1^2\, y2\, y4\, x3 + 9\, y\, x1^2\, y2\, y4\, x \\
&- 54\, x2\, y3\, x\, x4\, y^2 - 27\, y1^2\, x\, x3\, y4\, x2 + 36\, y1^2\, x\, x2\, x4\, y3 - 27\, x2\, y1\, x3\, x1\, y^2 + 81\, x2\, x1\, y2\, x3\, y^2 \\
&+ 27\, x2\, x1\, y2\, x4\, y^2 + 27\, y\, x1\, x4\, y3^2\, x - 54\, y\, x1\, y1\, y4\, x3^2 - 27\, y\, x1\, y1\, x4^2\, y2 - 162\, y\, y1\, x2^2\, x4\, y3 \\
&- 108\, y\, x2^2\, y4\, y2\, x + 81\, y\, y1\, x3\, y4\, x2^2 - 180\, y\, x1\, x3\, y2\, y4\, x + 81\, y\, y1\, x2^2\, y3\, x \\
&- 243\, y\, x2\, y3\, x4\, y2\, x - 63\, y1\, x^2\, x4\, y2\, y4 + 63\, y1\, x\, x4^2\, y2\, y + 27\, y1\, x^2\, x1\, y2\, y3 + 63\, y1^2\, x\, x3\, x4\, y \\
&- 27\, y1\, x^2\, y3\, x4\, y2 - 9\, y1\, x^2\, x1\, y2\, y4 - 27\, y\, x2\, x3\, y1^2\, x4 + 108\, y\, x2\, x4\, y2^2\, x + 54\, y\, x4\, y2\, x^2\, y3 \\
&+ 54\, y\, y1\, x4\, y4\, x2^2 + 54\, y\, y1\, x2^2\, x4\, y2 + 54\, y\, x1\, y2\, y4\, x2^2 - 54\, x2\, x3\, y^2\, y4\, x - 162\, x2\, y2\, x\, x3\, y^2 \\
&- 27\, y1^2\, x\, x4\, y4\, x2 - 108\, y\, x2\, x4\, y2\, x1\, y4 - 243\, y\, x2\, y1\, x3\, y3\, x + 27\, y\, y3\, x4^2\, y2\, x \\
&+ 81\, y\, x2\, x1\, y2\, y4\, x + 54\, y\, x2\, y1\, x3\, x1\, y4 - 9\, y\, y3\, x4^2\, y1\, x + 6\, y1^2\, x\, x4\, x1\, y4 + 153\, y1\, x^2\, x3\, y2\, y4 \\
&+ 54\, x3\, y3^2\, x1\, x4\, y + 81\, y\, x2\, y1\, x3\, y2\, x - 18\, y\, x4\, y2\, x^2\, y4 + 81\, y\, x4\, x3\, y2\, y1\, x2 + 18\, y\, x4\, y3\, x^2\, y4 \\
&- 6\, y\, y1\, x4^2\, y4\, x - 27\, y\, x1\, y2\, x4^2\, y3 - 54\, x3\, y3^2\, y1\, x4\, x1 - 54\, x3^2\, y3\, x1\, y4\, y + 54\, x3^2\, y3\, y1\, x1\, y4 \\
&- 54\, x3^2\, y3\, y1\, x1\, y - 54\, x3^2\, y3\, y1\, x4\, y + 18\, x2\, x4\, y^2\, y4\, x + 36\, y\, y1\, x2\, x4^2\, y3 + 6\, y\, y1\, x4^2\, x1\, y4 \\
&- 81\, y\, x4\, x1\, y2^2\, x3 - 126\, y\, x4\, y1\, x3\, y2\, x + 45\, y\, x4\, y1\, x3\, y4\, x + 6\, y\, x4\, x1\, y4^2\, x - 27\, y2^2\, x4^2\, x2\, y \\
&+ 18\, x2\, x1^2\, y4\, y^2 + 162\, x2\, x3\, y^2\, y3\, x - 54\, x1\, x2\, y^2\, y2\, x + 54\, x1\, x2\, y^2\, y3\, x + 27\, y2^2\, x4^2\, y1\, x2 \\
&- 18\, y1^2\, x^2\, x4\, y2 - 27\, x1^2\, y2\, x4\, y^2 + 153\, x1\, x3\, y2\, x4\, y^2 + 18\, x1\, x3\, y^2\, y4\, x - 18\, x1\, x2\, y^2\, y4\, x \\
&+ 3\, y1\, x4\, x1^2\, y4^2 + 3\, y1\, x4\, y4^2\, x^2 - 3\, y1^2\, x^2\, x1\, y4 + 27\, x3\, y3^2\, x1^2\, y + 27\, y1\, x\, x1\, y4^2\, x2 \\
&- 6\, y1\, x^2\, x1\, y\, y4 - 18\, y1\, x^2\, x1\, y\, y2 + 27\, y1\, x4\, x2^2\, y^2 - 3\, y1^2\, x4^2\, x1\, y4 + 9\, y\, x4\, x1\, y2\, y4\, x3 \\
&- 54\, y\, x4\, y1\, x3\, y4\, x2 + 81\, y\, x4\, x3\, y2^2\, x - 9\, y\, x4\, x3\, y2\, y4\, x - 81\, y\, x2\, x1\, y2\, y4\, x3 \\
&+ 54\, y\, x2\, y3\, x^2\, y4 + 54\, y1\, x3^3\, y4\, y - 27\, x3\, y3^2\, x1^2\, y4 + 27\, x3^2\, y3\, y1^2\, x4 - 81\, y\, x2\, x1\, y2\, y3\, x \\
&+ 27\, y\, y3\, x4\, y4\, x1^2 + 9\, y\, x1\, y4\, x3\, x4\, y1 + 162\, y\, x2\, y2\, x^2\, y3 - 27\, y\, x1\, y3\, x4^2\, y1 - 54\, y\, y1\, x2^2\, y2\, x \\
&+ 162\, y\, x2\, x1\, y2\, x4\, y3 - 54\, y\, x2\, y2\, x^2\, y4 - 45\, y\, x1\, y3\, x4\, y4\, x + 27\, y2^3\, x^3 + 9\, y4^2\, x^2\, x3\, y \\
&+ 18\, x4\, y^3\, x1\, x2 - 81\, x2^2\, y3\, x3\, y^2 + 27\, y4\, x\, x3^2\, y^2 - 3\, x4\, y\, y1^2\, x^2 - 27\, x4\, y\, y2^2\, x^2 - y1^3\, x^3 \\
&+ y1^3\, x4^3 + 81\, y3^2\, x^2\, x3\, y - 81\, y3\, x\, x3^2\, y^2 + 18\, y1\, x^3\, y3\, y4 - 3\, x1\, y4\, x4^2\, y^2 + 54\, x4\, y1\, x3^2\, y^2 \\
&- 9\, y1^2\, x^2\, x2\, y - 54\, y1\, x^2\, x1\, y3^2 + 54\, y1\, x^2\, x2\, y4^2 - 18\, y1\, x^3\, y2\, y4 - 27\, y1\, x\, x3^2\, y^2 \\
&- 27\, y3\, x\, y1^2\, x3^2 + 27\, y1\, x^2\, x3\, y3\, y4 + 27\, x4\, x2\, y4\, x3\, y^2 - 27\, y1\, x\, y4\, x3^2\, y + 18\, y1\, x^2\, x1\, y\, y3 \\
&+ 18\, y1\, x\, x1\, y^2\, x2 + 27\, y1^2\, x\, x2\, x3\, y + 18\, y\, x1\, y2\, x^2\, y4 - 54\, y\, x2\, x1\, y2^2\, x4 + 54\, y1\, x\, x1\, y3^2\, x4 \\
&+ 54\, y\, x2\, x1\, y2^2\, x - 18\, y\, x1\, y3\, x^2\, y4 - 54\, y\, x1\, y2\, x^2\, y3 - 3\, y1\, x\, x4^2\, y^2 + 54\, y1\, x^3\, y2\, y3 + x1^3\, y^3 \\
&+ y4^3\, x^3 - 81\, y1\, x^2\, x3\, y2\, y3 + 27\, y1\, x\, x3\, y4^2\, x1 - 81\, y1\, x\, y4\, x3^2\, y2 - 81\, y\, x2\, y2^2\, x^2 \\
&+ 108\, y1\, x\, x4\, y2\, y4\, x2 - 162\, y1\, x\, x4\, y3^2\, x2 + 81\, y1\, x\, x3\, y3\, x4\, y2 - 27\, x2\, y1\, x3\, x4\, y^2 \\
&- 153\, y1\, x^2\, y3\, x2\, y4 + 81\, y1\, x\, x3^2\, y2\, y - x4^3\, y^3 - 81\, y\, x2\, y3^2\, x^2 - 18\, y1\, x^2\, x3\, y\, y4 \\
&+ 27\, x2\, x1\, y4^2\, x3\, y - 27\, x2\, y1\, x3\, x1\, y4^2 - 9\, x2\, x1^2\, y4^2\, y + 27\, y\, x1\, y2^2\, x^2 + 27\, x2^2\, y4^2\, y2\, x
\end{aligned}
$$

$$+ 27\, x1\, y4^2\, x2^2\, y - 9\, y1^2\, x^2\, y3\, x4 - x1^3\, y4^3 - 54\, y\, x4^2\, y2^2\, x - 9\, y\, x2\, y4^2\, x^2 - 27\, y\, x4\, y3^2\, x^2$$
$$+ 243\, y\, x1\, x3\, y2\, y3\, x + 81\, y\, x4\, x3\, y2\, y3\, x - 81\, x2\, y3\, x3\, x4\, y^2 - 27\, y1^2\, x\, y4\, x3\, x4 + 9\, y\, x1\, x3\, y4^2\, x$$
$$+ 162\, y\, x1\, y4\, x3^2\, y2 - 18\, y1\, x^2\, y3\, x4\, y + 18\, y1\, x\, x3\, y^2\, x4 - 54\, y1\, x^2\, y2\, x3\, y + 162\, y1\, x^2\, x2\, y3^2$$
$$- 81\, y2^2\, x\, x1\, x3\, y + 81\, y2\, x^2\, x3\, y3\, y4 + 54\, y2\, x\, y1\, y4\, x2^2 - 162\, x1\, x3^2\, y2\, y^2$$
$$- 162\, y\, x1\, x3\, y3\, x4\, y2 + 18\, y2\, x\, x3\, y1^2\, x4 - 81\, y2\, x\, x1\, y3\, y4\, x3 - 27\, x2^3\, y^3 + 81\, y\, x2\, x3\, y3\, x4\, y1$$
$$+ 45\, y\, x1\, x4\, y2\, y4\, x - 81\, y\, x2\, y4\, x3^2\, y1 + 54\, y\, x2^2\, y4^2\, x - 81\, y1\, x\, x2^2\, y4^2 - 27\, y1\, x^2\, x3\, y4^2$$
$$- 81\, y\, x2\, y3\, x3\, y4\, x + 9\, y\, x2\, y3\, x4\, y4\, x + 81\, y\, x2\, x4\, y3^2\, x - 81\, y2\, x\, y4\, x3^2\, y + 81\, y2\, x\, x1\, y3^2\, x4$$
$$+ 27\, y2\, x\, x1^2\, y3\, y4 + 162\, x2^2\, y3\, x4\, y^2 - 27\, y1\, x\, x4^2\, y2^2 + 18\, y1^2\, x\, y3\, x4^2 + 81\, x2\, y4\, x3^2\, y^2$$
$$+ 3\, y1^2\, x4^2\, y4\, x + 81\, y2\, x\, y1\, x2\, x4\, y3 - 162\, y2\, x\, y1\, x3\, y4\, x2 - 54\, y2^2\, x\, y1\, x2\, x4$$
$$+ 27\, y2\, x^2\, x1\, y3\, y4 + 54\, y2\, x\, x1\, y4^2\, x2 - 54\, y2^2\, x\, x1\, y4\, x2 + 9\, y2\, x\, x4^2\, y^2 - 27\, y1\, x\, x2^2\, y^2$$
$$+ 9\, y1^2\, x^2\, x3\, y + 162\, y2^2\, x\, x1\, y4\, x3 - 81\, y2^2\, x\, x1\, x4\, y3 - 54\, y2^2\, x\, x4\, x1\, y4 - 3\, y1\, x\, x1^2\, y^2$$
$$- 18\, x2\, y3\, x4^2\, y^2 - 81\, x2^2\, y4\, x3\, y^2 - 81\, y2\, x^2\, y1\, x2\, y3 - 9\, x2\, y3\, y1\, x4\, x1\, y - 81\, x2\, y3\, x1\, y4\, x3\, y$$
$$- 18\, y1\, x3\, x4^2\, y^2 + 3\, x4^2\, y^2\, y4\, x + 3\, y\, x1\, y1^2\, x4^2 + 81\, x2\, y3^2\, x4\, y2\, x + 81\, x2^2\, y3\, y4\, x3\, y$$
$$+ 81\, x2\, y3^2\, x1\, x4\, y + 3\, y\, x1\, y4^2\, x^2 - 9\, y3\, x\, x4^2\, y^2 + 21\, y\, x1^2\, y4^2\, x - 9\, x2\, y3\, x1\, y4\, x4\, y$$
$$+ 81\, x2\, y3\, x1\, y2\, y4\, x3 - 81\, x2\, y3^2\, x1\, y2\, x4 + 27\, y\, x1\, y3^2\, x^2 + 27\, x3^3\, y^3 + 18\, y\, x2\, y1^2\, x4^2$$
$$+ 81\, y\, x1\, x4^2\, y2^2 - 27\, y1\, x^2\, x2\, y4\, y2 + 27\, y1\, x\, x2^2\, y4\, y + 63\, y1\, x^2\, x1\, y3\, y4 - 54\, y\, y1\, y4\, x2^3$$
$$- 54\, y2\, x4\, y1\, y4\, x2^2 - 162\, y2\, x^2\, x3\, y\, y3 + 54\, y2\, x^2\, x3\, y\, y4 - 81\, x4\, x3\, y2\, x2\, y^2$$
$$+ 9\, x2\, y3\, y1\, x4\, x1\, y4 - 81\, x2^2\, y3\, y1\, x3\, y4 + 54\, y2^2\, x^2\, x4\, y4 + 3\, y1\, x4\, x1^2\, y^2 - 9\, x3\, y2\, x4^2\, y^2$$
$$- 18\, y\, x3\, y4^2\, x1^2 - 3\, y\, x4\, y4^2\, x^2 - 3\, y\, x4\, x1^2\, y4^2 + 54\, y2^2\, x^2\, x2\, y4 - 27\, y2^2\, x^2\, x1\, y4$$
$$+ 81\, y2^2\, x^2\, y3\, x4 - 9\, y2\, x^2\, x3\, y1^2 - 27\, y3^3\, x^3 - 18\, y2\, x\, x1^2\, y4^2 - 54\, y1\, x^2\, x2\, y\, y3$$
$$+ 18\, y1\, x^2\, x2\, y\, y4 + 54\, y1\, x\, x2\, y^2\, x3 - 162\, y2^2\, x^2\, x3\, y4 - 54\, y2\, x^2\, x2\, y4^2 - 81\, y2\, x^2\, x1\, y3^2$$
$$- 81\, x3\, y3\, x1\, y2\, x2\, y + 27\, y1\, x4^2\, x1\, y2\, y3 - 9\, y1^2\, x4\, x\, x2\, y + 9\, y2\, x^2\, x1\, y4^2 - 18\, x2\, y3\, y1^2\, x4^2$$
$$+ 54\, y2^3\, x\, x1\, x4 - 81\, y2\, x^2\, x4\, y3^2 + 81\, x3\, y2\, y1\, x2\, y3\, x - 9\, x3\, y2\, y1\, x4\, x1\, y4 - 27\, x1\, y2\, y4^2\, x2^2$$
$$+ 9\, y1\, x4\, x1\, y^2\, x2 + 9\, x3\, y2\, y1\, x4\, x1\, y - 6\, y1\, x4\, x1\, y4^2\, x + 81\, x2^2\, y3^2\, y1\, x4 + 18\, x3\, y2\, y4^2\, x^2$$
$$- 81\, x3\, y2^2\, y3\, x4\, x + 27\, y1^2\, x4\, x3\, y4\, x2 + 9\, x3\, y2\, y1\, x4\, y4\, x - 9\, x3\, y2\, y1^2\, x4^2 + 27\, x3^2\, y3\, y^2\, x4$$
$$- 81\, y\, x1^2\, y3^2\, x4 + 54\, y\, x1^2\, y3^2\, x - 27\, y\, x4\, y1^2\, x3^2 + 54\, x2\, x3\, y^3\, x4 + 81\, x2^2\, y^2\, y2\, x$$
$$+ 27\, x2^2\, y^2\, y4\, x + 54\, y1\, x^2\, x2\, y\, y2 - 18\, y1\, x\, x1\, y^2\, x3 + 54\, y2\, x4\, x2^2\, y4\, y + 81\, x2\, y1\, x3^2\, y^2$$
$$+ 27\, x1\, y3\, x4^2\, y^2 - 27\, y3\, x^2\, x4\, y2\, y4 + 54\, y3^2\, x\, y1\, x3\, x1 - 9\, y1\, x\, y3\, x4\, y4\, x1 - 27\, x1\, y2\, x2^2\, y^2$$
$$+ 18\, x3\, y2\, y1\, x4^2\, y - 81\, x3\, y2^2\, x1\, y3\, x - 9\, y3\, x\, y1\, x2\, x1\, y4 - 18\, x1\, x3\, y^3\, x4 - 54\, x1\, x2\, y^3\, x3$$
$$+ 9\, y\, x3\, y1^2\, x4^2 + 9\, y1\, x^2\, y3\, x4\, y4 + 81\, x3\, y2\, x2\, y3\, x4\, y - 27\, x1\, y4\, x3^2\, y^2 + 9\, x2\, x1^2\, y3\, y^2$$
$$- 9\, x1^2\, y^2\, y3\, x + 9\, x1^2\, y^2\, y2\, x + 3\, x1^2\, y^2\, y4\, x - 54\, x1^2\, y3\, x3\, y^2 - 21\, x1^2\, y4\, x4\, y^2 + 9\, x1^2\, y4\, x3\, y^2$$
$$+ 54\, x1^2\, y3\, x4\, y^2 - 81\, x3^2\, y2\, x2\, y4\, y - 36\, x3\, y2\, x1\, y4^2\, x - 81\, x3^2\, y2\, y1\, x2\, y - 27\, y4^2\, x\, x2\, x3\, y$$
$$+ 81\, x2\, x1\, y3\, x3\, y^2 + 45\, y1\, x\, x1\, y2\, x4\, y + 54\, y3^2\, x\, x1\, y4\, x3 - 18\, y3\, x\, x1\, y4^2\, x2$$

$$- 108\, y1\, x\, x1\, y3\, y4\, x3 + 54\, y3\, x\, y4\, x3^2\, y + 81\, y3\, x^2\, x2\, y4\, y2 - 45\, y1\, x\, x2\, y4\, x4\, y$$
$$- 54\, x2^2\, y4\, x4\, y^2 - 54\, x1\, x4^2\, y2\, y^2 + 54\, x1\, y1\, x3^2\, y^2 - 54\, x4\, y2\, x2^2\, y^2 + 21\, x1\, y1\, x4^2\, y^2$$
$$+ 18\, x1^2\, y2\, x3\, y^2 - 54\, y3^2\, x\, x3\, x4\, y + 81\, y3\, x\, y1\, y4\, x2^2 - 45\, y1\, x\, x1\, y3\, x4\, y - 9\, x2\, y1\, x4^2\, y^2$$
$$- 81\, y1\, x2^2\, y^2\, x3 - 54\, x1\, y4\, x2^2\, y^2 + 3\, y1^2\, x^2\, x1\, y + 54\, x2\, x4^2\, y2\, y^2 + 6\, y1\, x\, x1^2\, y4\, y$$
$$+ 54\, y1\, x^2\, x3\, y\, y3 + 180\, y1\, x\, x2\, y3\, x4\, y + 27\, y4^2\, x\, y1\, x3\, x2 - 6\, y1^2\, x\, x4\, x1\, y$$
$$+ 162\, y1\, x\, x3\, y3\, y4\, x2 - 54\, y3\, x^2\, x3\, y\, y4 + 18\, x4\, y\, y1\, x^2\, y2 - 45\, y1\, x\, x1\, y4\, x2\, y + 6\, x4\, y^2\, x1\, y1\, x$$
$$- 18\, x4\, y^2\, x1\, y2\, x - 81\, y1\, x\, x3\, y3\, x4\, y + 81\, y2\, x\, x3^2\, y^2 + 81\, x3^2\, y2\, x2\, y^2 + 81\, x3\, y2^2\, x^2\, y3$$
$$- 81\, x3^2\, y2^2\, x\, y + 81\, x3^2\, y2^2\, y4\, x + 18\, x3\, y2\, x1^2\, y4^2 - 81\, x3^2\, y2^2\, x1\, y4 + 9\, x2\, y3\, x1^2\, y4^2$$
$$+ 81\, y2^2\, x^2\, x1\, y3 - 81\, x2^2\, y3^2\, x4\, y + 27\, y2^2\, x^2\, y1\, x2 + 54\, x3^2\, y3\, x1\, y^2 + 81\, x3^2\, y2^2\, x1\, y$$
$$- 81\, x2^2\, y^2\, y3\, x - 54\, y1^2\, x^2\, x3\, y4 - 18\, y1^2\, x^2\, x2\, y3 + 27\, y1^2\, x^2\, x2\, y4 - 21\, y1^2\, x\, x4^2\, y$$
$$+ 54\, y1^2\, x^2\, x3\, y3 + 21\, y1^2\, x^2\, x4\, y4 + 9\, y1^2\, x\, x4^2\, y2 + 54\, y1\, x^2\, x4\, y2^2 + 27\, y1\, x^2\, x4\, y3^2$$
$$+ 81\, y1^2\, x\, y4\, x3^2 - 54\, y1^2\, x\, x3^2\, y - 3\, y1\, x\, x1^2\, y4^2 - 21\, y1\, x^2\, x1\, y4^2 - 27\, y3^2\, x^2\, x3\, y4$$
$$+ 81\, y2^2\, x^2\, x3\, y - 54\, y2\, x^3\, y3\, y4 + 27\, y3^2\, x\, x1^2\, y4 - 54\, y3^2\, x^2\, x1\, y4 - 81\, x3\, y2\, y1\, x2\, x4\, y3$$
$$+ 81\, x3^2\, y2\, y1\, y4\, x2 + 18\, y3\, x^2\, x1\, y4^2 + 54\, x4\, y^2\, x2\, y2\, x - 18\, x4\, y^2\, y1\, x\, x2 + 45\, y1\, x\, x1\, y4\, x3\, y$$
$$- 9\, y3\, x\, x1^2\, y4^2 - 81\, y3^2\, x^2\, x2\, y2 - 54\, y3^3\, x\, x1\, x4 + 81\, y3^2\, x\, x2^2\, y + 9\, y3\, x^2\, x2\, y4^2$$
$$- 54\, y3^2\, x^2\, y1\, x3 - 81\, y3^2\, x\, y1\, x2^2 + 6\, y1\, x^2\, x4\, y\, y4 - 54\, y1^2\, x3\, y3\, x4\, x - 81\, y2\, x\, x3\, y3\, y4\, x2$$
$$+ 81\, x3\, y2^2\, x1\, x4\, y3 - 54\, x4\, y2\, x\, x3\, y^2 - 18\, x4\, x3\, y^2\, y4\, x - 27\, y2\, x4\, x1^2\, y3\, y4 - 54\, y2^2\, x4\, x2\, y4\, x$$
$$- 9\, y1\, x\, x1\, y2\, x3\, y + 54\, y2^2\, x4\, x1\, y4\, x2 + 108\, y3\, x\, y1\, x3^2\, y + 9\, y1\, x\, x4\, y2\, x1\, y4$$
$$+ 81\, y3^2\, x\, x1\, y2\, x2 - 27\, y3\, x\, y1\, x4^2\, y2 - 9\, y3\, x\, y1\, x4\, y4\, x2 - 27\, x4\, x3^2\, y^3 - 81\, y3\, x\, x2^2\, y4\, y$$
$$- 108\, y3^2\, x\, x1\, x3\, y - 81\, y3\, x\, x1\, y2\, y4\, x2 + 3\, y1^2\, x^3\, y4 + 54\, y3^2\, x\, y1\, x3\, x4 - 54\, y3\, x\, y1\, y4\, x3^2$$
$$+ 54\, y3\, x\, x4\, y2\, x1\, y4 + 27\, y3^3\, x4\, x1^2 - 9\, x1^2\, y^3\, x2 - 9\, y1^2\, x^3\, y3 + 54\, x4\, x3\, y^2\, y3\, x$$
$$- 9\, x1\, y4\, x3\, x4\, y^2 - 63\, y\, x2\, x1\, y4^2\, x + 3\, y1\, x4^3\, y^2 - 3\, y1^3\, x\, x4^2 + 27\, x1\, x3^2\, y^3 + 9\, y2\, x^3\, y4^2$$
$$- 9\, y3\, x^3\, y4^2 + 81\, y2\, x^3\, y3^2 - 9\, x2\, x4^2\, y^3 + 3\, y\, x1^3\, y4^2 - 3\, y1\, x^3\, y4^2 - 27\, y2^3\, x4^2\, x1 - 54\, y2^3\, x^2\, x4$$
$$- 3\, x4\, y^3\, x1^2 + 9\, y1^2\, x^3\, y2 + 27\, y2^3\, x4^2\, x - 3\, y\, y1^2\, x4^3 + 27\, x1\, x2^2\, y^3 - 27\, x4\, y^3\, x2^2 - 27\, y1\, x^3\, y3^2$$
$$- 27\, y1\, x^3\, y2^2 + 27\, y1^2\, x3^3\, y + 9\, x1^2\, y^3\, x3 + 27\, y3^3\, x^2\, x4 + 81\, x2^2\, y^3\, x3 + 9\, x3\, y^3\, x4^2$$
$$+ 27\, y3^2\, x^3\, y4 + 3\, y1^3\, x^2\, x4 + 54\, y3^3\, x^2\, x1 - 3\, x1^3\, y4\, y^2 - 27\, y3^3\, x\, x1^2 + 3\, y4^3\, x\, x1^2 - 81\, x2\, x3^2\, y^3$$
$$- 27\, y2^3\, x^2\, x1 + 27\, y2^2\, x^3\, y4 - 27\, x3^3\, y4\, y^2 - 81\, y2^2\, x^3\, y3 - 54\, y1\, x3^3\, y^2 + 18\, x1\, y3\, x\, x4\, y^2$$
$$+ 54\, x1\, y2\, x\, x3\, y^2 - 81\, y1\, x\, x4\, y2\, x2\, y + 63\, x2\, x1\, y4\, x4\, y^2 + 27\, x2\, x1\, y4\, x3\, y^2$$
$$- 54\, y1\, x\, x1\, y2\, x4\, y3 + 27\, y1\, x2^3\, y^2 + 3\, x1\, x4^2\, y^3 - 153\, x2\, x1\, y3\, x4\, y^2 - 6\, x1\, x4\, y^2\, y4\, x$$
$$+ 9\, y1\, x\, x1\, y2\, y4\, x3 + 27\, y1\, y4^2\, x2^3 + 54\, x2^3\, y4\, y^2 - 27\, x3^3\, y4\, y1^2 + 81\, x3\, y3\, y1\, x2^2\, y$$
$$- 27\, x2^3\, y4^2\, y - 3\, y4^3\, x^2\, x1 - 81\, y\, x2\, x1\, y3^2\, x - 54\, y\, x2\, y1\, x4^2\, y2 + 27\, y\, x1^2\, y2\, x4\, y3$$

```
> nops(g);
```

```
> maxmindegree(g,plex(x,y,x1,x2,x3,x4,y1,y2,y3,y4));
```

$$[6, 6]$$

```
>
```

Thus, above shows that it is a homogeneous polynomial in these 10 indeterminates of degree 6.

Rafal Ablamowicz
rablamowicz@tntech.edu

Cookeville, June 19, 2008

Help For:

Schur-Fkt - A Maple Package for the Hopf algebra of symmetric functions

Version 1.0.2 (9 vi 2008) -- designed for Maple 11

(*) Department of Mathematics, Box 5054
Tennessee Technological University
Cookeville, TN 38501 USA
Phone: (931) 372-3664, Fax: (931) 372-6353
E-mail: rablamowicz@tntech.edu
URL: http://math.tntech.edu/rafal/SchurFkt/

(§) University of Konstanz
Fachbereich Physik, Fach M678
Universitätsstrasse 10, D-78457 Konstanz, Germany
Phone: +49 (0)7531 693491
E-mail: Bertfried.Fauser@uni-konstanz.de
URL: http://clifford.physik.uni-konstanz.de/~fauser/

Last revised: June 19, 2008 (BF & RA)

Calling Sequence:

function(args) (if the package was loaded using **with(SchurFkt);**)
SchurFkt[function](args) (long form without loading the package)

Note:

SchurFkt _needs_ the package **define** of the Clifford/Bigebra packages, since it defined tensor products of symmetric functions. It also from
time to time needs Clifford/Bigebra, so we advice strongly to install the library file which contains all of these packages!

Description:

- SchurFkt provides essential operations on the Hopf algebra of (commutative) symmetric functions in formally infinite many variables. It provides several important bases which allow to implement products and coproducts by means of combinatorics of Young diagrams (Ferers diagrams, essentially a graphical display of partitions) and Young tableaux.
- Schur polynomials can be used to describe irreducible representations of general linear groups. The product of these polynomials resembles the Glebsch-Gordan decomposition of a tensor product of two irreducible representations (irreps) into irreducibles again. The decomposition of irreps gives a coproduct of Schur functions. Schur functions encode a huge number of

combinatorial identities. Schur functions have a second product, called 'inner product'. This product has to do with the product of irreps of the symmetric group. Since it does not add the weight of the tableaux but combines two tableaux of the same weight into other such tableaux.

- Power sum symmetric functions play a role in enumerative combinatorics (Polya counting theory, cycle indicators), as in algebras K-theory (Adams operations). In our setting, power sum function form the primitive elements of the outer Hopf algebra of symmetric functions.
- Monomial symmetric functions play an important role in the approach to symmetric functions proposed by Rota-Stein, 94. Currently SchurFkt does not fully implement these algorithms.
- The **general goal** of SchurFkt is to provide a proof of concept for some new developments in symmetric function and invariant theory. Since Maple (TM) is considerably slower as, e.g. SCHUR by Brian G Wybourne, serious calculations may need special purpose software. However, being able to code new algorithms provides new insights into the theory and last but not least proves the authors understanding of the subject.

Load SchurFkt in the following way:

```
> restart:with(SchurFkt);
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
```

[*AlexComp*, *CharHook*, *CompNM*, *FLAT*, *Frob2part*, *GesselThetaP*, *GesselThetaS*, *KostkaPC*, *KostkaTable*, *LaplaceM*, *LaplaceM_mon*, *LaplaceTable*, *MLIN*, *MurNak*, *MurNak2*, *PartNM*, *Scalar*, *ScalarHM*, *ScalarMH*, *ScalarP*, *antipE*, *antipH*, *antipM*, *antipMC*, *antipP*, *antipS*, *branch*, *cinner*, *cinnerP*, *cmp2part*, *cmp2prtMult*, *concatM*, *conjpart*, *counitInnerP*, *counitInnerS*, *couter*, *couterE*, *couterH*, *couterM*, *couterON*, *couterP*, *cplethP*, *cplethS*, *dimSN*, *e_to_h*, *e_to_s*, *evalJacobiTrudiMatrix*, *getSfktSeries*, *grAlexComp*, *h_to_m*, *h_to_s*, *inner*, *innerH*, *innerP*, *isLattice*, *m_to_p*, *maxlengthSymFkt*, *mset2part*, *outer*, *outerE*, *outerH*, *outerM*, *outerON*, *outerP*, *outerS*, *p_to_m*, *p_to_s*, *part2Frob*, *part2mset*, *plethP*, *plethS*, *plethSnm*, *s_to_h*, *s_to_hJT*, *s_to_hmat*, *s_to_p*, *s_to_x*, *skew*, *sq_coeff*, *truncLEN*, *truncWT*, *x_to_s*, *zee*]

```
>
```

Alphabetic (but Overview as the first topic) listing of available procedures in 'SchurFkt':

- Overview -- The main HELP FILE for SchurFkt (this file)
- AlexComp -- compares two compositions/partitions w.r.t. anti-lexicographic ordering
- antipS -- the antipode acting on symmetric functions in the Schur polynomial basis
- branch -- branch transforms a symmetric function representing a group character into

S-functions of another group via branching

- CharHook -- evaluation of a cycle indicator on a Hook Schur function
- cinner -- inner coproduct of symmetric function the Schur function basis
- cmp2prtMult -- computes the length of the orbit of compostions which project under sorting to the same partition
- CompNM -- produces a list of compositions of N into M parts
- concatM -- divided powers concatenation product (needed for Rota-Stein cliffordization)
- conjpart -- computes a conjugate partition
- couter -- the outer coproduct in the Schur function basis
- couterE -- the outer coproduct in the elementary symmetric function basis
- couterH -- the outer coproduct in the complete symmetric function basis
- couterM -- the outer coproduct in the monomial symmetric function basis
- couterON - outer coproduct for the O(n) groups in the stable limit N--> infinity
- couterP -- the outer coproduct in the power sum basis
- cplethP -- plethysm coproduct in the power sums basis
- cplethS -- plethysm coproduct in the Schur function basis
- dimSN -- computes the dimension of an sfkt polynom seen as S_n character
- FLAT -- flattens the function T() used by SchurFkt[MLIN] (hence T() is made associative this way), mainly for internal use!
- Frob2part -- converts a partition in Frobenius notation into a standard list notation of partitions
- GesselThetaP -- computes the Gessel map Theta for power sum symmetric functions
- GesselThetaS -- computes the Gessel map Theta for Schur functions
- getSfktSeries -- produces a Schur function series (or a list of its coefficients)
- grAlexComp -- compares two compositions/partitions w.r.t. graded anti-lexicographic ordering
- h_to_s -- convert a homogenous symmetric function into a Schur function
- inner -- the inner product in Schur function basis
- innerP -- inner product in the power sum basis
- isLattice -- checks if a Young tableau is a lattice permutation
- KostkaPC -- computes the Kostka coefficient between a composition and a partition

- KostkaTable -- computes the Kostka matrix in any dimension
- LaplaceM -- the Rota-Stein Laplace pairing internally used for 'cliffordization' of the concatenation product in the monomial basis into the outer product of monomial symmetric functions (internal use mostly)
- LaplaceM_mon -- Laplace pairing on monomials
- LaplaceTable -- tabulates the LaplaceM pairing of m-function monomials (exhibits some grading properties)
- m_to_p -- basis change from monomial to power sum symmetric functions
- MLIN -- makes the function T() multilinear over the integers, mainly for internal use!
- mset2part -- translates a partition in multiset notation into a partition in standard format
- MurNak -- the Murnaghan Nakayama character of the symmetric group, uses internally a rim-hook representation of partitions to optimize the algorithm
- MurNak2 -- MurNak2 uses a recursive algorithm and is much slower than MurNak (for comparison and educational/demonstration purpose only)
- outer -- outer product of two Schur functions (also known as SchurFkt[outerS])
- outerE -- outer product in the elementary symmetric function basis (E-basis)
- outerH -- outer product in the complete symmetric function basis (H-basis)
- outerM -- outer product of monomial symmetric functions (a la Rota-Stein)
- outerON -- outer product for orthogonal (symplectic) characters
- outerP -- the outer product of symmetric functions in the power sum basis
- outerS -- outer product of two Schur functions (same function as SchurFkt[outer] in this version of SchurFkt)
- p_to_m -- basis change from power sum to monomial symmetric functions
- p_to_s -- basis change from power sum symmetric functions to Schur functions
- part2Frob -- translates a standard partition (shape) into Frobenius notation
- part2mset -- translates a partition in standard representation into an multiset (exponential) representation
- PartNM -- returns a list of partitions of N with parts of size at most M
- plethP -- plethysm in the power sum basis
- plethS -- computes the plethysm of two Schur function polynoms
- plethSnm -- computes the plethysm of two sfunctions of the form s[n] (one part complete symmetric functions)

- s_to_h -- basis change from Schur function basis to complete symmetric function basis
- s_to_p -- basis change from Schur functions to power symmetric functions
- s_to_x -- translation of a Schur function into a polynom in variables x_i
- Scalar -- the Schur-Hall [Redfield cup] scalar product in the Schur function basis
- ScalarHM -- the Schur-Hall [Redfield cup] scalar product in the monomial-complete symmetric function basis
- ScalarMH -- the Schur-Hall [Redfield cup] scalar product in the complete-monomial symmetric function basis
- ScalarP -- the Schur-Hall [Redfield cup] scalar product in the power sum basis
- skew -- (outer) skew of two Schur functions
- sq_coeff -- returns the square of the coefficients of a Schur function
- truncWT -- truncates an Schur function expression by its weight
- x_to_s - translates a polynomial in the variables xi into a Schur function expression
- zee - the symmetry factor z associated to power symmetric functions (or cycles of the symmetric group)

New Types in 'SchurFkt':

We use 'fkt' derived from German `Funktion' (function) as in SchurFkt also for types. Typing is necessary to allow Maple(R) to decide about linearity of certain morphisms (procedures). Symmetric functions come with a number of standard bases, which have combinatorially different meanings and allow different algorithms to be used to perform calculations. The SchurFkt package knows currently the following types:

- Schur functions. This is the most important basis. Schur functions (sfunction for short) encode characters of irreducible representations of the symmetric and general linear groups. Schur functions (and all other bases) are indexed by integer partitions, written as index to the kernel-symbol (here `s'). We need to distinguish:

  -- `type/sfktmonom` -- Schur function monom, a basis element like s[3,2,2,1] with no prefactor.
  -- `type/sfktterm` -- A Schur function including a coefficient from the ground ring (usually integers) of type cliscalar like 4*s[4,1,1,1]
  -- `type/sfktpolynom` -- A linear combination of sfktterms like 2*s[2]+5*s[1,1].

  The types used are inclusive, so a check if an expression <foo> has type sfktpolynomial yields

true, if foo is an sfunction of type sfktmonom, skftterm, or sfktpolynom! The check for an sfktterm yields true, if <foo> is a term of a form coefficient times a Schur function monom or if it is a Schur function monom, while the check for Schur function monom yields true only for expressions like s[3,3] (irreps, basis monoms of the ring of symmetric functions).

Note: Schur functions are self dual wrt to the Schur-Hall inner product `ScalarS'. They form an orthonormal basis.
Note: Schur functions are not multiplicative (see below).

- Power sum symmetric functions have their origin in the invariant theory of the symmetric group. Considering polynomials in the indeterminates {x_i}_{i=1}^n it is obvious that the polynomials p_k(x) = \sum_{i=0}^n x_i^k are _invariant_ under the action of the symmetric group acting on n-letters (indeterminates). Furthermore, these polynomials are a complete set of invariants. Last but not least, the power sum symmetric functions are orthogonal but not normalized w.r.t. the Schur-Hall inner product `ScalarP'.

  We distinguish in the same fashion as for the S-functions, basis monoms, terms and polynomials in the power sum symmetric functions p_k(x):

  -- `type/pfktmonom` -- A basis monom like p[2,2,1].
  -- `type/pfktterm` -- A basis monom with an optional ring coefficient 5*p[2] of type cliscalar, but p[3]=1*p[1] is a term either.
  -- `type/pfktpolynom` -- A linear combination of pfktterms or a pfktterm or a pfktmonom.

  Note: Power sum symmetric functions are multiplicative. That is, the outer product of power sum symmetric functions is the (unordered) concatenation of power sum symmetric functions:

  p[3,1].p[2,1] = p[3,2,1,1]

  The outer product is particularly simple to compute for multiplicative bases!

- Complete symmetric functions are another special class of symmetric functions. Complete symmetric functions are the dual basis w.r.t. the Schur-Hall inner product of the monomial symmetric functions (see below). They are used to extract counting coefficients in generating functions in the Polya-Redfield theory of enumeration. The classical kernel symbol is `h`, we distinguish:

  -- `type/hfktmonom` -- A basis monom like h[2,2,1].
  -- `type/hfktterm` -- A basis monom with an optional ring coefficient 5*h[2] of type cliscalar, but h[3]=1*h[1] is a term either.

-- `type/hfktpolynom` -- A linear combination of hfktterms or a hfktterm or a hfktmonom.

- Monomial symmetric functions are _the_ classical symmetric functions. They are obtained by symmetrizing monomials x^\alpha=x_1^\alpha1...x_k^\alpha_k using the symmetric group S_k acting on the indices of the indeterminates, where only distinct terms are kept (no multiplicities). One has m_\lambda = \sum_{\sigma\in S_k distinct} x_{\sigma(1)}^\lambda_1...x_{\sigma(k)}^\lambda_k. Hence monomial symmetric functions appear by averaging over the symmetric group action on a single monomial. It is clear that a partition \lambda indexes such averages, while individual monomials are indexed by compositions (ordered integer decompositions). The monomial symmetric function basis is _not_ multiplicative. We distinguish:

  -- `type/mfktmonom` -- A basis monom like m[2,2,1].
  -- `type/mfktterm` -- A basis monom with an optional ring coefficient 5*m[2] of type cliscalar , but m[3]=1*m[1] is a term either.
  -- `type/mfktpolynom` -- A linear combination of mfktterms or a mfktterm or a mfktmonom.

  Note: The SchurFkt package has a second product employed on the basis of monomial symmetric functions. This is the `concatM' product which establishes the _multiplicative_ (unordered) concatenation product. This product is not usually considered in the theory of symmetric functions and is _not_ the outer product. However, the process of cliffordization described by Rota-Stein allows one to introduce the outer product in the monomial basis `outerM` as a Hopf algebra deformation of the (unordered) concatenation product `concatM'. (This is in analogy to how a Clifford algebra appears to be a deformation of the Grassmann algebra).

- Elementary symmetric functions are cousins of complete symmetric functions. The are obtained by conjugating the partitions indexing rows and columns of one part partitions and one row partitions. Elementary symmetric functions (while being symmetric functions) encode antisymmetric aspects of invariants. Rows in a Young diagram (tableau) are antisymmetrized. Elementary symmetric functions form a multiplicative basis, and we distinguish:

  -- `type/efktmonom` -- A basis monom like e[2,2,1].
  -- `type/efktterm` -- A basis monom with an optional ring coefficient 5*e[2] of type cliscalar, but e[3]=1*e[1] is a term either.
  -- `type/efktpolynom` -- A linear combination of efktterms or a efktterm or a efktmonom.

- The dual basis of the elementary symmetric functions is called forgotten functions (Doubilet

functions), since they played a minor (invisible) role in the combinatorial and enumerative approach to invariants and symmetric functions. The forgotten functions share many properties with the monomial symmetric functions, the basis is _not_ multiplicative. We distinguish:

-- `type/ffkmonom` -- A basis monom like f[2,2,1].
-- `type/ffktterm` -- A basis monom with an optional ring coefficient 5*f[2] of type cliscalar, but f[3]=1*f[1] is a term either.
-- `type/ffktpolynom` -- A linear combination of ffktterms or a ffktterm or a ffktmonom.

Note: Not much about forgotten functions is yet implemented in SchurFkt, ... nomen est omen.

- The general type symfkt[monom|term|polynom] was created to check if a general expression <foo> contains any of the above specified bases {s,p,h,m,e,}. This may allow to form expressions with mixed basis types like e[3,2,1]+h[3,3] and alike. Some internal functions of SchurFkt do not need to know what kind of basis they process unless it is, say, a multiplicative one. In order not to trigger a "wrong type" error, type checking is done only against symfkt[monom|term|polynom].

  Note: The usage of symfkt-types is _dangerous_ and should be done only in internally used functions! Beware!

See Also: define

NOTE: SchurFkt needs the patched define which ships with the Clifford/Bigebra packages!!

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[zee] - the symmetry factor z associated to power symmetric functions (or cycles of the symmetric group)

Calling Sequence:

int := zee(prt)

Parameters:

- prt : a partition (in standard notation list integer)

Output:

- int : nonnegative integer

WARNING:

--none--

Description:

- The z-factor (zee is SF package of Stembridge) provides a combinatorial symmetry factor.
- We have:

  zee(\mu) = \prod_i^length(\mu) i^r_i r_i!

  where the partition \mu is given in exponential (multiset) notation.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
```

```
> zee([1,1,1]);
  zee([2,1]);
  zee([3]);
```

$$6$$

$$2$$

$$3$$

```
> lst:=[seq([1$i],i=1..6)];
  map(x->zee(x),lst);
  lst:=[seq([i],i=1..6)];
  map(x->zee(x),lst);
```

$$lst := [[1], [1, 1], [1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1]]$$

$$[1, 2, 6, 24, 120, 720]$$

$$lst := [[1], [2], [3], [4], [5], [6]]$$

$$[1, 2, 3, 4, 5, 6]$$

```
> prt:=PartNM(5,5);
  map(x->zee(x),prt);
```

$$prt := [[5], [4, 1], [3, 2], [3, 1, 1], [2, 2, 1], [2, 1, 1, 1], [1, 1, 1, 1, 1]]$$

$$[5, 4, 6, 6, 8, 12, 120]$$

Critical cases:

```
> zee([]);
```

$$1$$

```
>
```

Algorithm used:

Implementation of the combinatorial formula.

See Also: SchurFkt[Overview]

Last modified: June 19, 2008, 2007/BF/RA

Function: SchurFkt[AlexComp] - compares two compositions/partitions w.r.t. anti-lexicographic ordering

Calling Sequence:

b := AlexComp(c1,c2)

Parameters:

- c1,c2 : compositions (or partitions)

Output:

- b : boolean value (true / false)

WARNING:

Note that Maple uses in its combinatorial packages lexicographical order of partitions and compositions.

Description:

- AlexComp allows to order compositions and partitions in anti-lexicographic order. Anti-lexicographic order is the standard order of Macdonald and other writers on symmetric functions. Note that Maple uses in the combinatorics packages lexicographic order!

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 says 'Good bye...'
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

Check AlexComp for special values:

```
> AlexComp([3,2],[2,1]);
  AlexComp([2,2,1,0],[3,0,2,0]);
```

$$true$$

$$false$$

Using AlexComp to sort a list of partitions:

```
> prt:=[ [3,2,1,0,0,0],[6,0,0,0,0,0],[5,2,0,0,0,0],[1,1,1,1,1,1]];
  sort(prt,AlexComp);
```

$$prt := [[3, 2, 1, 0, 0, 0], [6, 0, 0, 0, 0, 0], [5, 2, 0, 0, 0, 0], [1, 1, 1, 1, 1, 1]]$$

$$[[6, 0, 0, 0, 0, 0], [5, 2, 0, 0, 0, 0], [3, 2, 1, 0, 0, 0], [1, 1, 1, 1, 1, 1]]$$

Maples combinat produces lex ordered lists

```
> combinat[partition](3);
```

```
sort(%,AlexComp);
```

$$[[1, 1, 1], [1, 2], [3]]$$

$$[[3], [1, 2], [1, 1, 1]]$$

Note that CompNM and PartNM functions produce lists of compositions and partitions in anti-lexcicographical order

```
> prt:=PartNM(3,3);
  cmp:=CompNM(2,5);
```

$$prt := [[3], [2, 1], [1, 1, 1]]$$

$$cmp := [[2, 0, 0, 0, 0], [1, 1, 0, 0, 0], [1, 0, 1, 0, 0], [1, 0, 0, 1, 0], [1, 0, 0, 0, 1], [0, 2, 0, 0, 0], [0, 1, 1, 0, 0], [0, 1, 0, 1, 0], [0, 1, 0, 0, 1], [0, 0, 2, 0, 0], [0, 0, 1, 1, 0], [0, 0, 1, 0, 1], [0, 0, 0, 2, 0], [0, 0, 0, 1, 1], [0, 0, 0, 0, 2]]$$

Sorting in lexicographic order my be achieved by using 'not'

```
> prt;
  sort(prt,not AlexComp);
  cmp;
  sort(cmp,not AlexComp);
```

$$[[3], [2, 1], [1, 1, 1]]$$

$$[[1, 1, 1], [2, 1], [3]]$$

$$[[2, 0, 0, 0, 0], [1, 1, 0, 0, 0], [1, 0, 1, 0, 0], [1, 0, 0, 1, 0], [1, 0, 0, 0, 1], [0, 2, 0, 0, 0], [0, 1, 1, 0, 0], [0, 1, 0, 1, 0], [0, 1, 0, 0, 1], [0, 0, 2, 0, 0], [0, 0, 1, 1, 0], [0, 0, 1, 0, 1], [0, 0, 0, 2, 0], [0, 0, 0, 1, 1], [0, 0, 0, 0, 2]]$$

$$[[0, 0, 0, 0, 2], [0, 0, 0, 1, 1], [0, 0, 0, 2, 0], [0, 0, 1, 0, 1], [0, 0, 1, 1, 0], [0, 0, 2, 0, 0], [0, 1, 0, 0, 1], [0, 1, 0, 1, 0], [0, 1, 1, 0, 0], [0, 2, 0, 0, 0], [1, 0, 0, 0, 1], [1, 0, 0, 1, 0], [1, 0, 1, 0, 0], [1, 1, 0, 0, 0], [2, 0, 0, 0, 0]]$$

AlexComp can handle lists with different length, if necessary trailing zeros are appended internally:

```
> AlexComp([2],[3,2,1,2]);
  AlexComp([4],[1,1,1,1]);
```

$$false$$

$$true$$

```
>
```

Algorithm

Not available (obvious).

See Also: SchurFkt[Overview], SchurFkt[PartNM], SchurFkt[CompNM]

Last modified: June 19, 2008/BF/RA.

Function: SchurFkt[antipS] - the antipode acting on symmetric functions in the Schur polynomial basis

Calling Sequence:

s2 := antipS(s1)

Parameters:

- s1 : S-function polynom type/sfktpolynom

Output:

- s2 : S-function polynoms type/sfktpolynom

WARNING:

The antipode is currently implemented for Schur polynomials, do not apply this function to other bases, no type checking yet. Tensor products may be imported from the Bigebra[define] facility. Look there up how to specify the ground field of the tensor or how to define own tensors.

Description:

- The antipode of the outer Hopf algebra of symmetric functions is an anti algebra homomorphisms, which represents a generalized inverse. In a Hopf algebra stemming from a group, the antipode is the inverse S(g)=g^(-1), which illustrated the anti homomorphism rule S(gh) = S(h)S(g).
- In the Schur polynomial basis, the antipode is given by

  S(s[lambda]) = (-1)^length(lambda) s[lambda^`]

  where length(lambda) is the length of the partition lambda (number of parts) and lambda^` is the conjugated partition (mirrored partition).

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

The antipodes takes e.g. these special values:

```
> antipS(s[0]);
  antipS(s[1]);
  antipS(s[2]);
  antipS(s[1,1]);
  antipS(s[3]);
  antipS(s[2,1]);
```

```
antipS(s[1,1,1]);
```

$$s_0$$
$$-s_1$$
$$s_{1,1}$$
$$s_2$$
$$-s_{1,1,1}$$
$$-s_{2,1}$$
$$-s_3$$

```
> prt:=map(x->s[op(x)],PartNM(5,5));
  map(antipS,prt);
```

$$prt := [s_5, s_{4,1}, s_{3,2}, s_{3,1,1}, s_{2,2,1}, s_{2,1,1,1}, s_{1,1,1,1,1}]$$

$$[-s_{1,1,1,1,1}, -s_{2,1,1,1}, -s_{2,2,1}, -s_{3,1,1}, -s_{3,2}, -s_{4,1}, -s_5]$$

We check the axiom for the antipode:

f_(1) S(f_(2)) = \eta\epsilon(f) = S(f_(1)) f_(2)

where we have used the Sweedler notation for the coproduct \Delta(f) = f_(1) &t f_(2)

```
> out:=couter(s[2]);
```

$$out := (s_0 \,\&t\, s_2) + (s_2 \,\&t\, s_0) + (s_1 \,\&t\, s_1)$$

'f1' / 'f2' are a helper functions mapping the antipode to the first/second tensor slot:

```
> f1:=(x)->`&t`(antipS(op(x)[1]),op(x)[2]):
  f2:=(x)->`&t`(op(x)[1],antipS(op(x)[2])):
```

Check this functionality:

```
> f1(&t(s[2],s[1,1]));
  f2(&t(s[2],s[1,1]));
```

$$s_{1,1} \,\&t\, s_{1,1}$$
$$s_2 \,\&t\, s_2$$

Now map the antipode to the outer coproduct and map back the outer product for the tensor should give zero for all but the Schur function s[0]

```
> `+`(op(map(f1,[[op(couter(s[0]))]])));
  eval(subs(`&t`=outer,%));  # special case S(s[0]).s[0]=s[0]
  #
  `+`(op(map(f1,[op(couter(s[2,1]))])));
  eval(subs(`&t`=outer,%));  # general case S(s[\mu(1)]).s[mu(2)]
  = 0
```

$$s_0 \,\&t\, s_0$$
$$s_0$$
$$(s_0 \,\&t\, s_{2,1}) - (s_{2,1} \,\&t\, s_0) + (s_{1,1} \,\&t\, s_1) + (s_2 \,\&t\, s_1) - (s_1 \,\&t\, s_2) - (s_1 \,\&t\, s_{1,1})$$

$$0$$

A few more examples:

```
> f1(&t(op(couter(s[0]))));
> ` => `,eval(subs(`&t`=outer,%)),` special case for s[0]=1`;
  ## special case for s[0] !
  `+`(op(map(f1,[op(couter(s[1,1]))]))));
  ` => `,eval(subs(`&t`=outer,%));
  `+`(op(map(f2,[op(couter(s[4,1]))]))));
  ` => `,eval(subs(`&t`=outer,%));
  `+`(op(map(f2,[op(couter(s[1,1,1]))]))));
  ` => `,eval(subs(`&t`=outer,%));
```

$$s_0 \,\&t\, s_0$$

$$=> ,\ s_0,\ \textit{special case for s[0]=1}$$

$$(s_0 \,\&t\, s_{1,1}) + (s_2 \,\&t\, s_0) - (s_1 \,\&t\, s_1)$$

$$=> ,\ 0$$

$$-(s_0 \,\&t\, s_{2,1,1,1}) + (s_{4,1} \,\&t\, s_0) - (s_4 \,\&t\, s_1) - (s_{3,1} \,\&t\, s_1) + (s_3 \,\&t\, s_{1,1}) + (s_{2,1} \,\&t\, s_{1,1})$$
$$+ (s_3 \,\&t\, s_2) - (s_2 \,\&t\, s_{1,1,1}) - (s_{1,1} \,\&t\, s_{1,1,1}) - (s_2 \,\&t\, s_{2,1}) + (s_1 \,\&t\, s_{1,1,1,1}) + (s_1 \,\&t\, s_{2,1,1})$$

$$=> ,\ 0$$

$$-(s_0 \,\&t\, s_3) + (s_{1,1,1} \,\&t\, s_0) - (s_{1,1} \,\&t\, s_1) + (s_1 \,\&t\, s_2)$$

$$=> ,\ 0$$

```
>
```

Algorithm

The antipode in the function basis has a closed form

(1) antipS(s_\lambda) = (-1)^(|\lambda|) s_\lambda'

where |\lambda| is the weight (sum of boxes in tableaux) of the partition and \lambda' is the conjugated partition (transposed tableaux). The antipode is directly implemented via this formula.

See Also: SchurFkt[Overview], SchurFkt[couter], Bigebra[define]

Last modified: June 19, 2008/BF/RA.

Function: SchurFkt[branch] - branch transforms a symmetric function representing a group character into S-functions of another group via branching

Calling Sequence:

gr1 := branch(gr0,SeriesName)

Parameters:

- gr0 , SeriesName : a symmetric function (Schur function), a name of a recognized recognized series of Schur functions.

Output:

- gr1 : a branched group character (Schur function polynom)

WARNING:

SchurFkt does currently not provide any support for assigning a group to symmetric functions. It is the *users responsibility* to keep track which Schur functions stand for which type of characters.

Description:

- Branching is the process of reduction of characters or subduction of characters. A prominent example is the branching of GL(n) characters into O(n) or Sp(n) characters. In Littlewood notation s_\lambda in GL equals {\lambda}, in O equals [\lambda], and in Sp equals <\lambda>. Branch computes the transition from one picture into another.
- Branch can be used to define in a crude way the Glabsch-Gordan decomposition of a tensor product in subgroups, see BF,P. Jarvis, R.C. King and B.G. Wybourne, "New branching rules induced by plethysm", J. Phys. A: Math. Gen. 39, 2006:2611-2655, arXivmathph/0505037, where this problem is solved in a general setting.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
```

Example: GL(n) -> GL(n-1)

The input Schur functions represent GL(n) characters. A branching wrt to the M-series (complete one part symmetric functions) results in a decomposition of the GL(n) character in terms of GL(n-1) characters.

In a tensor notation this would mean to fix a particular vector in GL(n) and consider the centralizer of this vector.

```
> branch(s[2],M);
```

```
branch(s[3,1],M);
```

$$s_0 + s_1 + s_2$$

$$s_{3,1} + s_3 + s_{2,1} + s_2 + s_{1,1} + s_1$$

In terms of tableau, the branching by M is equivalent to a skew with a any row on k boxes which fit into the shape.

Note that the inverse series to M (in a convolutive sense obtained from the Hopf algebra induced convolution) is given by the L series (one part elementary symmetric functions including a sign (-1)^n). Hence the branching wrt to the L series is inverse and induced a GL(n-1) character into the GL(n) character.

```
> branch(s[2],L);
  branch(s[3,1],L);
```

$$s_2 - s_1$$

$$s_{3,1} - s_3 - s_{2,1} + s_2$$

Note that we pick up signs! Schur functions with a negative sign are *virtual characters* and have formally a *negative dimension*!

We show not that reduction followed by induction is the identity and vice versa.

```
> branch(s[5,2],M);   # GL(n)->GL(n-1)
  branch(%,L);        # GL(n-1)->GL(n)
  #
  branch(s[4,4,1],L);# GL(n-1)->GL(n)
  branch(%,M);        # GL(n)->GL(n-1)
```

$$s_{5,2} + s_{5,1} + s_{4,2} + s_5 + s_{4,1} + s_{3,2} + s_4 + s_{3,1} + s_{2,2} + s_3 + s_{2,1} + s_2$$

$$s_{5,2}$$

$$s_{4,4,1} - s_{4,4} - s_{4,3,1} + s_{4,3} + s_{3,3,1} - s_{3,3}$$

$$s_{4,4,1}$$

Using the A,B series we can branch to symplectic and orthogonal characters. The C,D series give the branching to the orthogonal characters.

```
> branch(s[2],D);   # GL(n)->O(n)
  branch(s[2],C);   # O(n)->GL(n)
  #
  branch(branch(s[3,2,1],C),D);
```

$$s_0 + s_2$$

$$s_2 - s_0$$

$$s_{3,2,1}$$

```
> branch(s[1,1],B);   # GL(n)->Sp(n)
  branch(s[1,1],A);   # Sp(n)->GL(n)
  #
```

```
branch(branch(s[4,4,1,1],B),A);
```

$$s_0 + s_{1,1}$$

$$s_{1,1} - s_0$$

$$s_{4,4,1,1}$$

In the language of tensors, these reductions have a precise meaning.

O(3): Consider an stress energy tensor which is a symmetric second rank tensor T_\mu\nu. If a system has an O(3) symmetry, this tensor decomposes into irreducible components:

{2} -> [2]+[0]

T_(\mu\nu) -> T^precisenu) + trace(T_())g_(\mu\nu)

The symmetric tensor T_(\mu\nu) is an irreducible GL character. T^0 is an trace free tensor and an irreducible O character, trace(T) is a trivial (scalar) O character.

```
> branch(s[2],D);
```

$$s_0 + s_2$$

In general relativity one needs the decomposition of the Weyl tensor/Riemann tensor into its irreducible symmetry components:

R^GL_ijkl -> R_ijkl + R_ij g_kl + perm + R g_ijg_kl + perm

where R_ijkl is the Riemann tensor, R_ij is the Ricci tensor, R is the Ricci scalar.

```
> branch(s[2,2],D);
```

$$s_{2,2} + s_2 + s_0$$

Glebsch-Gordan decomposition:

We can defined the outer product for symplectic and orthogonal character using this branching:

```
> outerOgrp:=(x,y)->branch(outer(branch(x,C),branch(y,C)),D):
  outerSpgrp:=(x,y)->branch(outer(branch(x,A),branch(y,A)),B):
```

Note: These type of products can be recognized as *cliffordizations* and are classified via Hopf algebra cohomology, see BF. P. Jarvis, "A Hopf laboratory for symmetric functions", J. Phys. A: Math. Gen 37(5), 2004:1633-1663.

We compute the product of two fundamental representations. Note that the bilinear forms defining O or Sp can be used to form traces and to reduce the weight of the character. In the O case we contract a s[2] in the Sp case an s[1,1]:

```
> outerOgrp(s[1],s[1]);
  outerSpgrp(s[1],s[1]);
```

$$s_0 + s_2 + s_{1,1}$$

$$s_0 + s_2 + s_{1,1}$$

Note that these are the same products. Indeed by accident of the HA deformation we find for Sp and O the same Glebsch-Gordan decomposition.

```
> outerOgrp(s[3],s[2,1]);
  outerSpgrp(s[3],s[2,1]);
  #
  outerOgrp(s[2,2],s[2,1]);
  outerSpgrp(s[2,2],s[2,1]);
  #
  outerOgrp(s[3,2],s[1,1]);
  outerSpgrp(s[3,2],s[1,1]);
```

$$s_2 + s_{1,1} + 2\, s_{3,1} + s_4 + s_{2,2} + s_{2,1,1} + s_{5,1} + s_{4,2} + s_{4,1,1} + s_{3,2,1}$$

$$s_2 + s_{1,1} + 2\, s_{3,1} + s_4 + s_{2,2} + s_{2,1,1} + s_{5,1} + s_{4,2} + s_{4,1,1} + s_{3,2,1}$$

$$s_1 + s_3 + 2\, s_{2,1} + s_{1,1,1} + 2\, s_{3,1,1} + s_{4,3} + s_{4,2,1} + s_{3,3,1} + s_{3,2,2} + s_{3,2,1,1} + s_{2,2,2,1} + s_{4,1} + 2\, s_{3,2} + 2\, s_{2,2,1} + s_{2,1,1,1}$$

$$s_1 + s_3 + 2\, s_{2,1} + s_{1,1,1} + 2\, s_{3,1,1} + s_{4,3} + s_{4,2,1} + s_{3,3,1} + s_{3,2,2} + s_{3,2,1,1} + s_{2,2,2,1} + s_{4,1} + 2\, s_{3,2} + 2\, s_{2,2,1} + s_{2,1,1,1}$$

$$s_{2,1} + s_{3,1,1} + s_{4,3} + s_{4,2,1} + s_{3,3,1} + s_{3,2,1,1} + s_{4,1} + 2\, s_{3,2} + s_{2,2,1}$$

$$s_{2,1} + s_{3,1,1} + s_{4,3} + s_{4,2,1} + s_{3,3,1} + s_{3,2,1,1} + s_{4,1} + 2\, s_{3,2} + s_{2,2,1}$$

```
>
```

Algorithm used:

Branch uses two ingredients. The function getSfktSeries and the outer coproduct of Schur functions. Consider

(1) branch(sfkt , X) = \sum_(x\in X) < x | sfkt(1)> sfkt(2)

we firstly decompose the character sfkt into two parts (tensor) using the outer coproduct. the we act with the linear form attached to the series X obtained by acting with the elements x in the series via the Schur-Hall scalar product.

branch does exactly this.

See Also: SchurFkt[getSfktSeries]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[CharHook] - evaluation of a cycle indicator on a Hook Schur function

Calling Sequence:

t1 := CharHook(s1,p1)

Parameters:

- s1 : S-function of hook shape (an element of `type/spolynom`) of any weight
  p1 : P-function of a **one-part** partition

Output:

- t1 : an integer (+1, 0, or -1) indicating if the hook shaped Schur function has the same amount of boxes as the power sum function and the sign is given by minus one to the power of the height of the hook (number of rows minus 1)

WARNING:

No type checking yet, especially in the second argument! If the second argument has more than one part the function does not produce an error!

Description:

- For the computation of the Murnaghan-Nakayama rule in a recursive way (SchurFkt[MurNak2]) one needs to check if a certain boundary strip decomposition of the shape {lambda} in terms of stripes {mu} can be obtained. The sign is given by the (-1)^height of the boundary strip (number of rows in the Ferrers diagram occupied by the boundary strip -1).
- This functions plays a role int proof of the Murnaghan-Nakayama rule. For actual computations, the recursive way is too slow, and one uses a different more efficient method (SchurFkt[MurNak]).
- Mainly internal use!

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

A hook shaped Schur function has a single row and a single column, both possibly degenerated.

```
> CharHook(s[4,1],p[5]);       # height is 1 -> sign is -1
  CharHook(s[4],p[4]);         # height is 0 -> sign in +1
  CharHook(s[1,1,1,1],p[4]);   # height is 3 -> sign is -1
```

$$-1$$

$$1$$

$$-1$$

Some cases where the Schur function is not of hook shape, hence zero is returned

```
> CharHook(s[2,2],p[4]);
  CharHook(s[3,2,1],p[6]);
  CharHook(s[2,2,1],p[5]);
```

$$0$$

$$0$$

$$0$$

Note that if the weight is unequal, no boundary strip can be found and the result is zero

```
> CharHook(s[4,1],p[2]);
  CharHook(s[2,1,1,1],p[8]);
```

$$0$$

$$0$$

It is **illegal** to put multipart power sum functions into CharHook as second argument, where a one-part partition power sums (primitive elements of the outer Hopf algebra) is expected. The function does not complain but returns the value as if the first part of the power sum function was given as input!

```
> CharHook(s[4,1],p[5,2]);  ## WRONG RESULT!
  CharHook(s[4,1],p[5]);
```

$$-1$$

$$-1$$

```
>
```

ALgorithm

An obvious check on the shape of the first argument and a comparison of weight.

See Also: SchurFkt[MurNak2], SchurFkt[MurNak]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[cinner] - inner coproduct of symmetric function the Schur function basis

Calling Sequence:

t1 := cinner(sfkt1sfkt2)

Parameters:

- sfkt1,sfkt2 : Schur function polynomials

Output:

- t1 : a tensor of Schur function monomials

WARNING:

--no typechecking--

Description:

- Schur functions come with a second product, the inner product which stems from the Cartesian product of alphabets. If dualized using the Schur-Hall scalar product, we obtain the inner coproduct.
- The inner coproduct has the same comultiplication table (section coefficients) as the inner products multiplication table. Therefore we can use the Murnaghan-Nakayama rule to evaluate this map.
- Note that the inner product and the inner coproduct do not form a Hopf algebra but only a bialgebra. There is no antipode.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> cinner(s[4]);
  cinner(s[2,2,1]);
```

$$(s_4 \,\&t\, s_4) + (s_{3,1} \,\&t\, s_{3,1}) + (s_{2,2} \,\&t\, s_{2,2}) + (s_{2,1,1} \,\&t\, s_{2,1,1}) + (s_{1,1,1,1} \,\&t\, s_{1,1,1,1})$$

$$\begin{aligned}(s_5 \,\&t\, s_{2,2,1}) &+ (s_{4,1} \,\&t\, s_{3,2}) + (s_{4,1} \,\&t\, s_{3,1,1}) + (s_{4,1} \,\&t\, s_{2,2,1}) + (s_{4,1} \,\&t\, s_{2,1,1,1}) \\ &+ (s_{3,2} \,\&t\, s_{4,1}) + (s_{3,2} \,\&t\, s_{3,2}) + (s_{3,2} \,\&t\, s_{3,1,1}) + (s_{3,2} \,\&t\, s_{2,2,1}) + (s_{3,2} \,\&t\, s_{2,1,1,1}) \\ &+ (s_{3,2} \,\&t\, s_{1,1,1,1,1}) + (s_{3,1,1} \,\&t\, s_{4,1}) + (s_{3,1,1} \,\&t\, s_{3,2}) + 2\,(s_{3,1,1} \,\&t\, s_{3,1,1}) \\ &+ (s_{3,1,1} \,\&t\, s_{2,2,1}) + (s_{3,1,1} \,\&t\, s_{2,1,1,1}) + (s_{2,2,1} \,\&t\, s_5) + (s_{2,2,1} \,\&t\, s_{4,1}) + (s_{2,2,1} \,\&t\, s_{3,2}) \\ &+ (s_{2,2,1} \,\&t\, s_{3,1,1}) + (s_{2,2,1} \,\&t\, s_{2,2,1}) + (s_{2,2,1} \,\&t\, s_{2,1,1,1}) + (s_{2,1,1,1} \,\&t\, s_{4,1}) \\ &+ (s_{2,1,1,1} \,\&t\, s_{3,2}) + (s_{2,1,1,1} \,\&t\, s_{3,1,1}) + (s_{2,1,1,1} \,\&t\, s_{2,2,1}) + (s_{1,1,1,1,1} \,\&t\, s_{3,2})\end{aligned}$$

The inner coproduct has a counit, the projection onto the one part complete symmetric function of

weight |\lambda| (recall s[n]=h[n])

```
> eps_inner:=(x,y)->if nops([op(x)])=1 and `+`(op(x))=`+`(op(y))
  then y else 0 end if:
> subs(`&t`=eps_inner,cinner(s[2]));
  eval(%);
```

$$\text{eps\_inner}(s_2, s_2) + \text{eps\_inner}(s_{1,1}, s_{1,1})$$

$$s_2$$

```
> subs(`&t`=eps_inner,cinner(s[1,1]));
  eval(%);
```

$$\text{eps\_inner}(s_2, s_{1,1}) + \text{eps\_inner}(s_{1,1}, s_2)$$

$$s_{1,1}$$

```
>
```

Algorithm used:

cinner is computed along the following lines:

\delta(x) = \sum_(\mu,\nu |-wt(x)) <inner(s[\mu],s[\nu] | x > s[\mu] &t s[\nu]

.

See Also: SchurFkt[inner]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[cmp2part] - projects a composition onto the related partition

Calling Sequence:

lprt := cmp2part(cmp)

Parameters:

- cmp : a composition (ordered list of integers)

Output:

- prt : a partition (unordered list of integers)

WARNING:

--none--

Description:

- To every composition of N (an ordered list of integers of fixed length summing up to N) one can assign its unordered representation (obtained by sorting the entries in a standard form). This is the associated partition. cmp2part is this projection from compositions to partitions.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> cmp2part([3,0,2,0,4]);
  cmp2part([1,1,2,2,3]);
```

$$[4, 3, 2]$$

$$[3, 2, 2, 1, 1]$$

Note that trailing zeros are pruned in the partition representation.

If we product all compositions we find the following

```
> cmp:=CompNM(3,7);
```

$$cmp := [[3, 0, 0, 0, 0, 0, 0], [2, 1, 0, 0, 0, 0, 0], [2, 0, 1, 0, 0, 0, 0], [2, 0, 0, 1, 0, 0, 0], [2, 0, 0, 0, 1, 0, 0], [2, 0, 0, 0, 0, 1, 0], [2, 0, 0, 0, 0, 0, 1], [1, 2, 0, 0, 0, 0, 0], [1, 1, 1, 0, 0, 0, 0], [1, 1, 0, 1, 0, 0, 0], [1, 1, 0, 0, 1, 0, 0], [1, 1, 0, 0, 0, 1, 0], [1, 1, 0, 0, 0, 0, 1], [1, 0, 2, 0, 0, 0, 0], [1, 0, 1, 1, 0, 0, 0], [1, 0, 1, 0, 1, 0, 0], [1, 0, 1, 0, 0, 1, 0], [1, 0, 1, 0, 0, 0, 1], [1, 0, 0, 2, 0, 0, 0], [1, 0, 0, 1, 1, 0, 0], [1, 0, 0, 1, 0, 1, 0], [1, 0, 0, 1, 0, 0, 1], [1, 0, 0, 0, 2, 0, 0], [1, 0, 0, 0, 1, 1, 0], [1, 0, 0, 0, 1, 0, 1], [1, 0, 0, 0, 0, 2, 0], [1, 0, 0, 0, 0, 1, 1], [1, 0, 0, 0, 0, 0, 2], [0, 3, 0, 0, 0, 0, 0], [0, 2, 1, 0, 0, 0, 0], [0, 2, 0, 1, 0, 0, 0], [0, 2, 0, 0, 1, 0, 0],$$

[0, 2, 0, 0, 0, 1, 0], [0, 2, 0, 0, 0, 0, 1], [0, 1, 2, 0, 0, 0, 0], [0, 1, 1, 1, 0, 0, 0], [0, 1, 1, 0, 1, 0, 0], [0, 1, 1, 0, 0, 1, 0], [0, 1, 1, 0, 0, 0, 1], [0, 1, 0, 2, 0, 0, 0], [0, 1, 0, 1, 1, 0, 0], [0, 1, 0, 1, 0, 1, 0], [0, 1, 0, 1, 0, 0, 1], [0, 1, 0, 0, 2, 0, 0], [0, 1, 0, 0, 1, 1, 0], [0, 1, 0, 0, 1, 0, 1], [0, 1, 0, 0, 0, 2, 0], [0, 1, 0, 0, 0, 1, 1], [0, 1, 0, 0, 0, 0, 2], [0, 0, 3, 0, 0, 0, 0], [0, 0, 2, 1, 0, 0, 0], [0, 0, 2, 0, 1, 0, 0], [0, 0, 2, 0, 0, 1, 0], [0, 0, 2, 0, 0, 0, 1], [0, 0, 1, 2, 0, 0, 0], [0, 0, 1, 1, 1, 0, 0], [0, 0, 1, 1, 0, 1, 0], [0, 0, 1, 1, 0, 0, 1], [0, 0, 1, 0, 2, 0, 0], [0, 0, 1, 0, 1, 1, 0], [0, 0, 1, 0, 1, 0, 1], [0, 0, 1, 0, 0, 2, 0], [0, 0, 1, 0, 0, 1, 1], [0, 0, 1, 0, 0, 0, 2], [0, 0, 0, 3, 0, 0, 0], [0, 0, 0, 2, 1, 0, 0], [0, 0, 0, 2, 0, 1, 0], [0, 0, 0, 2, 0, 0, 1], [0, 0, 0, 1, 2, 0, 0], [0, 0, 0, 1, 1, 1, 0], [0, 0, 0, 1, 1, 0, 1], [0, 0, 0, 1, 0, 2, 0], [0, 0, 0, 1, 0, 1, 1], [0, 0, 0, 1, 0, 0, 2], [0, 0, 0, 0, 3, 0, 0], [0, 0, 0, 0, 2, 1, 0], [0, 0, 0, 0, 2, 0, 1], [0, 0, 0, 0, 1, 2, 0], [0, 0, 0, 0, 1, 1, 1], [0, 0, 0, 0, 1, 0, 2], [0, 0, 0, 0, 0, 3, 0], [0, 0, 0, 0, 0, 2, 1], [0, 0, 0, 0, 0, 1, 2], [0, 0, 0, 0, 0, 0, 3]]

```
> map(cmp2part,cmp);
  sort(convert(convert(%,set),list),grAlexComp);
```

[[3], [2, 1], [2, 1], [2, 1], [2, 1], [2, 1], [2, 1], [2, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1], [2, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1], [2, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1], [2, 1], [1, 1, 1], [1, 1, 1], [2, 1], [1, 1, 1], [2, 1], [3], [2, 1], [2, 1], [2, 1], [2, 1], [2, 1], [2, 1], [2, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1], [2, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1], [2, 1], [1, 1, 1], [1, 1, 1], [2, 1], [1, 1, 1], [2, 1], [3], [2, 1], [2, 1], [2, 1], [2, 1], [2, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1], [2, 1], [1, 1, 1], [1, 1, 1], [2, 1], [1, 1, 1], [2, 1], [3], [2, 1], [2, 1], [2, 1], [2, 1], [2, 1], [2, 1], [1, 1, 1], [1, 1, 1], [2, 1], [1, 1, 1], [2, 1], [3], [2, 1], [2, 1], [2, 1], [1, 1, 1], [2, 1], [3], [2, 1], [2, 1], [3]]

[[3], [2, 1], [1, 1, 1]]

```
>
```

Compositions may be grouped into orbits as preimage of the partition obtained under cmp2part. The cardinality of this orbit is computed by cmp2prtMult.

Algorithm used:

cmp2part just sorts the composition into decreasing order and prunes the trailing zeros.

See Also: SchurFkt[cmp2prtMult], SchurFkt[CompNM]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[cmp2prtMult] - computes the length of the orbit of compostions which project under sorting to the same partition

Calling Sequence:

prt := cmp2prtMult(cmp))

Parameters:

- cmp : a composition

Output:

- int : an integer

WARNING:

--none--

Description:

- Internal use mainly.

Examples:

```
> restart:with(SchurFkt);
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
```

[*AlexComp*, *CharHook*, *CompNM*, *FLAT*, *Frob2part*, *GesselThetaP*, *GesselThetaS*, *KostkaPC*, *KostkaTable*, *LaplaceM*, *LaplaceM_mon*, *LaplaceTable*, *MLIN*, *MurNak*, *MurNak2*, *PartNM*, *Scalar*, *ScalarHM*, *ScalarMH*, *ScalarP*, *antipE*, *antipH*, *antipM*, *antipMC*, *antipP*, *antipS*, *branch*, *cinner*, *cinnerP*, *cmp2part*, *cmp2prtMult*, *concatM*, *conjpart*, *counitInnerP*, *counitInnerS*, *couter*, *couterE*, *couterH*, *couterM*, *couterON*, *couterP*, *cplethP*, *cplethS*, *dimSN*, *e_to_h*, *e_to_s*, *evalJacobiTrudiMatrix*, *getSfktSeries*, *grAlexComp*, *h_to_m*, *h_to_s*, *inner*, *innerH*, *innerP*, *isLattice*, *m_to_p*, *maxlengthSymFkt*, *mset2part*, *outer*, *outerE*, *outerH*, *outerM*, *outerON*, *outerP*, *outerS*, *p_to_m*, *p_to_s*, *part2Frob*, *part2mset*, *plethP*, *plethS*, *plethSnm*, *s_to_h*, *s_to_hJT*, *s_to_hmat*, *s_to_p*, *s_to_x*, *skew*, *sq_coeff*, *truncLEN*, *truncWT*, *x_to_s*, *zee*]

```
> cmp2prtMult([2,0,1,1]);
  cmp2prtMult([0,3,1,0]);
```

$$18$$

$$12$$

We study the orbits of the the projection of compositions of 4 into 6 parts onto partitions of 4:

```
> prt:=PartNM(4,6);
  cmp2prtMult([4,0,0,0,0,0]);  # orbit of [4]
```

```
cmp2prtMult([3,1,0,0,0,0]);  # orbit of [3,1]
cmp2prtMult([2,2,0,0,0,0]);  # orbit of [2,2]
cmp2prtMult([2,1,1,0,0,0]);  # orbit of [2,1,1]
cmp2prtMult([1,1,1,1,0,0]);  # orbit of [1,1,1,1]
cmp:=CompNM(4,6):
out1:=map(cmp2prtMult,cmp);  # compute the multiplicities
nops(out1);
convert(out1,set);
`+`(op(%));                  # add tem up as a test
```

$$prt := [[4], [3, 1], [2, 2], [2, 1, 1], [1, 1, 1, 1]]$$

$$6$$

$$30$$

$$15$$

$$75$$

$$15$$

$$out1 := [6, 30, 30, 30, 30, 30, 15, 75, 75, 75, 75, 15, 75, 75, 75, 15, 75, 75, 15, 75, 15, 30, 75, 75, 75, 75, 75, 15, 15, 15, 75, 15, 15, 75, 15, 75, 30, 75, 75, 75, 75, 15, 15, 75, 15, 75, 30, 75, 75, 75, 15, 75, 30, 75, 75, 30, 6, 30, 30, 30, 30, 15, 75, 75, 75, 15, 75, 75, 15, 75, 15, 30, 75, 75, 75, 75, 15, 15, 75, 15, 75, 30, 75, 75, 75, 15, 75, 30, 75, 75, 30, 6, 30, 30, 30, 15, 75, 75, 15, 75, 15, 30, 75, 75, 75, 15, 75, 30, 75, 75, 30, 6, 30, 30, 15, 75, 15, 30, 75, 75, 30, 6, 30, 15, 30, 6]$$

$$126$$

$$\{6, 15, 30, 75\}$$

$$126$$

```
>
```

Algorithm used:

--NA--

See Also: SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[CompNM] - produces a list of compositions of N into M parts

Calling Sequence:

lst := CompNM(N,M)

Parameters:

- N,M : nonnegative integers

Output:

- lst : list of compositions (list of integers) of N into M parts

WARNING:

Maple's combinat package contains a similar function which returns the partitions in a different (lexicographical) ordering.

Description:

- CompNM takes two integers as input and produces a list of lists of integers (a list of compositions). A composition is an ordered collection of nonnegative integers which add up to the given number N. The number of elements in this decomposition (including zeros and multiplicities) is M.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

Just a few examples:

```
> CompNM(5,2);
  CompNM(2,2);
  CompNM(2,4);
  CompNM(3,4);
```

$$[[5, 0], [4, 1], [3, 2], [2, 3], [1, 4], [0, 5]]$$

$$[[2, 0], [1, 1], [0, 2]]$$

$$[[2, 0, 0, 0], [1, 1, 0, 0], [1, 0, 1, 0], [1, 0, 0, 1], [0, 2, 0, 0], [0, 1, 1, 0], [0, 1, 0, 1], [0, 0, 2, 0], [0, 0, 1, 1], [0, 0, 0, 2]]$$

$$[[3, 0, 0, 0], [2, 1, 0, 0], [2, 0, 1, 0], [2, 0, 0, 1], [1, 2, 0, 0], [1, 1, 1, 0], [1, 1, 0, 1], [1, 0, 2, 0], [1, 0, 1, 1], [1, 0, 0, 2], [0, 3, 0, 0], [0, 2, 1, 0], [0, 2, 0, 1], [0, 1, 2, 0], [0, 1, 1, 1], [0, 1, 0, 2], [0, 0, 3, 0], [0, 0, 2, 1], [0, 0, 1, 2], [0, 0, 0, 3]]$$

Some dangerous or illegal cases:

```
> CompNM(0,0);  # no composition at all
  CompNM(-1,2); # illegal, but no error generated !
  CompNM(4,-1); # illegal, but no error generated !
```

$$[\ ]$$

$$[\ ]$$

$$[\ ]$$

Note that the list of compositions can be used to produce certain symmetrized complete symmetric functions in explicit form on the x variables

```
> h_to_x:=proc(m,nopsvar)
    local cmp,c1,f;
    c1:=`+`(op(m)):
    cmp:=CompNM(c1,nopsvar);
    f:=(lst)->mul(x[i]^lst[i],i=1..nops(lst));
    `+`(op(map(f,cmp)));
  end proc:
```

```
> h_to_x(h[2],2);
  h_to_x(h[2],3);
  h_to_x(h[2],4);
  h_to_x(h[2],5);
```

$$x_1^2 + x_1 x_2 + x_2^2$$

$$x_1^2 + x_1 x_2 + x_1 x_3 + x_2^2 + x_2 x_3 + x_3^2$$

$$x_1^2 + x_1 x_2 + x_1 x_3 + x_1 x_4 + x_2^2 + x_2 x_3 + x_2 x_4 + x_3^2 + x_3 x_4 + x_4^2$$

$$x_1^2 + x_1 x_2 + x_1 x_3 + x_1 x_4 + x_1 x_5 + x_2^2 + x_2 x_3 + x_2 x_4 + x_2 x_5 + x_3^2 + x_3 x_4 + x_3 x_5 + x_4^2 + x_4 x_5 + x_5^2$$

```
> h_to_x(h[2,1],3);
  h_to_x(h[2,2],3);  # terms may be missing
  h_to_x(h[1,1,1,1],2);
  h_to_x(h[1,1,1,1],4);  # terms may be missing
```

$$x_1^3 + x_1^2 x_2 + x_1^2 x_3 + x_1 x_2^2 + x_1 x_2 x_3 + x_1 x_3^2 + x_2^3 + x_2^2 x_3 + x_2 x_3^2 + x_3^3$$

$$x_1^4 + x_1^3 x_2 + x_1^3 x_3 + x_1^2 x_2^2 + x_1^2 x_2 x_3 + x_1^2 x_3^2 + x_1 x_2^3 + x_1 x_2^2 x_3 + x_1 x_2 x_3^2 + x_1 x_3^3 + x_2^4 + x_2^3 x_3 + x_2^2 x_3^2 + x_2 x_3^3 + x_3^4$$

$$x_1^4 + x_1^3 x_2 + x_1^2 x_2^2 + x_1 x_2^3 + x_2^4$$

$$x_2 x_4^3 + x_2^2 x_3 x_4 + x_1^3 x_4 + x_1^2 x_2 x_4 + x_1^2 x_3 x_4 + x_1^2 x_4^2 + x_1 x_2^2 x_4 + x_1 x_2 x_3 x_4 + x_1 x_2 x_4^2 + x_1 x_3^2 x_4 + x_1 x_3 x_4^2 + x_1 x_4^3 + x_2^3 x_3 + x_2^2 x_3^2 + x_2 x_3^3 + x_3^3 x_4 + x_3^2 x_4^2 + x_3 x_4^3 + x_1^3 x_2$$

$$+x_1^3 x_3 + x_1^2 x_2^2 + x_1^4 + x_2^4 + x_1^2 x_2 x_3 + x_1^2 x_3^2 + x_1 x_2^3 + x_1 x_2^2 x_3 + x_1 x_2 x_3^2 + x_1 x_3^3 + x_3^4$$
$$+x_2^3 x_4 + x_2^2 x_4^2 + x_2 x_3^2 x_4 + x_2 x_3 x_4^2 + x_4^4$$

Note: If a symmetric function is stable, then its form does not depend on the numbers of variables if that number N is large enough. In specifying the number of variables to be small, certain syzygies appear.

>

Algorithm:

The algorithm used is that of the Maple combinat package, but adapted to the inverse order.

See Also: SchurFkt[PartNM], Maple's combinat

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[concatM] - Divided powers concatenation product (needed for Rota-Stein cliffordization)

Calling Sequence:

mfkt := concatM(mfkt1,mfkt2,...)

Parameters:

- mfkti : monomial symmetric functions

Output:

- mfkt : a monomial symmetric function

WARNING:

-- note that this concatenation has nothing to do with the concatenation used to implement the outer product of symmetric function for the multiplicative bases (p,h,e), but is a new product!

Description:

- The concatM product is used to implement the outer product of symmetric function in the monomial symmetric function basis. It is **not the outer product** of monomial symmetric function, but a freely generated algebra of the underlying module.
- According to Rota and Stein one can represent monomial symmetric functions via a single generator (alphabet) in a special setting. This is done as follows:

  i) Consider the symmetric (tensor) algebra in one generator a and denote it as S(a). Take the module of the
  kernel of the counit of this Hopf algebra, which is denoted as S^+(a) = a+ (axa)_sym + (axaxa)_sym + ... = Z[a]^+
  (the unit element, that is the grade zero part, is missing).
  ii) Build the Divided powers algebra over the module S^+(a) spanned by the elements a^k. A basis monomial of this space reads

  m_\lambda = (a^1)^(r1) (a^2)^(r2) ... (a^l)^(rl)

  where the ri are the multiplicities of the partition \lambda=[1^r1 2^r2 3^r3 .... l^rl] . This module is denoted as Div[S^+(a)]. It carries the divided powers multiplication as algebra structure.
- The **concatenation product** |_| (free product) is implemented by demanding that

```
                                        [     (a^r)^(s)(a^u)^(v)                  if  r <> s
(1)  (a^r)^(s) |_| (a^u)^(v) =    {
                                        [     binomial( s+v, v) * (a^r)(s+v)      else
```

This turns Div[S^+(a)] into an divided powers algebra with infinitely many generators.

- The outer product for monomial symmetric function is then obtained by a cliffordization using the Laplace pairing LaplaceM , see outerM.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

concatM is symmetric:

```
> concatM(m[1],m[2]);
  concatM(m[2],m[1]);
  concatM(m[2],m[2]);       # divided powers prefactor!
  concatM(m[3,3],m[3,3]);   # divided powers prefactor!
```

$$m_{2,1}$$

$$m_{2,1}$$

$$2\, m_{2,2}$$

$$6\, m_{3,3,3,3}$$

concatM is multilinear and can handle arbitrarily many arguments (but needs at least one).

```
> concatM(m[3,2,1]);
  concatM(m[2],m[1],m[1]);
  concatM(m[3,2],m[3,2],m[1,1],m[1]);
```

$$m_{3,2,1}$$

$$2\, m_{2,1,1}$$

$$12\, m_{3,3,2,2,1,1,1}$$

```
> concatM(2*m[2]+m[3],m[1]+3*m[4]);
```

$$2\, m_{2,1} + 6\, m_{4,2} + m_{3,1} + 3\, m_{4,3}$$

Test for special cases:

```
> concatM(); # ERROR one argument needed!
Error, (in SchurFkt:-concatM) invalid subscript selector
```

```
> concatM(0),concatM(m[0]);
  concatM(m[]); ## should not occure as input...
```

$$0, m_0$$

$$m_{[\ ]}$$

```
>
```

Algorithm used:

The above described multiplicative representation is used to implement verbatim the divided powers structure.

See Also: SchurFkt[outerM], SchurFkt[outer], SchurFkt[LaplaceM]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[conjpart] - computes a conjugate partition

Calling Sequence:

prt2 := conjpart(prt1)

Parameters:

- prt1 : integer partition

Output:

- prt2 : integer partition

WARNING:

--none--

Description:

- An integer partition may be represented as a Young diagram (in Anglo-Saxon notation using indices as in matrices). Such a diagram has a main diagonal and we flip the diagram mirroring at this diagonal. The mirrored diagram gives rise to another partition which is called conjugate partition.
- conjpart is an involution. Since it is obtained from mirroring the diagram this is obvious.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> conjpart([3,2]);
  conjpart([5]);
  conjpart([2,2,2,2]);
```

$$[2, 2, 1]$$

$$[1, 1, 1, 1, 1]$$

$$[4, 4]$$

```
> prt:=PartNM(4,4):prt;
  map(conjpart,prt);
```

$$[[4], [3, 1], [2, 2], [2, 1, 1], [1, 1, 1, 1]]$$

$$[[1, 1, 1, 1], [2, 1, 1], [2, 2], [3, 1], [4]]$$

Note that conjpart inverses the grAlexComp total order which we use on partitions.

Show that conjpart is an involution:

```
> conjpart([3,3,1,1,1,1]);
  conjpart(%);
```

```
#
conjpart([4,4,2,1,1,1]);
conjpart(%);
```

$$[6, 2, 2]$$

$$[3, 3, 1, 1, 1, 1]$$

$$[6, 3, 2, 2]$$

$$[4, 4, 2, 1, 1, 1]$$

```
>
```

Algorithm used:

The length of the partition gives the first entry of the conjugate partition. Subtract the last element from all former element and add as many times the length of the partition as the last element is in magnitude, continue by recursion on the length.

See Also: SchurFkt[grAlexComp], SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[couter] - the outer coproduct in the Schur function basis

Calling Sequence:

stens := couter(sfkt,...)

Parameters:

- sfkt : Schur functions

Output:

- stens : 2 tensor made out of Schur function monomials

WARNING:

--none--

Description:

- couter implements the outer coproduct in the Schur function basis.
- Since the Hopf algebra of symmetric function is self dual, the outer coproduct of the Schur functions is obtained by dualizing the product on the dual basis. This implies that the outer coproduct can be derived directly using the Littlewood-Richardson rule:

  (1) \delta(s[\lambda]) = \sum_{\mu,\nu} c^\lambda_{\mu\nu} s[\mu] x s[\nu] (x = tensor)

  where the sum is over all pairs of partitions \mu,\nu such that the sum of their weight is equal to the weight of \lambda.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
```

```
> couter(s[2]);
  couter(s[2,2]);
  couter(s[2,1]);
  couter(2*s[3,1]+s[2,1]);
```

$$(s_0 \,\&t\, s_2) + (s_2 \,\&t\, s_0) + (s_1 \,\&t\, s_1)$$

$$(s_0 \,\&t\, s_{2,2}) + (s_{2,2} \,\&t\, s_0) + (s_{2,1} \,\&t\, s_1) + (s_2 \,\&t\, s_2) + (s_{1,1} \,\&t\, s_{1,1}) + (s_1 \,\&t\, s_{2,1})$$

$$(s_0 \,\&t\, s_{2,1}) + (s_{2,1} \,\&t\, s_0) + (s_2 \,\&t\, s_1) + (s_{1,1} \,\&t\, s_1) + (s_1 \,\&t\, s_2) + (s_1 \,\&t\, s_{1,1})$$

$$2\,(s_0 \,\&t\, s_{3,1}) + 2\,(s_{3,1} \,\&t\, s_0) + 2\,(s_3 \,\&t\, s_1) + 2\,(s_{2,1} \,\&t\, s_1) + 2\,(s_2 \,\&t\, s_2) + 2\,(s_{1,1} \,\&t\, s_2) + 2\,(s_2 \,\&t\, s_{1,1}) + 2\,(s_1 \,\&t\, s_3) + 2\,(s_1 \,\&t\, s_{2,1}) + (s_0 \,\&t\, s_{2,1}) + (s_{2,1} \,\&t\, s_0) + (s_2 \,\&t\, s_1)$$

$$+ (s_{1,1} \,\&t\, s_1) + (s_1 \,\&t\, s_2) + (s_1 \,\&t\, s_{1,1})$$

Show that the coproduct outer and the outer product outer fulfill the compatibility axiom for Hopf algebras

```
> f:=(a,b,c,d)->&t(outer(a,c),outer(b,d)):
  &t(couter(s[1]),couter(s[2]));
  eval(subs(`&t`=f,%));
  couter(outer(s[1],s[2]));
```

$$\&t(s_0, s_1, s_0, s_2) + \&t(s_0, s_1, s_2, s_0) + \&t(s_0, s_1, s_1, s_1) + \&t(s_1, s_0, s_0, s_2) + \&t(s_1, s_0, s_2, s_0) + \&t(s_1, s_0, s_1, s_1)$$

$$(s_0 \,\&t\, s_3) + (s_0 \,\&t\, s_{2,1}) + 2\,(s_2 \,\&t\, s_1) + 2\,(s_1 \,\&t\, s_2) + (s_1 \,\&t\, s_{1,1}) + (s_3 \,\&t\, s_0) + (s_{2,1} \,\&t\, s_0) + (s_{1,1} \,\&t\, s_1)$$

$$(s_0 \,\&t\, s_3) + (s_0 \,\&t\, s_{2,1}) + 2\,(s_2 \,\&t\, s_1) + 2\,(s_1 \,\&t\, s_2) + (s_1 \,\&t\, s_{1,1}) + (s_3 \,\&t\, s_0) + (s_{2,1} \,\&t\, s_0) + (s_{1,1} \,\&t\, s_1)$$

Test cases:

```
> couter(0);     # returns zero
  couter(s[0]); # special case since m[0] is te image of the
  counit
```

$$0$$

$$s_0 \,\&t\, s_0$$

```
>
```

Algorithm used:

A straight forward implementation of the rhs of (1) using the Littlewood-Richardson coefficients.

See Also: SchurFkt[outer], SchurFkt[Overview] (there are many outerA and couterA functions for basis 'A')

Last modified: June 19, 2008/BF/RA

Function:

SchurFkt[couterE] - the outer coproduct in the elementary symmetric function basis
SchurFkt[couterH] - the outer coproduct in the complete symmetric function basis
SchurFkt[couterP] - the outer coproduct in the power sum basis

Calling Sequence:

mtens := couterE(efkt)
mtens := couterH(hfkt)
mtens := couterP(pfkt)

Parameters:

- efkt : elementary symmetric function
  hfkt : complete symmetric function
  pfkt : power sum symmetric function

Output:

- mtens : 2 tensor made out of [elementary|complete|power sum] symmetric function monomials (input and output type coincide)

WARNING:

--none--

Description:

- couterE, couterH and couterP describe the **outer coproduct** in the elementary, complete and power sum basis.
- The procedures for couterE and couterH are the same. The outer coproduct of these bases is derived from the following facts:
  - The outer coproduct is an algebra homomorphism of the outer product algebra
    \Delta(x.y) = \Delta(x).\Delta(y)
  - The outer coproduct in the elementary and complete basis for *one part partitions* e[n] or h[n] are obtained from
    \Delta(e[n]) = \sum_{r=0}^n e[n-r] \otimes e[r]
    \Delta(h[n]) = \sum_{r=0}^n h[n-r] \otimes h[r]
    hence the one part functions in the e- and h-basis obey a coproduct of a **divided powers** algebra.
- The power sum coproduct is also reduced to one part power sum via the homomorphism property, however, the power sums enjoy a binomial coproduct rule steming from dualizing a polynomial algebra (recall that the power sum symmetric functions are orthogonal, but not normalized and hence self dual). One gets:
  \Delta(p[n]) = p[n]\stemming1 + 1 \otimes p[n]
  and the one part power sum functions are the **primitive elements** of the outer coalgebra of symmetric functions. This can be written as

p[n](x,y) = p[n](x)+p[n](y) and due to the fact that p[n](x) =\sum_{i=1}^N x_i^n one gets the binomial relation.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
```

```
> couterE(e[2]);
  couterE(e[2,2]);
  couterE(e[2,1]);
  couterE(2*e[3,1]+e[2,1]);
  "----";
  couterH(h[2]);
  couterH(h[2,2]);
  couterH(h[2,1]);
  couterH(2*h[3,1]+h[2,1]);
```

$$(e_2 \,\&t\, e_0) + (e_1 \,\&t\, e_1) + (e_0 \,\&t\, e_2)$$

$$(e_{2,2} \,\&t\, e_0) + 2\,(e_{2,1} \,\&t\, e_1) + 2\,(e_2 \,\&t\, e_2) + (e_{1,1} \,\&t\, e_{1,1}) + 2\,(e_1 \,\&t\, e_{2,1}) + (e_0 \,\&t\, e_{2,2})$$

$$(e_{2,1} \,\&t\, e_0) + (e_2 \,\&t\, e_1) + (e_{1,1} \,\&t\, e_1) + (e_1 \,\&t\, e_{1,1}) + (e_1 \,\&t\, e_2) + (e_0 \,\&t\, e_{2,1})$$

$$2\,(e_{3,1} \,\&t\, e_0) + 2\,(e_3 \,\&t\, e_1) + 2\,(e_{2,1} \,\&t\, e_1) + 2\,(e_2 \,\&t\, e_{1,1}) + 2\,(e_{1,1} \,\&t\, e_2) + 2\,(e_1 \,\&t\, e_{2,1}) + 2\,(e_1 \,\&t\, e_3) + 2\,(e_0 \,\&t\, e_{3,1}) + (e_{2,1} \,\&t\, e_0) + (e_2 \,\&t\, e_1) + (e_{1,1} \,\&t\, e_1) + (e_1 \,\&t\, e_{1,1}) + (e_1 \,\&t\, e_2) + (e_0 \,\&t\, e_{2,1})$$

"----"

$$(h_2 \,\&t\, h_0) + (h_1 \,\&t\, h_1) + (h_0 \,\&t\, h_2)$$

$$(h_{2,2} \,\&t\, h_0) + 2\,(h_{2,1} \,\&t\, h_1) + 2\,(h_2 \,\&t\, h_2) + (h_{1,1} \,\&t\, h_{1,1}) + 2\,(h_1 \,\&t\, h_{2,1}) + (h_0 \,\&t\, h_{2,2})$$

$$(h_{2,1} \,\&t\, h_0) + (h_2 \,\&t\, h_1) + (h_{1,1} \,\&t\, h_1) + (h_1 \,\&t\, h_{1,1}) + (h_1 \,\&t\, h_2) + (h_0 \,\&t\, h_{2,1})$$

$$2\,(h_{3,1} \,\&t\, h_0) + 2\,(h_3 \,\&t\, h_1) + 2\,(h_{2,1} \,\&t\, h_1) + 2\,(h_2 \,\&t\, h_{1,1}) + 2\,(h_{1,1} \,\&t\, h_2) + 2\,(h_1 \,\&t\, h_{2,1}) + 2\,(h_1 \,\&t\, h_3) + 2\,(h_0 \,\&t\, h_{3,1}) + (h_{2,1} \,\&t\, h_0) + (h_2 \,\&t\, h_1) + (h_{1,1} \,\&t\, h_1) + (h_1 \,\&t\, h_{1,1}) + (h_1 \,\&t\, h_2) + (h_0 \,\&t\, h_{2,1})$$

Hence the two coproducts behave the same way. However we find:

```
> couterP(p[2]);
  couterP(p[2,2]); # (x+y)^2= x^2 + 2*x*y + y^2
  couterP(p[2,1]);
  couterP(2*p[3,1]+p[2,1]);
```

$$(p_2 \,\&t\, p_0) + (p_0 \,\&t\, p_2)$$

$$(p_{2,2} \,\&t\, p_0) + 2\,(p_2 \,\&t\, p_2) + (p_0 \,\&t\, p_{2,2})$$

$$(p_{2,1} \,\&t\, p_0) + (p_2 \,\&t\, p_1) + (p_1 \,\&t\, p_2) + (p_0 \,\&t\, p_{2,1})$$

$$2\,(p_{3,1}\,\&t\,p_0)+2\,(p_3\,\&t\,p_1)+2\,(p_1\,\&t\,p_3)+2\,(p_0\,\&t\,p_{3,1})+(p_{2,1}\,\&t\,p_0)+(p_2\,\&t\,p_1)+(p_1\,\&t\,p_2)+(p_0\,\&t\,p_{2,1})$$

Show that the coproduct outerE and the outer product outerE fulfill the compatibility axiom for Hopf algebras

```
> f:=(a,b,c,d)->&t(outerE(a,c),outerE(b,d)):
  &t(couterE(e[1]),couterE(e[2])):
  RHS:=eval(subs(`&t`=f,%));
  LHS:=couterE(outerE(e[1],e[2]));
```

$$RHS := (e_{2,1}\,\&t\,e_0)+(e_2\,\&t\,e_1)+(e_{1,1}\,\&t\,e_1)+(e_1\,\&t\,e_{1,1})+(e_1\,\&t\,e_2)+(e_0\,\&t\,e_{2,1})$$

$$LHS := (e_{2,1}\,\&t\,e_0)+(e_2\,\&t\,e_1)+(e_{1,1}\,\&t\,e_1)+(e_1\,\&t\,e_{1,1})+(e_1\,\&t\,e_2)+(e_0\,\&t\,e_{2,1})$$

We proceed by showing a few critical cases:

```
> couterP(0),couterP(sin(x));
  couterE(0),couterE(1-q^2);
  couterH(0),couterH(a);
```

$$0,\ \sin(x)\,(p_0\,\&t\,p_0)$$

$$0,\ (1-q^2)\,(e_0\,\&t\,e_0)$$

$$0,\ a\,(h_0\,\&t\,h_0)$$

```
> couterP(p[0]);
  couterE(e[0]);
  couterH(h[0]);
```

$$p_0\,\&t\,p_0$$

$$e_0\,\&t\,e_0$$

$$h_0\,\&t\,h_0$$

we show that the coproduct is a linear map over <anything> but the basis monomials.

```
> expand(couterP(q^2*p[0]+q*p[1]+p[2]));
  expand(couterH(q^2*h[3]+q*h[2,1]+h[1,1,1]));
  expand(couterE(a*e[3]+b*e[2,1]+c*e[1,1,1]));
```

$$q^2\,(p_0\,\&t\,p_0)+q\,(p_1\,\&t\,p_0)+q\,(p_0\,\&t\,p_1)+(p_2\,\&t\,p_0)+(p_0\,\&t\,p_2)$$

$$q^2\,(h_3\,\&t\,h_0)+q^2\,(h_2\,\&t\,h_1)+q^2\,(h_1\,\&t\,h_2)+q^2\,(h_0\,\&t\,h_3)+q\,(h_{2,1}\,\&t\,h_0)+q\,(h_2\,\&t\,h_1)+q\,(h_{1,1}\,\&t\,h_1)+q\,(h_1\,\&t\,h_{1,1})+q\,(h_1\,\&t\,h_2)+q\,(h_0\,\&t\,h_{2,1})+(h_{1,1,1}\,\&t\,h_0)+3\,(h_{1,1}\,\&t\,h_1)+3\,(h_1\,\&t\,h_{1,1})+(h_0\,\&t\,h_{1,1,1})$$

$$a\,(e_3\,\&t\,e_0)+a\,(e_2\,\&t\,e_1)+a\,(e_1\,\&t\,e_2)+a\,(e_0\,\&t\,e_3)+b\,(e_{2,1}\,\&t\,e_0)+b\,(e_2\,\&t\,e_1)+b\,(e_{1,1}\,\&t\,e_1)+b\,(e_1\,\&t\,e_{1,1})+b\,(e_1\,\&t\,e_2)+b\,(e_0\,\&t\,e_{2,1})+c\,(e_{1,1,1}\,\&t\,e_0)+3\,c\,(e_{1,1}\,\&t\,e_1)+3\,c\,(e_1\,\&t\,e_{1,1})+c\,(e_0\,\&t\,e_{1,1,1})$$

```
>
```

Algorithm used:

see description above.

See Also: SchurFkt[outerP], SchurFkt[outerH]

Last modified: June 19, 2008/BF/RA.

Function:

SchurFkt[couterE] - the outer coproduct in the elementary symmetric function basis
SchurFkt[couterH] - the outer coproduct in the complete symmetric function basis
SchurFkt[couterP] - the outer coproduct in the power sum basis

Calling Sequence:

mtens := couterE(efkt)
mtens := couterH(hfkt)
mtens := couterP(pfkt)

Parameters:

- efkt : elementary symmetric function
  hfkt : complete symmetric function
  pfkt : power sum symmetric function

Output:

- mtens : 2 tensor made out of [elementary|complete|power sum] symmetric function monomials (input and output type coincide)

WARNING:

--none--

Description:

- couterE, couterH and couterP describe the **outer coproduct** in the elementary, complete and power sum basis.
- The procedures for couterE and couterH are the same. The outer coproduct of these bases is derived from the following facts:
  - The outer coproduct is an algebra homomorphism of the outer product algebra
    \Delta(x.y) = \Delta(x).\Delta(y)
  - The outer coproduct in the elementary and complete basis for *one part partitions* e[n] or h[n] are obtained from
    \Delta(e[n]) = \sum_{r=0}^n e[n-r] \otimes e[r]
    \Delta(h[n]) = \sum_{r=0}^n h[n-r] \otimes h[r]
    hence the one part functions in the e- and h-basis obey a coproduct of a **divided powers** algebra.
- The power sum coproduct is also reduced to one part power sum via the homomorphism property, however, the power sums enjoy a binomial coproduct rule steming from dualizing a polynomial algebra (recall that the power sum symmetric functions are orthogonal, but not normalized and hence self dual). One gets:
  \Delta(p[n]) = p[n]\stemming1 + 1 \otimes p[n]
  and the one part power sum functions are the **primitive elements** of the outer coalgebra of symmetric functions. This can be written as

p[n](x,y) = p[n](x)+p[n](y) and due to the fact that p[n](x) =\sum_{i=1}^N x_i^n one gets the binomial relation.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
```

```
> couterE(e[2]);
  couterE(e[2,2]);
  couterE(e[2,1]);
  couterE(2*e[3,1]+e[2,1]);
  "----";
  couterH(h[2]);
  couterH(h[2,2]);
  couterH(h[2,1]);
  couterH(2*h[3,1]+h[2,1]);
```

$$(e_2 \,\&t\, e_0) + (e_1 \,\&t\, e_1) + (e_0 \,\&t\, e_2)$$

$$(e_{2,2} \,\&t\, e_0) + 2\,(e_{2,1} \,\&t\, e_1) + 2\,(e_2 \,\&t\, e_2) + (e_{1,1} \,\&t\, e_{1,1}) + 2\,(e_1 \,\&t\, e_{2,1}) + (e_0 \,\&t\, e_{2,2})$$

$$(e_{2,1} \,\&t\, e_0) + (e_2 \,\&t\, e_1) + (e_{1,1} \,\&t\, e_1) + (e_1 \,\&t\, e_{1,1}) + (e_1 \,\&t\, e_2) + (e_0 \,\&t\, e_{2,1})$$

$$2\,(e_{3,1} \,\&t\, e_0) + 2\,(e_3 \,\&t\, e_1) + 2\,(e_{2,1} \,\&t\, e_1) + 2\,(e_2 \,\&t\, e_{1,1}) + 2\,(e_{1,1} \,\&t\, e_2) + 2\,(e_1 \,\&t\, e_{2,1}) + 2\,(e_1 \,\&t\, e_3) + 2\,(e_0 \,\&t\, e_{3,1}) + (e_{2,1} \,\&t\, e_0) + (e_2 \,\&t\, e_1) + (e_{1,1} \,\&t\, e_1) + (e_1 \,\&t\, e_{1,1}) + (e_1 \,\&t\, e_2) + (e_0 \,\&t\, e_{2,1})$$

"----"

$$(h_2 \,\&t\, h_0) + (h_1 \,\&t\, h_1) + (h_0 \,\&t\, h_2)$$

$$(h_{2,2} \,\&t\, h_0) + 2\,(h_{2,1} \,\&t\, h_1) + 2\,(h_2 \,\&t\, h_2) + (h_{1,1} \,\&t\, h_{1,1}) + 2\,(h_1 \,\&t\, h_{2,1}) + (h_0 \,\&t\, h_{2,2})$$

$$(h_{2,1} \,\&t\, h_0) + (h_2 \,\&t\, h_1) + (h_{1,1} \,\&t\, h_1) + (h_1 \,\&t\, h_{1,1}) + (h_1 \,\&t\, h_2) + (h_0 \,\&t\, h_{2,1})$$

$$2\,(h_{3,1} \,\&t\, h_0) + 2\,(h_3 \,\&t\, h_1) + 2\,(h_{2,1} \,\&t\, h_1) + 2\,(h_2 \,\&t\, h_{1,1}) + 2\,(h_{1,1} \,\&t\, h_2) + 2\,(h_1 \,\&t\, h_{2,1}) + 2\,(h_1 \,\&t\, h_3) + 2\,(h_0 \,\&t\, h_{3,1}) + (h_{2,1} \,\&t\, h_0) + (h_2 \,\&t\, h_1) + (h_{1,1} \,\&t\, h_1) + (h_1 \,\&t\, h_{1,1}) + (h_1 \,\&t\, h_2) + (h_0 \,\&t\, h_{2,1})$$

Hence the two coproducts behave the same way. However we find:

```
> couterP(p[2]);
  couterP(p[2,2]); # (x+y)^2= x^2 + 2*x*y + y^2
  couterP(p[2,1]);
  couterP(2*p[3,1]+p[2,1]);
```

$$(p_2 \,\&t\, p_0) + (p_0 \,\&t\, p_2)$$

$$(p_{2,2} \,\&t\, p_0) + 2\,(p_2 \,\&t\, p_2) + (p_0 \,\&t\, p_{2,2})$$

$$(p_{2,1} \,\&t\, p_0) + (p_2 \,\&t\, p_1) + (p_1 \,\&t\, p_2) + (p_0 \,\&t\, p_{2,1})$$

$$2\,(p_{3,1}\,\&t\,p_0)+2\,(p_3\,\&t\,p_1)+2\,(p_1\,\&t\,p_3)+2\,(p_0\,\&t\,p_{3,1})+(p_{2,1}\,\&t\,p_0)+(p_2\,\&t\,p_1)+(p_1\,\&t\,p_2)+(p_0\,\&t\,p_{2,1})$$

Show that the coproduct outerE and the outer product outerE fulfill the compatibility axiom for Hopf algebras

```
> f:=(a,b,c,d)->&t(outerE(a,c),outerE(b,d)):
  &t(couterE(e[1]),couterE(e[2])):
  RHS:=eval(subs(`&t`=f,%));
  LHS:=couterE(outerE(e[1],e[2]));
```

$$RHS := (e_{2,1}\,\&t\,e_0)+(e_2\,\&t\,e_1)+(e_{1,1}\,\&t\,e_1)+(e_1\,\&t\,e_{1,1})+(e_1\,\&t\,e_2)+(e_0\,\&t\,e_{2,1})$$

$$LHS := (e_{2,1}\,\&t\,e_0)+(e_2\,\&t\,e_1)+(e_{1,1}\,\&t\,e_1)+(e_1\,\&t\,e_{1,1})+(e_1\,\&t\,e_2)+(e_0\,\&t\,e_{2,1})$$

We proceed by showing a few critical cases:

```
> couterP(0),couterP(sin(x));
  couterE(0),couterE(1-q^2);
  couterH(0),couterH(a);
```

$$0,\ \sin(x)\,(p_0\,\&t\,p_0)$$

$$0,\ (1-q^2)\,(e_0\,\&t\,e_0)$$

$$0,\ a\,(h_0\,\&t\,h_0)$$

```
> couterP(p[0]);
  couterE(e[0]);
  couterH(h[0]);
```

$$p_0\,\&t\,p_0$$

$$e_0\,\&t\,e_0$$

$$h_0\,\&t\,h_0$$

we show that the coproduct is a linear map over <anything> but the basis monomials.

```
> expand(couterP(q^2*p[0]+q*p[1]+p[2]));
  expand(couterH(q^2*h[3]+q*h[2,1]+h[1,1,1]));
  expand(couterE(a*e[3]+b*e[2,1]+c*e[1,1,1]));
```

$$q^2\,(p_0\,\&t\,p_0)+q\,(p_1\,\&t\,p_0)+q\,(p_0\,\&t\,p_1)+(p_2\,\&t\,p_0)+(p_0\,\&t\,p_2)$$

$$q^2\,(h_3\,\&t\,h_0)+q^2\,(h_2\,\&t\,h_1)+q^2\,(h_1\,\&t\,h_2)+q^2\,(h_0\,\&t\,h_3)+q\,(h_{2,1}\,\&t\,h_0)+q\,(h_2\,\&t\,h_1)+q\,(h_{1,1}\,\&t\,h_1)+q\,(h_1\,\&t\,h_{1,1})+q\,(h_1\,\&t\,h_2)+q\,(h_0\,\&t\,h_{2,1})+(h_{1,1,1}\,\&t\,h_0)+3\,(h_{1,1}\,\&t\,h_1)+3\,(h_1\,\&t\,h_{1,1})+(h_0\,\&t\,h_{1,1,1})$$

$$a\,(e_3\,\&t\,e_0)+a\,(e_2\,\&t\,e_1)+a\,(e_1\,\&t\,e_2)+a\,(e_0\,\&t\,e_3)+b\,(e_{2,1}\,\&t\,e_0)+b\,(e_2\,\&t\,e_1)+b\,(e_{1,1}\,\&t\,e_1)+b\,(e_1\,\&t\,e_{1,1})+b\,(e_1\,\&t\,e_2)+b\,(e_0\,\&t\,e_{2,1})+c\,(e_{1,1,1}\,\&t\,e_0)+3\,c\,(e_{1,1}\,\&t\,e_1)+3\,c\,(e_1\,\&t\,e_{1,1})+c\,(e_0\,\&t\,e_{1,1,1})$$

```
>
```

Algorithm used:

see description above.

See Also: SchurFkt[outerP], SchurFkt[outerH]

Last modified: June 19, 2008/BF/RA.

Function: SchurFkt[couterM] - the outer coproduct in the monomial symmetric function basis

Calling Sequence:

mtens := couterM(mfkt)

Parameters:

- mfkt : monomial symmetric function

Output:

- mtens : 2 tensor made out of monomial symmetric function monomials

WARNING:

--none--

Description:

- couterM implements the outer coproduct in the monomial symmetric function basis.
- Since the Hopf algebra of symmetric function is self dual, the outer coproduct of the monomial symmetric functions is obtained by dualizing the product on the dual basis. This dual basis is the *multiplicative* basis of complete symmetric functions. Therefore is the coproduct given by **unordered deconcatenation**.

  \Delta m_\mu = \Delta (a^1)^(\mu1)...(a^l)^(\mul)
  = \sum_(\mu´+\mu´´) (a^1)^(\mu1´) ... (a^1)^(\mul´) &t (a^1)^(\mu1´´)...(a^l)^(\mul´´)

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> couterM(m[2]);
  couterM(m[2,2]);  # note, that we do not have multiplicities!
  couterM(m[2,1]);
  couterM(2*m[3,1]+m[2,1]);
```

$$(m_2 \,\&t\, m_0) + (m_0 \,\&t\, m_2)$$

$$(m_{2,2} \,\&t\, m_0) + (m_2 \,\&t\, m_2) + (m_0 \,\&t\, m_{2,2})$$

$$(m_{2,1} \,\&t\, m_0) + (m_1 \,\&t\, m_2) + (m_2 \,\&t\, m_1) + (m_0 \,\&t\, m_{2,1})$$

$$2\,(m_{3,1} \,\&t\, m_0) + 2\,(m_1 \,\&t\, m_3) + 2\,(m_3 \,\&t\, m_1) + 2\,(m_0 \,\&t\, m_{3,1}) + (m_{2,1} \,\&t\, m_0) + (m_1 \,\&t\, m_2) + (m_2 \,\&t\, m_1) + (m_0 \,\&t\, m_{2,1})$$

Show that the coproduct outerM and the outer product outerM fulfill the compatibility axiom for Hopf algebras.

```
> f:=(a,b,c,d)->&t(outerM(a,c),outerM(b,d)):
  &t(couterM(m[1]),couterM(m[2]));
  eval(subs(`&t`=f,%));
  couterM(outerM(m[1],m[2]));
```

$$\&t(m_1, m_0, m_2, m_0) + \&t(m_1, m_0, m_0, m_2) + \&t(m_0, m_1, m_2, m_0) + \&t(m_0, m_1, m_0, m_2)$$

$$(m_3 \,\&t\, m_0) + (m_{2,1} \,\&t\, m_0) + (m_1 \,\&t\, m_2) + (m_2 \,\&t\, m_1) + (m_0 \,\&t\, m_3) + (m_0 \,\&t\, m_{2,1})$$

$$(m_3 \,\&t\, m_0) + (m_{2,1} \,\&t\, m_0) + (m_1 \,\&t\, m_2) + (m_2 \,\&t\, m_1) + (m_0 \,\&t\, m_3) + (m_0 \,\&t\, m_{2,1})$$

Note: The pair of a product and coproduct (concatM,couterM) **also** fulfill the compatibility axiom of a Hopf algebra and can be shown to be a graded commutative cocommutative Hopf algebra too!

```
> f:=(a,b,c,d)->&t(concatM(a,c),concatM(b,d)):
  &t(couterM(m[1]),couterM(m[2]));
  eval(subs(`&t`=f,%));
  couterM(concatM(m[1],m[2]));
```

$$\&t(m_1, m_0, m_2, m_0) + \&t(m_1, m_0, m_0, m_2) + \&t(m_0, m_1, m_2, m_0) + \&t(m_0, m_1, m_0, m_2)$$

$$(m_{2,1} \,\&t\, m_0) + (m_1 \,\&t\, m_2) + (m_2 \,\&t\, m_1) + (m_0 \,\&t\, m_{2,1})$$

$$(m_{2,1} \,\&t\, m_0) + (m_1 \,\&t\, m_2) + (m_2 \,\&t\, m_1) + (m_0 \,\&t\, m_{2,1})$$

Note: The pair of a product and coproduct (LaplaceM,couterM) **also** fulfill the compatibility axiom but **do not** form a Hopf but only a bialgebra! The two Hopf algebras displayed above are related via a deformation by a bialgebra given by this pair!

```
> f:=(a,b,c,d)->&t(LaplaceM(a,c),LaplaceM(b,d)):
  &t(couterM(m[1]),couterM(m[2]));
  eval(subs(`&t`=f,%));
  couterM(LaplaceM(m[1],m[2]));
```

$$\&t(m_1, m_0, m_2, m_0) + \&t(m_1, m_0, m_0, m_2) + \&t(m_0, m_1, m_2, m_0) + \&t(m_0, m_1, m_0, m_2)$$

$$(m_3 \,\&t\, m_0) + (m_0 \,\&t\, m_3)$$

$$(m_3 \,\&t\, m_0) + (m_0 \,\&t\, m_3)$$

Test cases:

```
> concatM(0);     # returns zero
  concatM(m[0]); # special case since m[0] is te image of the
  counit
```

$$0$$

$$m_0$$

```
>
```

Algorithm used:

A straight forward implementation of the split of the partition into two pieces in the multiplicative representations.

See Also: SchurFkt[outerM], SchurFkt[concatM], SchurFkt[LaplaceM]

Last modified: June 19, 2008/BF/RA.

Function: SchurFkt[couterON] - outer coproduct for the O(n) groups in the stable limit N-> infinity

Calling Sequence:

sftk := outerON(sfkt1,sfkt2,...)

Parameters:

- sfkti : one or more orthogonal Schur functions

Output:

- sfkt : orthogonal Schur function

WARNING:

Note: SchurFkt does not distinguish bases for different groups. Hence a function s[3,2,2] may be regarded as a GL(N) character *or* as an orthogonal group character. The user is responsible for keeping track of this fact!
Note: The procedure outerON acts on characters in the stable limit, that is on formally infinite dimensional groups. This is to avoid syzygies between the group characters, which occur in low dimensions and require so-called modification rules to be applied.

Description:

- If we regard the Schur functions os_\lambda [in Littlewood notation this reads as [\lambda] in SchurFkt it is denoted as s[l1,l2,...]] as irreducible characters of the orthogonal group, we can ask the question what the Glebsch-Gordan decomposition for this character may be. This problem was solved by Newell and Littlewood for the orthogonal and symplectic case:

  (1) [\mu] x [\nu] = \sum_\zeta [\mu/\zeta . \nu/\zeta]

  where the . indicates the outer product of the GL representation ring and the sum is over all partitions.

- outerON can be obtained using the command branch .

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> outer(s[1],s[1]);
  outerON(s[1],s[1]); # additional s[0] appears
```

$$s_2 + s_{1,1}$$

$$s_2 + s_{1,1} + s_0$$

outerON is still symmetric

```
> outerON(s[2],s[2]+s[1,1]);
  outerON(s[2]+s[1,1],s[2]);
```

$$s_4 + 2\,s_{3,1} + s_{2,2} + 2\,s_2 + 2\,s_{1,1} + s_0 + s_{2,1,1}$$

$$s_4 + 2\,s_{3,1} + s_{2,2} + 2\,s_2 + 2\,s_{1,1} + s_0 + s_{2,1,1}$$

outerON is associative

```
> outerON(s[2],outerON(s[1],s[2,1]));
  outerON(outerON(s[2],s[1]),s[2,1]);
```

$$4\,s_2 + 4\,s_{1,1} + s_0 + 2\,s_4 + 6\,s_{3,1} + 4\,s_{2,2} + 5\,s_{2,1,1} + s_{5,1} + 2\,s_{4,2} + 2\,s_{4,1,1} + s_{3,3} + 3\,s_{3,2,1} + s_{1,1,1,1} + s_{2,2,2} + s_{3,1,1,1} + s_{2,2,1,1}$$

$$4\,s_2 + 4\,s_{1,1} + s_0 + 2\,s_4 + 6\,s_{3,1} + 4\,s_{2,2} + 5\,s_{2,1,1} + s_{5,1} + 2\,s_{4,2} + 2\,s_{4,1,1} + s_{3,3} + 3\,s_{3,2,1} + s_{1,1,1,1} + s_{2,2,2} + s_{3,1,1,1} + s_{2,2,1,1}$$

outerON can act on many slots since its associative.

```
> outer(s[1],s[1],s[1]);
  outerON(s[1],s[1],s[1]);
```

$$s_3 + 2\,s_{2,1} + s_{1,1,1}$$

$$s_3 + 2\,s_{2,1} + 3\,s_1 + s_{1,1,1}$$

Some special and critical cases:

```
> outerON(0);
  outerON(s[0]);
```

$$0$$

$$s_0$$

```
>
```

Algorithm used:

outerON is implemented along the formula (1) displayed above. There exists literature on domino tableaux which give direct combinatorial access to the multiplication coefficients which could be faster.

See Also: SchurFkt[outer], SchurFkt[couter], SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function:

SchurFkt[couterE] - the outer coproduct in the elementary symmetric function basis
SchurFkt[couterH] - the outer coproduct in the complete symmetric function basis
SchurFkt[couterP] - the outer coproduct in the power sum basis

Calling Sequence:

mtens := couterE(efkt)
mtens := couterH(hfkt)
mtens := couterP(pfkt)

Parameters:

- efkt : elementary symmetric function
  hfkt : complete symmetric function
  pfkt : power sum symmetric function

Output:

- mtens : 2 tensor made out of [elementary|complete|power sum] symmetric function monomials (input and output type coincide)

WARNING:

--none--

Description:

- couterE, couterH and couterP describe the **outer coproduct** in the elementary, complete and power sum basis.
- The procedures for couterE and couterH are the same. The outer coproduct of these bases is derived from the following facts:
  - The outer coproduct is an algebra homomorphism of the outer product algebra
    \Delta(x.y) = \Delta(x).\Delta(y)
  - The outer coproduct in the elementary and complete basis for *one part partitions* e[n] or h[n] are obtained from
    \Delta(e[n]) = \sum_{r=0}^n e[n-r] \otimes e[r]
    \Delta(h[n]) = \sum_{r=0}^n h[n-r] \otimes h[r]
    hence the one part functions in the e- and h-basis obey a coproduct of a **divided powers** algebra.
- The power sum coproduct is also reduced to one part power sum via the homomorphism property, however, the power sums enjoy a binomial coproduct rule steming from dualizing a polynomial algebra (recall that the power sum symmetric functions are orthogonal, but not normalized and hence self dual). One gets:
  \Delta(p[n]) = p[n]\stemming1 + 1 \otimes p[n]
  and the one part power sum functions are the **primitive elements** of the outer coalgebra of symmetric functions. This can be written as

p[n](x,y) = p[n](x)+p[n](y) and due to the fact that p[n](x) =\sum_{i=1}^N x_i^n one gets the binomial relation.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
```

```
> couterE(e[2]);
  couterE(e[2,2]);
  couterE(e[2,1]);
  couterE(2*e[3,1]+e[2,1]);
  "----";
  couterH(h[2]);
  couterH(h[2,2]);
  couterH(h[2,1]);
  couterH(2*h[3,1]+h[2,1]);
```

$$(e_2 \,\&t\, e_0) + (e_1 \,\&t\, e_1) + (e_0 \,\&t\, e_2)$$

$$(e_{2,2} \,\&t\, e_0) + 2\,(e_{2,1} \,\&t\, e_1) + 2\,(e_2 \,\&t\, e_2) + (e_{1,1} \,\&t\, e_{1,1}) + 2\,(e_1 \,\&t\, e_{2,1}) + (e_0 \,\&t\, e_{2,2})$$

$$(e_{2,1} \,\&t\, e_0) + (e_2 \,\&t\, e_1) + (e_{1,1} \,\&t\, e_1) + (e_1 \,\&t\, e_{1,1}) + (e_1 \,\&t\, e_2) + (e_0 \,\&t\, e_{2,1})$$

$$2\,(e_{3,1} \,\&t\, e_0) + 2\,(e_3 \,\&t\, e_1) + 2\,(e_{2,1} \,\&t\, e_1) + 2\,(e_2 \,\&t\, e_{1,1}) + 2\,(e_{1,1} \,\&t\, e_2) + 2\,(e_1 \,\&t\, e_{2,1}) + 2\,(e_1 \,\&t\, e_3) + 2\,(e_0 \,\&t\, e_{3,1}) + (e_{2,1} \,\&t\, e_0) + (e_2 \,\&t\, e_1) + (e_{1,1} \,\&t\, e_1) + (e_1 \,\&t\, e_{1,1}) + (e_1 \,\&t\, e_2) + (e_0 \,\&t\, e_{2,1})$$

"----"

$$(h_2 \,\&t\, h_0) + (h_1 \,\&t\, h_1) + (h_0 \,\&t\, h_2)$$

$$(h_{2,2} \,\&t\, h_0) + 2\,(h_{2,1} \,\&t\, h_1) + 2\,(h_2 \,\&t\, h_2) + (h_{1,1} \,\&t\, h_{1,1}) + 2\,(h_1 \,\&t\, h_{2,1}) + (h_0 \,\&t\, h_{2,2})$$

$$(h_{2,1} \,\&t\, h_0) + (h_2 \,\&t\, h_1) + (h_{1,1} \,\&t\, h_1) + (h_1 \,\&t\, h_{1,1}) + (h_1 \,\&t\, h_2) + (h_0 \,\&t\, h_{2,1})$$

$$2\,(h_{3,1} \,\&t\, h_0) + 2\,(h_3 \,\&t\, h_1) + 2\,(h_{2,1} \,\&t\, h_1) + 2\,(h_2 \,\&t\, h_{1,1}) + 2\,(h_{1,1} \,\&t\, h_2) + 2\,(h_1 \,\&t\, h_{2,1}) + 2\,(h_1 \,\&t\, h_3) + 2\,(h_0 \,\&t\, h_{3,1}) + (h_{2,1} \,\&t\, h_0) + (h_2 \,\&t\, h_1) + (h_{1,1} \,\&t\, h_1) + (h_1 \,\&t\, h_{1,1}) + (h_1 \,\&t\, h_2) + (h_0 \,\&t\, h_{2,1})$$

Hence the two coproducts behave the same way. However we find:

```
> couterP(p[2]);
  couterP(p[2,2]); # (x+y)^2= x^2 + 2*x*y + y^2
  couterP(p[2,1]);
  couterP(2*p[3,1]+p[2,1]);
```

$$(p_2 \,\&t\, p_0) + (p_0 \,\&t\, p_2)$$

$$(p_{2,2} \,\&t\, p_0) + 2\,(p_2 \,\&t\, p_2) + (p_0 \,\&t\, p_{2,2})$$

$$(p_{2,1} \,\&t\, p_0) + (p_2 \,\&t\, p_1) + (p_1 \,\&t\, p_2) + (p_0 \,\&t\, p_{2,1})$$

$$2\,(p_{3,1}\,\&t\,p_0)+2\,(p_3\,\&t\,p_1)+2\,(p_1\,\&t\,p_3)+2\,(p_0\,\&t\,p_{3,1})+(p_{2,1}\,\&t\,p_0)+(p_2\,\&t\,p_1)+(p_1\,\&t\,p_2)+(p_0\,\&t\,p_{2,1})$$

Show that the coproduct outerE and the outer product outerE fulfill the compatibility axiom for Hopf algebras

```
> f:=(a,b,c,d)->&t(outerE(a,c),outerE(b,d)):
  &t(couterE(e[1]),couterE(e[2])):
  RHS:=eval(subs(`&t`=f,%));
  LHS:=couterE(outerE(e[1],e[2]));
```

$$RHS := (e_{2,1}\,\&t\,e_0)+(e_2\,\&t\,e_1)+(e_{1,1}\,\&t\,e_1)+(e_1\,\&t\,e_{1,1})+(e_1\,\&t\,e_2)+(e_0\,\&t\,e_{2,1})$$

$$LHS := (e_{2,1}\,\&t\,e_0)+(e_2\,\&t\,e_1)+(e_{1,1}\,\&t\,e_1)+(e_1\,\&t\,e_{1,1})+(e_1\,\&t\,e_2)+(e_0\,\&t\,e_{2,1})$$

We proceed by showing a few critical cases:

```
> couterP(0),couterP(sin(x));
  couterE(0),couterE(1-q^2);
  couterH(0),couterH(a);
```

$$0,\ \sin(x)\,(p_0\,\&t\,p_0)$$

$$0,\ (1-q^2)\,(e_0\,\&t\,e_0)$$

$$0,\ a\,(h_0\,\&t\,h_0)$$

```
> couterP(p[0]);
  couterE(e[0]);
  couterH(h[0]);
```

$$p_0\,\&t\,p_0$$

$$e_0\,\&t\,e_0$$

$$h_0\,\&t\,h_0$$

we show that the coproduct is a linear map over <anything> but the basis monomials.

```
> expand(couterP(q^2*p[0]+q*p[1]+p[2]));
  expand(couterH(q^2*h[3]+q*h[2,1]+h[1,1,1]));
  expand(couterE(a*e[3]+b*e[2,1]+c*e[1,1,1]));
```

$$q^2\,(p_0\,\&t\,p_0)+q\,(p_1\,\&t\,p_0)+q\,(p_0\,\&t\,p_1)+(p_2\,\&t\,p_0)+(p_0\,\&t\,p_2)$$

$$q^2\,(h_3\,\&t\,h_0)+q^2\,(h_2\,\&t\,h_1)+q^2\,(h_1\,\&t\,h_2)+q^2\,(h_0\,\&t\,h_3)+q\,(h_{2,1}\,\&t\,h_0)+q\,(h_2\,\&t\,h_1)+q\,(h_{1,1}\,\&t\,h_1)+q\,(h_1\,\&t\,h_{1,1})+q\,(h_1\,\&t\,h_2)+q\,(h_0\,\&t\,h_{2,1})+(h_{1,1,1}\,\&t\,h_0)+3\,(h_{1,1}\,\&t\,h_1)+3\,(h_1\,\&t\,h_{1,1})+(h_0\,\&t\,h_{1,1,1})$$

$$a\,(e_3\,\&t\,e_0)+a\,(e_2\,\&t\,e_1)+a\,(e_1\,\&t\,e_2)+a\,(e_0\,\&t\,e_3)+b\,(e_{2,1}\,\&t\,e_0)+b\,(e_2\,\&t\,e_1)+b\,(e_{1,1}\,\&t\,e_1)+b\,(e_1\,\&t\,e_{1,1})+b\,(e_1\,\&t\,e_2)+b\,(e_0\,\&t\,e_{2,1})+c\,(e_{1,1,1}\,\&t\,e_0)+3\,c\,(e_{1,1}\,\&t\,e_1)+3\,c\,(e_1\,\&t\,e_{1,1})+c\,(e_0\,\&t\,e_{1,1,1})$$

```
>
```

Algorithm used:

see description above.

See Also: SchurFkt[outerP], SchurFkt[outerH]

Last modified: June 19, 2008/BF/RA.

Function: SchurFkt[cplethP] - plethysm coproduct in the power sums basis [do not use!!]

Calling Sequence:

ptens := cplethP(pfkt)

Parameters:

- pfkt : power sum symmetric function

Output:

- ptens : a 2 tensor polynom of power sum symmetric monoms.

WARNING:

Do not use!

Description:

- cplethP is intended to compute the *plethystic coproduct* of a power sum symmetric function.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> cplethP(p[1,1]);
  cplethP(2*p[1,1]);
  cplethP(outerP(p[1],p[1]));
  cplethP(p[1,1]+p[1,1]);
```

$$(p_1 \,\&t\, p_{1,1}) + (p_{1,1} \,\&t\, p_1)$$

$$2\,(p_1 \,\&t\, p_{1,1}) + 2\,(p_{1,1} \,\&t\, p_1)$$

$$(p_1 \,\&t\, p_{1,1}) + (p_{1,1} \,\&t\, p_1)$$

$$2\,(p_1 \,\&t\, p_{1,1}) + 2\,(p_{1,1} \,\&t\, p_1)$$

```
> cplethP(p[3,2]);
  cplethP(p[4]);
  cplethP(p[8]);
  cplethP(p[4,4]);
```

$$(p_1 \,\&t\, p_{3,2}) + (p_{3,2} \,\&t\, p_1)$$

$$(p_1 \,\&t\, p_4) + (p_2 \,\&t\, p_2) + (p_4 \,\&t\, p_1)$$

$$(p_1 \,\&t\, p_8) + (p_2 \,\&t\, p_4) + (p_4 \,\&t\, p_2) + (p_8 \,\&t\, p_1)$$

$$(p_1 \,\&t\, p_{4,4}) + 4\,(p_{1,1} \,\&t\, p_4) + 2\,(p_2 \,\&t\, p_{2,2}) + 2\,(p_{2,2} \,\&t\, p_2) + 4\,(p_4 \,\&t\, p_{1,1}) + (p_{4,4} \,\&t\, p_1)$$

```
> cplethP(p[4]);
```

```
eval(subs(`&t`=plethP,%));
```

$$(p_1 \,\&t\, p_4) + (p_2 \,\&t\, p_2) + (p_4 \,\&t\, p_1)$$

$$3\, p_4$$

```
>
```

Algorithm used:

The function cplethP is obtained from dualizing the function plethP via the Schur-Hall scalar product:

cplethP(x) := \sum_a,b < x, pleth(a,b)> &t(a,b)

See Also: SchurFkt[plethP], SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[cplethS] - plethysm coproduct in the Schur function basis

Calling Sequence:

ptens := cplethS(sfkt)

Parameters:

- sfkt : Schur function polynomial

Output:

- ptens : a 2 tensor polynom of Schur function monoms.

WARNING:

The used algorithm is by fare not the fastest available, don't expect good performance and be patient!

Description:

- cplethS is intended to compute the *plethystic coproduct* of a Schur function polynom.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> cplethS(s[1,1]);
  cplethS(s[3]);
  cplethS(s[4]);
  cplethS(s[5]);
```

$$(s_1 \,\&t\, s_{1,1}) + (s_{1,1} \,\&t\, s_1)$$

$$(s_1 \,\&t\, s_3) + (s_3 \,\&t\, s_1)$$

$$(s_1 \,\&t\, s_4) + (s_2 \,\&t\, s_2) + (s_4 \,\&t\, s_1)$$

$$(s_1 \,\&t\, s_5) + (s_5 \,\&t\, s_1)$$

Note that this is a non-cocommutative operation

```
> f:=(x,y)->&t(y,x); # switch
  out1:=cplethS(s[2,2]);
  out2:=eval(subs(`&t`=f,out1));
  is(out1=out2);
  out1-out2;
  "-------------";
  out1:=cplethS(s[4,4]);
  out2:=eval(subs(`&t`=f,out1));
  is(out1=out2);
```

```
out1-out2;
"-------------";
out1:=cplethS(s[4]);  ## IS symmetric!
out2:=eval(subs(`&t`=f,out1));
is(out1=out2);        ## true
out1-out2;
```

$$f := (x, y) \rightarrow y \,\&t\, x$$

$$out1 := (s_1 \,\&t\, s_{2,2}) + (s_2 \,\&t\, s_2) + (s_2 \,\&t\, s_{1,1}) + (s_{2,2} \,\&t\, s_1)$$

$$out2 := (s_{2,2} \,\&t\, s_1) + (s_2 \,\&t\, s_2) + (s_{1,1} \,\&t\, s_2) + (s_1 \,\&t\, s_{2,2})$$

$$false$$

$$-(s_{1,1} \,\&t\, s_2) + (s_2 \,\&t\, s_{1,1})$$

"-------------"

$$out1 := (s_1 \,\&t\, s_{4,4}) + (s_2 \,\&t\, s_4) + (s_2 \,\&t\, s_{3,1}) + (s_2 \,\&t\, s_{2,2}) + (s_4 \,\&t\, s_2) + (s_4 \,\&t\, s_{1,1}) + (s_{2,2} \,\&t\, s_2) + (s_{4,4} \,\&t\, s_1)$$

$$out2 := (s_{4,4} \,\&t\, s_1) + (s_4 \,\&t\, s_2) + (s_{3,1} \,\&t\, s_2) + (s_{2,2} \,\&t\, s_2) + (s_2 \,\&t\, s_4) + (s_{1,1} \,\&t\, s_4) + (s_2 \,\&t\, s_{2,2}) + (s_1 \,\&t\, s_{4,4})$$

$$false$$

$$(s_2 \,\&t\, s_{3,1}) + (s_4 \,\&t\, s_{1,1}) - (s_{3,1} \,\&t\, s_2) - (s_{1,1} \,\&t\, s_4)$$

"-------------"

$$out1 := (s_1 \,\&t\, s_4) + (s_2 \,\&t\, s_2) + (s_4 \,\&t\, s_1)$$

$$out2 := (s_1 \,\&t\, s_4) + (s_2 \,\&t\, s_2) + (s_4 \,\&t\, s_1)$$

$$true$$

$$0$$

>

Algorithm used:

The plethysm coproduct for S-functuions is obtained by dualizing the plethysm via the Schur-Hall scalar product:

cplethS(x) := \sum_{a,b} < x, plethS(a,b) &t(a,b)

See Also: SchurFkt[plethP], SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[dimSN] - computes the dimension of an sfkt polynom seen as S_n character

Calling Sequence:

n := dimSN(sfktpoly)

Parameters:

- sfktpoly : Schur function polynom

Output:

- n : integer

WARNING:

--not yet well tested--

Description:

- Schur functions have an evaluation, which describes the dimension of S_n modules described be these characters.
- We use the classical hook-rule for the characters, that is:

  dimSN(s[lambda]) = n! / \sum_{hooks at (i,j)} h_ij

  where h_ij is the hook-length, the number of boxes below and to the right of the box (i,j) in \lambda plus 1 (in British notation of Young diagrams).

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> seq(dimSN(s[n]),n=1..10); ## all one dime modules
```

$$1, 1, 1, 1, 1, 1, 1, 1, 1, 1$$

```
> seq(dimSN(s[1$n]),n=1..10); ## all one dime modules
```

$$1, 1, 1, 1, 1, 1, 1, 1, 1, 1$$

```
> dimSN(s[2,1]) = 3!/(3*1*1);
  dimSN(s[3,1,1]) = 5!/(5*2*2*1*1);
  dimSN(s[3,3,3]) = 9!/(5*4*4*3*3*3*2*2*1);
```

$$2 = 2$$

$$6 = 6$$

$$42 = 42$$

The function dimSN is linear in its argument:

```
> dimSN(a*s[3]+b*s[1,1,1]);
```

```
for n from 1 to 10 do
 dimSN(`+`(op(map((x)->s[op(x)],PartNM(n,n)))));
end do;
```

$$a+b$$
$$1$$
$$2$$
$$4$$
$$10$$
$$26$$
$$76$$
$$232$$
$$764$$
$$2620$$
$$9496$$

This series is Sloane integer database A000085, it counts the dimesnions of S_n representations of homogenous degree, and also the number of (standard) Young-tableau with n-cells as also the number of self inverse permutations with n-letters, etc...

Note that dimSN always used the appropriate S_n, so if an input has partitions of different weight, different S_n reps are used!! This is **different** from SCHUR!

```
> dimSN(s[1]+s[2,2]+s[3,3,3]+s[4,4,4,4]);
```

$$24069$$

```
> plethS(s[4],s[4]);
  dimSN(%);
```

$$s_{9,4,3}+s_{7,4,4,1}+s_{4,4,4,4}+s_{7,6,3}+s_{6,4,4,2}+s_{16}+2\,s_{12,4}+s_{11,3,2}+s_{10,5,1}+2\,s_{10,4,2}+s_{10,2,2,2}+s_{9,4,2,1}+s_{8,8}+s_{6,6,2,2}+s_{11,4,1}+s_{7,7,1,1}+2\,s_{8,4,4}+s_{7,5,3,1}+2\,s_{10,6}+s_{9,6,1}+s_{6,6,4}+s_{12,2,2}+2\,s_{8,6,2}+s_{9,5,2}+s_{8,4,2,2}+s_{8,5,2,1}+s_{14,2}+s_{13,3}$$

$$2627625$$

```
> # tested against SCHUR
  dimSN(s[6]),dimSN(s[5,1]),dimSN(s[4,2]),dimSN(s[4,1,1]),dimSN(s
  [3,3]),
  dimSN(s[3,2,1]),dimSN(s[3,1,1,1]),dimSN(s[2,2,2]),dimSN(s[2,2,1
  ,1]),
  dimSN(s[2,1,1,1,1]),dimSN(s[1,1,1,1,1,1]);
```

$$1, 5, 9, 10, 5, 16, 10, 5, 9, 5, 1$$

```
>
```

Algorithm used:

The algorithm used is based on the hook rule. We compute the conjugate partition, and use that

with (i,j) in lambda, one has (j,i) in the conjugated partition lambda'. The hook length at (i,j) is then given as: h_ij = \lambda[i]-j + \lambda'[j]-i +1 = armlength(i,j) + leglength(i,j) +1. Note that the leg-length at (i,j) is the arm-length of the conjugate partition at (j,i), the 1 counts the box in the corder. The dimesnion (hook formula) is then given as

dimSN(\lambda) = |\lambda|! / (Sum_{(i,j) box in lambda} h_ij

See Also: SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[dummy] - <description>

Calling Sequence:

lst := dummy(N,M)

Parameters:

- N,M : <input type>

Output:

- lst : <output type>

WARNING:

iff applicable

Description:

- Bullet item list of properties of the function

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
>
```

```
>
```

mandatory test cases and most likely cases a user want to type in

Algorithm used:

The presently implemented algorithm, if possible with certificate.

See Also: SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[FLAT] - flattens the function T() used by SchurFkt[MLIN] (hence T() is made associative this way).

Calling Sequence:

out := FLAT(foo)

Parameters:

- <foo> an expression containing nested (unnested or none) calls of the operator T().

Output:

- out : the expression <foo> where the internal nested appearances of T() have been deleted and the arguments passed over to the outer most T(args) call.

WARNING:

This is an experimental function for internal use mainly. It is **not** a general device to implement associativity! FLAT calls internally MLIN and hence is affected by multilinearity and refers to a ground field (here integers).

Description:

- FLAT deletes nested calls of the operator T() passing arguments to its outer most call. Flat calls MLIN and hence is affected by multilinearity. It is only a helper function to get a quick and dirty multilinearity for some functions over the integers.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

Just a few examples:

```
> FLAT(T(a,T(b,c)));
  FLAT(T(T(a,b),c));
```

$$\mathrm{T}(a, b, c)$$

$$\mathrm{T}(a, b, c)$$

Some more complicated cases showing the internal call of MLIN:

```
> FLAT(T(a+b,T(c+d)));
  FLAT(T(2*a,3*T(b,c)));
```

$$\mathrm{T}(a, c) + \mathrm{T}(a, d) + \mathrm{T}(b, c) + \mathrm{T}(b, d)$$

$$6\,\mathrm{T}(a, b, c)$$

The second case shows that FLAT does implements multilinearity over the integers using MLIN....

```
>
```

Algorithm

Note: FLAT and MLIN are hacks and not serious code. Don't use this code it will most likely be replaced by a better device, for example, a tensor product using the (patched) define(`&t`,...) facility of BIGEBRA.

If FLAT sees an expression of type `+` (sum) then
    it maps MLIN onto the summands and drops internal occurrences of T().
elif FLAT sees an expression of type `*` (product) then
    it maps MLIN onto the term and drops internal occurrences of T() keeping the coefficient
else
    FLAT applies MLIN to the argument dropping thereby internal occurrences of T()

>

See Also: SchurFkt[MLIN], Bigebra[define]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[Frob2part] - converts a partition in Frobenius notation into a standard list notation of partitions

Calling Sequence:

prt := Frob2part(fpart)

Parameters:

- fpart : a partition in Frobenius notation (list lists of integers)

Output:

- prt : partition ins standard list notation

WARNING:

internal use mainly

Description:

- A partition is represented in SchurFkt usually in a list of integer (shape) notation (e.g. [3,2,2,1]) which describes the rowlength of the associated Young diagram (Ferrers diagram).
- The Frobenius notation for partitions is given by a list of pairs of integers, where the pair of integers describes the 'arm length' and 'leg length' of a partition seen as a Young diagram. The box on the diagram does not count, so the Frobenius partition [[0],[0]] is a one box partition [1] in standard notation. The number of boxes on the diagonal of the Young diagram is called the (Frobenius) rank of the partition.
- This function is for internal use. The policy of the SchurFkt package is that partitions are described in standard notation at the level where the user interacts with the package.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> Frob2part([[0],[0]]);
```

$$[1]$$

Trivial, illegal and special cases

```
> Frob2part([[],[]]);       ## OK
  Frob2part([[-1],[2]]);    ## illegal, no negative entries allowed => UNCONTROLLED OUTPUT
  Frob2part([[1,1],[0]]);   ## ERROR: the pair of lists needs to be of the same length
  Frob2part([[1,1],[2,0]]);## illegal, produces not a partition (non decreasing list of integers)
```

```
## ==> UNCONTROLLED OUTPUT
```

$$[0]$$

$$[0, 1, 1]$$

```
Error, (in SchurFkt:-Frob2part) invalid subscript selector
```

$$[2, 3, 1]$$

Standard cases:

```
> Frob2part([[2,1,1],[1,1,0]]);
```

$$[3, 3, 3]$$

The inverse function is called SchurFkt[part2Frob]:

```
> part2Frob([3,3,1,1]);
  Frob2part(%);
  part2Frob([1,1,1,1]);
  Frob2part(%);
  part2Frob([4,3,3,2,1]);
  Frob2part(%);
```

$$[[2, 1], [3, 0]]$$

$$[3, 3, 1, 1]$$

$$[[0], [3]]$$

$$[1, 1, 1, 1]$$

$$[[3, 1, 0], [4, 2, 0]]$$

$$[4, 3, 3, 2, 1]$$

```
>
```

Algorithm used:

The Frobenius representation of a partition is obtained by inspection of the standard list form of the partitions.

See Also: SchurFkt[part2Frob]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[GesselThetaS] - computes the Gessel map Theta for Schur functions
SchurFkt[GesselThetaP] - computes the Gessel map Theta for power sum symmetric functions

Calling Sequence:

zpoly := GesselThetaS(sfktpoly,var)
zpoly := GesselThetaP(pfktpoly,var)

Parameters:

- sfktpoly,var : Schur function polynomial, var a name for a variable
  pfktpoly,var : power sum polynomial, var a name for a variable

Output:

- zpoly : an polynomial expression in teh variable var (usually taken as 'z')

WARNING:

-- none --

Description:

- The Gessel map Theta is used in counting problems in Young's lattice, for graphs, etc. It is defined as follows.
  Let U = u1+u2+u3+... be the alphabet of the symmetric function ring /\. This ring is hence an polynomial ring infinitly many (finitely many) variables ui. The Gessel map is a map from this ring into a polynomial ring in one variable. One has:

  i) Theta(1)=1
  ii) Theta(u1u2...un) = z^n/n!

  this yields the two cases:

- GesselThetaS:

  Theta(s_\lambda(u)) = f^\lambda z^|\lambda|/|\lambda|!

  where f^\lambda = dimSN(s[\lambda]) is the number of standard Young tableau of shape \lambda, that is teh dimension of s[\lambda] in S_|\lambda«, and |\lambda| is the weight of the partition.

- GesselThetaP:

  Theta(p_n(u)) = z if n=1 else 0

Examples:

```
> restart:with(SchurFkt):
```

```
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> GesselThetaS(1,z);
  GesselThetaS(s[0],z);
  GesselThetaS(s[2,2],z),dimSN(s[2,2])/factorial(4)*z^4;
```

$$1$$

$$1$$

$$\frac{z^4}{12}, \frac{z^4}{12}$$

The power sum version:

```
> GesselThetaP(1,z);
  GesselThetaP(p[0],z);
  GesselThetaP(p[1,1],z),z^2;
```

$$1$$

$$1$$

$$z^2, z^2$$

For the series L,M as also for A,B and C,D one knows the result of the Theta map gives exponential generating series:

```
> Mser:=getSfktSeries(M,10,1);
  GesselThetaS(Mser,z);
  convert(series(exp(z),z=0,11),polynom);
```

$$Mser := s_0 + s_1 + s_2 + s_3 + s_4 + s_5 + s_6 + s_7 + s_8 + s_9 + s_{10}$$

$$1 + z + \frac{1}{2}z^2 + \frac{1}{6}z^3 + \frac{1}{24}z^4 + \frac{1}{120}z^5 + \frac{1}{720}z^6 + \frac{1}{5040}z^7 + \frac{1}{40320}z^8 + \frac{1}{362880}z^9 + \frac{1}{3628800}z^{10}$$

$$1 + z + \frac{1}{2}z^2 + \frac{1}{6}z^3 + \frac{1}{24}z^4 + \frac{1}{120}z^5 + \frac{1}{720}z^6 + \frac{1}{5040}z^7 + \frac{1}{40320}z^8 + \frac{1}{362880}z^9 + \frac{1}{3628800}z^{10}$$

```
> Lser:=getSfktSeries(L,10,1);
  GesselThetaS(Lser,z);
  convert(series(exp(z),z=0,11),polynom);
```

$$Lser := s_0 - s_1 + s_{1,1} - s_{1,1,1} + s_{1,1,1,1} - s_{1,1,1,1,1} + s_{1,1,1,1,1,1} - s_{1,1,1,1,1,1,1} + s_{1,1,1,1,1,1,1,1} - s_{1,1,1,1,1,1,1,1,1} + s_{1,1,1,1,1,1,1,1,1,1}$$

$$1 - z + \frac{1}{2}z^2 - \frac{1}{6}z^3 + \frac{1}{24}z^4 - \frac{1}{120}z^5 + \frac{1}{720}z^6 - \frac{1}{5040}z^7 + \frac{1}{40320}z^8 - \frac{1}{362880}z^9 + \frac{1}{3628800}z^{10}$$

$$1 + z + \frac{1}{2}z^2 + \frac{1}{6}z^3 + \frac{1}{24}z^4 + \frac{1}{120}z^5 + \frac{1}{720}z^6 + \frac{1}{5040}z^7 + \frac{1}{40320}z^8 + \frac{1}{362880}z^9 + \frac{1}{3628800}z^{10}$$

```
> Aser:=getSfktSeries(A,10,1);
  GesselThetaS(Aser,z);
  convert(series(exp(z^2/2),z=0,11),polynom);
```

$$Aser := s_0 - s_{1,1} + s_{2,1,1} - s_{3,1,1,1} - s_{2,2,2} + s_{4,1,1,1,1} + s_{3,2,2,1} - s_{5,1,1,1,1,1} - s_{4,2,2,1,1} - s_{3,3,2,2}$$

$$1 - \frac{1}{2}z^2 + \frac{1}{8}z^4 - \frac{1}{48}z^6 + \frac{1}{384}z^8 - \frac{1}{3840}z^{10}$$

$$1 + \frac{1}{2}z^2 + \frac{1}{8}z^4 + \frac{1}{48}z^6 + \frac{1}{384}z^8 + \frac{1}{3840}z^{10}$$

```
> Bser:=getSfktSeries(B,10,1);
  GesselThetaS(Bser,z);
  convert(series(exp(z^2/2),z=0,11),polynom);
```

$$Bser := s_0 + s_{1,1} + s_{1,1,1,1} + s_{2,2} + s_{1,1,1,1,1,1} + s_{2,2,1,1} + s_{3,3} + s_{1,1,1,1,1,1,1,1} + s_{2,2,1,1,1,1} + s_{2,2,2,2} + s_{3,3,1,1} + s_{4,4} + s_{1,1,1,1,1,1,1,1,1,1} + s_{2,2,1,1,1,1,1,1} + s_{2,2,2,2,1,1} + s_{3,3,1,1,1,1} + s_{3,3,2,2} + s_{4,4,1,1} + s_{5,5}$$

$$1 + \frac{1}{2}z^2 + \frac{1}{8}z^4 + \frac{1}{48}z^6 + \frac{1}{384}z^8 + \frac{1}{3840}z^{10}$$

$$1 + \frac{1}{2}z^2 + \frac{1}{8}z^4 + \frac{1}{48}z^6 + \frac{1}{384}z^8 + \frac{1}{3840}z^{10}$$

```
> Cser:=getSfktSeries(C,10,1);
  GesselThetaS(Cser,z);
  convert(series(exp(z^2/2),z=0,11),polynom);
```

$$Cser := s_0 - s_2 + s_{3,1} - s_{4,1,1} - s_{3,3} + s_{5,1,1,1} + s_{4,3,1} - s_{6,1,1,1,1} - s_{5,3,1,1} - s_{4,4,2}$$

$$1 - \frac{1}{2}z^2 + \frac{1}{8}z^4 - \frac{1}{48}z^6 + \frac{1}{384}z^8 - \frac{1}{3840}z^{10}$$

$$1 + \frac{1}{2}z^2 + \frac{1}{8}z^4 + \frac{1}{48}z^6 + \frac{1}{384}z^8 + \frac{1}{3840}z^{10}$$

```
> Dser:=getSfktSeries('D',10,1);
  GesselThetaS(Dser,z);
  convert(series(exp(z^2/2),z=0,11),polynom);
```

$$Dser := s_0 + s_2 + s_4 + s_{2,2} + s_6 + s_{4,2} + s_{2,2,2} + s_8 + s_{6,2} + s_{4,4} + s_{4,2,2} + s_{2,2,2,2} + s_{10} + s_{8,2} + s_{6,4} + s_{6,2,2} + s_{4,4,2} + s_{4,2,2,2} + s_{2,2,2,2,2}$$

$$1 + \frac{1}{2}z^2 + \frac{1}{8}z^4 + \frac{1}{48}z^6 + \frac{1}{384}z^8 + \frac{1}{3840}z^{10}$$

$$1 + \frac{1}{2}z^2 + \frac{1}{8}z^4 + \frac{1}{48}z^6 + \frac{1}{384}z^8 + \frac{1}{3840}z^{10}$$

```
> Fser:=getSfktSeries(F,10,1);
  GesselThetaS(Fser,z);
  convert(series(exp(z^2/2+z),z=0,11),polynom);
```

$$Fser := s_0 + s_{2,2} + s_1 + s_2 + s_3 + s_4 + s_5 + s_6 + s_7 + s_8 + s_9 + s_{10} + s_{1,1} + s_{1,1,1} + s_{1,1,1,1}$$

$$+ s_{1,1,1,1,1} + s_{1,1,1,1,1,1} + s_{1,1,1,1,1,1,1} + s_{1,1,1,1,1,1,1,1} + s_{1,1,1,1,1,1,1,1,1} + s_{1,1,1,1,1,1,1,1,1,1}$$
$$+ s_{2,1} + s_{3,1} + s_{2,1,1} + s_{4,1} + s_{3,2} + s_{3,1,1} + s_{2,2,1} + s_{2,1,1,1} + s_{5,1} + s_{4,2} + s_{4,1,1} + s_{3,3} + s_{3,2,1}$$
$$+ s_{3,1,1,1} + s_{2,2,2} + s_{2,2,1,1} + s_{2,1,1,1,1} + s_{6,1} + s_{5,2} + s_{5,1,1} + s_{4,3} + s_{4,2,1} + s_{4,1,1,1} + s_{3,3,1}$$
$$+ s_{3,2,2} + s_{3,2,1,1} + s_{3,1,1,1,1} + s_{2,2,2,1} + s_{2,2,1,1,1} + s_{2,1,1,1,1,1} + s_{7,1} + s_{6,2} + s_{6,1,1} + s_{5,3}$$
$$+ s_{5,2,1} + s_{5,1,1,1} + s_{4,4} + s_{4,3,1} + s_{4,2,2} + s_{4,2,1,1} + s_{4,1,1,1,1} + s_{3,3,2} + s_{3,3,1,1} + s_{3,2,2,1}$$
$$+ s_{3,2,1,1,1} + s_{3,1,1,1,1,1} + s_{2,2,2,2} + s_{2,2,2,1,1} + s_{2,2,1,1,1,1} + s_{2,1,1,1,1,1,1} + s_{8,1} + s_{7,2} + s_{7,1,1}$$
$$+ s_{6,3} + s_{6,2,1} + s_{6,1,1,1} + s_{5,4} + s_{5,3,1} + s_{5,2,2} + s_{5,2,1,1} + s_{5,1,1,1,1} + s_{4,4,1} + s_{4,3,2} + s_{4,3,1,1}$$
$$+ s_{4,2,2,1} + s_{4,2,1,1,1} + s_{4,1,1,1,1,1} + s_{3,3,3} + s_{3,3,2,1} + s_{3,3,1,1,1} + s_{3,2,2,2} + s_{3,2,2,1,1}$$
$$+ s_{3,2,1,1,1,1} + s_{3,1,1,1,1,1,1} + s_{2,2,2,2,1} + s_{2,2,2,1,1,1} + s_{2,2,1,1,1,1,1} + s_{2,1,1,1,1,1,1,1} + s_{9,1}$$
$$+ s_{8,2} + s_{8,1,1} + s_{7,3} + s_{7,2,1} + s_{7,1,1,1} + s_{6,4} + s_{6,3,1} + s_{6,2,2} + s_{6,2,1,1} + s_{6,1,1,1,1} + s_{5,5} + s_{5,4,1}$$
$$+ s_{5,3,2} + s_{5,3,1,1} + s_{5,2,2,1} + s_{5,2,1,1,1} + s_{5,1,1,1,1,1} + s_{4,4,2} + s_{4,4,1,1} + s_{4,3,3} + s_{4,3,2,1}$$
$$+ s_{4,3,1,1,1} + s_{4,2,2,2} + s_{4,2,2,1,1} + s_{4,2,1,1,1,1} + s_{4,1,1,1,1,1,1} + s_{3,3,3,1} + s_{3,3,2,2} + s_{3,3,2,1,1}$$
$$+ s_{3,3,1,1,1,1} + s_{3,2,2,2,1} + s_{3,2,2,1,1,1} + s_{3,2,1,1,1,1,1} + s_{3,1,1,1,1,1,1,1} + s_{2,2,2,2,2} + s_{2,2,2,2,1,1}$$
$$+ s_{2,2,2,1,1,1,1} + s_{2,2,1,1,1,1,1,1} + s_{2,1,1,1,1,1,1,1,1}$$

$$1 + z + \frac{5}{12} z^4 + z^2 + \frac{2}{3} z^3 + \frac{13}{60} z^5 + \frac{19}{180} z^6 + \frac{29}{630} z^7 + \frac{191}{10080} z^8 + \frac{131}{18144} z^9 + \frac{1187}{453600} z^{10}$$

$$1 + z + \frac{5}{12} z^4 + z^2 + \frac{2}{3} z^3 + \frac{13}{60} z^5 + \frac{19}{180} z^6 + \frac{29}{630} z^7 + \frac{191}{10080} z^8 + \frac{131}{18144} z^9 + \frac{1187}{453600} z^{10}$$

>

Algorithm used:

The Gessel map is exacly computed along the lines given in the description above.

See Also: SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[getSfktSeries] - produces a Schur function series (or a list of its coefficients)

Calling Sequence:

ser := getSfktSeries('name',ord,var)
lst := getSfktSeries('name',ord)
void := getSfktSeries('names')

Parameters:

- 'name[s]',ord,var : a name of a series, the order (weight of the last displayed term, =order), a formal variable for the series

Output:

- ser,lst,void : a series, a list of coefficient, void (a usage message)

WARNING:

--none--

Description:

- Schur functions series are formal series with Schur functions as its arguments. Such series are important in the branching process of groups, see SchurFkt[branch] .
- Schur function series come in pairs which are mutually inverse under convolution (point wise product).
- If the function is called with one argument 'names' it returns a list of known names.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
```

Display known series:

```
> getSfktSeries('names');
```

"Known Series are: A,B,C,D,E,F,L,M"

We display the known series and explain the coefficients (up to a sign):

```
> A(t)=getSfktSeries('A',6,t)+O(t^7);
  printf("Partitions of Frobenius form
  ((a1,a1+1)(a2,a2+1),...)");
  B(t)=getSfktSeries('B',6,t)+O(t^7);
  printf("Partitions with even column length (even row length of
  the conjugated partition (2*l_1,2*l_2,...)^' )");
  C(t)=getSfktSeries('C',6,t)+O(t^7);
  printf("Partitions of Frobenius form
```

```
((a1+1,a1)(a2+1,a2),...)");
D(t)=getSfktSeries('D',6,t)+O(t^7);
printf("Partitions with even row length (2*l_1,2*l_2,...)");
E(t)=getSfktSeries('E',6,t)+O(t^7);
printf("Self conjugate partitions \lambda=\lambda^' ");
F(t)=getSfktSeries('F',6,t)+O(t^7);
printf("All partitions");
L(t)=getSfktSeries('L',6,t)+O(t^7);
printf("One row partitions (one part elementary symmetric
functions)");
M(t)=getSfktSeries('M',6,t)+O(t^7);
printf("One row partitions (one part complete symmetric
functions)");
```

$$A(t) = s_0 - s_{1,1}\, t^2 + s_{2,1,1}\, t^4 - s_{3,1,1,1}\, t^6 - s_{2,2,2}\, t^6 + O(t^7)$$

```
Partitions of Frobenius form ((a1,a1+1)(a2,a2+1),...)
```

$$B(t) = s_0 + s_{1,1}\, t^2 + s_{1,1,1,1}\, t^4 + s_{2,2}\, t^4 + s_{1,1,1,1,1,1}\, t^6 + s_{2,2,1,1}\, t^6 + s_{3,3}\, t^6 + O(t^7)$$

```
Partitions with even column length (even row length of the conjugated partition
(2*l_1,2*l_2,...)^' )
```

$$C(t) = s_0 - s_2\, t^2 + s_{3,1}\, t^4 - s_{4,1,1}\, t^6 - s_{3,3}\, t^6 + O(t^7)$$

```
Partitions of Frobenius form ((a1+1,a1)(a2+1,a2),...)
```

$$D(t) = s_0 + s_2\, t^2 + s_4\, t^4 + s_{2,2}\, t^4 + s_6\, t^6 + s_{4,2}\, t^6 + s_{2,2,2}\, t^6 + O(t^7)$$

```
Partitions with even row length (2*l_1,2*l_2,...)
```

$$E(t) = s_0 - s_1\, t + s_{2,1}\, t^3 - s_{2,2}\, t^4 - s_{3,1,1}\, t^5 + s_{3,2,1}\, t^6 + O(t^7)$$

```
Self conjugate partitions lambda=lambda^'
```

$$\begin{aligned} F(t) = {} & O(t^7) + s_0 + s_{1,1}\, t^2 + s_{2,1,1}\, t^4 + s_{3,1,1,1}\, t^6 + s_{2,2,2}\, t^6 + s_{1,1,1,1}\, t^4 + s_{2,2}\, t^4 + s_{1,1,1,1,1,1}\, t^6 \\ & + s_{2,2,1,1}\, t^6 + s_{3,3}\, t^6 + s_2\, t^2 + s_{3,1}\, t^4 + s_{4,1,1}\, t^6 + s_{2,1}\, t^3 + s_4\, t^4 + s_6\, t^6 + s_{4,2}\, t^6 + s_1\, t + s_{3,1,1}\, t^5 \\ & + s_3\, t^3 + s_{3,2,1}\, t^6 + s_{1,1,1}\, t^3 + s_5\, t^5 + s_{4,1}\, t^5 + s_{3,2}\, t^5 + s_{2,2,1}\, t^5 + s_{2,1,1,1}\, t^5 + s_{1,1,1,1,1}\, t^5 + s_{5,1}\, t^6 \\ & + s_{2,1,1,1,1}\, t^6 \end{aligned}$$

```
All partitions
```

$$L(t) = s_0 - s_1\, t + s_{1,1}\, t^2 - s_{1,1,1}\, t^3 + s_{1,1,1,1}\, t^4 - s_{1,1,1,1,1}\, t^5 + s_{1,1,1,1,1,1}\, t^6 + O(t^7)$$

```
One row partitions (one part elementary symmetric functions)
```

$$M(t) = s_0 + s_1\, t + s_2\, t^2 + s_3\, t^3 + s_4\, t^4 + s_5\, t^5 + s_6\, t^6 + O(t^7)$$

```
One row partitions (one part complete symmetric functions)
```

Show that A,B and C,D and E,F and L,M are mutual inverse series:

```
> getSfktSeries(A,4,1); # A series
  getSfktSeries(B,4,1); # B series
  truncWT(outer(%%,%),4); # skip terms of weight greather than 4
```

$$s_0 - s_{1,1} + s_{2,1,1}$$

$$s_0 + s_{1,1} + s_{1,1,1,1} + s_{2,2}$$

$$s_0$$

```
> getSfktSeries(C,4,1); # C series
  getSfktSeries(D,4,1); # D series
  truncWT(outer(%%,%),4); # skip terms of weight greather than 4
```

$$s_0 - s_2 + s_{3,1}$$

$$s_0 + s_2 + s_4 + s_{2,2}$$

$$s_0$$

```
> getSfktSeries(E,6,1); # E series
  getSfktSeries(F,6,1); # F series
  truncWT(outer(%%,%),6); # skip terms of weight greather than 4
  ==>>ERROR E wrong!
```

$$s_0 - s_1 + s_{2,1} - s_{2,2} - s_{3,1,1} + s_{3,2,1}$$

$$s_0 + s_1 + s_2 + s_{1,1} + s_3 + s_{2,1} + s_{1,1,1} + s_4 + s_{3,1} + s_{2,2} + s_{2,1,1} + s_{1,1,1,1} + s_5 + s_{4,1} + s_{3,2} + s_{3,1,1} + s_{2,2,1} + s_{2,1,1,1} + s_{1,1,1,1,1} + s_6 + s_{5,1} + s_{4,2} + s_{4,1,1} + s_{3,3} + s_{3,2,1} + s_{3,1,1,1} + s_{2,2,2} + s_{2,2,1,1} + s_{2,1,1,1,1} + s_{1,1,1,1,1,1}$$

$$s_0$$

Note that the product of the L series and the A series is the E series:

```
> truncWT(outer(getSfktSeries(L,6,1),getSfktSeries(A,6,1)),6);
  getSfktSeries(E,6,1); # E series
```

$$s_0 - s_1 + s_{2,1} - s_{2,2} - s_{3,1,1} + s_{3,2,1}$$

$$s_0 - s_1 + s_{2,1} - s_{2,2} - s_{3,1,1} + s_{3,2,1}$$

```
> getSfktSeries(L,4,1); # L series
  getSfktSeries(M,4,1); # M series
  truncWT(outer(%%,%),4); # skip terms of weight greather than 4
```

$$s_0 - s_1 + s_{1,1} - s_{1,1,1} + s_{1,1,1,1}$$

$$s_0 + s_1 + s_2 + s_3 + s_4$$

$$s_0$$

```
>
```

Algorithm used:

The series are obtained by stupidly examining the partitions ... .

See Also: SchurFkt[branch]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[grAlexComp] - compares two compositions/partitions w.r.t. graded anti-lexicographic ordering

Calling Sequence:

b := grAlexComp(c1,c2)

Parameters:

- c1,c2 : compositions (or partitions)

Output:

- b : boolean value (true / false)

WARNING:

Note that Maple uses in its combinatorial packages lexicographical order of partitions and compositions.

Description:

- grAlexComp allows to order compositions and partitions in graded anti-lexicographic order. Graded anti-lexicographic order may display some gradings in the Laplace pairing for the Rota-Stein cliffordizations of monomial symmetric functions.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

Compare AlexComp and grAlexComp for special values:

```
> AlexComp([3,2],[2,1]);
  grAlexComp([3,2],[2,1]);
  AlexComp([2,2,1,0],[3,0,2,0]);
  grAlexComp([2,2,1,0],[3,0,2,0]);
```

$$true$$

$$true$$

$$false$$

$$false$$

Using AlexComp and grAlexComp to sort a list of partitions:

```
> prt:=[ [3,2],[6],[5,1],[1,1,1,1,1,1] ];
  sort(prt,AlexComp);
  sort(prt,grAlexComp);
```

$$prt := [[3, 2], [6], [5, 1], [1, 1, 1, 1, 1, 1]]$$

$$[[6], [5, 1], [3, 2], [1, 1, 1, 1, 1, 1]]$$

$$[[6], [5, 1], [3, 2], [1, 1, 1, 1, 1, 1]]$$

Maple's combinat package produces lex ordered lists.

```
> combinat[partition](3);
  sort(%,grAlexComp);
```

$$[[1, 1, 1], [1, 2], [3]]$$

$$[[3], [1, 2], [1, 1, 1]]$$

Note that CompNM and PartNM functions produce lists of compositions and partitions in anti-lexicographical order.

```
> prt:=PartNM(3,3);
  cmp:=CompNM(2,5);
```

$$prt := [[3], [2, 1], [1, 1, 1]]$$

$$cmp := [[2, 0, 0, 0, 0], [1, 1, 0, 0, 0], [1, 0, 1, 0, 0], [1, 0, 0, 1, 0], [1, 0, 0, 0, 1], [0, 2, 0, 0, 0], [0, 1, 1, 0, 0], [0, 1, 0, 1, 0], [0, 1, 0, 0, 1], [0, 0, 2, 0, 0], [0, 0, 1, 1, 0], [0, 0, 1, 0, 1], [0, 0, 0, 2, 0], [0, 0, 0, 1, 1], [0, 0, 0, 0, 2]]$$

Sorting in anti lex order can be accomplished by using 'not':

```
> prt;
  sort(prt,not grAlexComp);
  cmp;
  sort(cmp,not grAlexComp);
```

$$[[3], [2, 1], [1, 1, 1]]$$

$$[[1, 1, 1], [2, 1], [3]]$$

$$[[2, 0, 0, 0, 0], [1, 1, 0, 0, 0], [1, 0, 1, 0, 0], [1, 0, 0, 1, 0], [1, 0, 0, 0, 1], [0, 2, 0, 0, 0], [0, 1, 1, 0, 0], [0, 1, 0, 1, 0], [0, 1, 0, 0, 1], [0, 0, 2, 0, 0], [0, 0, 1, 1, 0], [0, 0, 1, 0, 1], [0, 0, 0, 2, 0], [0, 0, 0, 1, 1], [0, 0, 0, 0, 2]]$$

$$[[0, 0, 0, 0, 2], [0, 0, 0, 1, 1], [0, 0, 0, 2, 0], [0, 0, 1, 0, 1], [0, 0, 1, 1, 0], [0, 0, 2, 0, 0], [0, 1, 0, 0, 1], [0, 1, 0, 1, 0], [0, 1, 1, 0, 0], [0, 2, 0, 0, 0], [1, 0, 0, 0, 1], [1, 0, 0, 1, 0], [1, 0, 1, 0, 0], [1, 1, 0, 0, 0], [2, 0, 0, 0, 0]]$$

AlexComp can handle lists with different length, if necessary trailing zeros are appended internally:

```
> AlexComp([2],[3,2,1,2]);
  AlexComp([4],[1,1,1,1]);
```

$$false$$

$$true$$

```
>
```

Algorithm

Not available (obvious).

See Also: SchurFkt[Overview], SchurFkt[PartNM], SchurFkt[CompNM], SchurFkt[AlexComp]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[h_to_s] - convert a homogenous symmetric function into a Schur function

Calling Sequence:

sfkt := h_to_s(hfkt)

Parameters:

- hfkt : complete symmetric function

Output:

- sfkt : Schur function

WARNING:

--none--

Description:

- This function takes a an expression in complete symmetric functions and returns it in the Schur function basis.
- Basis changes are considered to be a major part of the theory of symmetric functions. This particular basis change can be done in various ways. SchurFkt uses in general Hopf algebra methods to perform such basis changes. These algorithms are possibly not the fastest, but they neatly generalize to Schur functors and are therefore important as a proof-of-concept issue.
- h_to_s could be done by using the Kostka numbers, but is implemented by using the multiplicativity of the h-basis and the isomorphism h[n]=s[n] for one part partitions.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
```

```
> h_to_s(h[2]);
  h_to_s(h[1,1]);
```

$$s_2$$

$$s_2 + s_{1,1}$$

```
> h_to_s(2*h[3,1]+3*h[1,1]);
```

$$2\, s_4 + 2\, s_{3,1} + 3\, s_2 + 3\, s_{1,1}$$

```
> prt:=PartNM(4,4):
  prt:=map((x)->h[op(x)],prt);
  map(h_to_s,prt);
```

$$prt := [h_4, h_{3,1}, h_{2,2}, h_{2,1,1}, h_{1,1,1,1}]$$

$$[s_4, s_4 + s_{3,1}, s_4 + s_{3,1} + s_{2,2}, s_4 + 2\, s_{3,1} + s_{2,2} + s_{2,1,1}, s_4 + 3\, s_{3,1} + 2\, s_{2,2} + 3\, s_{2,1,1} + s_{1,1,1,1}]$$

Critical cases:

```
> h_to_s(0);
  h_to_s(h[0]);
```

$$0$$

$$s_0$$

```
>
```

Algorithm used:

Due to the multiplicativity of the h-basis we use the following fact:

(1) h[\mu] = h[\mu1] . h[\mu2] . h[\mu3] . ... . h[\mul]

furthermore we use the isomorphism h[n]=s[n] for one part complete symmetric functions and evaluate the outer product in the Schur function basis

(2) h[\mu] = eval(s[\mu1] . s[\mu2] . s[\mu3] . ... . s[\mul])
= s[\mu] + other terms

See Also: SchurFkt[Overview], there are many a_to_b functions for other bases, check out in the overview.

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[inner] - the inner product in Schur function basis

Calling Sequence:

sfkt := inner(sfkt1,sfkt2,...)

Parameters:

- sfkti : Schur function polynoms

Output:

- sfkt : Schur function polynom

WARNING:

--none--

Description:

- inner implements the inner product of Schur functions.
- The implementation is based on the Murnaghan-Nakayama rule to derive the coefficients of the multiplication table in this basis

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
```

```
> inner(s[1],s[1]);
  inner(s[2],s[1,1]);
  inner(s[3],s[1]);  # not the samne weight => 0
```

$$s_1$$

$$s_{1,1}$$

$$0$$

Inner allows more than one input (due to associativity) and is multilinear over Z.

```
> inner(s[1],s[1],s[1]);
  inner(3*s[3]+2*s[2,1]+s[1,1,1],7*s[1,1,1]+9*s[3]);
```

$$s_1$$

$$30\, s_{1,1,1} + 34\, s_3 + 32\, s_{2,1}$$

Critical cases:

```
> inner(0);
  inner(s[3,3]);  # inner returns a single argument
  inner(s[2,1],0);
```

$$0$$

$$s_{3,\,3}$$

$$0$$

>

Algorithm used:

The multiplication table of the inner multiplication is given as (\lambda,\mu,\nu of weight N, else zero)

\gamma^\lambda_\mu\nu = (\sum_\pi\in S_N) \chi_\lamba(\pi) \chi_\mu(\pi) \chi_nu(\pi)

where \chi_\mu(\pi) is the irreducible character \chi_\mu of S_N evaluated on a permutation \pi. For efficiency reason it is sufficient to evaluate on cycle types and multiply by the length of the orbit of this cycle type. The chi_\mu(\cmp2part(\pi)) are computed using the Murnaghan-Nakayama rule .

See Also: SchurFkt[Overview], there are other version of the inner product for other bases 'B' called : innerB

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[innerP] - inner product in the power sum basis

Calling Sequence:

pfkt := innerP(pfkt1,pfkt2)

Parameters:

- pfkt1,pfkt2 : power sum symmetric functions

Output:

- pfkt : power sum symmetric function

WARNING:

--none--

Description:

- This function implements the inner product in power sum symmetric function basis.
- The inner product of power sum symmetric functions has the following properties:

  (1) if the weight of the partitions \mu and \nu is different, then p[\mu]*p[\nu]=0
  (2) the p[\mu] are almost idempotent elements under the inner product
  p[\mu]*p[\nu] = \z_\mu \delta_\mu,\nu p[\mu]
  where z\mu] is the zee-factor .

- In a generating function realization of symmetric functions the inner product belongs to the Haddamard product of these generating functions.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> innerP(p[2,1],p[2,1]);
  zee([2,1]);
```

$$2\,p_{2,1}$$

$$2$$

```
> prt:=map((x)->p[op(x)],PartNM(5,5));
  linalg[matrix](nops(prt),nops(prt),(i,j)->innerP(prt[i],prt[j]));
```

$$prt := [p_5, p_{4,1}, p_{3,2}, p_{3,1,1}, p_{2,2,1}, p_{2,1,1,1}, p_{1,1,1,1,1}]$$

$$\begin{bmatrix} 5\,p_5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4\,p_{4,1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6\,p_{3,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6\,p_{3,1,1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 8\,p_{2,2,1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 12\,p_{2,1,1,1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 120\,p_{1,1,1,1,1} \end{bmatrix}$$

Compare with the inner product on Schur functions based on the Murnaghan-Nakayama rule.

```
> inner(p_to_s(p[3,2]),p_to_s(p[3,2]));
  s_to_p(%);
  innerP(p[3,2],p[3,2]);
```

$$6\,s_5 - 6\,s_{4,1} + 6\,s_{3,2} - 6\,s_{2,2,1} + 6\,s_{2,1,1,1} - 6\,s_{1,1,1,1,1}$$

$$6\,p_{3,2}$$

$$6\,p_{3,2}$$

Critical cases:

```
> innerP(p[2]); # one input is returned (maybe should be zero?)
  innerP(p[3,2],0);
  innerP(0,0);
  innerP(p[1],p[1]);
```

$$p_2$$

$$0$$

$$0$$

$$p_1$$

```
>
```

Algorithm used:

A trivial implementation of the above given properties.

See Also: SchurFkt[inner]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[isLattice] - checks if a Young tableau is a lattice permutation

Calling Sequence:

bool := isLattice(tab)

Parameters:

- tab : a Young tableau (list list integer)

Output:

- bool : Boolean value

WARNING:

--internal use mainly--

Description:

- A Young tableau is a Young diagram where the boxes are filed with an ordered alphabet (e.g., integers in [1..n]). A Young tableau is called standard if in each row and column the entries are strictly increasing (in out Anglo-Saxonia indexing). A semistandard Young tableau is a Young tableau with nondecreasing rows and strictly increasing columns. One can extract from a Young tableau a word by forming a string out of the entries along a reading order. We use the following reading order: start in the top row rightmost and read leftward and then continue in the same way in the following column.

```
1 2 3              1 3 3
4 5   std tableau    2 4    semi std tableau
5 6                5 5

3215465            3314255
```

  A word extracted from a Young tableau is called a lattice permutation (or ballot sequence) if the number of occurrences of letters ai are greater or equal to the number of occurrences of letters aj is ai<aj.

- In the implementation of the Littewood-Richardson rule along the lines one calculates by hand (there is a more effective algorithm proposed by Lascoux and Schützenberger using Schubert calculus and Specht modules) we need to add boxes from one Young tableau to a second Young tableau in such a way, that the resulting tableau is a lattice permutation. This is a condition which ensures that the produces tableaux are in the so called plactic monoid (fulfilling the Knuth correspondence)
- A skew tableaux is represented by an ordinary tableau, where the removed boxes are filled with zeros.

0 0 0 1 1 2
0 0 2 2 2 3
0 2 3 4
1 3

A semistandard tableau (6 6 4 2/3 2 1) with a filling from [1..4].

- isLattice is the implementation of the boolean valued function which checks if a given tableau is a lattice permutation.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> isLattice([[1,2,3],[4,5]]);                # 32145
  isLattice([[1,1,1,1],[2,2,2],[3,3,3]]);   # 1111222333
  isLattice([1,1,1,2],[2,2],[3]);
```

false

true

true

Critical cases:

```
> isLattice([[]]);
  isLattice([[0]]);
  isLattice([[1]]);
```

true

true

true

```
>
```

Algorithm used:

The function is implemented along the lines described above. To any tableau the associated word is formed as a sequence. Zeros are eliminated, then a check is performed if the sequence is a ballot sequence (lattice permutation).

See Also: SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[KostkaPC] - computes the Kostka coefficient between a composition and a partition

Calling Sequence:

int := KostkaPC(part,comp)

Parameters:

- part : a partition
  comp : a composition (note that a partition is also a composition!)

Output:

- int : an integer. The Kostka number.

WARNING:

No type check, the first input has to be a partition!

Description:

- The procedure KostkaPC computes the Kostka number between a composition and a partition. KostkaPC is constant on all compositions which can be projected onto a single partition. Therefore KostkaPC computes the Kostka number between the partition part1 which is the projection from the composition comp deleting zeros and sorting the positive entries.
- The Kostka number is the number of semi-standard tableau of shape part1 (projection of comp) filled by the numbers 1^r1 2^r2 ... k^rk where the ri are the multiplicities of occurrences of i's in the partition part.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

A single semi-standrd tableau:

```
> KostkaPC([3,2,1],[3,2,1]);
  KostkaPC([2,2],[2,1,1]);
```

$$1$$

$$1$$

Orbits of compositions (compositions which after sorting have the same partition form) have the same Kostka number

```
> KostkaPC([2,1],[2,1]);
  KostkaPC([2,1],[1,2]);
  KostkaPC([2,1],[0,2,0,1]);
```

$$1$$

$$1$$

$$1$$

Further nontrivial examples:

```
> KostkaPC([2,2,1],[1,1,1,1,1]);
  KostkaPC([4,3,2,1],[2,2,2,2,2]);
```

$$5$$

$$24$$

```
>
```

Algorithm:

The Kostka numbers can be obtained effectively using hives.

SchurFkt uses the fact, that the Scalar product between a Schur function and a complete symmetric function gives the Kostka numbers as:

(1) <s_\mu | h_\nu> = K_\mu\rho < m_\rho| h_\nu> = K_\mu\nu

due to the mutual duality of monomial and complete symmetric functions. Now we can apply the Laplace expansion

(2) <s_\mu | h_\nu> = <s_\mu | h_\nu1 . h_\nu2 . ... . h_\nul >
= <\delta^(l-1)s_\mu | h_\nu1 x h_\nu2 x ... x h_\nul > x= tensor product
= < s_\mu(1) x s_\mu(2) x ... x s_\mu(l) | h_\nu1 x ... x h_\nul >
= <s_\mu(1) | h_\nu1 > . ... . < s_\mu(l) | h_\nul>

This works for compositions in the second argument too, since the product of the h_n is commutative and since h_0=1 is the identity.

```
>
```

See Also: SchurFkt[Overview], SchurFkt[KostkaTable]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[Table] - computes the Kostka matrix in any dimension

Calling Sequence:

eq := KostkaPC(int)

Parameters:

- int : integer specifying the weight of the partitions for the Kostka table

Output:

- eq : an equation KostkaN=matrix, with a matrix of dimension number of partitions of int (in general not int itself!) containing the Kostka numbers. The matrix is given in the basis where the partitions are ordered in anti-lexicographical ordering (AlexComp).

WARNING:

<none>

Description:

- The Kostka matrix is the matrix which governs many transitions of interger bases of symmetric functions. They have a combinatorial meaning, see SchurFkt[KostkaPC].
- An equation for the integer number is given as:
  (1) s_\lambda = \sum_\mu K_\lambda\mu m_\mu = \sum <s_\lambda | h_\mu> m_\mu
  which is used in KostkaPC to compute these numbers.
- The combinatorial description of the Kostka number is that they give the number of standard fillings of \mu1 ones, \mu2 twos etc into the Young diagram of shape \lambda. For that reason the number is a nonnegative integer.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

Some examples:

```
> KostkaTable(2);
  KostkaTable(3);
```

$$Kostka2 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

$$Kostka3 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

A bigger one, be careful with increasing N though

```
> tmp:=time():loc_system:="Pentium M 2.13 GHz, 2 GB RAM":
```

```
KostkaTable(7);
printf("Computation needed %f second on a
%s\n",time()-tmp,loc_system);
```

$$Kostka7 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 2 & 1 & 2 & 3 & 2 & 2 & 3 & 4 & 3 & 4 & 5 & 6 \\ 0 & 0 & 1 & 1 & 1 & 2 & 3 & 2 & 3 & 4 & 6 & 5 & 7 & 10 & 14 \\ 0 & 0 & 0 & 1 & 0 & 1 & 3 & 1 & 1 & 3 & 6 & 3 & 6 & 10 & 15 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 3 & 4 & 4 & 6 & 9 & 14 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 & 2 & 4 & 8 & 6 & 11 & 20 & 35 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 4 & 1 & 4 & 10 & 20 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 2 & 3 & 3 & 6 & 11 & 21 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 2 & 3 & 5 & 10 & 21 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 3 & 2 & 6 & 15 & 35 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 5 & 15 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 5 & 14 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 4 & 14 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

```
Computation needed 0.392000 second on a Pentium M 2.13 GHz, 2 GB RAM
>
```

Algorithm:

KostkaTable just produces the partitions and a matrix using KostkaPC.

See Also: SchurFkt[Overview], SchurFkt[KostkaPC]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[LaplaceM] - Laplace pairing for the monomial symmetric function concatenation algebra

SchurFkt[LaplaceTable] - a tabulated output of the LaplaceM pairing

SchurFkt[LaplaceM_mon] - Laplace pairing on monomials

Calling Sequence:

o1 := LaplaceM(m1,m2);
o1 := LaplaceM_mon(mon1,mon2);
M1 := LaplaceTable(N,M);

Parameters:

- m1,m2 : monomial symmetric functions (elements of `type/mfktpolynom`) of any weight
  mon1,mon2: monomials (not terms or polynomials)
  N,M : integers, weights of two sets of partitions, ordered in grAlexComp.

Output:

- t1 : monomial symmetric function polynom of type `type/mfktpolynom`, in the same basis as the input type
  M1 : a table which gives the LaplaceM pairing on the Cartesian product of two sets of partitions of weight N,M

WARNING:

LaplaceM is for internal use mainly! LaplaceTable is for better display and may disappear in later versions
LaplaceM_mon many be used for performance reasons, but be cautious with the input (no type checking)!

Description:

- LaplaceM is the **algebra valued** pairing of two monomial symmetric functions. This pairing is needed to compute the cliffordization (aka Hopf algebra deformation, aka Drinfeld twist) which deforms the divided powers concatenation product concatM into the outer product of symmetric functions in the monomial basis SchurFkt[outerM].
- The definition of LaplaceM is as follows:

  i) LaplaceM(m[0],m[0]) := m[0]

  ii) let n,...,n_r and m,...,m_s have the r respectively s repetitions of the same parts
  LaplaceM(m[n,...,n_r],m[m,...,m_s]) = \delta_{r,s} m[n+m,...,n+m_s]

iii) extend the above definition my Laplace expansion using the outer coproduct SchurFkt[couterM] and the divided powers concatenation product SchurFkt[concatM].

Laplace(concatM(m[lambda],m[mu]), m[rho]) = concatM(LaplaceM(m[lambda], m[rho(1)]),LaplaceM(m[mu],m[rho(2)]))
Laplace(m[lambda],concatM(m[mu], m[rho])) = concatM(LaplaceM(m[lambda(1)],m[mu]),LaplaceM(m[lambda(2),m[rho]))

- The LaplaceM pairing is graded in two ways. First, LaplaceM it is a graded map from /\ x /\ --> /\. Hence it maps two monomial symmetric functions of weight n and m onto a homogenous polynomial of monomial symmetric functions of weight n+m. Further LaplaceM is graded by the number of parts of a partition. As such, it is composed of infinite dimensional subblocks! However, for finite input we get a finite output.
- **LaplaceM_mon** is the Laplace pairing acting on monomials. In tensor expression this is faster since we know that the arguments are monomials and not monomial symmetric function (polynomials or terms).
- Note that there is a function LaplaceTable(N,M) which displays the output of LaplaceM on the set of partitions of N, M.

  LaplaceTable shows a sort of 'multiplication table' since the LaplaceM pairing is algebra valued. In the first row and columns, the partitions of the monomial symmetric functions used are given, in the entry 1,1 the Laplace pairing <x|y> is displayed. For better display and understanding of the procedure, the partitions are ordered in grAlecCopm order, that is ordered by the number-of-parts grading fist and then anti-lexicographicaly.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

some examples for LaplaceM:

```
> LaplaceM(m[0],m[0]);
  LaplaceM(m[1,1,1],m[3,1,1]);
  LaplaceM(m[2,1],m[2,1]);
```

$$m_0$$

$$m_{4,2,2}$$

$$m_{4,2} + 2\, m_{3,3}$$

check for zeros because of the grading by parts:

```
> LaplaceM(m[2,1],m[1]);
  LaplaceM(m[3,2,2],m[2,2]);
  LaplaceM(m[3,2,1],m[1,1,1,1,1]);
```

$$0$$

$$0$$

$$0$$

Note that LaplaceM is a **graded multiplication** by the weight of the partitions, and can be nonzero for any pairs of weights!

```
> LaplaceM(m[2,1,1],m[4,2,1]);
  LaplaceM(m[5,5],m[9,1]);
```

$$m_{6,3,2} + m_{5,4,2} + 2\, m_{5,3,3}$$

$$m_{14,6}$$

Multilinearity:

```
> LaplaceM(3*m[3],m[1]);
  LaplaceM(3*m[2]+2*m[3],m[1]);
  LaplaceM(3*m[2,1]+2*m[3],m[1]+m[2,1]);
```

$$3\, m_4$$

$$3\, m_3 + 2\, m_4$$

$$3\, m_{4,2} + 6\, m_{3,3} + 2\, m_4$$

WARNING: Since LaplaceM_mon is for internal use mainly, there is no type checking!

```
> LaplaceM_mon(s[2],e[2]);  ## no typecheck!! results as if
  LaplaceM(m[2],m[2]) was given
  LaplaceM_mon(m[2],m[2]);
```

$$m_4$$

$$m_4$$

Now we show a few tables of the LaplaceM pairing using the LaplaceTable function:

For better readability a first row and column is added showing the partitions of the monomial basis elements of the input of LaplaceM:

```
> LaplaceTable(2,2);
  LaplaceTable(7,4);
  LaplaceTable(6,6);
```

$$\begin{bmatrix} <x|y> & [2] & [1,1] \\ [2] & m_4 & 0 \\ [1,1] & 0 & m_{2,2} \end{bmatrix}$$

$$\begin{bmatrix} <x|y> & [4] & [3,1] & [2,2] & [2,1,1] & [1,1,1,1] \\ [7] & m_{11} & 0 & 0 & 0 & 0 \\ [6,1] & 0 & m_{9,2}+m_{7,4} & m_{8,3} & 0 & 0 \\ [5,2] & 0 & m_{8,3}+m_{6,5} & m_{7,4} & 0 & 0 \\ [5,1,1] & 0 & 0 & 0 & m_{7,2,2}+m_{6,3,2} & 0 \\ [4,3] & 0 & m_{7,4}+m_{6,5} & m_{6,5} & 0 & 0 \\ [4,2,1] & 0 & 0 & 0 & m_{6,3,2}+m_{5,4,2}+2\,m_{5,3,3} & 0 \\ [4,1,1,1] & 0 & 0 & 0 & 0 & m_{5,2,2,2} \\ [3,3,1] & 0 & 0 & 0 & m_{5,4,2}+m_{4,4,3} & 0 \\ [3,2,2] & 0 & 0 & 0 & m_{5,3,3}+2\,m_{4,4,3} & 0 \\ [3,2,1,1] & 0 & 0 & 0 & 0 & m_{4,3,2,2} \\ [3,1,1,1,1] & 0 & 0 & 0 & 0 & 0 \\ [2,2,2,1] & 0 & 0 & 0 & 0 & m_{3,3,3,2} \\ [2,2,1,1,1] & 0 & 0 & 0 & 0 & 0 \\ [2,1,1,1,1,1] & 0 & 0 & 0 & 0 & 0 \\ [1,1,1,1,1,1,1] & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$[<x|y>, [6], [5,1], [4,2], [4,1,1], [3,3], [3,2,1], [3,1,1,1], [2,2,2], [2,2,1], [2,1,1,1,1], [1,1,1,1,1,1]]$

$[[6], m_{12}, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$

$[[5,1], 0, m_{10,2}+2\,m_{6,6}, m_{9,3}+m_{7,5}, 0, m_{8,4}, 0, 0, 0, 0, 0, 0]$

$[[4,2], 0, m_{9,3}+m_{7,5}, m_{8,4}+2\,m_{6,6}, 0, m_{7,5}, 0, 0, 0, 0, 0, 0]$

$[[4,1,1], 0, 0, 0, m_{8,2,2}+2\,m_{5,5,2}, 0, m_{7,3,2}+m_{6,4,2}+m_{5,4,3}, 0, m_{6,3,3}, 0, 0, 0]$

$[[3,3], 0, m_{8,4}, m_{7,5}, 0, m_{6,6}, 0, 0, 0, 0, 0, 0]$

$[[3,2,1], 0, 0, 0, m_{7,3,2}+m_{6,4,2}+m_{5,4,3}, 0,$
$m_{6,4,2}+2\,m_{5,5,2}+2\,m_{6,3,3}+2\,m_{5,4,3}+6\,m_{4,4,4}, 0, m_{5,4,3}, 0, 0, 0]$

$[[3,1,1,1], 0, 0, 0, 0, 0, 0, m_{6,2,2,2}+2\,m_{4,4,2,2}, 0, m_{5,3,2,2}+m_{4,3,3,2}, 0, 0]$

$[[2,2,2], 0, 0, 0, m_{6,3,3}, 0, m_{5,4,3}, 0, m_{4,4,4}, 0, 0, 0]$

$[[2,2,1,1], 0, 0, 0, 0, 0, 0, m_{5,3,2,2}+m_{4,3,3,2}, 0, m_{4,4,2,2}+2\,m_{4,3,3,2}+6\,m_{3,3,3,3}, 0,$
$0]$

$[[2,1,1,1,1], 0, 0, 0, 0, 0, 0, 0, 0, 0, m_{4,2,2,2,2}+2\,m_{3,3,2,2,2}, 0]$

$[[1,1,1,1,1,1], 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, m_{2,2,2,2,2,2}]$

```
>
```

Algorithm:

The Laplace pairing is almost verbatim implemented as described above. For efficiency reasons, internally an sparse exponential representation of partitions (see SchurFkt[part2mset]) is used.

The internal function doing this is LaplaceM_mset (not exported). LaplaceM_mon is a wrapper for this function to make it applicable on ordinary partition representations. LaplaceM is the multilinear version of LaplaceM_mon.

See Also: SchurFkt[outerM], SchurFkt[couter], SchurFkt[outerH,E,P], SchurFkt[couterM]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[LaplaceM] - Laplace pairing for the monomial symmetric function concatenation algebra
SchurFkt[LaplaceTable] - a tabulated output of the LaplaceM pairing
SchurFkt[LaplaceM_mon] - Laplace pairing on monomials

Calling Sequence:

o1 := LaplaceM(m1,m2);
o1 := LaplaceM_mon(mon1,mon2);
M1 := LaplaceTable(N,M);

Parameters:

- m1,m2 : monomial symmetric functions (elements of `type/mfktpolynom`) of any weight
  mon1,mon2: monomials (not terms or polynomials)
  N,M : integers, weights of two sets of partitions, ordered in grAlexComp.

Output:

- t1 : monomial symmetric function polynom of type `type/mfktpolynom` , in the same basis as the input type
  M1 : a table which gives the LaplaceM pairing on the Cartesian product of two sets of partitions of weight N,M

WARNING:

LaplaceM is for internal use mainly! LaplaceTable is for better display and may disappear in later versions
LaplaceM_mon many be used for performance reasons, but be cautious with the input (no type checking)!

Description:

- LaplaceM is the **algebra valued** pairing of two monomial symmetric functions. This pairing is needed to compute the cliffordization (aka Hopf algebra deformation, aka Drinfeld twist) which deforms the divided powers concatenation product concatM into the outer product of symmetric functions in the monomial basis SchurFkt[outerM] .
- The definition of LaplaceM is as follows:

  i) LaplaceM(m[0],m[0]) := m[0]

  ii) let n,...,n_r and m,...,m_s have the r respectively s repetitions of the same parts
  LaplaceM(m[n,...,n_r],m[m,...,m_s]) = \delta_{r,s} m[n+m,...,n+m_s]

iii) extend the above definition my Laplace expansion using the outer coproduct (SchurFkt[couterM]) and the divided powers concatenation product SchurFkt[concatM] .

Laplace(concatM(m[lambda],m[mu]), m[rho]) = concatM(LaplaceM(m[lambda], m[rho(1)]),LaplaceM(m[mu],m[rho(2)]))

Laplace(m[lambda],concatM(m[mu], m[rho])) = concatM(LaplaceM(m[lambda(1)],m[mu]),LaplaceM(m[lambda(2),m[rho]))

- The LaplaceM pairing is graded in two ways. First, LaplaceM it is a graded map from /\ x /\ --> /\. Hence it maps two monomial symmetric functions of weight n and m onto a homogenous polynomial of monomial symmetric functions of weight n+m. Further LaplaceM is graded by the number of parts of a partition. As such, it is composed of infinite dimensional subblocks! However, for finite input we get a finite output.
- **LaplaceM_mon** is the Laplace pairing acting on monomials. In tensor expression this is faster since we know that the arguments are monomials and not monomial symmetric function (polynomials or terms).
- Note that there is a function LaplaceTable(N,M) which displays the output of LaplaceM on the set of partitions of N, M.

  LaplaceTable shows a sort of 'multiplication table' since the LaplaceM pairing is algebra valued. In the first row and columns, the partitions of the monomial symmetric functions used are given, in the entry 1,1 the Laplace pairing <x|y> is displayed. For better display and understanding of the procedure, the partitions are ordered in grAlecCopm order, that is ordered by the number-of-parts grading first and then anti-lexicographicaly.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

Some examples for LaplaceM:

```
> LaplaceM(m[0],m[0]);
  LaplaceM(m[1,1,1],m[3,1,1]);
  LaplaceM(m[2,1],m[2,1]);
```

$$m_0$$

$$m_{4,2,2}$$

$$m_{4,2} + 2\, m_{3,3}$$

Check for zeros because of the grading by parts:

```
> LaplaceM(m[2,1],m[1]);
  LaplaceM(m[3,2,2],m[2,2]);
  LaplaceM(m[3,2,1],m[1,1,1,1,1]);
```

$$0$$

$$0$$

$$0$$

Note that LaplaceM is a **graded multiplication** by the weight of the partitions, and can be nonzero for any pairs of weights!

```
> LaplaceM(m[2,1,1],m[4,2,1]);
  LaplaceM(m[5,5],m[9,1]);
```

$$m_{6,3,2} + m_{5,4,2} + 2\, m_{5,3,3}$$

$$m_{14,6}$$

Multilinearity:

```
> LaplaceM(3*m[3],m[1]);
  LaplaceM(3*m[2]+2*m[3],m[1]);
  LaplaceM(3*m[2,1]+2*m[3],m[1]+m[2,1]);
```

$$3\, m_4$$

$$3\, m_3 + 2\, m_4$$

$$3\, m_{4,2} + 6\, m_{3,3} + 2\, m_4$$

WARNING: Since LaplaceM_mon is for internal use mainly, there is no type checking!

```
> LaplaceM_mon(s[2],e[2]);  ## no typecheck!! results as if
  LaplaceM(m[2],m[2]) was given
  LaplaceM_mon(m[2],m[2]);
```

$$m_4$$

$$m_4$$

Now we show a few tables of the LaplaceM pairing using the LaplaceTable function:

For better readability a first row and column is added showing the partitions of the monomial basis elements of the input of LaplaceM:

```
> LaplaceTable(2,2);
  LaplaceTable(7,4);
  LaplaceTable(6,6);
```

$$\begin{bmatrix} <x|y> & [2] & [1,1] \\ [2] & m_4 & 0 \\ [1,1] & 0 & m_{2,2} \end{bmatrix}$$

$$\begin{bmatrix} <x|y> & [4] & [3,1] & [2,2] & [2,1,1] & [1,1,1,1] \\ [7] & m_{11} & 0 & 0 & 0 & 0 \\ [6,1] & 0 & m_{9,2}+m_{7,4} & m_{8,3} & 0 & 0 \\ [5,2] & 0 & m_{8,3}+m_{6,5} & m_{7,4} & 0 & 0 \\ [5,1,1] & 0 & 0 & 0 & m_{7,2,2}+m_{6,3,2} & 0 \\ [4,3] & 0 & m_{7,4}+m_{6,5} & m_{6,5} & 0 & 0 \\ [4,2,1] & 0 & 0 & 0 & m_{6,3,2}+m_{5,4,2}+2\,m_{5,3,3} & 0 \\ [4,1,1,1] & 0 & 0 & 0 & 0 & m_{5,2,2,2} \\ [3,3,1] & 0 & 0 & 0 & m_{5,4,2}+m_{4,4,3} & 0 \\ [3,2,2] & 0 & 0 & 0 & m_{5,3,3}+2\,m_{4,4,3} & 0 \\ [3,2,1,1] & 0 & 0 & 0 & 0 & m_{4,3,2,2} \\ [3,1,1,1,1] & 0 & 0 & 0 & 0 & 0 \\ [2,2,2,1] & 0 & 0 & 0 & 0 & m_{3,3,3,2} \\ [2,2,1,1,1] & 0 & 0 & 0 & 0 & 0 \\ [2,1,1,1,1,1] & 0 & 0 & 0 & 0 & 0 \\ [1,1,1,1,1,1,1] & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$[<x|y>, [6], [5,1], [4,2], [4,1,1], [3,3], [3,2,1], [3,1,1,1], [2,2,2], [2,2,1], [2,1,1,1,1], [1,1,1,1,1,1]]$

$[[6], m_{12}, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$

$[[5,1], 0, m_{10,2}+2\,m_{6,6}, m_{9,3}+m_{7,5}, 0, m_{8,4}, 0, 0, 0, 0, 0, 0]$

$[[4,2], 0, m_{9,3}+m_{7,5}, m_{8,4}+2\,m_{6,6}, 0, m_{7,5}, 0, 0, 0, 0, 0, 0]$

$[[4,1,1], 0, 0, 0, m_{8,2,2}+2\,m_{5,5,2}, 0, m_{7,3,2}+m_{6,4,2}+m_{5,4,3}, 0, m_{6,3,3}, 0, 0, 0]$

$[[3,3], 0, m_{8,4}, m_{7,5}, 0, m_{6,6}, 0, 0, 0, 0, 0, 0]$

$[[3,2,1], 0, 0, 0, m_{7,3,2}+m_{6,4,2}+m_{5,4,3}, 0,$
$m_{6,4,2}+2\,m_{5,5,2}+2\,m_{6,3,3}+2\,m_{5,4,3}+6\,m_{4,4,4}, 0, m_{5,4,3}, 0, 0, 0]$

$[[3,1,1,1], 0, 0, 0, 0, 0, 0, m_{6,2,2,2}+2\,m_{4,4,2,2}, 0, m_{5,3,2,2}+m_{4,3,3,2}, 0, 0]$

$[[2,2,2], 0, 0, 0, m_{6,3,3}, 0, m_{5,4,3}, 0, m_{4,4,4}, 0, 0, 0]$

$[[2,2,1,1], 0, 0, 0, 0, 0, 0, m_{5,3,2,2}+m_{4,3,3,2}, 0, m_{4,4,2,2}+2\,m_{4,3,3,2}+6\,m_{3,3,3,3}, 0,$
$0]$

$[[2,1,1,1,1], 0, 0, 0, 0, 0, 0, 0, 0, 0, m_{4,2,2,2,2}+2\,m_{3,3,2,2,2}, 0]$

$[[1,1,1,1,1,1], 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, m_{2,2,2,2,2,2}]$

>

Algorithm:

The Laplace pairing is almost verbatim implemented as described above. For efficiency reasons, internally an sparse exponential representation of partitions (see SchurFkt[part2mset]) is used.

The internal function doing this is LaplaceM_mset (not exported). LaplaceM_mon is a wrapper for this function to make it applicable on ordinary partition representations. LaplaceM is the multilinear version of LaplaceM_mon.

See Also: SchurFkt[outerM], SchurFkt[couter], SchurFkt[outerH,E,P], SchurFkt[couterM]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[LaplaceM] - Laplace pairing for the monomial symmetric function concatenation algebra
SchurFkt[LaplaceTable] - a tabulated output of the LaplaceM pairing
SchurFkt[LaplaceM_mon] - Laplace pairing on monomials

Calling Sequence:

o1 := LaplaceM(m1,m2);
o1 := LaplaceM_mon(mon1,mon2);
M1 := LaplaceTable(N,M);

Parameters:

- m1,m2 : monomial symmetric functions (elements of `type/mfktpolynom`) of any weight
  mon1,mon2: monomials (not terms or polynomials)
  N,M : integers, weights of two sets of partitions, ordered in grAlexComp.

Output:

- t1 : monomial symmetric function polynom of type `type/mfktpolynom`, in the same basis as the input type
  M1 : a table which gives the LaplaceM pairing on the Cartesian product of two sets of partitions of weight N,M

WARNING:

LaplaceM is for internal use mainly! LaplaceTable is for better display and may disappear in later versions
LaplaceM_mon many be used for performance reasons, but be cautious with the input (no type checking)!

Description:

- LaplaceM is the **algebra valued** pairing of two monomial symmetric functions. This pairing is needed to compute the cliffordization (aka Hopf algebra deformation, aka Drinfeld twist) which deforms the divided powers concatenation product concatM into the outer product of symmetric functions in the monomial basis SchurFkt[outerM] .
- The definition of LaplaceM is as follows:

  i) LaplaceM(m[0],m[0]) := m[0]

  ii) let n,...,n_r and m,...,m_s have the r respectively s repetitions of the same parts
  LaplaceM(m[n,...,n_r],m[m,...,m_s]) = \delta_{r,s} m[n+m,...,n+m_s]

iii) extend the above definition my Laplace expansion using the outer coproduct (SchurFkt[couterM]) and the divided powers concatenation product SchurFkt[concatM] .

Laplace(concatM(m[lambda],m[mu]), m[rho]) = concatM(LaplaceM(m[lambda], m[rho(1)]),LaplaceM(m[mu],m[rho(2)]))

Laplace(m[lambda],concatM(m[mu], m[rho])) = concatM(LaplaceM(m[lambda(1)],m[mu]),LaplaceM(m[lambda(2),m[rho]))

- The LaplaceM pairing is graded in two ways. First, LaplaceM it is a graded map from /\ x /\ --> /\. Hence it maps two monomial symmetric functions of weight n and m onto a homogenous polynomial of monomial symmetric functions of weight n+m. Further LaplaceM is graded by the number of parts of a partition. As such, it is composed of infinite dimensional subblocks! However, for finite input we get a finite output.
- **LaplaceM_mon** is the Laplace pairing acting on monomials. In tensor expression this is faster since we know that the arguments are monomials and not monomial symmetric function (polynomials or terms).
- Note that there is a function LaplaceTable(N,M) which displays the output of LaplaceM on the set of partitions of N, M.

  LaplaceTable shows a sort of 'multiplication table' since the LaplaceM pairing is algebra valued. In the first row and columns, the partitions of the monomial symmetric functions used are given, in the entry 1,1 the Laplace pairing <x|y> is displayed. For better display and understanding of the procedure, the partitions are ordered in grAlecCopm order, that is ordered by the number-of-parts grading first and then anti-lexicographicaly.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

Some examples for LaplaceM:

```
> LaplaceM(m[0],m[0]);
  LaplaceM(m[1,1,1],m[3,1,1]);
  LaplaceM(m[2,1],m[2,1]);
```

$$m_0$$

$$m_{4,2,2}$$

$$m_{4,2} + 2\, m_{3,3}$$

Check for zeros because of the grading by parts:

```
> LaplaceM(m[2,1],m[1]);
  LaplaceM(m[3,2,2],m[2,2]);
  LaplaceM(m[3,2,1],m[1,1,1,1,1]);
```

$$0$$

$$0$$

$$0$$

Note that LaplaceM is a **graded multiplication** by the weight of the partitions, and can be nonzero for any pairs of weights!

```
> LaplaceM(m[2,1,1],m[4,2,1]);
  LaplaceM(m[5,5],m[9,1]);
```

$$m_{6,3,2} + m_{5,4,2} + 2\, m_{5,3,3}$$

$$m_{14,6}$$

Multilinearity:

```
> LaplaceM(3*m[3],m[1]);
  LaplaceM(3*m[2]+2*m[3],m[1]);
  LaplaceM(3*m[2,1]+2*m[3],m[1]+m[2,1]);
```

$$3\, m_4$$

$$3\, m_3 + 2\, m_4$$

$$3\, m_{4,2} + 6\, m_{3,3} + 2\, m_4$$

WARNING: Since LaplaceM_mon is for internal use mainly, there is no type checking!

```
> LaplaceM_mon(s[2],e[2]);  ## no typecheck!! results as if
  LaplaceM(m[2],m[2]) was given
  LaplaceM_mon(m[2],m[2]);
```

$$m_4$$

$$m_4$$

Now we show a few tables of the LaplaceM pairing using the LaplaceTable function:

For better readability a first row and column is added showing the partitions of the monomial basis elements of the input of LaplaceM:

```
> LaplaceTable(2,2);
  LaplaceTable(7,4);
  LaplaceTable(6,6);
```

$$\begin{bmatrix} <x|y> & [2] & [1,1] \\ [2] & m_4 & 0 \\ [1,1] & 0 & m_{2,2} \end{bmatrix}$$

$$\begin{bmatrix} <x|y> & [4] & [3,1] & [2,2] & [2,1,1] & [1,1,1,1] \\ [7] & m_{11} & 0 & 0 & 0 & 0 \\ [6,1] & 0 & m_{9,2}+m_{7,4} & m_{8,3} & 0 & 0 \\ [5,2] & 0 & m_{8,3}+m_{6,5} & m_{7,4} & 0 & 0 \\ [5,1,1] & 0 & 0 & 0 & m_{7,2,2}+m_{6,3,2} & 0 \\ [4,3] & 0 & m_{7,4}+m_{6,5} & m_{6,5} & 0 & 0 \\ [4,2,1] & 0 & 0 & 0 & m_{6,3,2}+m_{5,4,2}+2\,m_{5,3,3} & 0 \\ [4,1,1,1] & 0 & 0 & 0 & 0 & m_{5,2,2,2} \\ [3,3,1] & 0 & 0 & 0 & m_{5,4,2}+m_{4,4,3} & 0 \\ [3,2,2] & 0 & 0 & 0 & m_{5,3,3}+2\,m_{4,4,3} & 0 \\ [3,2,1,1] & 0 & 0 & 0 & 0 & m_{4,3,2,2} \\ [3,1,1,1,1] & 0 & 0 & 0 & 0 & 0 \\ [2,2,2,1] & 0 & 0 & 0 & 0 & m_{3,3,3,2} \\ [2,2,1,1,1] & 0 & 0 & 0 & 0 & 0 \\ [2,1,1,1,1,1] & 0 & 0 & 0 & 0 & 0 \\ [1,1,1,1,1,1,1] & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$[<x|y>, [6], [5,1], [4,2], [4,1,1], [3,3], [3,2,1], [3,1,1,1], [2,2,2], [2,2,1], [2,1,1,1,1], [1,1,1,1,1,1]]$

$[[6], m_{12}, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$

$[[5,1], 0, m_{10,2}+2\,m_{6,6}, m_{9,3}+m_{7,5}, 0, m_{8,4}, 0, 0, 0, 0, 0, 0]$

$[[4,2], 0, m_{9,3}+m_{7,5}, m_{8,4}+2\,m_{6,6}, 0, m_{7,5}, 0, 0, 0, 0, 0, 0]$

$[[4,1,1], 0, 0, 0, m_{8,2,2}+2\,m_{5,5,2}, 0, m_{7,3,2}+m_{6,4,2}+m_{5,4,3}, 0, m_{6,3,3}, 0, 0, 0]$

$[[3,3], 0, m_{8,4}, m_{7,5}, 0, m_{6,6}, 0, 0, 0, 0, 0, 0]$

$[[3,2,1], 0, 0, 0, m_{7,3,2}+m_{6,4,2}+m_{5,4,3}, 0,$
$m_{6,4,2}+2\,m_{5,5,2}+2\,m_{6,3,3}+2\,m_{5,4,3}+6\,m_{4,4,4}, 0, m_{5,4,3}, 0, 0, 0]$

$[[3,1,1,1], 0, 0, 0, 0, 0, 0, m_{6,2,2,2}+2\,m_{4,4,2,2}, 0, m_{5,3,2,2}+m_{4,3,3,2}, 0, 0]$

$[[2,2,2], 0, 0, 0, m_{6,3,3}, 0, m_{5,4,3}, 0, m_{4,4,4}, 0, 0, 0]$

$[[2,2,1,1], 0, 0, 0, 0, 0, 0, m_{5,3,2,2}+m_{4,3,3,2}, 0, m_{4,4,2,2}+2\,m_{4,3,3,2}+6\,m_{3,3,3,3}, 0,$
$0]$

$[[2,1,1,1,1], 0, 0, 0, 0, 0, 0, 0, 0, 0, m_{4,2,2,2,2}+2\,m_{3,3,2,2,2}, 0]$

$[[1,1,1,1,1,1], 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, m_{2,2,2,2,2,2}]$

>

Algorithm:

The Laplace pairing is almost verbatim implemented as described above. For efficiency reasons, internally an sparse exponential representation of partitions (see SchurFkt[part2mset]) is used.

The internal function doing this is LaplaceM_mset (not exported). LaplaceM_mon is a wrapper for this function to make it applicable on ordinary partition representations. LaplaceM is the multilinear version of LaplaceM_mon.

See Also: SchurFkt[outerM], SchurFkt[couter], SchurFkt[outerH,E,P], SchurFkt[couterM]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[m_to_p] - basis change from monomial to power sum symmetric functions

Calling Sequence:

pfkt := dummy(mfkt)

Parameters:

- mfkt : monomial symmetric function

Output:

- pfkt : power sum symmetric function

WARNING:

--quick and dirty hack, needs to be replaced by a combinatorial algorithm--

Description:

- The basis change from monomial symmetric functions to power sum symmetric functions.
- The current implementation does not use a combinatorial algorithm buy computes the basis change p_to_m and inverts the linear operator (matrix) using maples matrix inverse.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> m_to_p(m[1]);
  m_to_p(m[2]);
  m_to_p(m[3]);
  m_to_p(m[4]);
```

$$p_1$$

$$p_2$$

$$p_3$$

$$p_4$$

```
> m_to_p(m[1,1]);
  m_to_p(m[2,1]);
  m_to_p(m[1,1,1]);
```

$$-\frac{1}{2}p_2+\frac{1}{2}p_{1,1}$$

$$-p_3+p_{2,1}$$

$$\frac{1}{3}p_3 - \frac{1}{2}p_{2,1} + \frac{1}{6}p_{1,1,1}$$

Linearity:

```
> m_to_p(2*p[2]+3*p[3,2,1]);
```

$$2\,p_2\,p_1 + 3\,p_{3,2,1}\,p_1$$

```
> m_to_p(m[2,2,1,1]);
  p_to_m(%);  # inverse
```

$$-\frac{3}{2}p_6 + p_{5,1} + \frac{5}{4}p_{4,2} - \frac{1}{4}p_{4,1,1} + \frac{1}{2}p_{3,3} - p_{3,2,1} - \frac{1}{4}p_{2,2,2} + \frac{1}{4}p_{2,2,1,1}$$

$$m_{2,2,1,1}$$

Critical cases:

```
> m_to_p(0);
  m_to_p(m[0]);
```

$$0$$

$$p_0$$

```
>
```

Algorithm used:

--void, a quick hack--

See Also:

SchurFkt[p_to_m], SchurFkt[Overview] (there are many functions b_to_b changing from A-basis to B-basis)

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[MLIN] - makes the function T() multilinear over the integers

Calling Sequence:

ex2 := MLIN(ex1)

Parameters:

- ex1 : expression of type symfktpolynom

Output:

- ex2 : linearized (over the integers) expression of type symfktpolynom

WARNING:

this function is for **internal use only** and may disappear in later versions of SchurFkt. In a later version of SchurFkt this function may disappear and may be replaced by the tensor based on the (patched) define command of Bigebra .

Description:

- MLIN is applied in a substitution for T(expression) and needs an explicit evaluation. If substituted for T(expression) it makes this function multilinear over the integers.
- MLIN is a quick an dirty hack to achieve a special form of multilinearity. It may be replaced in later versions of SchurFkt by a better device and should not be used in derived work!

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> T(2*s[3]+3*s[2,1],2*s[1,1]+s[2]);
```

$$\mathrm{T}(2\,s_3 + 3\,s_{2,1}, 2\,s_{1,1} + s_2)$$

```
> eval(subs(T=MLIN,%));
```

$$4\,\mathrm{T}(s_3, s_{1,1}) + 2\,\mathrm{T}(s_3, s_2) + 6\,\mathrm{T}(s_{2,1}, s_{1,1}) + 3\,\mathrm{T}(s_{2,1}, s_2)$$

```
>
```

mandatory test cases and most likely cases a user want to type in [not yet done]

Algorithm used:

Quick and dirty hack, not to be documented.

See Also: SchurFkt[FLAT]

Last modified: June 19, 2008/BF/RA.

Function: SchurFkt[mset2part] - translates a partition in multiset notation into a partition in standard format

Calling Sequence:

prt := mset2part(expprt)

Parameters:

- expprt : exponential of power or multiset notation for a partition

Output:

- prt : standard (shape form) of a partition

WARNING:

--none-

Description:

- mset2part translates partitions in exponential or multiset notation into standard partitions, e.g., : \lambda=[1^2 2^0 3^1 4^0 ...] = (3 1 1)

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> mset2part([1,3]);
  mset2part([0,0,0,0,0,1]);
  mset2part([1,1,1,1]);
  mset2part([6]);
```

$$[2, 2, 2, 1]$$
$$[6]$$
$$[4, 3, 2, 1]$$
$$[1, 1, 1, 1, 1, 1]$$

```
> part2mset([3,3,2,2,1,1]);
  mset2part(%);
  part2mset([3,2,2,1,1,1]);
  mset2part(%);
```

$$[2, 2, 2]$$
$$[3, 3, 2, 2, 1, 1]$$
$$[3, 2, 1]$$
$$[3, 2, 2, 1, 1, 1]$$

Critical cases:

```
> mset2part([]);
```

$$[0]$$

```
>
```

Algorithm used:

The exponents given in the mset representations give direct a way (if read in reverse) to construct the partition in ordinary standard notation.

See Also: SchurFkt[part2mset]

Last modified: June 19, 2008/BF/RA.

Function: SchurFkt[MurNak2] - computes the Murnaghan-Nakayama rule in a recursive way

Calling Sequence:

int := MurNak2(sfkt,pfkt)

Parameters:

- sfkt,pfkt : And sftmonom and a pfktmonom

Output:

- int : integer (-1,0,+1)

WARNING:

function is for demonstration only and is **not (multi)linear**!

Description:

- **MurNak2 is not used in the package**. It is a demonstration of certain mathematical facts and can be used to check the correctness of the much faster SchurFkt[MurNak] function which is a direct combinatorial implementation of the Murnaghan-Nakayama rule.
- MurNak2 is a recursive implementation it uses tow facts: The fact that the Murnaghan-Nakayama rule is known on hook shaped Schur functions (1) (see also SchurFkt[CharHook]) and the fact that the Murnaghan-Nakayama rule is nothing else than the Schur-Hall scalar product between the Schur- and power sum bases. This turns the Murnaghan-Nakayama map MurNak2 : /_s x /_p ---> Integers into a Laplace pairing (2).

  (1) MurNak2(s_{a+1,1^b}, p_{n}) = (-1)^b * \delta(a+b+1,n)

  (2) MurNak2(s_\nu,p_\mu) = MurNak2(s_\nu(1),p_\mu1) * MurNak2(s_\nu(2),p_\mu[2..l])

- This recursion shows, that the characters of the symmetric group can be computed via a Glebsch-Gordan decomposition (used in the outer coproduct for Schur functions) and the relation for hook shaped Schur functions. It also shows that the Murnaghan-Nakayama rule is nothing but the Schur-Hall scalar product in the s-p-basis.
- The later fact can be used to compute the Murnaghan-Nakayama rule in a third way. We can transform Schur functions into complete functions using the Kostka matrix and the power sums functions into monomial symmetric functions using the Rota-Stein deformation of the monomial symmetric functions, which provides a method to implement the p_to_m basis change:

  (3) MurNak2(s_\mu,p_nu) = <s_\mu,p_\nu> = \sum_\rho K_\mu\rho <h_\rho, p_to_m(p_\nu) >

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> MurNak2(s[3],p[3]);     # fits in one row    (-1)^(1-1) = +1
  MurNak2(s[2,1],p[3]);   # fits in two rows   (-1)^(2-1) = -1
  MurNak2(s[1,1,1],p[3]);# fits in three rows (-1)^(3-1) = +1
  MurNak2(s[3,3],p[2,2,2]);  # there is more than one
  decomposition into boundary strips
```

$$1$$

$$-1$$

$$1$$

$$-3$$

We exemplify our claim (3) from above.

```
> 'MurNak2'(s[3,3],p[2,2,2])=MurNak2(s[3,3],p[2,2,2]);
  (s[3,3],p[2,2,2])='ScalarMH'(s_to_h(s[3,3]),p_to_m(p[2,2,2]));
  eval(subs(s[0]=1,%));
```

$$SchurFkt{:}\text{-}MurNak2(s_{3,3}, p_{2,2,2}) = -3$$

$$(s_{3,3}, p_{2,2,2}) = SchurFkt{:}\text{-}ScalarMH(-h_{4,2} + h_{3,3}, m_6 + 3\, m_{4,2} + 6\, m_{2,2,2})$$

$$(s_{3,3}, p_{2,2,2}) = -3$$

```
>
```

Algorithm used:

We use the above described algorithm splitting of parts from p_\mu and computing the coproducts of s_\nu unless we can stop the recursion using CharHook.

See Also: SchurFkt[MurNak]

Last modified: June 19, 2008/BF/RA.

Function: SchurFkt[MurNak] - computes the Murnaghan-Nakayama, hence the characters of the symmetric group

Calling Sequence:

int := MurNak(part1,part2)

Parameters:

- part1,part2 : two partitions corresponding to a power sum symmetric function and a Schur function.

Output:

- int : an integer

WARNING:

no type checking! (for internal use mainly! in s_to_p, p_to_s etc)

Description:

- MurNak computes the rimhook decomposition of a Young diagram by rimhooks given by the parts of a power symmetric function. This is an efficient way to compute the characters of the symmetric group.
- For efficiency reasons, MurNak uses internally a rim representation of partitions. This is an encoding of the shape of a partition via north (=0) and east (=1) steps. A shape translates into an infinite set of steps describing the path from (0,-infinity) to (infinity,0). (The infinite pre and suffixes are of course not used internally).

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

Some simple cases, where the power sum argument (first argument) has one part (one hook) and the Schur function part has 1,2,3 rows, which yields the signs (internally1)^1 and (-1)^2

```
> MurNak([3],[3]);
  MurNak([3],[2,1]);
  MurNak([3],[1,1,1]);
```

$$1$$

$$-1$$

$$1$$

```
> MurNak([4],[4]);        ##  1 row
  MurNak([4],[3,1]);      ##  2 rows
  MurNak([4],[2,2]);      ##  not a hook, => 0
```

```
MurNak([4],[2,1,1]);    ##  3 rows
MurNak([4],[1,1,1,1]); ##  4 rows
```

1

-1

0

1

-1

```
>
```

Algorithm used:

MurNak uses several internal functions:

makeRimRep: this function transforms the shape of a partition (Young diagram) into its rim representation (north-east-sequesnce of steps). Note that rime represented partitions are not available on the user level but only internally to the module SchurFkt.

removeRimHook: This is the function which actually does the job of removing a rimhook from a partition. It looks for a pair of east-north step in the distance of the length of the hook. If such a pair is found, the pair is exchanged (the hook removed) and a sign is computed according to the number of north steps in between the removed pair. This gives the height of the hook and hence the desired sign of the reduction. The function returns pair: the rimhook represented partition with removed hook and the sign.

MurNakRim: This is the Murnaghan-Nakayama rule on the rimhook representation level. It calls removeRimHook several times according to the partition corresponding to the power sum removing for each part of the power sum partition exactly one hook. The function keeps track of all possible removals. The result is given in standard list notation for partitions.

Finally MurNak is the wrapper function taking user supplied arguments and preparing them for usage in MurNakRim.

See Also: SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA.

Function: SchurFkt[outer] - outer product of two Schur functions

Calling Sequence:

o1 := outer(s1,s2)
o1 := outer(s1,s2,...sn)

Parameters:

- s1,s2 : S-function polynoms (an element of `type/spolynom`) of any weight
  s1,...,sn : S-function polynoms, n ranging in the positive integers

Output:

- o1 : S-function polynom of type `type/sfktpolynom`.

WARNING:

make sure to use Schur function basis, no type checking yet !!.

NOTE: This functions is also known as SchurFkt[outerS].

Description:

- The outer Schur function product multiplies two Schur polynomials using the Littlewood-Richardson rule. If the two Schur functions are of weight (number of boxes in the corresponding Young diagram, aka partition) n and m, then the output is a polynomial with integer coefficients in Schur polynomials of weight n+m.
- The outer product is multilinear over the integers (fractions of integers) as the ring of symmetric functions Λ is build as a polynomial ring over the integers or the fractions.
- In principle this product depends not on a chosen basis. However, the combinatorial algorithms which implement this product depend heavily on the fact that the Schur polynomials are related to the basis of irreducibles of Λ.
- For the outer product in other bases see SchurFkt[outerH,E,P].
- NOTE: A Schur polynomial is represented as s[lambda], where lambda is a partition of n. This object is treated as an identity, even if itself is a polynomials in (possibly infinite many) variables x1,...xn,... . A S-function monom is a single Schur polynom with coefficient unity, an S-function term is a Schur polynom with an integer (fractional) prefactor, a S-function polynomial is an integer (fractional) linear combination of Schur polynomials.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

Some some elementary examples of outer products of Schur polynomials.

```
> outer(s[1],s[0])=s[1];
  outer(s[0],s[3,1,1])=s[3,1,1];
```

$$s_1 = s_1$$

$$s_{3,1,1} = s_{3,1,1}$$

This shows that s[0]=1(x)=1 is the unit of the polynomial ring.

```
> outer(s[1],s[1]);
  outer(s[2],s[1]);
  outer(s[1,1],s[1]);
  outer(s[4,2,2,1,1],s[1]);
```

$$s_2 + s_{1,1}$$

$$s_3 + s_{2,1}$$

$$s_{2,1} + s_{1,1,1}$$

$$s_{5,2,2,1,1} + s_{4,3,2,1,1} + s_{4,2,2,2,1} + s_{4,2,2,1,1,1}$$

Adding a single box to another single box can be done in two ways, horizontally and vertically, giving two boxes in a row {2} or two boxes in a column {11}. More generally a single box is added in all possible ways to the pile of boxes of another Schur polynomial. No multiplicities can occur.

```
> outer(s[1],s[1],s[1]);
  outer(s[2]+s[1,1],s[1]);
```

$$s_3 + 2\, s_{2,1} + s_{1,1,1}$$

$$s_3 + 2\, s_{2,1} + s_{1,1,1}$$

Putting a polynomial into the product (or iterating the product) may give multiplicities. These coefficients make up the multiplication table of the ring and can be obtained by the Schur-Hall scalar product:

```
> out:=outer(s[1],s[2]);
  Scalar(out,s[3]);
  Scalar(out,s[2,1]);
  Scalar(out,s[1,1,1]);
```

$$out := s_3 + s_{2,1}$$

$$s_0$$

$$s_0$$

$$0$$

Note that s[0]=1 is the unit of the product.

```
> outer(s[4]+3*s[2,1],s[0]);
```

$$s_4 + 3\, s_{2,1}$$

Critical cases:

```
> outer(0);
```

```
outer(s[]); # should not occur but should possibly return s[0]
```

$$0$$

$$s_{[\ ]}$$

```
>
```

See Also: SchurFkt[Overview], SchurFkt[couter], e.g., SchurFkt[outerH] (see many other incarnations of outerA for A-basis)

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[outerE] - outer product in the elementary symmetric function basis (E-basis)

SchurFkt[outerH] - outer product in the complete symmetric function basis (H-basis)

Calling Sequence:

efkt := outerE(efkt1,efkt2,...)
hfkt := outerH(hfkt1,hfkt2,...)

Parameters:

- efkti : one or more elementary symmetric functions
  hfkti : one or more complete symmetric functions

Output:

- efkt : elementary symmetric function
  hfkt : complete symmetric function

WARNING:

--none--

Description:

- The E-basis (H-basis) is multiplicative. That is, the outer product is given by unordered concatenation.
- Note that the concatenation product of monomial symmetric functions concatM is different since it involved divided powers and therefore introduces prefactors.

Examples:

```
> restart:with(SchurFkt):
```

```
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> outerE(e[4],e[4]);
  outerE(e[2],e[1]);
  outerE(e[1],e[2]);  # unordered !
```

$$e_{4,4}$$

$$e_{2,1}$$

$$e_{2,1}$$

```
> outerH(h[4,4],h[2,2]);
  outerH(h[2,1],h[1,1]);
  outerH(h[1,1],h[2,1]);  # unordered !
```

$$h_{4,4,2,2}$$

$$h_{2,1,1,1}$$

$$h_{2,1,1,1}$$

More than one entry and linearity in each slot:

```
> outerE(e[3],e[2],e[1]);
  outerE(e[1]+e[2],e[3],e[4]+e[5]);
  outerE(3*e[3,2,2,1],4*e[4,2,2]);
  #
  outerH(h[3],h[2],h[1]);
  outerH(h[1]+h[2],h[3],h[4]+h[5]);
  outerH(3*h[3,2,2,1],4*h[4,2,2]);
```

$$e_{3,2,1}$$

$$e_{4,3,1} + e_{5,3,1} + e_{4,3,2} + e_{5,3,2}$$

$$12\, e_{4,3,2,2,2,2,1}$$

$$h_{3,2,1}$$

$$h_{4,3,1} + h_{5,3,1} + h_{4,3,2} + h_{5,3,2}$$

$$12\, h_{4,3,2,2,2,2,1}$$

Critical cases:

```
> outerE(0);
  outerE(e[]);  # should not occure but possibly return e[0]
  #
  outerH(0);
  outerH(e[]);  # should not occure but possibly return e[0]
```

$$0$$

$$e_{[\ ]}$$

$$0$$

$$e_{[\ ]}$$

```
>
```

Algorithm used:

On monomials combine the index sets and sort. Generalize to many input arguments and implement linearity..

See Also: SchurFkt[outer], SchurFkt[outerP], SchurFkt[outerH], SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[outerE] - outer product in the elementary symmetric function basis (E-basis)

SchurFkt[outerH] - outer product in the complete symmetric function basis (H-basis)

Calling Sequence:

efkt := outerE(efkt1,efkt2,...)
hfkt := outerH(hfkt1,hfkt2,...)

Parameters:

- efkti : one or more elementary symmetric functions
  hfkti : one or more complete symmetric functions

Output:

- efkt : elementary symmetric function
  hfkt : complete symmetric function

WARNING:

--none--

Description:

- The E-basis (H-basis) is multiplicative. That is, the outer product is given by unordered concatenation.
- Note that the concatenation product of monomial symmetric functions concatM is different since it involved divided powers and therefore introduces prefactors.

Examples:

```
> restart:with(SchurFkt):
```

```
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
```

```
> outerE(e[4],e[4]);
  outerE(e[2],e[1]);
  outerE(e[1],e[2]);  # unordered !
```

$$e_{4,4}$$

$$e_{2,1}$$

$$e_{2,1}$$

```
> outerH(h[4,4],h[2,2]);
  outerH(h[2,1],h[1,1]);
  outerH(h[1,1],h[2,1]);  # unordered !
```

$$h_{4,4,2,2}$$

$$h_{2,1,1,1}$$

$$h_{2,1,1,1}$$

More than one entry and linearity in each slot:

```
> outerE(e[3],e[2],e[1]);
  outerE(e[1]+e[2],e[3],e[4]+e[5]);
  outerE(3*e[3,2,2,1],4*e[4,2,2]);
  #
  outerH(h[3],h[2],h[1]);
  outerH(h[1]+h[2],h[3],h[4]+h[5]);
  outerH(3*h[3,2,2,1],4*h[4,2,2]);
```

$$e_{3,2,1}$$

$$e_{4,3,1} + e_{5,3,1} + e_{4,3,2} + e_{5,3,2}$$

$$12\, e_{4,3,2,2,2,2,1}$$

$$h_{3,2,1}$$

$$h_{4,3,1} + h_{5,3,1} + h_{4,3,2} + h_{5,3,2}$$

$$12\, h_{4,3,2,2,2,2,1}$$

Critical cases:

```
> outerE(0);
  outerE(e[]);  # should not occure but possibly return e[0]
  #
  outerH(0);
  outerH(e[]);  # should not occure but possibly return e[0]
```

$$0$$

$$e_{[\ ]}$$

$$0$$

$$e_{[\ ]}$$

```
>
```

Algorithm used:

On monomials combine the index sets and sort. Generalize to many input arguments and implement linearity..

See Also: SchurFkt[outer], SchurFkt[outerP], SchurFkt[outerH], SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[outerM] - outer product of monomial symmetric functions (a la Rota-Stein)

Calling Sequence:

mfkt := outerM(mfkt1,mfkt2,...)

Parameters:

- mfkti : one or more monomial symmetric functions

Output:

- mfkt : monomial symmetric function

WARNING:

--none--

Description:

- Monomial symmetric functions are *the* classical symmetric functions. The outer product is involved since this basis is not multiplicative.
- Using the concatM product on the module spanned by monomial symmetric functions, we can deform this product to become the outer product by using the LaplaceM pairing. Such a deformation is called a **cliffordization**.
- A direct approach to the multiplication in monomial symmetric functions was given by Rota and Stein and yields an alternative way to compute the Littlewood-Richardson coefficients (up to the basis change m_to_s).

  (1) m_\mu . m_\nu = \sum_\lambda r^\lambda_\mu\nu m_\lambda

- In a Hopf algebraic notation with Sweedler indices, the outerM product reads

  (2) outerM(x,y) = concatM(LaplaceM(x_(1),y_(1)) , concatM(x_(2),y_(2)))

  where one should note that the LaplaceM pairing is algebra valued and not scalar valued!

- This function allows to construct the basis change p_to_m by using the fact that the p-basis is indeed multiplicative.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> outerM(m[1],m[2]);
  outerM(m[2],m[1]);
```

```
outerM(m[1],m[1]);
outerM(m[2],m[2]);
```

$$m_3 + m_{2,1}$$

$$m_3 + m_{2,1}$$

$$m_2 + 2\, m_{1,1}$$

$$m_4 + 2\, m_{2,2}$$

```
> outerM(m[2,1],m[3,1]);
outerM(m[2,1],m[2,1]);
```

$$m_{5,2} + m_{4,3} + 2\, m_{3,2,2} + m_{4,2,1} + 2\, m_{3,3,1} + 2\, m_{5,1,1} + 2\, m_{3,2,1,1}$$

$$m_{4,2} + 2\, m_{3,3} + 6\, m_{2,2,2} + 2\, m_{3,2,1} + 2\, m_{4,1,1} + 4\, m_{2,2,1,1}$$

Linearity and multiple entries (due to associativity):

```
> outerM(m[1],m[1],m[1]);
outerM(m[2],m[1],m[3]);
```

$$m_3 + 3\, m_{2,1} + 6\, m_{1,1,1}$$

$$m_6 + 2\, m_{3,3} + m_{4,2} + m_{5,1} + m_{3,2,1}$$

```
> outerM(2*m[1]+3*m[2,1],5*m[1,1]+m[2,2]);
```

$$10\, m_{2,1} + 30\, m_{1,1,1} + 17\, m_{3,2} + 32\, m_{2,2,1} + 30\, m_{3,1,1} + 45\, m_{2,1,1,1} + 3\, m_{4,3} + 6\, m_{3,2,2} + 3\, m_{4,2,1} + 9\, m_{2,2,2,1}$$

Critical cases:

```
> outerM(0);
outerM(m[0]);
outerM(m[]);  # should not occure and should possibly return
m[0]
```

$$0$$

$$m_0$$

$$m_{[\ ]}$$

```
>
```

Algorithm used:

The algorithm implements the cliffordization along the equation (2). This is done by using some internal helper functions of the module, which for efficiency reasons use the exponential representation of the partitions indexing the monomial symmetric functions. Most of these functions are used to implement the LaplaceM pairing. Note that the coproduct couterM is also needed to write down (2)

See Also: SchurFkt[outer], SchurFkt[LaplaceM], SchurFkt[concatM], SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[outerON] - outer product for orthogonal (symplectic) characters

Calling Sequence:

os := outerON(os1,os2,...)

Parameters:

- osi : one or more (orthogonal / symplectic) Schur functions

Output:

- os : (orthogonal / symplectic) Schur function

WARNING:

Note: SchurFkt does not distinguish between different types of Schur functions for different groups (yet). Hence orthogonal and symplectic Schur functions are denoted as s[\lambda] as are the classical Schur functions for GL

Description:

- We denote the characters of the orthogonal and symplectic Schur functions by s[\lambda] (do not confuse with GL characters!). outerON implements the Clebsch-Gordan decomposition of the tensor products of such characters w.r.t. the orthogonal (symplectic) group. This is done using the branching process.
- It is a well known result of Newell and Littlewood, that the decomposition of the tensor product of orthogonal characters (also denoted as [\lambda]) and symplectic characters (also denoted as <\lambda>) is given by:

  (1a) [\mu] . [\nu] = \sum_{\zeta} [\mu/\zeta . \nu/\zeta]

  (1b) <\mu> . < \nu> = \sum_{\zeta} <\mu/\zeta . \nu/\zeta>

  The fact that these decompositions are identical is an accident of this pair of groups (see BF, P.D. Jarvis, "A Hopf laboratory of symmetric functions", J. Phys.A: Math. Gen: 37(5) 2004:1633-1663)
- outerON can take one or more arguments and is commutative, associative and unital w.r.t. s[0]. In the stable limit this HA is another isomorphic incarnation of the symmetric function Hopf algebra /\ (see BF, P.D. Jarvis, R.C. King, "On the Hopf algebra structure of the character rings of orthogonal and symplectic groups"arXiv:0704.2029)
- **Note:** outerON is not graded by weight but only filtered!
- Note that the outer coproduct of the outer Hopf algebra for orthogonal and symplectic groups are no longer finitary expressions but formal power series. At this place it is worth using vertex operator algebra techniques which are beyond the current version of SchurFkt.

Examples:

```
> restart:with(SchurFkt):
```

```
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> outerON(s[1],s[1]);  # see additonal term s[0]
  outerON(s[1],s[1],s[1]);
  outerON(s[3],s[3]);
```

$$s_2 + s_{1,1} + s_0$$

$$s_3 + 2\, s_{2,1} + 3\, s_1 + s_{1,1,1}$$

$$s_6 + s_{5,1} + s_{4,2} + s_{3,3} + s_4 + s_{3,1} + s_{2,2} + s_2 + s_{1,1} + s_0$$

Multilinearity and associativity:

```
> outerON(s[3],outerON(s[1],s[2,1]));
  outerON(outerON(s[3],s[1]),s[2,1]);
```

$$2\, s_1 + 4\, s_3 + 6\, s_{2,1} + 2\, s_{1,1,1} + s_{6,1} + 2\, s_{5,2} + 2\, s_{5,1,1} + s_{4,3} + 3\, s_{4,2,1} + s_{3,3,1} + 2\, s_5 + 6\, s_{4,1} + 5\, s_{3,2} + 5\, s_{3,1,1} + 3\, s_{2,2,1} + s_{3,2,2} + s_{4,1,1,1} + s_{3,2,1,1} + s_{2,1,1,1}$$

$$2\, s_1 + 4\, s_3 + 6\, s_{2,1} + 2\, s_{1,1,1} + s_{6,1} + 2\, s_{5,2} + 2\, s_{5,1,1} + s_{4,3} + 3\, s_{4,2,1} + s_{3,3,1} + 2\, s_5 + 6\, s_{4,1} + 5\, s_{3,2} + 5\, s_{3,1,1} + 3\, s_{2,2,1} + s_{3,2,2} + s_{4,1,1,1} + s_{3,2,1,1} + s_{2,1,1,1}$$

```
> outerON(2*s[2]+3*s[1,1],s[3,2,1]+s[4]);
```

$$2\, s_2 + 2\, s_6 + 5\, s_{5,1} + 7\, s_{4,2} + 5\, s_{3,3} + 5\, s_4 + 10\, s_{3,1} + 5\, s_{2,2} + 5\, s_{2,1,1} + 15\, s_{3,2,1} + 2\, s_{5,2,1} + 5\, s_{4,3,1} + 5\, s_{4,2,2} + 5\, s_{4,2,1,1} + 5\, s_{3,3,2} + 5\, s_{3,3,1,1} + 5\, s_{3,2,2,1} + 8\, s_{4,1,1} + 5\, s_{3,1,1,1} + 5\, s_{2,2,2} + 5\, s_{2,2,1,1} + 3\, s_{3,2,1,1,1}$$

Critical cases:

```
> outerON(0);
  outerON(s[]);  # should not occure, but should possibly return s[0]
```

$$0$$

$$s_{[\ ]}$$

```
>
```

Algorithm used:

outerON is implemented along the Newell-Littlewood formula (1a).

See Also: SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[outerP] - the outer product of symmetric functions in the power sum basis

Calling Sequence:

pfkt := dummy(pfkt1,pfkt2,...)

Parameters:

- pfkti : one or more power sum symmetric functions

Output:

- pfkt : a power sum symmetric function

WARNING:

--none--

Description:

- Power sums symmetric functions for a multiplicative basis. Hence they have the same outer product as outerH and outerE . However, the outer product fro power sum symmetric functions has further importance, since this is a orthogonal basis w.r.t. the Schur-Hall scalar product .
- Power sum symmetric functions correspond to characters of the symmetric group. The multiplicativity of the power sum functions and the Schur-Hall scalar product between a one part power sum and a hook shaped Schur function and the outer coproduct of Schur functions allows to compute the Murnaghan-Nakayama rule in yet another way.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
```

```
> outerP(p[1],p[1]);
  outerP(p[1,1],p[1,1]);
  outerP(p[2,1],p[3,1]);
  outerP(p[2,2],p[4,3,1]);   # unordered concatenation
```

$$p_{1,1}$$

$$p_{1,1,1,1}$$

$$p_{3,2,1,1}$$

$$p_{4,3,2,2,1}$$

Check how this is related to characters:

```
> MurNak([3],[3]);
  CharHook(s[3],p[3]);
  #
```

```
MurNak([3,3],[5,1]); ## input p,s
Scalar(p_to_s(p[3,3]),s[5,1]);
```

$$1$$

$$1$$

$$-1$$

$$-s_0$$

Critical cases:

```
> outerP(0);
  outerP(p[]);
```

$$0$$

$$p_{[\ ]}$$

```
>
```

Algorithm used:

On monomials this is just the unsorted concatenations (= concatenate the indices and sort the entries to get a standard representation)

See Also: SchurFkt[MurNak], SchurFkt[MurNak2], SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[outerS] - outer product of two Schur functions

Calling Sequence:

o1 := outer(s1,s2)
o1 := outer(s1,s2,...sn)

Parameters:

- s1,s2 : S-function polynoms (an element of `type/spolynom`) of any weight
  s1,...,sn : S-function polynoms, n ranging in the positive integers

Output:

- o1 : S-function polynom of type `type/sfktpolynom`.

WARNING:

make sure to use Schur function basis, no type checking yet !!.

NOTE: This is the same function as SchurFkt[outer].

Description:

- The outer Schur function product multiplies two Schur polynomials using the Littlewood-Richardson rule. If the two Schur functions are of weight (number of boxes in the corresponding Young diagram, aka partition) n and m, then the output is a polynomial with integer coefficients in Schur polynomials of weight n+m.
- The outer product is multilinear over the integers (fractions of integers) as the ring of symmetric functions Λ is build as a polynomial ring over the integers or the fractions.
- In principle this product depends not on a chosen basis. However, the combinatorial algorithms which implement this product depend heavily on the fact that the Schur polynomials are related to the basis of irreducibles of Λ.
- For the outer product in other bases see SchurFkt[outerH,E,P].
- NOTE: A Schur polynomial is represented as s[lambda], where lambda is a partition of n. This object is treated as an identity, even if itself is a polynomials in (possibly infinite many) variables x1,...xn,... . A S-function monom is a single Schur polynom with coefficient unity, an S-function term is a Schur polynom with an integer (fractional) prefactor, a S-function polynomial is an integer (fractional) linear combination of Schur polynomials.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

Some some elementary examples of outer products of Schur polynomials.

```
> outerS(s[1],s[0])=s[1];
  outerS(s[0],s[3,1,1])=s[3,1,1];
```

$$s_1 = s_1$$

$$s_{3,1,1} = s_{3,1,1}$$

This shows that s[0]=1(x)=1 is the unit of the polynomial ring.

```
> outerS(s[1],s[1]);
  outerS(s[2],s[1]);
  outerS(s[1,1],s[1]);
  outerS(s[4,2,2,1,1],s[1]);
```

$$s_2 + s_{1,1}$$

$$s_3 + s_{2,1}$$

$$s_{2,1} + s_{1,1,1}$$

$$s_{5,2,2,1,1} + s_{4,3,2,1,1} + s_{4,2,2,2,1} + s_{4,2,2,1,1,1}$$

Adding a single box to another single box can be done in two ways, horizontally and vertically, giving two boxes in a row {2} or two boxes in a column {11}. More generally a single box is added in all possible ways to the pile of boxes of another Schur polynomial. No multiplicities can occur.

```
> outerS(s[1],s[1],s[1]);
  outerS(s[2]+s[1,1],s[1]);
```

$$s_3 + 2\, s_{2,1} + s_{1,1,1}$$

$$s_3 + 2\, s_{2,1} + s_{1,1,1}$$

Putting a polynomial into the product (or iterating the product) may give multiplicities. These coefficients make up the multiplication table of the ring and can be obtained by the Schur-Hall scalar product:

```
> out:=outerS(s[1],s[2]);
  Scalar(out,s[3]);
  Scalar(out,s[2,1]);
  Scalar(out,s[1,1,1]);
```

$$out := s_3 + s_{2,1}$$

$$s_0$$

$$s_0$$

$$0$$

Note that s[0]=1 is the unit of the product.

```
> outer(s[4]+3*s[2,1],s[0]);
```

$$s_4 + 3\, s_{2,1}$$

Critical cases:

```
> outer(0);
```

```
outer(s[]); # should not occur but should possibly return s[0]
```

$$0$$

$$s_{[\ ]}$$

```
>
```

See Also: SchurFkt[Overview], SchurFkt[couter], e.g., SchurFkt[outerH] (see many other incarnations of outerA for A-basis)

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[p_to_m] - basis change from power sum to monomial symmetric functions

Calling Sequence:

mfkt := p_to_m(pfkt)

Parameters:

- pfkt : power sum symmetric function

Output:

- mfkt : monomial symmetric function

WARNING:

--none--

Description:

- The basis change from power sum symmetric function to monomials symmetric functions is important at many areas of symmetric function theory. Here it is implemented (for the first time) using Rota-Stein cliffordization techniques for the outer product of monomials symmetric functions (see outerM).

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> p_to_m(p[3]);  # on primitive elements this is an isomorphism
  p_to_m(p[7]);
  p_to_m(p[1]);
  p_to_m(p[9]);
```

$$m_3$$

$$m_7$$

$$m_1$$

$$m_9$$

Note the difference for identical and non indentical indices coming from the concatM product underlying outerM

```
> p_to_m(p[2,1]),outerM(m[2],m[1]);
  p_to_m(p[1,1]),outerM(m[1],m[1]);
```

$$m_3 + m_{2,1}, m_3 + m_{2,1}$$

$$m_2 + 2\, m_{1,1}, m_2 + 2\, m_{1,1}$$

Note that the basis change can be described by looking at all possible contractions of indices in a

successive way adding certain multiplicities if the same indices are contracted

```
> p_to_m(p[3,2,1]);  # [3,2,1], [3+1,2], [3,2+1], [3+2,1],
  [3+2+1]
  outerM(m[3],m[2],m[1]);
  p_to_m(p[2,2,2]);  # [2,2,2], [2+2,2], [2+2+2]
  outerM(m[2],m[2],m[2]);
```

$$m_6 + m_{5,1} + 2\, m_{3,3} + m_{4,2} + m_{3,2,1}$$

$$m_6 + m_{5,1} + 2\, m_{3,3} + m_{4,2} + m_{3,2,1}$$

$$m_6 + 3\, m_{4,2} + 6\, m_{2,2,2}$$

$$m_6 + 3\, m_{4,2} + 6\, m_{2,2,2}$$

Critical cases:

```
> p_to_m(0);
  p_to_m(p[]);
  p_to_m(p[0]);
```

$$0$$

$$m_0$$

$$m_0$$

```
>
```

Algorithm used:

We write p[\mu] = p[\mu1] . p[\mu2] .... p[\mul] and use this isomorphism p[n]=m[n] for all n on primitive elements. Then we multiply back using the outer product in the monomial symmetric function basis outerM .

See Also:

SchurFkt[Overview], see there many functions called a_to_b for basis change from A-basis to B-basis.

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[p_to_s] - basis change from power sum symmetric functions to Schur functions

Calling Sequence:

sfkt := p_to_s(pfkt)

Parameters:

- pfkt : power sum symmetric function

Output:

- sfkt : Schur function

WARNING:

--none--

Description:

- This basis change of symmetric functions encodes the *inverse* transformations as the character table of the symmetric group. It is realized by an implementation of the Murnaghan-Nakayama rule (with rim-represented Young diagrams for efficiency reasons) for the s_to_p basis change and inverts the whole matrix.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
```

```
> p_to_s(p[1]);
  p_to_s(p[2]);
  p_to_s(p[1,1]);
  p_to_s(p[3]);
  p_to_s(p[2,1]);
  p_to_s(p[1,1,1]);
```

$$s_1$$

$$s_2 - s_{1,1}$$

$$s_2 + s_{1,1}$$

$$s_3 - s_{2,1} + s_{1,1,1}$$

$$s_3 - s_{1,1,1}$$

$$s_3 + 2\, s_{2,1} + s_{1,1,1}$$

```
> InvcharTab:=proc(N)
    local prt;
```

```
    prt:=map(x->s[op(x)],PartNM(N,N));
    linalg[matrix](nops(prt),nops(prt),

  (i,j)->subs(s[0]=1,Scalar(s[op(prt[i])],p_to_s(p[op(prt[j])]))))
  );
  end proc:
```

```
> InvcharTab(2);
  evalm(2!*%^(-1));
  #
  InvcharTab(3);
  evalm(3!*%^(-1));
```

$$\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 2 \\ 1 & -1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 2 & -2 & 2 \\ 3 & 0 & -3 \\ 1 & 2 & 1 \end{bmatrix}$$

p_to_s is inverse to s_to_p:

```
> p_to_s(p[3,2,2,1]);
  s_to_p(%);
```

$$s_8 + s_{6,2} - 2\, s_{6,1,1} + s_{5,3} + s_{5,1,1,1} - s_{4,4} - s_{4,3,1} + s_{4,1,1,1,1} + 2\, s_{3,3,2} - s_{3,2,2,1} - 2\, s_{3,1,1,1,1,1} - s_{2,2,2,2} + s_{2,2,2,1,1} + s_{2,2,1,1,1,1} + s_{1,1,1,1,1,1,1}$$

$$p_{3,2,2,1}$$

Critical cases:

```
> p_to_s(0);
  p_to_s(p[]);
  p_to_s(p[0]);
```

$$0$$

$$s_0$$

$$s_0$$

```
>
```

Algorithm used:

Direct usage of the Murnaghan-Nakayama rule.

See Also: SchurFkt[s_to_p], SchurFkt[Overview], see also other functions a_to_b for basis

changes from A-basis to B-basis.

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[part2Frob] - translates a standard partition (shape) into Frobenius notation

Calling Sequence:

Frob := part2Frob(prt)

Parameters:

- prt : a partition in standard notation (list integer)

Output:

- Frob : a Frobenius partition list list integer

WARNING:

--none--

Description:

- Partitions have many representations. SchurFkt uses standard partitions (list of the row length of the Young diagram) as a standard. The Frobenius notation for partitions is given by two equally long lists describing the arm length and leg length of the diagram seen from the diagonal boxes.
- Let (i,j) be the indices (Anglo-Saxon = matrix indexing) of a box in a Young diagram. The diagonal boxes have index (i,i) and the Frobenius rank of a partition is the number of such diagonal boxes. The arm length of (i,i) is given by the numbers (may be zero) of boxes in the row right to (i,i). The leg length is given by boxes below (i,i) (may be zero). A Frobenius partition is then given by the lists of arm and leg lengths.
- Hook shaped partitions have Frobenius representations by exactly one arm and one leg (may be zero).

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> part2Frob([3,3,1,1]);
  part2Frob([1,1,1,1]);
  part2Frob([4]);
  part2Frob([3,2,1]);
  part2Frob([4,3,2,1]);
  part2Frob([4,4,4,4]);
```

$$[[2, 1], [3, 0]]$$

$$[[0], [3]]$$

$$[[3], [0]]$$

$$[[2, 0], [2, 0]]$$
$$[[3, 1], [3, 1]]$$
$$[[3, 2, 1, 0], [3, 2, 1, 0]]$$

Hooks:

```
> part2Frob([4,1,1,1]);
  part2Frob([1,1,1,1]);
  part2Frob([2,1,1,1,1,1]);
  part2Frob([9,1,1,1]);
```

$$[[3], [3]]$$
$$[[0], [3]]$$
$$[[1], [5]]$$
$$[[8], [3]]$$

Frob2part is the inverse to part2Frob:

```
> part2Frob([6,6,2,2,1]);
  Frob2part(%);
```

$$[[5, 4], [4, 2]]$$
$$[6, 6, 2, 2, 1]$$

Critical cases:

```
> part2Frob([]);
```

$$[[\ ], [\ ]]$$

```
>
```

Algorithm used:

Just detecting arm and leg lengths and preparing the two lists.

See Also: SchurFkt[Frob2part], SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[part2mset] - translates a partition in standard representation into an multiset (exponential) representation

Calling Sequence:

mprt := part2mset(prt)

Parameters:

- prt : a partition in standard notation (a shape)

Output:

- mset : a partition in multiplicity notation

WARNING:

--mainly internal use--

Description:

- part2mset transforms a partition in standard notation (list integer, a shape of a Young diagram) into the multiplicity representation. This reads \mu=(\mu_1,\mu_2,...,\mu_l) = [1^r_1 2^r_2 ... n^r_n] where the weight (number of boxes) of the partition is \sum \mu_i = n. We have hence that the r_i are an integer solution of 1r_1+2r_2+3r_3...+nr_n=n.
- multiplicity represented partitions are used internally and are not intended for user purpose.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> part2mset([3,3,1,1]); # 2 ones 2 threes
  part2mset([1,1,1,1]); # 4 ones
  part2mset([4,3,2,1]); # one one, one two , one three one four
```

$$[2, 0, 2]$$

$$[4]$$

$$[1, 1, 1, 1]$$

part2mset is the inverse of mset2part:

```
> mset2part([2,0,0,2,1,1]);
  part2mset(%);
```

$$[6, 5, 4, 4, 1, 1]$$

$$[2, 0, 0, 2, 1, 1]$$

Critical cases:

```
> part2mset([]);
  part2mset([0]);
```

[ ]

[ ]

[>

Algorithm used:

[It is just a matter of counting to get the multiplicities and the procedure does just this.

See Also: SchurFkt[mset2part], SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[PartNM] - returns a list of partitions of N with parts of size at most M.

Calling Sequence:

lst := PartNM(N,M)

Parameters:

- N,M : integers

Output:

- lst : list of partitions in standard representation and ordered by AlexComp.

WARNING:

Note that the Maple combinat package does use the inverse ordering of partitions.

Description:

- PartNM returns a list of partitions of N into parts of site at most M..
- PartNM returns the list in AlexComp order.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> PartNM(3,3);
  PartNM(3,2);
```

$$[[3], [2, 1], [1, 1, 1]]$$

$$[[2, 1], [1, 1, 1]]$$

```
> PartNM(4,1);
  PartNM(4,2);
  PartNM(4,3);
  PartNM(4,4); # all partitions of 4
  PartNM(4,5); # no restriction anyhow
```

$$[[1, 1, 1, 1]]$$

$$[[2, 2], [2, 1, 1], [1, 1, 1, 1]]$$

$$[[3, 1], [2, 2], [2, 1, 1], [1, 1, 1, 1]]$$

$$[[4], [3, 1], [2, 2], [2, 1, 1], [1, 1, 1, 1]]$$

$$[[4], [3, 1], [2, 2], [2, 1, 1], [1, 1, 1, 1]]$$

```
>
```

Algorithm used:

The algorithm is the same as in the Maple combinat package with the difference that the order of partitions is given in the reversed order.

See Also: SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[plethP] - plethysm in the power sum basis

Calling Sequence:

pfkt := plethP(pfkt1,pfkt2)

Paameters:

- pfkti : power sum symmetric functions

Output:

- pfkt : power sum symmetric functions

WARNING:

--disfunctional--

Description:

- plethP computes the plethysm of power sum symmetric functions.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
```

```
> plethP(p[3],p[2]);
  plethP(p[2,2],p[1,1]);
  plethP(3*p[2],p[3]);
  plethP(p[2],p[3]+p[2]);
  plethP(p[2]+p[3],p[2]);   # <===
```

$$p_6$$

$$p_{2,2,2,2}$$

$$3\,p_6$$

$$p_6 + p_4$$

$$p_6 + p_4$$

```
>
```

Critical input:

```
> plethP(p[5]); ## error intended
  plethP(p[0],p[4]);
  plethP(p[4],p[0]);
Error, invalid input: SchurFkt:-plethP uses a 2nd argument, y, which is
missing
```

$$p_0$$

$$p_0$$

```
> plethP(p[3,2],0);  # illegal input -> error
  plethP(0,p[3,2]);  # illegal input -> error
Error, (in &t) too many levels of recursion

Error, (in type/pfktmonom) too many levels of recursion
>
```

Algorithm used:

--void--

See Also: SchurFkt[pleth], SchurFkt[cplethP], SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: plethS - computes plethysm of two Schur function polynoms
plethSnm - computes plethysm of two sfunctions of the form s[n] (one part complete symmetric functions)

Calling Sequence:

sfkt3 := plethS(sfkt1,sfkt2)
sfkt3 := plethSnm(sfkt1,sfkt2)

Parameters:

- sfkt1,sfk2 : Schur function polynoms, sfkt2 must be a polynomial over the integers!

Output:

- sfkt3 : an Schur function polynom

WARNING:

Note that the second input has to be a Schur function polynomial over the integers. This procedure of plethysm is based on the notation s_\lambda[s_\mu] = P[Q] = plethS(P,Q), which is _opposite_ to the notion 'pleth' in SCHUR!

Description:

- The plethysm is the functional composition of Schur maps. We consider symmetric functions as maps on G-modules, where G is a group acting on the modules (GL(n), or S_m). The notation for plethysm is plethS(P,Q) = P[Q].
- Plethysms are computed via the power sum symmetric functions. This is not optimal but works out.
- Plethysm is left distributive (linear in the first factor):
  (P1+P1)[Q] = P1[Q] + P2[Q]
- Plethysm is not right distributive, but enjoys a set of relations with the outer and inner product. Let Q=Q^+ - Q^-. We consider the Schur positive part Q^+ (coefficient in the positive integers) and let Q^- be minus the Schur negative part. We have:

  i) P[Q+R] = P_(1)[Q] P_(2)[R] with \Delta(P)=P_(1)\otimes P_(2) the outer coproduct
  ii) P[QR] = P_[1][Q] P_[2][R] with \delta(P)=P_[1]\otimes P_[2] the inner coproduct
  iii) P[-Q] = (antipS(P))[Q] antipode acting on P

  Note that case ii) will not ocure during calculations, since unevaluated outer products are not defined in SchurFkt.
- Note that one has to distinguish between the negative one -1 in the ring and the negative of an indeterminate X ---> `-X'. These transformations differ (for explanations see the book of Lascoux on symmetric functions, published by the AMS).

- Evaluation of plethysms is very storage and processor intensive and our algorithm is not optimal, so be patient and/or modest.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

First we compute a few plethysms using the restricted function plethSnm which works only (no typecheck!) in one part sfunctions without prefactors!

```
> plethSnm(s[2],s[2]);
  plethSnm(s[3],s[2]);
  plethSnm(s[2],s[3]);
```

$$s_4 + s_{2,2}$$

$$s_6 + s_{2,2,2} + s_{4,2}$$

$$s_6 + s_{4,2}$$

The following **fails**:

```
> plethSnm(3*s[2],s[2]);  # intended error
Error, (in SchurFkt:-plethSnm) ambiguous multiple assignment
```

```
> plethSnm(s[2],3*s[3]);  # intended error
Error, (in SchurFkt:-plethSnm) ambiguous multiple assignment
```

Now we give some examples for more general plethysms.

```
> plethS(s[2,1],s[2,1]);
  plethS(s[2],s[2]-s[1,1]);
```

$$s_{6,2,1} + s_{5,4} + 2\,s_{5,3,1} + s_{5,2,2} + 2\,s_{5,2,1,1} + s_{5,1,1,1,1} + s_{4,4,1} + 3\,s_{4,3,2} + 3\,s_{4,3,1,1} + 3\,s_{4,2,2,1} + 2\,s_{4,2,1,1,1} + 3\,s_{3,3,2,1} + s_{3,3,1,1,1} + s_{3,2,2,2} + 2\,s_{3,2,2,1,1} + s_{3,2,1,1,1,1} + s_{2,2,2,2,1}$$

$$s_4 - s_{3,1} + s_{2,2}$$

We can now generate a wealth of series out of plethysms of standard type checking series, as the M,L series, A,B,C,D series etc...

```
> M[t].[2]=`+`(op(map(x->plethSnm(x,s[2]),getSfktSeries(M,4))));
  D[t] = getSfktSeries(D,8,1);
```

$$(M_t).[2] = s_0 + s_2 + s_4 + s_{2,2} + s_6 + s_{2,2,2} + s_{4,2} + s_8 + s_{6,2} + s_{4,4} + s_{4,2,2} + s_{2,2,2,2}$$

$$\mathrm{D}_t = s_0 + s_2 + s_4 + s_{2,2} + s_6 + s_{2,2,2} + s_{4,2} + s_8 + s_{6,2} + s_{4,4} + s_{4,2,2} + s_{2,2,2,2}$$

```
> M[t].[3]=subs(t=t^3,map(x->plethS(x,s[3]),getSfktSeries(M,4,t)));
  L[t].[3]=subs(t=t^3,map(x->plethS(x,s[3]),getSfktSeries(L,4,t)));
```

$$(M_t).[3] = s_0 + s_3\,t^3 + t^6\,(s_6 + s_{4,2}) + t^9\,(s_9 + s_{7,2} + s_{6,3} + s_{5,2,2} + s_{4,4,1}) +$$

$$t^{12}\left(s_{6,6}+s_{7,4,1}+s_{8,2,2}+s_{10,2}+s_{9,3}+s_{12}+s_{8,4}+s_{7,3,2}+s_{6,4,2}+s_{6,2,2,2}+s_{5,4,2,1}+s_{4,4,4}\right)$$

$$(L_t).[3]=s_0-s_3\,t^3+t^6\left(s_{5,1}+s_{3,3}\right)-t^9\left(s_{7,1,1}+s_{6,3}+s_{5,3,1}+s_{3,3,3}\right)$$
$$+t^{12}\left(s_{3,3,3,3}+s_{6,6}+s_{6,3,3}+s_{7,4,1}+s_{7,3,1,1}+s_{9,1,1,1}+s_{8,3,1}+s_{6,4,2}+s_{5,5,1,1}+s_{5,3,3,1}\right)$$

```
> M[t].[2,1]=subs(t=t^3,map(x->plethS(x,s[2,1]),getSfktSeries(M,4
  ,t)));
  L[t].[2,1]=subs(t=t^3,map(x->plethS(x,s[2,1]),getSfktSeries(L,4
  ,t)));
```

$$(M_t).[2,1]=s_0+t^3\,s_{2,1}+t^6\left(s_{3,1,1,1}+s_{4,2}+s_{2,2,2}+s_{3,2,1}\right)+t^9\left(s_{6,3}+s_{5,3,1}+s_{5,2,2}\right.$$
$$+s_{5,2,1,1}+s_{4,4,1}+s_{4,3,2}+s_{4,3,1,1}+2\,s_{4,2,2,1}+s_{4,2,1,1,1}+s_{4,1,1,1,1,1}+s_{3,3,3}+s_{3,3,2,1}$$
$$\left.+s_{3,3,1,1,1}+s_{3,2,2,2}+s_{3,2,2,1,1}\right)+t^{12}\left(2\,s_{3,3,3,2,1}+s_{4,4,3,1}+s_{7,4,1}+s_{7,3,1,1}+s_{6,4,1,1}\right.$$
$$+3\,s_{6,3,2,1}+2\,s_{5,4,3}+3\,s_{4,4,2,2}+s_{8,4}+s_{7,3,2}+s_{6,5,1}+2\,s_{6,4,2}+2\,s_{6,2,2,2}+s_{5,5,1,1}$$
$$+3\,s_{5,4,2,1}+2\,s_{5,3,2,2}+s_{5,1,1,1,1,1,1,1}+s_{4,2,2,2,1,1}+2\,s_{4,4,2,1,1}+s_{4,4,1,1,1,1}+3\,s_{4,3,2,2,1}$$
$$+s_{4,3,1,1,1,1,1}+2\,s_{4,2,2,2,2}+2\,s_{3,3,2,2,1,1}+3\,s_{5,3,3,1}+2\,s_{4,3,3,2}+3\,s_{4,3,3,1,1}+s_{4,4,4}$$
$$+3\,s_{4,3,2,1,1,1}+s_{4,2,2,1,1,1,1}+s_{6,3,1,1,1}+s_{6,2,2,1,1}+s_{6,2,1,1,1,1}+2\,s_{5,4,1,1,1}+4\,s_{5,3,2,1,1}$$
$$\left.+s_{5,3,1,1,1,1}+3\,s_{5,2,2,2,1}+2\,s_{5,2,2,1,1,1}+s_{5,2,1,1,1,1,1}\right)$$

$$(L_t).[2,1]=s_0-t^3\,s_{2,1}+t^6\left(s_{4,1,1}+s_{3,3}+s_{2,2,1,1}+s_{3,2,1}\right)-t^9\left(s_{6,1,1,1}+s_{5,3,1}+s_{5,2,2}\right.$$
$$+s_{5,2,1,1}+s_{4,4,1}+s_{4,3,2}+2\,s_{4,3,1,1}+s_{4,2,2,1}+s_{4,2,1,1,1}+s_{3,3,3}+s_{3,3,2,1}+s_{3,3,1,1,1}$$
$$\left.+s_{3,2,2,2}+s_{3,2,2,1,1}+s_{2,2,2,1,1,1}\right)+t^{12}\left(s_{3,3,3,3}+2\,s_{3,3,3,2,1}+2\,s_{4,4,3,1}+s_{7,3,1,1}+s_{6,4,1,1}\right.$$
$$+3\,s_{6,3,2,1}+2\,s_{5,4,3}+3\,s_{4,4,2,2}+s_{8,1,1,1,1}+s_{3,2,2,2,1,1,1}+s_{7,2,2,1}+2\,s_{6,4,2}+s_{6,2,2,2}$$
$$+2\,s_{5,5,1,1}+3\,s_{5,4,2,1}+2\,s_{5,3,2,2}+s_{4,2,2,2,1,1}+2\,s_{4,4,2,1,1}+2\,s_{4,4,1,1,1,1}+3\,s_{4,3,2,2,1}$$
$$+s_{4,2,2,2,2}+2\,s_{3,3,2,2,1,1}+s_{3,3,2,1,1,1,1}+s_{3,2,2,2,2,1}+s_{2,2,2,2,1,1,1,1}+3\,s_{5,3,3,1}+s_{4,3,3,2}$$
$$+3\,s_{4,3,3,1,1}+3\,s_{4,3,2,1,1,1}+s_{4,2,2,1,1,1,1}+s_{7,2,1,1,1}+2\,s_{6,3,1,1,1}+s_{6,2,2,1,1}+s_{6,2,1,1,1,1}$$
$$\left.+3\,s_{5,4,1,1,1}+4\,s_{5,3,2,1,1}+s_{5,3,1,1,1,1}+2\,s_{5,2,2,2,1}+s_{5,2,2,1,1,1}\right)$$

```
> M[t].[1,1,1]=subs(t=t^3,map(x->plethS(x,s[1,1,1]),getSfktSeries
  (M,4,t)));
  L[t].[1,1,1]=subs(t=t^3,map(x->plethS(x,s[1,1,1]),getSfktSeries
  (L,4,t)));
```

$$(M_t).[1,1,1]=s_0+s_{1,1,1}\,t^3+t^6\left(s_{2,1,1,1,1}+s_{2,2,2}\right)$$
$$+t^9\left(s_{3,3,3}+s_{3,2,2,1,1}+s_{2,2,2,1,1,1}+s_{3,1,1,1,1,1,1}\right)+t^{12}\left(s_{3,3,3,1,1,1}+s_{3,2,2,2,1,1,1}\right.$$
$$+s_{2,2,2,2,2,2}+s_{4,2,2,2,2}+s_{4,1,1,1,1,1,1,1,1}+s_{3,3,2,2,1,1}+s_{3,2,2,1,1,1,1,1}+s_{4,3,3,1,1}+s_{4,4,4}$$
$$\left.+s_{4,2,2,1,1,1,1}\right)$$

$$(L_t).[1,1,1]=s_0-s_{1,1,1}\,t^3+t^6\left(s_{1,1,1,1,1,1}+s_{2,2,1,1}\right)$$
$$-t^9\left(s_{3,3,1,1,1}+s_{3,2,2,2}+s_{2,2,2,1,1,1}+s_{2,2,1,1,1,1,1}+s_{1,1,1,1,1,1,1,1,1}\right)+t^{12}\left(s_{3,3,3,3}\right.$$

$$+ s_{2,2,2,1,1,1,1,1,1} + s_{2,2,1,1,1,1,1,1,1,1} + s_{3,3,1,1,1,1,1,1} + s_{3,2,2,2,1,1,1} + s_{2,2,2,2,2,2}$$
$$+ s_{4,4,1,1,1,1} + s_{4,3,2,2,1} + s_{3,3,2,2,1,1} + s_{3,3,2,1,1,1,1} + s_{2,2,2,2,1,1,1,1} + s_{1,1,1,1,1,1,1,1,1,1,1,1})$$

Plethysms can be used to count several types of graphs, using Redfield-Polya counting theory.

Suppose, we have 4 vertices, 1 of order 3, two of order 2 and one of order 1 (order = connectivity). We represent these vertices by complete symmetric functions (which are dual to the monomial symmetric functions) but write them as Schur functions h[n]=s[n]

```
> vert:=outer(s[3],s[2],s[2],s[1]);
```

$$vert := s_8 + 3\, s_{7,1} + 5\, s_{6,2} + 3\, s_{6,1,1} + 5\, s_{5,3} + 6\, s_{5,2,1} + s_{5,1,1,1} + 2\, s_{4,4} + 5\, s_{4,3,1} + 3\, s_{4,2,2}$$
$$+ 2\, s_{4,2,1,1} + 2\, s_{3,3,2} + s_{3,3,1,1} + s_{3,2,2,1}$$

To count the correct type of graphs, we need to extract the plethysm for s_4[Q] or s_(1,1,1,1)[Q] from this result, where Q is a weight 2 Schur function s[2] or s[1,1]. These computations where exemplified by Burge:

```
> MultilinesLoops:=plethS(s[4],s[2]);
  MultilinesNoLoops:=plethS(s[4],s[1,1]);
  SingleLinesLoops:=plethS(s[1,1,1,1],s[2]);
  SingleLinesNoLoops:=plethS(s[1,1,1,1],s[1,1]);
```

$$MultilinesLoops := s_8 + s_{6,2} + s_{4,4} + s_{4,2,2} + s_{2,2,2,2}$$
$$MultilinesNoLoops := s_{4,4} + s_{3,3,1,1} + s_{2,2,2,2} + s_{2,2,1,1,1,1} + s_{1,1,1,1,1,1,1,1}$$
$$SingleLinesLoops := s_{5,1,1,1} + s_{4,3,1}$$
$$SingleLinesNoLoops := s_{3,2,2,1} + s_{4,1,1,1,1}$$

To count these we need to pair the two results via the Schur-Hall scalar product:

```
> "number of ML  :",subs(s[0]=1,Scalar(MultilinesLoops,vert));
  "number of MNL :",subs(s[0]=1,Scalar(MultilinesNoLoops,vert));
```

"number of ML :", 11

"number of MNL :", 3

```
> "number of SLL  :",subs(s[0]=1,Scalar(SingleLinesLoops,vert));
  "number of SLNL
  :",subs(s[0]=1,Scalar(SingleLinesNoLoops,vert));
```

"number of SLL :", 6

"number of SLNL :", 1

Let x be a vertex without loop and O a vertex with a loop. The graphs we are concerned with look like:

```
x === x   x -- x    O --- x   O    O   x === x    O    O
|       | \/        |   |        \       \
|       | /\        |   |         \      \    ...etc
x --- x   x    x    O    x   x --- x    O    x    O    x
```

Plethysm has many more applications in group representation theory, for example in branching rules etc.

We give now some test cases for critical input of plethysms, especially for **zero** the numerical null = 0 and the zero
partition Schur function s[0]. There is some asymetry for the s[0] case

```
> plethS(0,0);
  plethS(0,s[2]);
  plethS(s[3],0);
```

$$0$$
$$0$$
$$0$$

```
> plethS(s[0],s[3]);
  plethS(s[0],s[3,2]);
  plethS(s[3],s[0]);
  plethS(s[3,2],s[0]); # --> numerical zero!
```

$$s_0$$
$$s_0$$
$$s_0$$
$$0$$

Further critical cases are:

```
> plethS(a*s[2]+b*s[1,1],s[3]);
  plethS(s[2],a*s[1,1]+b*s[2]); # Error intended, only integer
  polynomials allowed
  plethS(s[2],2*s[1,1]+2*s[2]);
  plethS(s[2],s[1,1]+s[1,1]+s[2]+s[2]);
```

$$a\,(s_6 + s_{4,2}) + b\,(s_{5,1} + s_{3,3})$$

```
Error, (in SchurFkt:-plethS) Second input must be a polynomial over the
integers, but received , b*s[2]
```

$$3\,s_4 + 6\,s_{2,2} + 5\,s_{3,1} + 3\,s_{1,1,1,1} + 5\,s_{2,1,1}$$
$$3\,s_4 + 6\,s_{2,2} + 5\,s_{3,1} + 3\,s_{1,1,1,1} + 5\,s_{2,1,1}$$

```
>
```

Algorithm used:

Stupidly using the translation to power sums and their relatively easy plethysm formulae, then converting back to sfunctions.

See Also: SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: plethS - computes plethysm of two Schur function polynoms
plethSnm - computes plethysm of two sfunctions of the form s[n] (one part complete symmetric functions)

Calling Sequence:

sfkt3 := plethS(sfkt1,sfkt2)

Parameters:

- sfkt1,sfk2 : Schur function polynoms, sfkt2 must be a polynomial over the integers!

Output:

- sfkt3 : an sfunction polynom

WARNING:

Note that the second input has to be a sfunction polynomial over the integers. This procedure of plethysm is based on the notation s_\lambda[s_\mu] = P[Q] = plethS(P,Q), which is _opposite_ to the notion 'pleth' in SCHUR!

Description:

- The plethysm is the functional composition of Schur maps. We consider symmetric functions as maps on G-modules, where G is a group acting on the modules (GL(n), or S_m). The notation for plethysm is plethS(P,Q)=P[Q].
- Plethysms are computed via the power symmetric functions. This is not optimal but works out.
- Plethysm is left distributive (linear in the first factor):
  (P1+P1)[Q] = P1[Q] + P2[Q]
- Plethysm is not right distributive, but enjoys a set of relations with the outer and inner product. Let Q=Q^+ - Q^- be the Schur positive (coefficient in the positive integers) and Q^- be the Schur negative part. We have:

  P[Q+R] = P_(1)[Q] P_(2)[R] with \Delta(P)=P_(1)\otimes P_(2) the outer coproduct
  P[QR] = P_[1][Q] P_[2][R] with \delta(P)=P_[1]\otimes P_[2] the inner coproduct
  P[-Q] = (antipS(P))[Q] antipode acting on P
- Note that one has to distinguish between the negative one -1 in the ring and the negative of an indeterminate X ---> `-X'. These transformations differ (for explanations see the book of Lascoux on symmetric functions AMS).
- Evaluation of plethysms is very storage and processor intensive and our algorithm is not optimal, so be patient and/or modest.

Examples:

```
> restart:with(SchurFkt):
```

```
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

First we compute a few plethysms using the restricted function plethSnm which works only (no typecheck!) in one part sfunctions without prefactors!

```
> plethSnm(s[2],s[2]);
  plethSnm(s[3],s[2]);
  plethSnm(s[2],s[3]);
```

$$s_4 + s_{2,2}$$

$$s_6 + s_{2,2,2} + s_{4,2}$$

$$s_6 + s_{4,2}$$

Now we give some examples for more general plethysms.

```
> plethS(s[2,1],s[2,1]);
  plethS(s[2],s[2]-s[1,1]);
```

$$s_{6,2,1} + s_{5,4} + 2\,s_{5,3,1} + s_{5,2,2} + 2\,s_{5,2,1,1} + s_{5,1,1,1,1} + s_{4,4,1} + 3\,s_{4,3,2} + 3\,s_{4,3,1,1} + 3\,s_{4,2,2,1} + 2\,s_{4,2,1,1,1} + 3\,s_{3,3,2,1} + s_{3,3,1,1,1} + s_{3,2,2,2} + 2\,s_{3,2,2,1,1} + s_{3,2,1,1,1,1} + s_{2,2,2,2,1}$$

$$s_4 - s_{3,1} + s_{2,2}$$

We can now generate a wealth of series out of plethysms of standard type checking series, as the M,L series, A,B,C,D series etc...

```
> M[t].[2]=`+`(op(map(x->plethSnm(x,s[2]),getSfktSeries(M,4))));
  D[t] = getSfktSeries(D,8,1);
```

$$(M_t).[2] = s_0 + s_2 + s_4 + s_{2,2} + s_6 + s_{2,2,2} + s_{4,2} + s_8 + s_{6,2} + s_{4,4} + s_{4,2,2} + s_{2,2,2,2}$$

$$\mathrm{D}_t = s_0 + s_2 + s_4 + s_{2,2} + s_6 + s_{2,2,2} + s_{4,2} + s_8 + s_{6,2} + s_{4,4} + s_{4,2,2} + s_{2,2,2,2}$$

```
> M[t].[3]=subs(t=t^3,map(x->plethS(x,s[3]),getSfktSeries(M,4,t)));
  L[t].[3]=subs(t=t^3,map(x->plethS(x,s[3]),getSfktSeries(L,4,t)));
```

$$(M_t).[3] = s_0 + s_3\,t^3 + t^6\,(s_6 + s_{4,2}) + t^9\,(s_9 + s_{7,2} + s_{6,3} + s_{5,2,2} + s_{4,4,1}) + t^{12}\,(s_{7,4,1} + s_{6,6} + s_{10,2} + s_{9,3} + s_{8,2,2} + s_{12} + s_{8,4} + s_{7,3,2} + s_{6,4,2} + s_{6,2,2,2} + s_{5,4,2,1} + s_{4,4,4})$$

$$(L_t).[3] = s_0 - s_3\,t^3 + t^6\,(s_{5,1} + s_{3,3}) - t^9\,(s_{7,1,1} + s_{6,3} + s_{5,3,1} + s_{3,3,3}) + t^{12}\,(s_{7,4,1} + s_{7,3,1,1} + s_{6,6} + s_{6,3,3} + s_{5,3,3,1} + s_{3,3,3,3} + s_{9,1,1,1} + s_{8,3,1} + s_{6,4,2} + s_{5,5,1,1})$$

```
> M[t].[2,1]=subs(t=t^3,map(x->plethS(x,s[2,1]),getSfktSeries(M,4,t)));
  L[t].[2,1]=subs(t=t^3,map(x->plethS(x,s[2,1]),getSfktSeries(L,4,t)));
```

$$(M_t).[2,1] = s_0 + t^3\,s_{2,1} + t^6\,(s_{3,1,1,1} + s_{4,2} + s_{2,2,2} + s_{3,2,1}) + t^9\,(s_{6,3} + s_{5,3,1} + s_{5,2,2}$$

$$+ s_{5,2,1,1} + s_{4,4,1} + s_{4,3,2} + s_{4,3,1,1} + 2\, s_{4,2,2,1} + s_{4,2,1,1,1} + s_{4,1,1,1,1,1} + s_{3,3,3} + s_{3,3,2,1}$$
$$+ s_{3,3,1,1,1} + s_{3,2,2,2} + s_{3,2,2,1,1}) + t^{12} (s_{7,4,1} + s_{7,3,1,1} + 3\, s_{4,4,2,2} + 3\, s_{4,3,2,1,1,1}$$
$$+ s_{4,2,2,1,1,1,1} + s_{4,2,2,2,1,1} + 2\, s_{5,4,3} + s_{6,4,1,1} + 3\, s_{6,3,2,1} + 3\, s_{5,3,3,1} + s_{8,4} + s_{7,3,2} + s_{6,5,1}$$
$$+ 2\, s_{6,4,2} + 2\, s_{6,2,2,2} + s_{5,5,1,1} + 3\, s_{5,4,2,1} + 2\, s_{5,3,2,2} + s_{5,1,1,1,1,1,1} + s_{4,4,4} + 2\, s_{4,4,2,1,1}$$
$$+ s_{4,4,1,1,1,1} + 3\, s_{4,3,2,2,1} + s_{4,3,1,1,1,1,1} + 2\, s_{4,2,2,2,2} + 2\, s_{3,3,2,2,1,1} + 2\, s_{4,3,3,2} + 3\, s_{4,3,3,1,1}$$
$$+ 2\, s_{3,3,3,2,1} + s_{4,4,3,1} + s_{6,3,1,1,1} + s_{6,2,2,1,1} + s_{6,2,1,1,1,1} + 2\, s_{5,4,1,1,1} + 4\, s_{5,3,2,1,1}$$
$$+ s_{5,3,1,1,1,1} + 3\, s_{5,2,2,2,1} + 2\, s_{5,2,2,1,1,1} + s_{5,2,1,1,1,1,1})$$

$$(L_t).[2,1] = s_0 - t^3 s_{2,1} + t^6 (s_{4,1,1} + s_{3,3} + s_{2,2,1,1} + s_{3,2,1}) - t^9 (s_{6,1,1,1} + s_{5,3,1} + s_{5,2,2}$$
$$+ s_{5,2,1,1} + s_{4,4,1} + s_{4,3,2} + 2\, s_{4,3,1,1} + s_{4,2,2,1} + s_{4,2,1,1,1} + s_{3,3,3} + s_{3,3,2,1} + s_{3,3,1,1,1}$$
$$+ s_{3,2,2,2} + s_{3,2,2,1,1} + s_{2,2,2,1,1,1}) + t^{12} (s_{7,3,1,1} + 3\, s_{4,4,2,2} + 3\, s_{4,3,2,1,1,1} + s_{4,2,2,1,1,1,1}$$
$$+ s_{4,2,2,2,1,1} + 2\, s_{5,4,3} + s_{6,4,1,1} + 3\, s_{6,3,2,1} + 3\, s_{5,3,3,1} + s_{3,3,3,3} + s_{8,1,1,1,1} + s_{7,2,2,1}$$
$$+ 2\, s_{6,4,2} + s_{6,2,2,2} + 2\, s_{5,5,1,1} + 3\, s_{5,4,2,1} + 2\, s_{5,3,2,2} + 2\, s_{4,4,2,1,1} + 2\, s_{4,4,1,1,1,1}$$
$$+ 3\, s_{4,3,2,2,1} + s_{4,2,2,2,2} + s_{3,2,2,2,1,1,1} + 2\, s_{3,3,2,2,1,1} + s_{3,3,2,1,1,1,1} + s_{4,3,3,2} + s_{3,2,2,2,2,1}$$
$$+ s_{2,2,2,2,1,1,1,1} + 3\, s_{4,3,3,1,1} + 2\, s_{3,3,3,2,1} + 2\, s_{4,4,3,1} + s_{7,2,1,1,1} + 2\, s_{6,3,1,1,1} + s_{6,2,2,1,1}$$
$$+ s_{6,2,1,1,1,1} + 3\, s_{5,4,1,1,1} + 4\, s_{5,3,2,1,1} + s_{5,3,1,1,1,1} + 2\, s_{5,2,2,2,1} + s_{5,2,2,1,1,1})$$

```
> M[t].[1,1,1]=subs(t=t^3,map(x->plethS(x,s[1,1,1]),getSfktSeries
  (M,4,t)));
  L[t].[1,1,1]=subs(t=t^3,map(x->plethS(x,s[1,1,1]),getSfktSeries
  (L,4,t)));
```

$$(M_t).[1,1,1] = s_0 + s_{1,1,1}\, t^3 + t^6 (s_{2,1,1,1,1} + s_{2,2,2})$$
$$+ t^9 (s_{3,3,3} + s_{3,2,2,1,1} + s_{2,2,2,1,1,1} + s_{3,1,1,1,1,1,1}) + t^{12} (s_{3,3,3,1,1,1} + s_{2,2,2,2,2,2}$$
$$+ s_{4,2,2,1,1,1,1} + s_{4,4,4} + s_{4,2,2,2,2,2} + s_{4,1,1,1,1,1,1,1,1} + s_{3,2,2,2,1,1,1} + s_{3,3,2,2,1,1}$$
$$+ s_{3,2,2,1,1,1,1,1} + s_{4,3,3,1,1})$$

$$(L_t).[1,1,1] = s_0 - s_{1,1,1}\, t^3 + t^6 (s_{1,1,1,1,1,1} + s_{2,2,1,1})$$
$$- t^9 (s_{3,3,1,1,1} + s_{3,2,2,2} + s_{2,2,2,1,1,1} + s_{2,2,1,1,1,1,1} + s_{1,1,1,1,1,1,1,1,1}) + t^{12} (s_{2,2,2,2,2,2}$$
$$+ s_{2,2,2,1,1,1,1,1,1} + s_{2,2,1,1,1,1,1,1,1,1} + s_{3,3,3,3} + s_{4,4,1,1,1,1} + s_{4,3,2,2,1} + s_{3,3,1,1,1,1,1,1}$$
$$+ s_{3,2,2,2,1,1,1} + s_{3,3,2,2,1,1} + s_{3,3,2,1,1,1,1} + s_{2,2,2,2,1,1,1,1} + s_{1,1,1,1,1,1,1,1,1,1,1,1})$$

```
>
```

Plethysms can be used to count several types of graphs, using Redfield-Polya counting theory.

Suppose, we have 4 vertices, 1 of order 3, two of order 2 and one of order 1 (order = connectivity). We represent these vertices by their complete symmetric functions (which are dual to the monomial symmetric functions) but write them as Schur functions h[n]=s[n]

```
> vert:=outer(s[3],s[2],s[2],s[1]);
```

$$vert := s_8 + 3\,s_{7,1} + 5\,s_{6,2} + 3\,s_{6,1,1} + 5\,s_{5,3} + 6\,s_{5,2,1} + s_{5,1,1,1} + 2\,s_{4,4} + 5\,s_{4,3,1} + 3\,s_{4,2,2} + 2\,s_{4,2,1,1} + 2\,s_{3,3,2} + s_{3,3,1,1} + s_{3,2,2,1}$$

To count the correct type of graphs, we need to extract the plethysm for s_4[Q] or s_(1,1,1,1)[Q] from this result, where Q is a weight 2 Schur function s[2],s[1,1]. These computations where exemplified by Burge:

```
> MultilinesLoops:=plethS(s[4],s[2]);
  MultilinesNoLoops:=plethS(s[4],s[1,1]);
  SingleLinesLoops:=plethS(s[1,1,1,1],s[2]);
  SingleLinesNoLoops:=plethS(s[1,1,1,1],s[1,1]);
```

$$MultilinesLoops := s_8 + s_{6,2} + s_{4,4} + s_{4,2,2} + s_{2,2,2,2}$$

$$MultilinesNoLoops := s_{4,4} + s_{3,3,1,1} + s_{2,2,2,2} + s_{2,2,1,1,1,1} + s_{1,1,1,1,1,1,1,1}$$

$$SingleLinesLoops := s_{5,1,1,1} + s_{4,3,1}$$

$$SingleLinesNoLoops := s_{3,2,2,1} + s_{4,1,1,1,1}$$

To count these we need to pair the two results via the Schur-Hall scalar product:

```
> "number of ML  :",subs(s[0]=1,Scalar(MultilinesLoops,vert));
  "number of MNL :",subs(s[0]=1,Scalar(MultilinesNoLoops,vert));
```

"number of ML :", 11

"number of MNL :", 3

```
> "number of SLL  :",subs(s[0]=1,Scalar(SingleLinesLoops,vert));
  "number of SLNL
  :",subs(s[0]=1,Scalar(SingleLinesNoLoops,vert));
```

"number of SLL :", 6

"number of SLNL :", 1

Let x be a vertex without loop and O a vertex with a loop. The graphs we are concerned with look like:

```
x === x   x -- x    O --- x   O    O   x === x   O   O
|       | \/        |  |        \       \
|       | /\        |  |         \      \    ...etc
x --- x   x   x     O    x   x --- x   O    x   O   x
```

Plethysm has many more applications in group representation theory, for example in branching rules etc.

```
>
```

Algorithm used:

Stupidly using the translation to power sums and their relatively easy plethysm formulae, then converting back to sfunctions.

See Also: SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[s_to_h] - basis change from Schur function basis to complete symmetric function basis

Calling Sequence:

pfkt := s_to_h(sfkt)

Parameters:

- sfkt : Schur function

Output:

- hfkt : complete symmetric function

WARNING:

--slow, uses matrix inversion: -> should use Jacobi-Trudi or a Hopf-Laplace algorithm

Description:

- s_to_h transforms a Schur function into a complete symmetric function. This transformation is governed by the Kostka matrix. We conclude from

  (1) s_\mu = \sum_\nu K_\mu\nu m_\nu
  (2) <h_\rho | s_\mu> = \sum_\mu K_\mu\nu <h_\rho|m_\nu> = K_\mu\rho

  that this transformation is obtained via K^(-1).

- A better way to compute this transition is to use the Jacobi-Trudi formula

  (3) s_\mu = det(h_{\mu_i-i+j})

  giving a direct combinatorial device to produce the result.

- A more complicated but potentially more effective way to compute this transition would e to use the Laplace property of the outer coproduct and product to evaluate the transition using a multiplicative basis inserted into the expression

  (4) s_\mu = \sum_\nu <s_\mu| m_nu> h_nu
  = \sum_\nu\rho <s_\mu|p_\rho>1/z_\rho <p_\rho|m_\nu> h_\nu
  = \sum_\nu\mu \sum_(\mu)(\nu) \prod_(i,j)^length(\rho) 1/z_\rho <s_\mu(i)|p_\rho_i> <p_rho_j| m_\nu(j)> h_\nu

  which implements the Jacobi-Trudi expansion via a Hopf algebraic mechanism.

Examples:

```
> restart:with(SchurFkt):
```

```
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> s_to_h(s[3,3,1]);
  s_to_h(s[2,2,1,1]);
```

$$h_{5,2} - h_{4,3} - h_{4,2,1} + h_{3,3,1}$$

$$-h_{5,1} + h_{4,2} + h_{4,1,1} - h_{3,1,1,1} - h_{2,2,2} + h_{2,2,1,1}$$

```
> 2*s_to_h(s[2])+3*s_to_h(s[1,1]);
  s_to_h(2*s[2]+3*s[1,1]);
```

$$-h_2 + 3\,h_{1,1}$$

$$-h_2 + 3\,h_{1,1}$$

Inverse transformation is h_to_s:

```
> h_to_s(3*h[2,1]+5*h[4,2,1]);
```

$$3\,s_3 + 3\,s_{2,1} + 5\,s_7 + 10\,s_{6,1} + 10\,s_{5,2} + 5\,s_{5,1,1} + 5\,s_{4,3} + 5\,s_{4,2,1}$$

```
> s_to_h(%);
```

$$3\,h_{2,1} + 5\,h_{4,2,1}$$

Critical cases:

```
> s_to_h(s[0]);
  s_to_h(0);
```

$$h_0$$

$$0$$

```
>
```

Algorithm used:

Quick and dirty hack, no documentation intended.

See Also: SchurFkt[h_to_s], SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[s_to_p] - basis change from Schur functions to power symmetric functions

Calling Sequence:

pfkt := s_to_p(sfkt)

Parameters:

- sfkt : Schur function

Output:

- pfkt : power sum symmetric function

WARNING:

--none--

Description:

- The basis change from Schur function to power sum symmetric functions. This function relates the characters of the symmetric group (via the Murnaghan-Nakayama rule) to the Littlewood-Richardson rule, via the Laplace expansion.
- We find the following formulae:

(1) s_\mu = \sum_\nu <s_\mu | p_\nu> 1/z_\nu p_\nu
= \sum_\nu 1/z_\nu \chi^\mu(\nu) p_\nu

by the Murnaghan-Nakayama rule

(2) s_\mu = \sum_\nu <s_\mu | p_\nu > 1/z_\nu p_\nu
= \sum_\nu \prod_i^length(\nu) < s_\mu(i) | p_\nui > p_\nu

using the Littlewood-Richardson coefficients for the *outer coproduct*.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
```

```
> s_to_p(s[4,3]);
  s_to_p(s[1]);
  s_to_p(p[2]);
  s_to_p(p[1,1]);
```

$$-\frac{1}{10}p_{5,2}-\frac{1}{10}p_{5,1,1}+\frac{1}{12}p_{4,3}-\frac{1}{12}p_{4,1,1,1}+\frac{1}{9}p_{3,3,1}-\frac{1}{24}p_{3,2,2}+\frac{1}{12}p_{3,2,1,1}-\frac{1}{72}p_{3,1,1,1,1}$$

$$+\frac{1}{24}p_{2,2,1,1,1}+\frac{1}{60}p_{2,1,1,1,1,1}+\frac{1}{360}p_{1,1,1,1,1,1,1}$$

$$p_1$$

$$\frac{1}{2}p_2+\frac{1}{2}p_{1,1}$$

$$-\frac{1}{2}p_2+\frac{1}{2}p_{1,1}$$

```
> p_to_s(p[3,3,3]);
  s_to_p(%);
```

$$s_9-s_{8,1}+s_{7,1,1}+3\,s_{6,3}-3\,s_{6,2,1}+2\,s_{6,1,1,1}-3\,s_{5,4}+3\,s_{5,2,2}-2\,s_{5,1,1,1,1}+3\,s_{4,4,1}$$
$$-3\,s_{4,3,2}+2\,s_{4,1,1,1,1,1}+6\,s_{3,3,3}-3\,s_{3,3,2,1}+3\,s_{3,3,1,1,1}+3\,s_{3,2,2,2}-3\,s_{3,2,1,1,1,1}$$
$$+s_{3,1,1,1,1,1,1}-3\,s_{2,2,2,2,1}+3\,s_{2,2,2,1,1,1}-s_{2,1,1,1,1,1,1,1}+s_{1,1,1,1,1,1,1,1,1}$$

$$p_{3,3,3}$$

Critical cases:

```
> s_to_p(0);
  s_to_p(s[0]);
```

$$0$$

$$p_0$$

```
>
```

Algorithm used:

The algorithm is an implementation of the formula (1) using the Murnaghan-Nakayama rule .

See Also: SchurFkt[p_to_s], SchurFkt[Overview] (see Overview for more basis changes a_to_b from A-basis to B-basis)

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[s_to_x] - translation of a Schur function into a polynom in variables x_i

Calling Sequence:

xpoly := s_to_x(sfkt,N)

Parameters:

- sfktmon : Schur function
  N : nonnegative integer

Output:

- xpoly : polynom in variables xi, i in [1..N]

WARNING:

--none--

Description:

- Schur functions for a finite number of variables are polynomials. We can represent a Schur function as such a polynom using the function s_to_x, specifying the number of variables. The outer product translates into the polynomial ring product of the polynoms.
- Note that if the number N of variables xi is less than the number of boxes of the partition (its weight) then some Schur functions may vanish and syzygies occur.
- The translation from Schur functions to polynomials is given by the expansion

  (1) s_\lambda(x1..xN) = \sum_\mu K\lambda\mu x_\mu

  where the sum is over all *compositions* of N. The Kostka numbers are computed using KostkaPC .

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> s_to_x(s[3],2);
  s_to_x(s[2,1],2);
  s_to_x(s[3]+s[2,1],2);
```

$$x_{3,0} + x_{2,1} + x_{1,2} + x_{0,3}$$

$$x_{2,1} + x_{1,2}$$

$$x_{3,0} + 2\,x_{2,1} + 2\,x_{1,2} + x_{0,3}$$

```
> s_to_x(s[1],3);
```

```
CompNM(1,3);
s_to_x(s[2],3);
CompNM(2,3);
s_to_x(s[3],3);
CompNM(3,3);
s_to_x(s[4],3);
CompNM(4,3);
```

$$x_{1,0,0} + x_{0,1,0} + x_{0,0,1}$$

$$[[1,0,0],[0,1,0],[0,0,1]]$$

$$x_{2,0,0} + x_{1,1,0} + x_{1,0,1} + x_{0,2,0} + x_{0,1,1} + x_{0,0,2}$$

$$[[2,0,0],[1,1,0],[1,0,1],[0,2,0],[0,1,1],[0,0,2]]$$

$$x_{3,0,0} + x_{2,1,0} + x_{2,0,1} + x_{1,2,0} + x_{1,1,1} + x_{1,0,2} + x_{0,3,0} + x_{0,2,1} + x_{0,1,2} + x_{0,0,3}$$

$$[[3,0,0],[2,1,0],[2,0,1],[1,2,0],[1,1,1],[1,0,2],[0,3,0],[0,2,1],[0,1,2],[0,0,3]]$$

$$x_{4,0,0} + x_{3,1,0} + x_{3,0,1} + x_{2,2,0} + x_{2,1,1} + x_{2,0,2} + x_{1,3,0} + x_{1,2,1} + x_{1,1,2} + x_{1,0,3} + x_{0,4,0} + x_{0,3,1} + x_{0,2,2} + x_{0,1,3} + x_{0,0,4}$$

$$[[4,0,0],[3,1,0],[3,0,1],[2,2,0],[2,1,1],[2,0,2],[1,3,0],[1,2,1],[1,1,2],[1,0,3],[0,4,0],[0,3,1],[0,2,2],[0,1,3],[0,0,4]]$$

```
> s_to_x(s[3,1],3);
```

$$x_{3,1,0} + x_{3,0,1} + x_{2,2,0} + 2\,x_{2,1,1} + x_{2,0,2} + x_{1,3,0} + 2\,x_{1,2,1} + 2\,x_{1,1,2} + x_{1,0,3} + x_{0,3,1} + x_{0,2,2} + x_{0,1,3}$$

```
> x_to_s(%); ## error in x_to_s
```

$$s_{3,1} - 3\,s_{1,1,1,1}$$

Critical cases:

```
> s_to_x(s[0],2);
  s_to_x(0,3);
```

$$x_{0,0}$$

$$0$$

```
>
```

Algorithm used:

An implementation of (1).

See Also: SchurFkt[x_to_s], SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[Scalar] - the Schur-Hall scalar product of symmetric function for Schur functions

Calling Sequence:

int := Scalar(s1,s2)

Parameters:

- s1,s2 : Schur functions

Output:

- int : an integer times the Schur function s[0]=1.

WARNING:

In functioning version of this function may return an integer, not the Schur function s[0] times an integer.

Description:

- The Schur-Hall scalarproduct (introduced by Redfield earlier) renders the Schur functions to be an orthonormal basis. This reflects the fact that the Schur functions correspond to irreducible GL-characters. Due to Schur's lemma there is one or non isomorphism between such characters if they are isomorphic or not.
- The Schur-Hall scalar product is used to identify the dual Hopf algebra of the Hopf algebra of symmetric functions /\ to be isomorphic to the later. Hence /\ is a selfdual Hopf algebra. This fact leads to a wealth of combinatorial identities.
- The Schur-Hall scalar product is the inverse (in a certain categorical sense, as a tangle it is a cup) of the Cauchy kernel (as a tangle a cap):

  (1) C(x,y) = \prod_(i ,j) 1 / (1-x_iy_j) = \sum_\lambda s_\lambda(x) s_\lambda(y)

  where the product is over all letters x_i and y_j of the alphabets X and Y and the sum is over all partitions \lambda.
- Note that the Schur-Hall scalar product identifies as modules the dual space /\^* with /\.
- A proper name for this function would be ScalarSS due to our naming conventions.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
> Scalar(s[2],s[1]);
```

```
Scalar(s[2],s[2]);
```

$$0$$

$$s_0$$

Scalar is multilinear

```
> Scalar(s[1,1]+s[2],s[1,1]+s[2]);
```

$$2\, s_0$$

```
>
```

Algorithm used:

Scalar is implemented straight forwardly by comparing the partitions.

See Also: SchurFkt[Overview], (see also under ScalarAB where A,B are other basis types)

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[ScalarP] - ScalarP computes the scalarproduct between two power sum symmetric functions

Calling Sequence:

int := ScalarP(p1,p2)

Parameters:

- p1,p2 : power sum symmetric functions

Output:

- int : integer

WARNING:

--none--

Description:

- ScalarP is the instance of the Schur-Hall scalar product which renders the Schur functions orthonormal. The power sum functions form an orthogonal (not normalized, remind yourself we are working over the ring of integers) basis for this scalar product.
- We find for two power sum monomials:

  (1) < p_n | p_m> = n \delta_n,m

  (2) <p_\mu | p_\nu> = z_\mu \delta_\mu,\nu

  where z_\mu is a combinatorial normalization factor, see zee . And \delta-\mu,\nu is the Kronecker delta extended to partitions.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
```

```
> ScalarP(p[3],p[4]);
  ScalarP(p[3],p[3]);
```

$$0$$

$$3$$

```
> ScalarP(p[3,2,1,1],p[3,2,1,1])=zee([3,2,1,1]);
```

$$12 = 12$$

We have rudimentary type checking:

```
> ScalarP(h[3,2,1,1],p[3,2,1,1]);  # ERROR ScalarP checks for its
```

```
input
Error, (in SchurFkt:-ScalarP) wrong type
```

ScalarP is multilinear

```
> ScalarP(p[3,2]+p[1,1],p[3,2]+p[1,1]+p[4])=
  ScalarP(p[3,2],p[3,2]+p[1,1]+p[4])+ScalarP(p[1,1],p[3,2]+p[1,1]
  +p[4]);
```

$$8 = 8$$

```
>
```

Algorithm used:

ScalarP compares the input partitions component wise and computes the zee factor straightforwardly.

See Also:

SchurFkt[Scalar], SchurFkt[Overview] (there are further incarnations called ScalarAB for A B bases)

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[type/sfktmonom] - SchurFkt[type/sfktterm] - SchurFkt[type/sfktpolynom]

SchurFkt[type/sfktmonom] - SchurFkt[type/sfktterm] - SchurFkt[type/sfktpolynom]
SchurFkt[type/pfktmonom] - SchurFkt[type/pfktterm] - SchurFkt[type/pfktpolynom]
SchurFkt[type/hfktmonom] - SchurFkt[type/hfktterm] - SchurFkt[type/hfktpolynom]
SchurFkt[type/mfktmonom] - SchurFkt[type/mfktterm] - SchurFkt[type/mfktpolynom]
SchurFkt[type/efktmonom] - SchurFkt[type/efktterm] - SchurFkt[type/efktpolynom]
SchurFkt[type/ffktmonom] - SchurFkt[type/ffktterm] - SchurFkt[type/ffktpolynom]
SchurFkt[type/symfktmonom] - SchurFkt[type/symfktterm] - SchurFkt[type/symfktpolynom]

Calling Sequence:

bool := type(x,'sfktmonom');
bool := type(x,'<any SchurFkt type>');

Parameters:

- x : an expression
  'sfktmonom' : a valid type of the SchurFkt package
  <any SchurFkt type> any of the above listed types.

Output:

- bool : true or false

WARNING:

Not all SchurFkt procedures deal properly with type identification. Because of performance issues and due to the experimental nature of this code, many functions do **_no typechecking_** of their input. However, some functions will check if they receive proper input, these are critical cases.

Do **not assign** any value to the basis monomials used by the package, they are currently not protected to allow easy manipulations of them in worksheets. Hence if you assign say an elementary symmetric function e[2,1]:=foo to the value foo, the functionality of SchurFkt is surely spoiled!

There is one general type 'symfktmonom' which allows to check for any of the standard bases {s,p,h,m,e,f}. Later the user might add own bases to allow Littlewood, Hall-Littlewood or Macdonald symmetric functions.

Note that all(?) (but MLIN, FLAT) procedures are linearized wrt the basis is use, hence are linear over any expression not containing a particular basis monomial. To construct tensors one needs to use the Bigebra facility of tensor products.

Description:

- Types are used to check for basis elements in certain bases and to make procedures (multi)linear over that basis of symmetric functions..

Examples:

```
> restart:with(SchurFkt):
```

```
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

Just a few examples:

```
> type(s[2,2,1],'sfktmonom');
  type(s[2,2,1],'pfktmonom');    # => false
  type(s[2,2,1],'symfktmonom');
  type(p[2,2,1],'pfktmonom');
  type(p[2,2,1],'sfktmonom');    # => false
  type(p[2,2,1],'symfktmonom');
```

$$true$$
$$false$$
$$true$$
$$true$$
$$false$$
$$true$$

More generally we have:

```
> type(3*s[2,1],'sfktterm');
  selectremove(type,2*s[2,1],'sfktmonom');
  type(2*s[2]-2*s[1,1],'sfktpolynom');
```

$$true$$
$$s_{2,1}, 2$$
$$true$$

And over more general rings we get:

```
> selectremove(type,q^3*4*s[3,2,1],'sfktmonom');
```

$$s_{3,2,1}, 4\,q^3$$

So that for example the outer product is linear over 'anything', especially over q-polynomials:

```
> out1:=expand(outerS(q*s[2]+q^2*s[1,1],q^3*s[1,1]+q^2*s[2]+q*s[3]));
  vars:={seq( op(map(x->s[op(x)],PartNM(i,i))), i=1..5)};
  collect(out1,vars);
```

$$out1 := 2\,q^4\,s_{3,1} + 2\,q^4\,s_{2,1,1} + q^3\,s_4 + q^3\,s_{3,1} + q^3\,s_{2,2} + q^2\,s_5 + q^2\,s_{4,1} + q^2\,s_{3,2} + q^5\,s_{2,2} + q^5\,s_{2,1,1} + q^5\,s_{1,1,1,1} + q^3\,s_{4,1} + q^3\,s_{3,1,1}$$

$$vars := \{s_{2,2,1}, s_{2,1}, s_2, s_{1,1}, s_3, s_{3,1}, s_{2,1,1}, s_4, s_{2,2}, s_5, s_{4,1}, s_{3,2}, s_{1,1,1,1}, s_{3,1,1}, s_1, s_{1,1,1}, s_{2,1,1,1}, s_{1,1,1,1,1}\}$$

$$(q^3 + 2\,q^4)\,s_{3,1} + (2\,q^4 + q^5)\,s_{2,1,1} + q^3\,s_4 + (q^3 + q^5)\,s_{2,2} + q^2\,s_5 + (q^3 + q^2)\,s_{4,1} + q^3\,s_{3,1,1} + q^2\,s_{3,2} + q^5\,s_{1,1,1,1}$$

Algorithm

Note: The type functions check for the letter indexing the basis. So basis monoms are just 'indexed' object. These objects are not protected!

```
>
```

See Also: SchurFkt[Overview], Bigebra[define]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[skew] - (outer) skew of two Schur functions

Calling Sequence:

sfkt := skew(sfkt1,sfkt2)

Parameters:

- sfkti : Schur functions

Output:

- sfkt : Schur function

WARNING:

--none--

Description:

- The skew of two Schur functions is given as the Young diagram where from a larger partition \lambda a smaller partition (fitting graphically into \lambda) is removed. The removed partition is top-left adjusted:

```
1 1 1                  1 1
2 2    skewed by 1   =>   2 2
3                     3
```

  such skewed diagrams lead to Schur functions described by these skew diagrams. There exists an expansion of such skew diagrams into the Schur ufnction basis (indexed by ordinary diagrams). The procedure skew performs this expansion.

- The skew operation (denoted by '/') can be recognized as the adjoint of outer multiplication w.r.t the Schur-Hall scalar product.

  (1) < s_\lambda \ s_\mu s_\nu > = < s_\lambda / s_\mu | s_\nu > = < s_\mu^* s_\lambda | s_\nu>

  where D_\m u= s_\mu^* is the adjoint operator M_\mu of multiplying with s_\mu.

- One can show that D_\mu is a derivation called the Foulkes derivative. This derivation property is a consequence of the Laplace pairing property of the scalar product with respect to the outer product and coproduct. This fact is deeply rooted in invariant theory, see Richard Kane "Invariant Theory and Pseudo Reflection Groups", Springer NNNN.

- A major application of the skew comes from the fact that group branchings are obtained by skewing with respect to a series of Schur functions.

Examples:

```
> restart:with(SchurFkt):
```

```
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> skew(s[3,2,1],s[1]);  # above example
  skew(s[4,4],s[5]);    # s[5] does not fit into s[4,4]
```

$$s_{3,2} + s_{3,1,1} + s_{2,2,1}$$

$$0$$

```
> skew(2*s[2]+3*s[1,1],s[1]);
  skew(s[4,4],s[2]+s[1,1]);
```

$$5\, s_1$$

$$s_{4,2} + s_{3,3}$$

```
> M4:=getSfktSeries(M,4,1);
  skew(s[4],M4);
  skew(s[3,3],M4);
  skew(s[3,2,1],M4);
```

$$M4 := s_0 + s_1 + s_2 + s_3 + s_4$$

$$s_0 + s_1 + s_2 + s_3 + s_4$$

$$s_{3,3} + s_{3,2} + s_{3,1} + s_3$$

$$s_{3,2,1} + s_{3,2} + s_{3,1,1} + s_{2,2,1} + s_{3,1} + s_{2,2} + s_{2,1,1} + s_{2,1}$$

```
> L4:=getSfktSeries(L,4,1);
  truncWT(outer(M4,L4),4);  # mutually inverse (up to order 4)
```

$$L4 := s_0 - s_1 + s_{1,1} - s_{1,1,1} + s_{1,1,1,1}$$

$$s_0$$

```
> skew(s[3,2,1],M4);
  skew(%,L4);
```

$$s_{3,2,1} + s_{3,2} + s_{3,1,1} + s_{2,2,1} + s_{3,1} + s_{2,2} + s_{2,1,1} + s_{2,1}$$

$$s_{3,2,1}$$

This established the fact that A /(B.C) = ((A/B) /C)

Critical tests:

```
> skew(0,s[3,2]);
  skew(s[3,2],0);
  skew(0,0);
```

$$0$$

$$0$$

$$0$$

```
> skew(s[3,2],s[0]);
```

```
skew(s[3,2,1],s[]);  # should not occure as input
```

$$s_{3,\,2}$$

$$s_{3,\,2,\,1}$$

```
>
```

Algorithm used:

The skew is implemented as a direct consequence of formula (1) using the Littlewood-Richardson coefficients.

See Also: SchurFkt[outer], SchurFkt[couter], SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[sq_coeff] - returns the square of the coefficients of a Schur function

Calling Sequence:

int := sq_coeff(sfkt)

Parameters:

- sfkt : Schur function (polynomial)

Output:

- int : integer

WARNING:

--none--

Description:

- The square of the coefficients of a Schur function expression is an important number in group representation theory. sq_coeff computes this number.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
```

```
> sq_coeff(s[2,2,1],sfktmonom);
```

$$1$$

```
> sq_coeff(2*s[2]+s[1,1],sfktmonom);
  sq_coeff(s[2]+s[1,1],sfktmonom);
```

$$5$$

$$2$$

```
> for N from 2 to 5 do
  outer(s[1]$N);
  sq_coeff(%,sfktmonom);
  end do;
```

$$s_2 + s_{1,1}$$

$$2$$

$$s_3 + 2\, s_{2,1} + s_{1,1,1}$$

$$6$$

$$s_4 + 3\, s_{3,1} + 2\, s_{2,2} + 3\, s_{2,1,1} + s_{1,1,1,1}$$

$$24$$

$$s_5 + 4\, s_{4,1} + 5\, s_{3,2} + 6\, s_{3,1,1} + 5\, s_{2,2,1} + 4\, s_{2,1,1,1} + s_{1,1,1,1,1}$$

$$120$$

This checks that sq_coeff of s[1]^n = n!

Critical cases:

```
> sq_coeff(4*s[3],nonsensetype);
Error, invalid input: SchurFkt:-sq_coeff expects its 2nd argument, typ, to be
of type type, but received nonsensetype
```

```
> sq_coeff(0,sfktmonom);
```

$$0$$

```
>
```

Algorithm used:

sq_coeff very stupidly extracts the coefficients and returns them squared and summed up.

See Also: SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[truncWT] - truncates an Schur function expression by its weight

Calling Sequence:

prsfkt := truncWT(sfkt,N)

Parameters:

- sfkt : Schur function
  N : integer (weight)

Output:

- prsfkt : pruned Schur function where all terms of weight greater than N are missing

WARNING:

--none--

Description:

- truncWT just deletes terms of weight greater than N in an expression. This function is used when one computes modulo weight for example with Schur function series.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> truncWT(s[4]+s[5],5);
  truncWT(s[4]+s[5],4);
  truncWT(s[4]+s[5],3);
```

$$s_4 + s_5$$

$$s_4$$

$$0$$

```
> truncWT(s[3,2],4);
```

$$0$$

Critical cases:

```
> truncWT(s[2,1],0);
  truncWT(s[0]+s[3],0);
  truncWT(0,4);
```

$$0$$

$$s_0$$

$$0$$

```
>
```

Algorithm used:

We just delete the unwanted terms.

See Also: SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

Function: SchurFkt[x_to_s] - Translates a polynomial in the variables xi into a Schur function expression

Calling Sequence:

sfkt := x_to_s(xpoly)

Parameters:

- xpoly : a symmetric polynomial

Output:

- sfkt : Schur function

WARNING:

--disfunctional--

Description:

- Translates a polynomial in the variables xi into a Schur function expression.

Examples:

```
> restart:with(SchurFkt):
SchurFkt Version 1.0.2 (9 vi 2008) at your service
(c) 2003-2008 BF&RA, no warranty, no fitness for anything!
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> s_to_x(s[1],2);
  x_to_s(x[1,0]+x[0,1]);
  s_to_x(s[2]+s[1,1],2);
  x_to_s(x[2,0]+2*x[1,1]+x[0,2]);
```

$$x_{1,0} + x_{0,1}$$

$$s_1$$

$$x_{2,0} + 2\,x_{1,1} + x_{0,2}$$

$$s_2 + s_{1,1}$$

```
> s_to_x(s[2,1,1],4);
  x_to_s(%);  ## error
```

$$x_{2,1,1,0} + x_{2,1,0,1} + x_{2,0,1,1} + x_{1,2,1,0} + x_{1,2,0,1} + x_{1,1,2,0} + 3\,x_{1,1,1,1} + x_{1,1,0,2} + x_{1,0,2,1} + x_{1,0,1,2} + x_{0,2,1,1} + x_{0,1,2,1} + x_{0,1,1,2}$$

$$\frac{2}{3}\,s_{2,1,1} + s_{1,1,1,1}$$

```
>
```

Algorithm used:

[--void--

See Also: SchurFkt[s_to_x], SchurFkt[Overview]

Last modified: June 19, 2008/BF/RA

```
> restart:
> libname;
  with(SINGULARPLURALlink);
```

"C:\Maple11/Cliffordlib", "C:\Maple11/lib", "C:\Brachey.Troy/TNB", "C:\Maple11/SINGULARPLURALlinklib"

```
SINGULARPLURALlink 0.3 beta (May 15, 2008) says Hello...
3 function(s) exported
===>If you find this packages useful, please let us know about your derived work.
===>You can contact us at http://math.tntech.edu/rafal/ or http://clifford.physik.
uni-konstanz.de/~fauser/
===>This package requires CLIFFORD/Bigebra for Maple available from http://math.tn
tech.edu/rafal/
===>Singular:Plural requires nctools.lib and clifford.lib libraries to be installe
d. Consult http://www.singular.uni-kl.de/
Clifford package with 84 functions loaded...
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help
]
Bigebra package with 33 functions loaded...
```

[*PLURALforClink*, *PLURALforGlink*, *SINGULARlink*, *linkversion*]

```
> linkversion();
```

++

SINGULARPLURALlink - A Maple 11 Package for Linking Maple with Singular and Plural

Last revised: May 15, 2008 (Source file: SINGULARPLURALlink_M11_0515.mws)

(*) Department of Mathematics, Box 5054

Tennessee Technological University, Cookeville, TN 38505

tel: USA (931) 372-3662, fax: USA (931) 372-6353

rablamowicz@tntech.edu

http://math.tntech.edu/rafal/

($) Universit"at Konstanz, Fachbereich Physik, Fach M678

78457 Konstanz, Germany

Bertfried.Fauser@uni-konstanz.de

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

http://math.tntech.edu/rafal/

This package requires CLIFFORD/Bigebra for Maple available from http://math.tntech.edu/rafal/
Singular:Plural requires nctools.lib and clifford.lib libraries to be installed. Consult http://www.s
ingular.uni-kl.de

++++++++++++*This is SINGULARPLURALlink for Maple 11 version 0.3 with drp*+++++++\

++++++

>

Package **SINGULARPLURALlink** provides an interface between Maple and Singular with Plural for the purpose of computing commutative Groebner basis in Singular and non commutative Groebner bases in non commutative algebras, e.g., Grassmann, Clifford, etc., in its Plural extension. For now, the link permits only non commutative Groebner bases computation in Grassmann and Clifford algebras of a quadratic form. Input from Maple is sent to Singular:Plural, and results are returned to Maple.

NOTE: In order to compute with Grassmann or Clifford algebras in Maple, download and install package CLIFFORD/Bigebra by Ablamowicz and Fauser from http://www.math.tntech.edu/rafal/

NOTE: In order to compute with Grassmann or Clifford algebras in Singular:Plural, make sure that your Singular:Plural library located in C:\cygwin\usr\share\Singular\LIB directory contains two Singular:Plural libraries "nctools.lib" and "clifford.".

-- Procedure SINGULARlink provides an interface between Maple and Singular for the purpose of computing commutative Groebner basis in Singular and returning result back to Maple. It has been succsessfully tested with Maple 8, 9, 9.5, 10, 11; and 3-0-4. It can be easily modified for the use of other commands in Singular than "std" and "groebner". This procedures is the same as the one posted on Singular's web page

http://www.mathematik.uni-kl.de/ftp/pub/Math/Singular/misc/

as SINGULARlink2.mws, dated 05-Aug-2006. The only difference now is that this procedure that has been successfully tested, is now part of a package (Maple module) called "SINGULARPLURALlink".

Usage: Procedure SINGULARlink uses the following input arguments:

SINGULARlink(Id,C,tord,elimorder,input_for_Singular,input_for_Maple,wait);
SINGULARlink(Id,C,tord,elimorder,input_for_Singular,input_for_Maple,wait,'p');

- Id - type "list(polynom)" - a list of polynomials [f1, f2,..., fn] that generate ideal Id, that is, Id = <f1, f2, ..., fn>.
- C - type "nonnegint" - non negative characteristic for the ground field of polynomial coefficients, set 0 for the rationals
- tord - type "symbol" - one of total orders used by Singular/Plural: lp, dp, rp, Dp, ls, ds, Ds, or drp
  NOTE: In this version of the link, tord can really be anything as long as Singular:Plural can understand it. For example, in Maple we define a new order drp in n noncommutng variables (x(1..n)) as (a(1:n),rp).
  This will be then equivalent to Deg[InvLex] order or a degree order

such that x^\alpha > x^\beta if |\alpha| > |\beta|, or, in case of equality |\alpha| = |\beta|, if the right-most nonzero entry in $\alpha - \beta$ is positive.

This order has no special name in Singular:Plural, but we call it for short 'drp'.

- elimorder - type "list" - list of polynomial variables that are to be eliminated in the order in which they appear in the list. This list must be identical to the list of polynomial variables.
- input_for_Singular - type "string" - path to a directory where input file In.txt for Singular will be written to
- input_for_Maple - type "string" - path to a directory where output file Out.txt from Singular for Maple will be written to
- wait - type "nonnegint" or "symbol" - when used with a positive integer it will pass on that integer to "groebner(Id,wait)" as its wait parameter; when used as symbol 'infty', procedure "groebner(Id)" will be invoked

'p' - (optional) type symbol 'd' - for saving files to the disk; 't' for displaying In.txt to the terminal only (see examples below)

-- Procedure PLURALforGlink provides an interface between Maple and Plural for the purpose of computing non commutative Groebner basis in Grassmann algebra using Plural and returning result back to Maple. It has been succsessfully tested with Maple 11; and Singular 3-0-3 and 3-0-4.

Usage: Procedure PLURALforGlink uses the following input arguments:

PLURALforGlink(G,C,tord,vars,input_for_Singular,input_for_Maple,wait);
PLURALforGlink(G,C,tord,vars,input_for_Singular,input_for_Maple,wait,'p');

- G - type "list::({clipolynom,climon,clibasmom})" - a list of Clifford or Grassmann polynomials [f1, f2,..., fn] that generate ideal G that is, G = <f1, f2, ..., fn>, in a Grassmann algebra.
- C - type "nonnegint" - non negative characteristic for the ground field of polynomial coefficients, set 0 for the rationals
- tord - type "symbol" - one of total orders used by Singular/Plural: lp, dp, rp, Dp, ls, ds, Ds, or drp:

NOTE: In this version of the link, tord can really be anything as long as Singular:Plural can understand it. For example, in Maple we define a new order drp in n noncommutng variables (x(1..n)) as (a(1:n),rp).

This will be then equivalent to Deg[InvLex] order or a degree order such that x^\alpha > x^\beta if |\alpha| > |\beta|, or, in case of equality |\alpha| = |\beta|, if the right-most nonzero entry in $\alpha - \beta$ is positive.

This order has no special name in Singular:Plural, but we call it for short 'drp'.

- vars - type "list" - list of generators eg, eq,e2,e3,..., e9, of the Grassmann algebra. This list must be identical or larder than the list of variables appearing in the polynomials generating G.
- input_for_Singular - type "string" - path to a directory where input file In.txt for Singular will be

written to
- input_for_Maple - type "string" - path to a directory where output file Out.txt from Singular for Maple will be written to
- wait - type "nonnegint" or "symbol" - when used with a positive integer it will pass on that integer to "groebner(Id,wait)" as its wait parameter; when used as symbol 'infty', procedure "groebner(Id)" will be invoked
'p' - (optional) type symbol 'd' - for saving files to the disk; 't' for displaying In.txt to the terminal only (see examples below)

-- Procedure PLURALforClink provides an interface between Maple and Plural for the purpose of computing non commutative Groebner basis in Clifford algebra Cl(Q) of a quadratic form which could be degenerate, using Plural and returning result back to Maple. It has been succsessfully tested with Maple 11; and Singular 3-0-4.

Usage: Procedure PLURALforClink uses the following input arguments:

PLURALforClink(G,C,tord,vars,B,input_for_Singular,input_for_Maple,wait);
PLURALforClink(G,C,tord,vars,B,input_for_Singular,input_for_Maple,wait,'p');

- G - type "list::({clipolynom,climon,clibasmom})" - a list of Clifford or Grassmann polynomials [f1, f2,..., fn] that generate ideal G that is, G = <f1, f2, ..., fn>, in a Grassmann algebra.
- C - type "nonnegint" - non negative characteristic for the ground field of polynomial coefficients, set 0 for the rationals
- tord - type "symbol" - one of total orders used by Singular/Plural: lp, dp, rp, Dp, ls, ds, Ds, or drp
NOTE: In this version of the link, tord can really be anything as long as Singular:Plural can understand it. For example, in Maple we define a new order drp in n noncommutng variables (x(1..n)) as (a(1:n),rp).
This will be then equivalent to Deg[InvLex] order or a degree order such that x^\alpha > x^\beta if |\alpha| > |\beta|, or, in case of equality |\alpha| = |\beta|, if the right-most nonzero entry in $\alpha - \beta$ is positive.
This order has no special name in Singular:Plural, but we call it for short 'drp'.
- vars - type "list" - list of generators eg, eq,e2,e3,..., e9, of the Grassmann algebra. This list must be identical or larder than the list of variables appearing in the polynomials generating G.
- B - type "diagmatrix", a diagonal matrix that defines a quadratic form Q for Cl(Q). The size of this matrix must but at least equal to the largest index in generators appearing in the list vars
- input_for_Singular - type "string" - path to a directory where input file In.txt for Singular will be written to
- input_for_Maple - type "string" - path to a directory where output file Out.txt from Singular for Maple will be written to

- wait - type "nonnegint" or "symbol" - when used with a positive integer it will pass on that integer to "groebner(Id,wait)" as its wait parameter; when used as symbol 'infty', procedure "groebner(Id)" will be invoked
'p' - (optional) type symbol 'd' - for saving files to the disk; 't' for displaying In.txt to the terminal only (see examples below)

Comments:

1. Global variable "dirpathSexe" must be defined (see below) with a path to bash.exe (version 3-0-4) to execute Singular 3-0-4.
2. The transfer directories "input_for_Singular" and "input_for_Maple" need not be the same. If either one does not exist, it will be created. These directories are not automatically deleted when Singular is finished computing unless appropriate lines system(cat("rm -r ",pathS)); and system(cat("rm -r ",pathM)); are unremarked. Be VERY CAREFUL when you unremark these lines as you may accidently delete directories you wanted to keep!
3. The eighth parameter 'p' is optional and can only take values 't' and 'd'. When 't' is used, the input script for Singular is displayed to the terminal only: No transfer directory is created and nothing is written to the hard disk. When 'd' is used, the input file In.txt is created, written to the hard disk, then Singular is invoked to execute it and write its output as file Out.txt. Then, Out.txt file is read in and displayed to the terminal/screen. When this parameter is not used, the procedure behaves like if it were used with the parameter 'd'.
4.The output of this procedure is the Groebner basis returned by Singular's "groebner" command. One can easily use "std" command instead.
5. Some error messaging is built in.
6. No warranties of any kind can be made. Permission is granted to use this procedure and modify it at will, although I would appreciate an acknowledgment.
7. A complete interface module allowing a Maple user to use other command from Singular could easily be written.

References

[1] T. Stokes, Groebner Bases in Exterior Algebra, J. Automated Reasoning 6 (1990), 233--250

[2] J.L. Bueso, J.Gomez-Torrecillas and A. Verschoren, "Algorithmic Methods in Non-Commutative Algebra - Applications to Quantum Groups", Kluwer Academic Publishers, Dordrecht-Boston-London, 2003

[3] R. Ab{\l}amowicz and B. Fauser, `CLIFFORD' - Maple 11 package for Clifford algebra computations, ver. 11, http://math.tntech.edu/rafal/cliff11/, 2008.

[4] R. Ab{\l}amowicz and B. Fauser, `GfG' - Maple 11 package for \Grobner Bases for Grassmann Algebras, ver. 0.5, http://math.tntech.edu/rafal/GfG/, 2008.

[5] T. Brachey, `TNB - A Maple 11 Package for Computing Groebner bases in Grassmann algebras`, ver. delta 0.3 (March 28, 2008), http://www.math.tntech.edu/rafal/TNB0328/TNB_delta_03_28.zip

[6] ViktorLevandovskyy, Manipulations within Clifford Algebras in Singular:Plural, Private Communication, 2006

Rafal Ablamowicz
Department of Mathematics
Tennessee Technological University, Box 5054
Cookeville, TN 38501
Email: rablamowicz@tntech.edu
Tel: (931) 372-3662
Fax: (931) 372-6353

Cookeville, May 15, 2008

First, we define some paths and directories:

```
>
```

1. Procedure **'linkversion()'** returns information about the current version of the link,s authors' names, etc.

```
> linkversion();
```

++

SINGULARPLURALlink - A Maple 11 Package for Linking Maple with Singular and Plural

Last revised: May 15, 2008 (Source file: SINGULARPLURALlink_M11_0515.mws)

Copyright 2008 by Rafal Ablamowicz (*) and Bertfried Fauser ($)

(*) Department of Mathematics, Box 5054

Tennessee Technological University, Cookeville, TN 38505

tel: USA (931) 372-3662, fax: USA (931) 372-6353

rablamowicz@tntech.edu

http://math.tntech.edu/rafal/

($) Universit"at Konstanz, Fachbereich Physik, Fach M678

78457 Konstanz, Germany

Bertfried.Fauser@uni-konstanz.de

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

http://math.tntech.edu/rafal/

This package requires CLIFFORD/Bigebra for Maple available from http://math.tntech.edu/rafal/ Singular:Plural requires nctools.lib and clifford.lib libraries to be installed. Consult http://www.s\ ingular.uni-kl.de

++++++++++++This is SINGULARPLURALlink for Maple 11 version 0.3 with drp+++++++\ +++++

```
>
```

First, we define some paths and directories:

```
> #dirpathSexe:="C:/cygwin/usr/local/Singular/2-0-3/ix86-Win/Singula
  r.exe": #A path to Singular executable VERSION 2-0-3
  dirpathSexe:="C:/cygwin/bin/bash -l C:/cygwin/bin/Singular";
  #A path to Singular executable VERSION 3-0-4
  transfer_directoryM:="C:/transferM":      #A user-defined path to
  a directory where Singular should write its output file Out.txt
  for Maple.
  transfer_directoryS:="C:/transferS":      #A user-defined path to
  a directory where Maple should write its input file In.txt for
  Singular.
  input_for_Maple:=transfer_directoryM;     #A path to a directory
  where Singular will write its output file Out.txt for Maple.
  input_for_Singular:=transfer_directoryS;  #A path to a directory
  where Maple will write its input file In.txt for Singular.
```

dirpathSexe := "C:/cygwin/bin/bash -l C:/cygwin/bin/Singular"

input_for_Maple := "C:/transferM"

input_for_Singular := "C:/transferS"

```
>
```

Example 1: To test option 't' in the procedure **SINGULARlink**. When used with option 't', the procedure outputs a script file In.txt containing input for Singular to the terminal/screen. It does not write it to the hard disk nor it invokes Singular to execute it:

```
> f:=2*x^10+4*x^9;
  g:=4*y-8-x;
  h:=f*g^2+z^4;
  k:=-5*x^10+x^9*z^2;
  m:=x*y*z;
  L:=map(expand,[f,g]);
  vars:=sort(convert(indets(L),list));
```

```
N:=nops(vars);
```

$$f := 2\,x^{10} + 4\,x^{9}$$
$$g := 4\,y - 8 - x$$
$$h := (2\,x^{10} + 4\,x^{9})\,(4\,y - 8 - x)^{2} + z^{4}$$
$$k := -5\,x^{10} + x^{9}\,z^{2}$$
$$m := x\,y\,z$$
$$L := [2\,x^{10} + 4\,x^{9}, 4\,y - 8 - x]$$
$$vars := [x, y]$$
$$N := 2$$

```
> SINGULARlink([f,g,h,k,m],0,lp,[x,y,z],input_for_Singular,input_for
  _Maple,'infty','t');
```

Trying to create and display input file for Singular... proceeding...

```
ring R = 0,(x,y,z),lp;
ideal I =
2*x^10
+ 4*x^9
,
4*y
+ -8
+ -x
,
32*x^10*y^2
+ -160*x^10*y
+ -16*x^11*y
+ 192*x^10
+ 36*x^11
+ 2*x^12
+ 64*x^9*y^2
+ -256*x^9*y
+ 256*x^9
+ z^4
,
-5*x^10
+ x^9*z^2
,
x*y*z
;
short=0;
option(redSB);
ideal J = groebner(I);
write(":w C:/transferM/Out.txt",J);
quit;
```

```
>
```

Example 2: To test option 'd' in the above procedure. When used with option 'd', the procedure creates and saves file In.txt file containing input for Singular to the hard disk. It then invokes Singular to execute it and write its output file Out.txt for Maple. It then reads and displays the file Out.txt to the terminal/screen. Option 'infty' means that Maple will wait for Singular to complete its computation infinitely long or, as long as it takes for Singular to complete its computation. If instead 'infty' some

positive integer is entered, say 30, then Maple will wait 30 CPU seconds for the output.

```
> f:=2*x^10+4*x^9;
  g:=4*y-8-x;
  h:=f*g^2+z^4;
  k:=-5*x^10+x^9*z^2;
  L:=map(expand,[f,g]);
  vars:=sort(convert(indets(L),list));
  N:=nops(vars);
```

$$f := 2\,x^{10} + 4\,x^{9}$$

$$g := 4\,y - 8 - x$$

$$h := (2\,x^{10} + 4\,x^{9})\,(4\,y - 8 - x)^{2} + z^{4}$$

$$k := -5\,x^{10} + x^{9}\,z^{2}$$

$$L := [2\,x^{10} + 4\,x^{9}, 4\,y - 8 - x]$$

$$vars := [x, y]$$

$$N := 2$$

```
> SINGULARlink([f,g,h],0,dp,[x,y,z],input_for_Singular,input_for_Maple,'infty','d');
```

Transfer directory for Maple already exists... proceeding... waiting for Singular results...

Transfer directory for Singular already exists... proceeding... waiting for Singular results...

$$[x - 4\,y + 8, z^{4}, 2\,y^{10} - 39\,y^{9} + 342\,y^{8} - 1776\,y^{7} + 6048\,y^{6} - 14112\,y^{5} + 22848\,y^{4} - 25344\,y^{3} + 18432\,y^{2} - 7936\,y + 1536]$$

When neither option 't' or 'd' is used, the deafult is to execute Singular:Plural.

```
> SINGULARlink([f,g,h],0,dp,[z,y,x],input_for_Singular,input_for_Maple,'infty');
```

Transfer directory for Maple already exists... proceeding... waiting for Singular results...

Transfer directory for Singular already exists... proceeding... waiting for Singular results...

$$[4\,y - 8 - x, z^{4}, x^{10} + 2\,x^{9}]$$

```
> SINGULARlink([f,g,h],0,lp,[z,y,x],input_for_Singular,input_for_Maple,30);
```

Transfer directory for Maple already exists... proceeding... waiting for Singular results...

Transfer directory for Singular already exists... proceeding... waiting for Singular results...

$$[x^{10} + 2\,x^{9}, 4\,y - 8 - x, z^{4}]$$

```
>
```

Example 3. Testing some intended error messages:

```
> SINGULARlink([f,g,h],0,LP,[z,y,x],input_for_Singular,input_for_Maple,30);
Warning, third argument, total order, must be one of lp, dp, rp, Dp, ls, ds, or
```

```
Ds. Order drp only makes sense with noncommuting variables! Proceeding anyway...
```

Transfer directory for Maple already exists... proceeding... waiting for Singular results...
Transfer directory for Singular already exists... proceeding... waiting for Singular results...
Singular has not returned any results... increase 'wait' time or make it 'infty'

[ ]

```
> SINGULARlink([f,g,h],0,lp,[z,y,X],input_for_Singular,input_for_Maple,30);
Error, (in SINGULARPLURALlink:-SINGULARlink) extra variables found in polynomials that do not appear in elimination list
```

```
> SINGULARlink([f,g,h],0,lp,[z,y,x],input_for_Singular,input_for_Maple,'infty','p');
Error, (in SINGULARPLURALlink:-SINGULARlink) last optional argument needs to be 't' (for displaying to terminal) or 'd' (for saving to disk)
```

```
> SINGULARlink([f,g,h],-1,lp,[z,y,x],input_for_Singular,input_for_Maple,30);
Error, invalid input: SINGULARPLURALlink:-SINGULARlink expects its 2nd argument, C, to be of type nonnegint, but received -1
```

```
>
```

2. Procedure **PLURALforGlink** formats input for PLURAL to compute a Groebner basis in an ideal I = <F> = <f1,f2,...,fn> in a Grassmann algebra ER. The ideal is defined in PLURAL as an ideal I generated by the said polynomials, for example, as follows:

ring r = 0, (e1,e2,e3,e3,e4,e5,e6,e7,e8,e9), dp;
def ER = Exterior();
setring ER;
poly f1=.....;
poly f2=.....;
.....
poly fn=.....;
ideal I=f1,f2,....,fn;
ideal GB=std(I);
GB;.

We prepare to work with Grassmann algebra $\bigwedge_9$.

```
> B:=diag(0$9):
  MonOrderS:=Deg[Lex]:### same as Plural's dp monomial order
  eval(Clifford:-makealiases(9,'ordered')):
  e12;
```

e12

Example 4: Simple computation of a Groebner basis in a Grassmann algebra. Let I <f1,f2> be an ideal in $\bigwedge_4.$

```
>
```

```
> f1,f2:=2*e1we2+e2-4*e3we4,e1;
  F:=[f1,f2];
  vars:=[seq(e||i,i=1..9)];
  wedge(f1,f2);
```

$$f1, f2 := 2\ e12 + e2 - 4\ e34, e1$$

$$F := [2\ e12 + e2 - 4\ e34, e1]$$

$$vars := [e1, e2, e3, e4, e5, e6, e7, e8, e9]$$

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

$$-e12 - 4\ e134$$

First, we show a typical input file In.txt prepared for Plural.

Note that in CLIFFORD/Bigebra package, that has been loaded by the link, a Grassmann monomial built out of, for example, e1 and e2 is denoted by e1we2 whereas the wedge product is computed as wedge(e1,e2). In general, the wedge product of any two Grasssmann polynomials p1 and p2 is computed with wedge(p1,p2) where wedge is a procedure in the CLIFFORD/Bigebra package. For more information type ?Clifford once you have installed CLIFFORD/Bigebra on your system. Furthermore, note that in Singular:Plural, the wedge product of e1 and e2 is just written as e1*e2 which is different than e2*e1: That is, the star `*` implicitly is non-commutative and represents the wedge product in the Grassmann algebra defined as a ring ER with the command Exterior().

Procedure PLURALforGlink translates Maple input, like Grassmann polynomials with wedge product, into polynomials in the exterior algebra ER with the wedge product denoted by `*`. That is why in in the input file below, the first polynomial is written as 2*e1*e2+e2-4*e3*e4. Then, before Plural's output is displayed back in Maple, it needs to be translated back into input that CLIFFORD/Bigebra understand.

Note also that a special library package "nctools.lib" is loaded by Singular:Plural.

```
> PLURALforGlink(F,0,dp,vars,input_for_Singular,input_for_Maple,'inf
  ty','t');
```

Trying to create and display input file for Singular... proceeding...

```
LIB "nctools.lib";
ring R = 0,(e1,e2,e3,e4,e5,e6,e7,e8,e9),dp;
def ER = Exterior();
setring ER;
ideal I =
2*e1*e2
+ e2
+ -4*e3*e4
,
```

```
e1
;
short=0;
option(redSB);
ideal GB = std(I);
write(":w C:/transferM/Out.txt",GB);
quit;
```

Next, we actually compute a Groebner basis for the ideal I generated by the above polynomials. This basis can be compared with the output from Troy Brachey's TNB package for Maple available from

http://math.tntech.edu/rafal/

REMEMBER: Deg[Lex] is the same as Plural's dp monomial order whereas Deg[InvLex] order is the same as (a(1:n),rp), called shortly drp.

```
> GB1:=PLURALforGlink(F,0,dp,vars,input_for_Singular,input_for_Maple
  ,'infty','d');
  nops(GB1);
```

Transfer directory for Maple already exists... proceeding... waiting for Singular results...

Transfer directory for Singular already exists... proceeding... waiting for Singular results...

$$GB1 := [e1, 4\,e34 - e2, e24, e23]$$

$$4$$

Thus, GB1 is a Groebner basis for the ideal I = <f1,f2> for the dp order.

Example 5. We change order in the above example to drp = (a(1:n),rp) which is the same as Deg[InvLex] order in CLIFFORD/Bigebra. This is a graded rp order such that x^\alpha > x^\beta if |\alpha| > |\beta|, or, in case of equality |\alpha| = |\beta|, if the right-most nonzero entry in $\alpha - \beta$ is positive. **This order has no special name in Singular:Plural**, but we call it for short 'drp' as it is a global order based on Plurals' rp order with the added gradation.

```
> PLURALforGlink(F,0,drp,vars,input_for_Singular,input_for_Maple,'in
  fty','t');
```

Trying to create and display input file for Singular... proceeding...

```
LIB "nctools.lib";
ring R = 0,(e1,e2,e3,e4,e5,e6,e7,e8,e9),(a(1:9),rp);
def ER = Exterior();
setring ER;
ideal I =
2*e1*e2
+ e2
+ -4*e3*e4
,
e1
;
short=0;
option(redSB);
ideal GB = std(I);
write(":w C:/transferM/Out.txt",GB);
```

```
quit;
```

```
>
```

```
> PLURALforGlink(F,0,drp,vars,input_for_Singular,input_for_Maple,'infty','d');
```

Transfer directory for Maple already exists... proceeding... waiting for Singular results...

Transfer directory for Singular already exists... proceeding... waiting for Singular results...

$$[e1, e23, e24, 4\ e34 - e2]$$

```
>
```

Example 6: Simple computation of a Groebner basis in a Grassmann algebra. Let I = <f1,f2> be an ideal in $\bigwedge_6.$ We use monomial order drp = Deg[InvLex].

```
>
```

```
> f1,f2:=e56-e23,e45-e13;
  F:=[f1,f2];
  vars:=[seq(e||i,i=1..6)];
  wedge(f1,f2);
```

$$f1, f2 := e56 - e23, e45 - e13$$

$$F := [e56 - e23, e45 - e13]$$

$$vars := [e1, e2, e3, e4, e5, e6]$$

$$-e2345 - e1356$$

```
> PLURALforGlink(F,0,drp,vars,input_for_Singular,input_for_Maple,'infty','t');
```

Trying to create and display input file for Singular... proceeding...

```
LIB "nctools.lib";
ring R = 0,(e1,e2,e3,e4,e5,e6),(a(1:6),rp);
def ER = Exterior();
setring ER;
ideal I =
e5*e6
+ -1*e2*e3
,
e4*e5
+ -1*e1*e3
;
short=0;
option(redSB);
ideal GB = std(I);
write(":w C:/transferM/Out.txt",GB);
quit;
```

```
>
```

```
> PLURALforGlink(F,0,drp,vars,input_for_Singular,input_for_Maple,'infty','d');
```

Transfer directory for Maple already exists... proceeding... waiting for Singular results...

Transfer directory for Singular already exists... proceeding... waiting for Singular results...

$$[e45 - e13, e56 - e23, e134, e135, e235, e136 - e234, e236]$$

```
>
```

3. Procedure **PLURALforClink** formats input for PLURAL to compute a Groebner basis in an ideal I = <F> = <f1,f2,...,fn> in a Clifford algebra Cl(Q) of a quadratic form Q which may be degenerate.

Notice that this time, once an appropriate Clifford algebra is defined in Plural, the asterisk `*` denotes Clifford product in the Clifford algebra Cl(Q). Thus, procedure **PLURALforClink** translates Maple input which, as default, is expressed as Grassmann polynomials with the wedge product, into polynomials in the Clifford algebra with the wedge product denoted by `*`. That is why in in the input file below, the second polynomial is written as 3*e1*e2+e1*e2. Then, before Plural's output is displayed back in Maple, it is reduced NF modulo the defining relations in the Clifford algebra. Then, it is written to a file Out.txt, read into Maple, and translated back into input that CLIFFORD/Bigebra understand.

We prepare to work with a Clifford algebra Cl(R^2) by defining a bilinear form B which is diagonal, that is, Q = B.

Example 7: (Following Example 4.1 from ViktorLevandovskyy, Private Communication, 2006)

We compute a Groebner basis for the ideal F in Clifford algebra Cl(R^2) w.r.t. to the order dp=Deg[Lex].

```
> with(linalg):
  B:=diag(1,1);
  f1:=e1+2*e2;
  f2:=3*e1+cmul(e1,e2);
  F:=[f1,f2];
  clidata([2,0]);
  vrs:=[e1,e2];
```

$$B := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$f1 := e1 + 2\,e2$$

$$f2 := 3\,e1 + e12$$

$$F := [e1 + 2\,e2, 3\,e1 + e12]$$

$$\left[real, 2, simple, \frac{Id}{2} + \frac{e1}{2}, [Id, e2], [Id], [Id, e2]\right]$$

$$vrs := [e1, e2]$$

Showing an input file for Singular:Plural:

```
> PLURALforClink(F,0,dp,vrs,B,input_for_Singular,input_for_Maple,'in
  fty','t');
```

Trying to create and display input file for Singular... proceeding...

```
LIB "clifford.lib";
ring R = 0,(e1,e2),dp;
option(redSB);
option(redTail);
matrix M[2][2];
M[1,1]=2;M[2,2]=2;
clifAlgebra(M);
qring Q =twostd(clQuot);
ideal I =
e1
+ 2*e2
,
3*e1
+ e1*e2
;
short=0;
ideal GB = std(I);
if(size(GB)>=1){int j=1; for ( j=1; j <= size(GB); j++) {GB[j]=NF(GB[j],std(0));}}
;
write(":w C:/transferM/Out.txt",GB);
quit;
```

```
> PLURALforClink(F,0,dp,vrs,B,input_for_Singular,input_for_Maple,'in
  fty','d');
```

Transfer directory for Maple already exists... proceeding... waiting for Singular results...

Transfer directory for Singular already exists... proceeding... waiting for Singular results...

$$[Id]$$

Thus, the above does show that the ideal is the entire algebra Cl(R^2) since it contains the identity element Id. However, we can check that as follows:

Example 8: Following Example 4.1 from ViktorLevandovskyy, Private Communication, 2006, but using drp order defined above.

We compute a Groebner basis for the ideal F in Clifford algebra Cl(R^2) w.r.t. to the order drp=Deg[InvLex].

```
> with(linalg):
  B:=diag(1,1);
  f1:=e1+2*e2;
  f2:=3*e1+cmul(e1,e2);
  F:=[f1,f2];
  clidata([2,0]);
  vrs:=[e1,e2];
```

$$B := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$f1 := e1 + 2\,e2$$

$$f2 := 3\,e1 + e12$$

$$F := [e1 + 2\,e2, 3\,e1 + e12]$$

$$\left[real, 2, simple, \frac{Id}{2} + \frac{e1}{2}, [Id, e2], [Id], [Id, e2]\right]$$

$$vrs := [e1, e2]$$

Showing an input file for Singular:Plural:

```
> PLURALforClink(F,0,drp,vrs,B,input_for_Singular,input_for_Maple,'i
  nfty','t');
```

Trying to create and display input file for Singular... proceeding...

```
LIB "clifford.lib";
ring R = 0,(e1,e2),(a(1:2),rp);
option(redSB);
option(redTail);
matrix M[2][2];
M[1,1]=2;M[2,2]=2;
clifAlgebra(M);
qring Q =twostd(clQuot);
ideal I =
e1
+ 2*e2
,
3*e1
+ e1*e2
;
short=0;
ideal GB = std(I);
if(size(GB)>=1){int j=1; for ( j=1; j <= size(GB); j++) {GB[j]=NF(GB[j],std(0));}}
;
write(":w C:/transferM/Out.txt",GB);
quit;
```

```
> F:=PLURALforClink(F,0,drp,vrs,B,input_for_Singular,input_for_Maple
  ,'infty','d');
```

Transfer directory for Maple already exists... proceeding... waiting for Singular results...

Transfer directory for Singular already exists... proceeding... waiting for Singular results...

$$F := [Id]$$

Thus, the above does show that the ideal is the entire algebra Cl(R^2) since it contains only identity element Id.

```
>
```

Example 9 (Following Example 4.2 from ViktorLevandovskyy, Private Communication, 2006)

We compute a Groebner basis for the ideal F in Clifford algebra Cl(R^3) w.r.t. to the order dp=Deg[Lex].

```
> B:=diag(1,1,1);
  f1:=3*Id-e1;
  f2:=cmul(e1,e3)+e2;
  F:=[f1,f2];
```

```
clidata([3,0]);
vrs:=[e1,e2,e3];
```

$$B := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$f1 := 3\,Id - e1$$

$$f2 := e13 + e2$$

$$F := [3\,Id - e1, e13 + e2]$$

$$\left[complex, 2, simple, \frac{Id}{2} + \frac{e1}{2}, [Id, e2, e3, e23], [Id, e23], [Id, e2]\right]$$

$$vrs := [e1, e2, e3]$$

```
> PLURALforClink(F,0,dp,vrs,B,input_for_Singular,input_for_Maple,'infty','t');
```

Trying to create and display input file for Singular... proceeding...

```
LIB "clifford.lib";
ring R = 0,(e1,e2,e3),dp;
option(redSB);
option(redTail);
matrix M[3][3];
M[1,1]=2;M[2,2]=2;M[3,3]=2;
clifAlgebra(M);
qring Q =twostd(clQuot);
ideal I =
3*1
+ -1*e1
,
e1*e3
+ e2
;
short=0;
ideal GB = std(I);
if(size(GB)>=1){int j=1; for ( j=1; j <= size(GB); j++) {GB[j]=NF(GB[j],std(0));}}
;
write(":w C:/transferM/Out.txt",GB);
quit;
```

```
> PLURALforClink(F,0,dp,vrs,B,input_for_Singular,input_for_Maple,'infty','d');
```

Transfer directory for Maple already exists... proceeding... waiting for Singular results...

Transfer directory for Singular already exists... proceeding... waiting for Singular results...

$$[Id]$$

```
>
```

Example 10 We compute a Groebner basis for the ideal F in Clifford algebra Cl(R^3) w.r.t. to the order drp=Deg[InvLex].

Notice that the ideal I = <F> = <f1,f2,f3,f4> = Cl(R^3)f is a spinor ideal in Cl(R^3) generated by a primitive idempotent f=f1.

```
> B:=diag(1,1,1);
  f1:=1/2*(Id+e1);
  f2:=cmul(e2,f1);
  f3:=cmul(e3,f1);
  f4:=cmul(e23,f1);
  F:=[f1,f2,f3,f4];
  clidata([3,0]);
  vrs:=[e1,e2,e3];
```

$$B := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$f1 := \frac{Id}{2} + \frac{e1}{2}$$

$$f2 := -\frac{e12}{2} + \frac{e2}{2}$$

$$f3 := -\frac{e13}{2} + \frac{e3}{2}$$

$$f4 := \frac{e23}{2} + \frac{e123}{2}$$

$$F := \left[\frac{Id}{2} + \frac{e1}{2}, -\frac{e12}{2} + \frac{e2}{2}, -\frac{e13}{2} + \frac{e3}{2}, \frac{e23}{2} + \frac{e123}{2}\right]$$

$$\left[complex, 2, simple, \frac{Id}{2} + \frac{e1}{2}, [Id, e2, e3, e23], [Id, e23], [Id, e2]\right]$$

$$vrs := [e1, e2, e3]$$

```
> PLURALforClink(F,0,drp,vrs,B,input_for_Singular,input_for_Maple,'i
  nfty','t');
```

Trying to create and display input file for Singular... proceeding...

```
LIB "clifford.lib";
ring R = 0,(e1,e2,e3),(a(1:3),rp);
option(redSB);
option(redTail);
matrix M[3][3];
M[1,1]=2;M[2,2]=2;M[3,3]=2;
clifAlgebra(M);
qring Q =twostd(clQuot);
ideal I =
1/2*1
+ 1/2*e1
,
-1/2*e1*e2
+ 1/2*e2
,
-1/2*e1*e3
+ 1/2*e3
```

```
,
1/2*e2*e3
+ 1/2*e1*e2*e3
;
short=0;
ideal GB = std(I);
if(size(GB)>=1){int j=1; for ( j=1; j <= size(GB); j++) {GB[j]=NF(GB[j],std(0));}}
;
write(":w C:/transferM/Out.txt",GB);
quit;
```

```
> F:=PLURALforClink(F,0,drp,vrs,B,input_for_Singular,input_for_Maple
  ,'infty','d');
```

Transfer directory for Maple already exists... proceeding... waiting for Singular results...

Transfer directory for Singular already exists... proceeding... waiting for Singular results...

$$F := [e1 + Id]$$

Notice, as it should not be a surprise, that the Groebner basis for the left ideal is, up to a scalar coefficient, the idempotent f1 as S = Cl(R^3)f1 = <f1>.

```
>
```

Example 11 (following example 4.3 from ViktorLevandovskyy, Private Communication)

We compute a Groebner basis for the ideal F in Clifford algebra Cl(B) w.r.t. to the order dp=Deg[Lex]. However, in this case B=0, hence the Clifford algebra Cl(B) = Grassmann algebra. We can also use here procedure PLURALforGlink.

```
> B:=diag(0,0,0); #Grassmann algebra
  f1:=Id+e1+cmul(e2,e3);
  f2:=cmul(e1,e3)+cmul(e1,e2,e3);
  f3:=e1-2*e3;
  F:=[f1,f2,f3];
  vrs:=[e1,e2,e3];
```

$$B := \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$f1 := Id + e1 + e23$$

$$f2 := e13 + e123$$

$$f3 := e1 - 2\,e3$$

$$F := [Id + e1 + e23, e13 + e123, e1 - 2\,e3]$$

$$vrs := [e1, e2, e3]$$

```
> PLURALforClink(F,0,dp,vrs,B,input_for_Singular,input_for_Maple,'in
  fty','t');
```

Trying to create and display input file for Singular... proceeding...

```
LIB "clifford.lib";
ring R = 0,(e1,e2,e3),dp;
option(redSB);
option(redTail);
```

```
matrix M[3][3];
M[1,1]=0;M[2,2]=0;M[3,3]=0;
clifAlgebra(M);
qring Q =twostd(clQuot);
ideal I =
1
+ e1
+ e2*e3
,
e1*e3
+ e1*e2*e3
,
e1
+ -2*e3
;
short=0;
ideal GB = std(I);
if(size(GB)>=1){int j=1; for ( j=1; j <= size(GB); j++) {GB[j]=NF(GB[j],std(0));}}
;
write(":w C:/transferM/Out.txt",GB);
quit;
```

```
> PLURALforGlink(F,0,dp,vrs,input_for_Singular,input_for_Maple,'infty','t');
```

Trying to create and display input file for Singular... proceeding...

```
LIB "nctools.lib";
ring R = 0,(e1,e2,e3),dp;
def ER = Exterior();
setring ER;
ideal I =
1
+ e1
+ e2*e3
,
e1*e3
+ e1*e2*e3
,
e1
+ -2*e3
;
short=0;
option(redSB);
ideal GB = std(I);
write(":w C:/transferM/Out.txt",GB);
quit;
```

```
>
```

```
> G1:=PLURALforClink(F,0,dp,vrs,B,input_for_Singular,input_for_Maple,'infty','d');
```

Transfer directory for Maple already exists... proceeding... waiting for Singular results...

Transfer directory for Singular already exists... proceeding... waiting for Singular results...

$$G1 := [Id]$$

```
> G2:=PLURALforGlink(F,0,dp,vrs,input_for_Singular,input_for_Maple,'infty','d');
```

Transfer directory for Maple already exists... proceeding... waiting for Singular results...

Transfer directory for Singular already exists... proceeding... waiting for Singular results...

$$G2 := [Id]$$

>

Example 12 Following example 4.3 from ViktorLevandovskyy, Private Communication, 2006, but for drp order.

We compute a Groebner basis for the ideal F in Clifford algebra Cl(B) w.r.t. to the order dp=Deg[Lex]. However, in this case B=0, hence the Clifford algebra Cl(B) = Grassmann algebra. We can also use here procedure **PLURALforGlink**.

```
> B:=diag(0,0,0); #Grassmann algebra
  f1:=Id+e1+cmul(e2,e3);
  f2:=cmul(e1,e3)+cmul(e1,e2,e3);
  f3:=e1-2*e3;
  F:=[f1,f2,f3];
  vrs:=[e1,e2,e3];
```

$$B := \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$f1 := Id + e1 + e23$$

$$f2 := e13 + e123$$

$$f3 := e1 - 2\,e3$$

$$F := [Id + e1 + e23, e13 + e123, e1 - 2\,e3]$$

$$vrs := [e1, e2, e3]$$

```
> PLURALforClink(F,0,drp,vrs,B,input_for_Singular,input_for_Maple,'i
  nfty','t');
```

Trying to create and display input file for Singular... proceeding...

```
LIB "clifford.lib";
ring R = 0,(e1,e2,e3),(a(1:3),rp);
option(redSB);
option(redTail);
matrix M[3][3];
M[1,1]=0;M[2,2]=0;M[3,3]=0;
clifAlgebra(M);
qring Q =twostd(clQuot);
ideal I =
1
+ e1
+ e2*e3
,
e1*e3
+ e1*e2*e3
,
e1
+ -2*e3
;
short=0;
```

```
ideal GB = std(I);
if(size(GB)>=1){int j=1; for ( j=1; j <= size(GB); j++) {GB[j]=NF(GB[j],std(0));}}
;
write(":w C:/transferM/Out.txt",GB);
quit;
```

```
> PLURALforClink(F,0,drp,vrs,B,input_for_Singular,input_for_Maple,'i
  nfty','d');
  PLURALforGlink(F,0,drp,vrs,input_for_Singular,input_for_Maple,'inf
  ty','d');
```

Transfer directory for Maple already exists... proceeding... waiting for Singular results...

Transfer directory for Singular already exists... proceeding... waiting for Singular results...

$$[Id]$$

Transfer directory for Maple already exists... proceeding... waiting for Singular results...

Transfer directory for Singular already exists... proceeding... waiting for Singular results...

$$[Id]$$

```
>
```

Example 13 (following example 4.4 from ViktorLevandovskyy, Private Communication)

We compute a Groebner basis for the ideal F in Clifford algebra Cl(B) w.r.t. to the order dp=Deg[Lex] whene B=0, that is, Cl(B) = Exterior Algebra. However, ideal is smaller.

```
> B:=diag(0,0,0); #Grassmann algebra
  f1:=cmul(e1,e2,e3)+cmul(e2,e3);
  f2:=e1-2*e3;
  F:=[f1,f2];
  vrs:=[e1,e2,e3];
```

$$B := \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$f1 := e123 + e23$$

$$f2 := e1 - 2\,e3$$

$$F := [e123 + e23, e1 - 2\,e3]$$

$$vrs := [e1, e2, e3]$$

```
> PLURALforClink(F,0,dp,vrs,B,input_for_Singular,input_for_Maple,'in
  fty','t');
```

Trying to create and display input file for Singular... proceeding...

```
LIB "clifford.lib";
ring R = 0,(e1,e2,e3),dp;
option(redSB);
option(redTail);
matrix M[3][3];
M[1,1]=0;M[2,2]=0;M[3,3]=0;
clifAlgebra(M);
qring Q =twostd(clQuot);
```

```
ideal I =
e1*e2*e3
+ e2*e3
,
e1
+ -2*e3
;
short=0;
ideal GB = std(I);
if(size(GB)>=1){int j=1; for ( j=1; j <= size(GB); j++) {GB[j]=NF(GB[j],std(0));}}
;
write(":w C:/transferM/Out.txt",GB);
quit;
```

```
> PLURALforGlink(F,0,dp,vrs,input_for_Singular,input_for_Maple,'infty','t');
```

Trying to create and display input file for Singular... proceeding...

```
LIB "nctools.lib";
ring R = 0,(e1,e2,e3),dp;
def ER = Exterior();
setring ER;
ideal I =
e1*e2*e3
+ e2*e3
,
e1
+ -2*e3
;
short=0;
option(redSB);
ideal GB = std(I);
write(":w C:/transferM/Out.txt",GB);
quit;
```

```
> PLURALforClink(F,0,dp,vrs,B,input_for_Singular,input_for_Maple,'infty','d');
```

Transfer directory for Maple already exists... proceeding... waiting for Singular results...

Transfer directory for Singular already exists... proceeding... waiting for Singular results...

$$[e1 - 2\,e3, e23]$$

We can convert the above basis expressed in Grassmann basis to Clifford unevaluated basis using command 'Cliplus:-cliexpand'.

Note: Package 'Cliplus' is an extension of CLIFFORD/Bigebra. It loads automatically with CLIFFORD.:

```
> map(Cliplus:-cliexpand,%);
```

$$[e1 - 2\,e3, e2\ \&C\ e3]$$

```
> map(Cliplus:-clieval,%);
```

$$[e1 - 2\,e3, e23]$$

```
> PLURALforGlink(F,0,dp,vrs,input_for_Singular,input_for_Maple,'infty','d');
```

Transfer directory for Maple already exists... proceeding... waiting for Singular results...

Transfer directory for Singular already exists... proceeding... waiting for Singular results...

$$[e1 - 2\,e3, e23]$$

```
> map(Cliplus:-cliexpand,%);
```

$$[e1 - 2\,e3, e2 \,\&C\, e3]$$

```
> map(Cliplus:-clieval,%);
```

$$[e1 - 2\,e3, e23]$$

We can convert the above basis expressed in Grassmann basis to Clifford unevaluated basis using command 'Cliplus:-cliexpand'. The result is the same when using PLURALforClink procedure.

Additional examples:

Example 14:

We continue with this algebra Cl(2,0):

```
> B:=diag(1,1);
  eval(makealiases(2)):
  clid:=clidata([2,0]);
  N:=nops(clid[5]);
```

$$B := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$clid := \left[real, 2, simple, \frac{Id}{2} + \frac{e1}{2}, [Id, e2], [Id], [Id, e2] \right]$$

$$N := 2$$

```
> f:=clid[4];
  f:=eval(f);
  for i from 1 to N do
        g||i:=clid[5][i];
  end do;
  for i from 1 to N do
      f||i:=cmul(g||i,f);
  end do;
  F:=[f1,f2];
```

$$f := \frac{Id}{2} + \frac{e1}{2}$$

$$f := \frac{Id}{2} + \frac{e1}{2}$$

$$g1 := Id$$

$$g2 := e2$$

$$f1 := \frac{Id}{2} + \frac{e1}{2}$$

$$f2 := -\frac{e12}{2} + \frac{e2}{2}$$

$$F := \left[\frac{Id}{2} + \frac{e1}{2}, -\frac{e12}{2} + \frac{e2}{2}\right]$$

```
> vrs:=[seq(e||k,k=1..maxindex(F))];
```

$$vrs := [e1, e2]$$

```
> PLURALforClink(F,0,dp,vrs,B,input_for_Singular,input_for_Maple,'infty','t');
```

Trying to create and display input file for Singular... proceeding...

```
LIB "clifford.lib";
ring R = 0,(e1,e2),dp;
option(redSB);
option(redTail);
matrix M[2][2];
M[1,1]=2;M[2,2]=2;
clifAlgebra(M);
qring Q =twostd(clQuot);
ideal I =
1/2*1
+ 1/2*e1
,
-1/2*e1*e2
+ 1/2*e2
;
short=0;
ideal GB = std(I);
if(size(GB)>=1){int j=1; for ( j=1; j <= size(GB); j++) {GB[j]=NF(GB[j],std(0));}}
;
write(":w C:/transferM/Out.txt",GB);
quit;
```

```
> GB:=PLURALforClink(F,0,dp,vrs,B,input_for_Singular,input_for_Maple,'infty','d');
  ff:=op(GB)/2;
```

Transfer directory for Maple already exists... proceeding... waiting for Singular results...

Transfer directory for Singular already exists... proceeding... waiting for Singular results...

$$GB := [e1 + Id]$$

$$ff := \frac{Id}{2} + \frac{e1}{2}$$

```
> 'ff' = cmul(ff,ff);
```

$$ff = \frac{Id}{2} + \frac{e1}{2}$$

```
>
```

Example 15: We continue with this algebra Cl(3,1):

```
> B:=diag(1,1,1,-1);
  eval(makealiases(4)):
```

```
clid:=clidata([3,1]);
N:=nops(clid[5]);
```

$$B := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

$$clid := \left[real, 4, simple, \text{'}Clifford{:}\text{-}cmulQ\text{'}\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e34}{2}\right), [Id, e2, e3, e23], [Id], [Id, e2, e3, e23] \right]$$

$$N := 4$$

```
> f:=clid[4];
  f:=eval(f);
  for i from i to N do
        g||i:=clid[5][i];
  end do;
  for i from 1 to N do
      p||i:=cmul(g||i,f);
  end do;
  F:=[p1,p2,p3,p4];
```

$$f := Clifford{:}\text{-}cmulQ\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e34}{2}\right)$$

$$f := \frac{Id}{4}+\frac{e1}{4}+\frac{e34}{4}+\frac{e134}{4}$$

$$g3 := e3$$

$$g4 := e23$$

$$p1 := \frac{Id}{4}+\frac{e34}{4}+\frac{e1}{4}+\frac{e134}{4}$$

$$p2 := -\frac{e12}{4}+\frac{e2}{4}+\frac{e234}{4}-\frac{e1234}{4}$$

$$p3 := -\frac{e13}{4}-\frac{e14}{4}+\frac{e3}{4}+\frac{e4}{4}$$

$$p4 := \frac{e23}{4}+\frac{e123}{4}+\frac{e124}{4}+\frac{e24}{4}$$

$$F := \left[\frac{Id}{4}+\frac{e34}{4}+\frac{e1}{4}+\frac{e134}{4}, -\frac{e12}{4}+\frac{e2}{4}+\frac{e234}{4}-\frac{e1234}{4}, -\frac{e13}{4}-\frac{e14}{4}+\frac{e3}{4}+\frac{e4}{4}, \frac{e23}{4}+\frac{e123}{4}+\frac{e124}{4}+\frac{e24}{4}\right]$$

```
> vrs:=[seq(e||k,k=1..maxindex(F))];
```

$$vrs := [e1, e2, e3, e4]$$

We proceed to compute a left Groebner basis in the spinor ideal S = Cl(3,1)f1.

```
> PLURALforClink(F,0,dp,vrs,B,input_for_Singular,input_for_Maple,'infty','t');
```

Trying to create and display input file for Singular... proceeding...

```
LIB "clifford.lib";
ring R = 0,(e1,e2,e3,e4),dp;
option(redSB);
option(redTail);
matrix M[4][4];
M[1,1]=2;M[2,2]=2;M[3,3]=2;M[4,4]=-2;
clifAlgebra(M);
qring Q =twostd(clQuot);
ideal I =
1/4*1
+ 1/4*e3*e4
+ 1/4*e1
+ 1/4*e1*e3*e4
,
-1/4*e1*e2
+ 1/4*e2
+ 1/4*e2*e3*e4
+ -1/4*e1*e2*e3*e4
,
-1/4*e1*e3
+ -1/4*e1*e4
+ 1/4*e3
+ 1/4*e4
,
1/4*e2*e3
+ 1/4*e1*e2*e3
+ 1/4*e1*e2*e4
+ 1/4*e2*e4
;
short=0;
ideal GB = std(I);
if(size(GB)>=1){int j=1; for ( j=1; j <= size(GB); j++) {GB[j]=NF(GB[j],std(0));}}
;
write(":w C:/transferM/Out.txt",GB);
quit;
```

```
> GB:=PLURALforClink(F,0,dp,vrs,B,input_for_Singular,input_for_Maple,'infty','d');
  g:=op(GB)/4;
```

Transfer directory for Maple already exists... proceeding... waiting for Singular results...

Transfer directory for Singular already exists... proceeding... waiting for Singular results...

$$GB := [e13 + e14 - e3 - e4]$$

$$g := \frac{e13}{4} + \frac{e14}{4} - \frac{e3}{4} - \frac{e4}{4}$$

Notice that this element is nilpotent:

```
> cmul(g,g);
```

$$0$$

However, as Cl(3,1) is a simple Artinian algebra, any of its left and right ideals is generated by an idempotent. It turns out that in this case g = -cmul(e3,f):

```
> e3f:=-cmul(e3,f);
```

$$e3f := \frac{e13}{4} + \frac{e14}{4} - \frac{e3}{4} - \frac{e4}{4}$$

```
> g-e3f;
```

$$0$$

```
> cmul(e3,f,e3,f);
```

$$0$$

```
> -cmul(e3,g)=f;
```

$$\frac{Id}{4} + \frac{e34}{4} + \frac{e1}{4} + \frac{e134}{4} = \frac{Id}{4} + \frac{e34}{4} + \frac{e1}{4} + \frac{e134}{4}$$

Example 16: We continue with a semisimple algebra.

```
>
> all_sigs(1..9, 'real', 'semisimple');
```

$$[[0, 7], [1, 0], [1, 8], [2, 1], [3, 2], [4, 3], [5, 4], [9, 0]]$$

```
> clidata([2,1]);
```

$$\left[real, 2, semisimple, 'Clifford{:}\text{-}cmulQ'\left(\frac{Id}{2} + \frac{e1}{2}, \frac{Id}{2} + \frac{e23}{2}\right), [Id, e2], [Id], [Id, e2]\right]$$

```
> B:=diag(1,1,-1);
  eval(makealiases(4)):
  clid:=clidata([2,1]);
  N:=nops(clid[5]);
```

$$B := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

$$clid := \left[real, 2, semisimple, 'Clifford{:}\text{-}cmulQ'\left(\frac{Id}{2} + \frac{e1}{2}, \frac{Id}{2} + \frac{e23}{2}\right), [Id, e2], [Id], [Id, e2]\right]$$

$$N := 2$$

```
> f:=clid[4];
  f:=eval(f);
  frev:=gradeinv(f);
  for i from i to N do
        g||i:=clid[5][i];
  end do;
```

```
for i from 1 to N do
    p||i:=cmul(g||i,f);
end do;
F:=[p1,p2];
```

$$f := Clifford{:}\text{-}cmulQ\left(\frac{Id}{2}+\frac{e1}{2},\frac{Id}{2}+\frac{e23}{2}\right)$$

$$f := \frac{Id}{4}+\frac{e1}{4}+\frac{e23}{4}+\frac{e123}{4}$$

$$frev := \frac{Id}{4}-\frac{e1}{4}+\frac{e23}{4}-\frac{e123}{4}$$

$$p1 := \frac{Id}{4}+\frac{e23}{4}+\frac{e123}{4}+\frac{e1}{4}$$

$$p2 := -\frac{e13}{4}-\frac{e12}{4}+\frac{e2}{4}+\frac{e3}{4}$$

$$F := \left[\frac{Id}{4}+\frac{e23}{4}+\frac{e123}{4}+\frac{e1}{4},-\frac{e13}{4}-\frac{e12}{4}+\frac{e2}{4}+\frac{e3}{4}\right]$$

```
> vrs:=[seq(e||k,k=1..maxindex(F))];
```

$$vrs := [e1, e2, e3]$$

```
> GB:=PLURALforClink(F,0,dp,vrs,B,input_for_Singular,input_for_Maple
,'infty','d');
```

Transfer directory for Maple already exists... proceeding... waiting for Singular results...

Transfer directory for Singular already exists... proceeding... waiting for Singular results...

$$GB := [e12+e13-e2-e3]$$

```
> g:=op(GB)/4;
```

$$g := \frac{e12}{4}+\frac{e13}{4}-\frac{e2}{4}-\frac{e3}{4}$$

```
> cmul(g,g);
```

$$0$$

```
> e2f:=-cmul(e2,f);
```

$$e2f := \frac{e12}{4}+\frac{e13}{4}-\frac{e2}{4}-\frac{e3}{4}$$

```
> g-e2f;
```

$$0$$

```
>
>
>
>
```

End

Cookeville, May 15, 2008

Function: SP:-SPversion - displays current version of the SP Package for Symmetric Polynomials

Calling Sequence:

SPversion();

Parameters:

- none

Output:

- Information about the package version, release date, etc.,

Description:

- This procedure takes no argument (parameter) and it returns information about the current version of the package.
- Observe that upon loading the package remember tables of several procedures are assigned through the procedure ModuleLoad. This procedure uses another procedure called 'load_remember_table' which reads and assigns the remember tables. Then, upon unloading the package which happens when Maple is restarted or when the Maple worksheet is closed, the remember tables are again saved in the library. This is accomplished through the procedure ModuleUnload.

Examples:

```
> restart:with(SP);
Remember table of SymmetricGroup has been read and assigned
Remember table of AlternatingGroup has been read and assigned
Remember table of Reynolds has been read and assigned
Remember table of FiniteGroups has been read and assigned
Remember table of generateGinvariants has been read and assigned
```

[*AlternatingGroup*, *Dpolynom*, *FiniteGroups*, *Hilbert_series*, *MatrixAction*, *ModuleLoad*, *ModuleUnload*, *Molien_series*, *Reynolds*, *SPversion*, *Schur_polynom*, Σ, *SymmetricGroup*, *SyzygyIdeal*, *a_polynom*, *create_partitions*, *generateGinvariants*, *gpolynom*, *hpolynom*, *isContained*, *isGinvariant*, *isSymmetric*, *ispartition*, *load_remember_table*, *maxmindegree*, *permsign*, *powersum*, *reduceGinvariants*, *sigma_to_powersum*]

```
> SPversion();
```

++

SP - A Maple 11 Small Package for Symmetric Polynomials

Last revised: December 20, 2007 (Source file: SP_06_M11.mws)

Copyright 2007-2008 by Rafal Ablamowicz(*) and Bertfried Fauser(**)

(*) Department of Mathematics, Box 5054
Tennessee Technological University, Cookeville, TN 38505
tel: USA (931) 372-3622, fax: USA (931) 372-6353
rablamowicz@tntech.edu
http://math.tntech.edu/rafal/
(**) University of Konstanz
Fachbereich Physik, Fach M678
Universitätsstrasse 10, D-78457 Konstanz, Germany
Phone: +49 (0)7531 693491
E-mail: Bertfried.Fauser@uni-konstanz.de
http://clifford.physik.uni-konstanz.de/~fauser/

*++++++++++++This is Symmetric Polynomials Package (SP) for Maple 11 version 06++\
++++++++++*

```
>
>
```

Algorithm used:

None

See Also: SP:-Sigma, SP:-isSymmetric

Last modified: June 19, 2008

Function: SP:-SyzygyIdeal - determine nontrivial relations among polynomials, if any

Calling Sequence:

SyzygyIdeal(F);

Parameters:

- F = [f1,f2,...,fm] is a list of polynomials in the ring *k*[x[1], x[2], x[3],..., x[n]].

Output:

- A list with polynomials in k[_y[1], _y[2], ..., _y[m]] which give a Groebner basis for the syzygy ideal I_F of relations.
- If the syzygy ideal I_F does not contain non-trivial identities, then the procedure returns [0].

Description:

- Procedure 'SyzygyIdeal' (or "ideal of relations") computes non-trivial relations, if any, between polynomials f1, f2,..., fm in a polynomial ring k[x[1],x[2],...,x[n]].
- The ideal of relations I_F is defined as a collection of polynomials h in the polynomial ring *k*[y[1], y[2],..., y[m]] such that h(f1, f2,..., fm) = 0. It is a prime ideal of *k*[y[1], y[2],..., y[m]].
- The ideal I_F can be computed directly using the elimination theory (see Section 7.4 in [1]]. According to Proposition 3 on page 339 in [1], the ideal I_F is the n-th elimination ideal of an ideal J_F = <f1 - y[1], f2 - y[2], ..., fm - y[m]> in the polynomial ring *k*[x[1], x[2], ..., x[n], y[1], y[2],..., y[m]]. That is, I_F is the intersection of J_F with *k*[y[1], y[2],..., y[m]].
- A Groebner basis for I_F can be computed then using the standard elimination theory as follows: Fix any monomial order T where any monomial involving one of x[1], x[2], ..., x[n] is greater than all monomials in *k*[y[1], y[2],... y[m]]. For example, we can set T to be the lexicographic order x[1] > x[2] > ...> x[n] > y[1] > y[2] >...> y[m]. Compute a Groebner basis G for J_F for the order T. Then, the intersection of G with the ring *k*[y[1], y[2],... y[m]] provides a Groebner basis for I_F. That is, the Groebner basis for I_F consists of those polynomials in G, if any, which belong to *k*[y[1], y[2],... y[m]].
- The procedure returns the Groebner basis for I_F or a list [0] if the intersection between G with the ring *k*[y[1], y[2],... y[m]] is empty. In that latter case, this means that the generators f1, f2, ..., fm are algebraically independent, or, in another words, that they do not satisfy any non-trivial relation.
- The Groebner basis for I_F may not be minimal: This is because the original list F of polynomials may contain polynomials which are algebraically dependent. Thus, in order to obtain a minimal Groebner basis for I_F, the smallest number of generating syzygy relations, apply the procedure

SP:-reduceGinvariants to the list F.

- Care needs to be excercised because variables _y[1], _y[2], ..., _y[m] returned by the procedure are defined as local to the procedure. This is so that Maple would not make automatic substitutions for these polynomials should they be defined in the worksheet. In order to be able to replace them with the polynomials f1, f2,..., fm, e.g., to verify that indeed these polynomials satisfy the syzygy relations, the local attribute of these variables need to be changed to global. See example below.
- In practice, the polynomials f1, f2,..., fm, will often be some elementary invariants of a finite group G, for example, like the elementary symmetric functions are invariant under the group S[n], and they will generate a ring of invariants k[x[1], x[2],..., x[n]]^G. Then, we will be interested to find all relations between the generators, if any, in order to reduce these generators to elementary ones.
- References:
  [1] D. Cox, J. Little, D. O'Shea: "Ideals, varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra", Third Edition, Springer, New York, 2007
  [2] B. Sturmfels: "Algorithms in Invariant Theory", Texts and Monographs in Symbolic Computation, Springer-Verlag/Wien, 1993

Examples:

```
> restart:with(SP);
Remember table of SymmetricGroup has been read and assigned
Remember table of AlternatingGroup has been read and assigned
Remember table of Reynolds has been read and assigned
Remember table of FiniteGroups has been read and assigned
Remember table of generateGinvariants has been read and assigned
```

[*AlternatingGroup*, *Dpolynom*, *FiniteGroups*, *Hilbert_series*, *MatrixAction*, *ModuleLoad*, *ModuleUnload*, *Molien_series*, *Reynolds*, *SPversion*, *Schur_polynom*, Σ, *SymmetricGroup*, *SyzygyIdeal*, *a_polynom*, *create_partitions*, *generateGinvariants*, *gpolynom*, *hpolynom*, *isContained*, *isGinvariant*, *isSymmetric*, *ispartition*, *load_remember_table*, *maxmindegree*, *permsign*, *powersum*, *reduceGinvariants*, *sigma_to_powersum*]

```
>
```

Example 1: Find a nontrivial syzygy relations among invariants of C4:

```
> C4:=FiniteGroups('C4');
```

$$C4 := \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\right]$$

```
> F:=generateGinvariants[C4]([x,y]);
```

$$F := [x^2 + y^2, y^4 + x^4, x\,y^3 - x^3\,y, x^2\,y^2, x^3\,y - x\,y^3]$$

```
> I_F:=SyzygyIdeal(F);
```

```
nops(I_F);
I_F:=map(convert,I_F,`global`);
```

```
Path set to C:\Maple11\bin.win/libfgbuni.so
    FGb/Maple interface package Version 1.34
    JC Faugere (jcf@calfor.lip6.fr)
    Type ?FGb for documentation
```

$$I\_F := [\_y_1^2 - \_y_2 - 2\,\_y_4, \_y_2\,\_y_4 - 2\,\_y_4^2 - \_y_5^2, \_y_3 + \_y_5]$$

$$3$$

$$I\_F := [\_y_1^2 - \_y_2 - 2\,\_y_4, \_y_2\,\_y_4 - 2\,\_y_4^2 - \_y_5^2, \_y_3 + \_y_5]$$

```
> map(simplify,subs({seq(_y[i]=F[i],i=1..nops(F))},I_F));
```

$$[0, 0, 0]$$

Since the generators in the list F are algebraically related, the ideal I_F is returned with three polynomial relations. However, these relations are not independent because if dependent relations are removed from the list F, the final list of relations reduces to just one polynomial:

```
> 'F'=F;
  FF:=reduceGinvariants[C4](F,[x,y]);
```

$$F = [x^2 + y^2, y^4 + x^4, x\,y^3 - x^3\,y, x^2\,y^2, x^3\,y - x\,y^3]$$

$$FF := [x\,y^3 - x^3\,y, x^2\,y^2, x^2 + y^2]$$

```
> I_FF:=SyzygyIdeal(FF);
  nops(I_FF);
  I_FF:=map(convert,I_FF,`global`);
```

$$I\_FF := [\_y_2\,\_y_3^2 - \_y_1^2 - 4\,\_y_2^2]$$

$$1$$

$$I\_FF := [\_y_2\,\_y_3^2 - \_y_1^2 - 4\,\_y_2^2]$$

```
> map(simplify,subs({seq(_y[i]=FF[i],i=1..nops(FF))},I_FF));
```

$$[0]$$

```
>
```

Example 2: In the above example we found out that the ring *k*[x,y]^C4 = *k*[FF] where the generating C4 polynomials in FF satisfy one syzygy relation displayed in the list I_FF. This means, that although any polynomial f invariant under C4, that is, any polynomial f from *k*[x,y]^C4, can be experssed in terms of polynomials in the list FF, this representation is not unique: It is modulo the syzygy relation. Here is an example:

```
> f:=2*x*y-y^4+x^6;
```

$$f := 2\,x\,y - y^4 + x^6$$

```
> isGinvariant[C4](f,[x,y]);
```

$$false$$

Thus, the polynomial f defined above is not C4 invariant. We can easily generate out of f a new polynomial that will be C4-invariant by using the Reynolds operator (see SP:-Reynolds) for C4:

```
> f:=Reynolds[C4](f,[x,y]);
```

$$f := -\frac{1}{2}y^4 + \frac{1}{2}x^6 - \frac{1}{2}x^4 + \frac{1}{2}y^6$$

```
> isGinvariant[C4](f,[x,y]);
```

$$true$$

Thus, this redefined polynomial f is now C4 invariant. Hence, it is contained in the polynomial ideal <FF>. We can verify this with the procedure SP:-isContained.

```
> isContained(f,FF);
  g:=isContained(f,FF,'r');
  g:=convert(g,`global`);
```

$$true$$

$$g := \_f_2 - \frac{1}{2}\_f_3^{\,2} - \frac{3}{2}\_f_2\,\_f_3 + \frac{1}{2}\_f_3^{\,3}$$

$$g := \_f_2 - \frac{1}{2}\_f_3^{\,2} - \frac{3}{2}\_f_2\,\_f_3 + \frac{1}{2}\_f_3^{\,3}$$

Thus, observe, that upon substituting FF[1], FF[2], FF[3] for _f[1], _f[2], _f[3] in the polynomial g, we get back the polynomial f:

```
> f = simplify(subs({_f[1]=FF[1],_f[2]=FF[2],_f[3]=FF[3]},g));
```

$$-\frac{1}{2}y^4 + \frac{1}{2}x^6 - \frac{1}{2}x^4 + \frac{1}{2}y^6 = -\frac{1}{2}y^4 + \frac{1}{2}x^6 - \frac{1}{2}x^4 + \frac{1}{2}y^6$$

Of course, the syzygy relation in I_FF is one nontrivial relation satisfied by the three polynomials FF[1], FF[2], and FF[3]. Let's call it h:

```
> h:=op(I_FF);
```

$$h := \_y_2\,\_y_3^{\,2} - \_y_1^{\,2} - 4\,\_y_2^{\,2}$$

Then, obviously polynomial g + h also gives f:

```
> f =
  simplify(subs({_f[1]=FF[1],_f[2]=FF[2],_f[3]=FF[3],_y[1]=FF[1],
  _y[2]=FF[2],_y[3]=FF[3]},g+h));
```

$$-\frac{1}{2}y^4 + \frac{1}{2}x^6 - \frac{1}{2}x^4 + \frac{1}{2}y^6 = -\frac{1}{2}y^4 + \frac{1}{2}x^6 - \frac{1}{2}x^4 + \frac{1}{2}y^6$$

```
>
```

Example 3: Since the elementary symmetric polynomials sigma[1], sigma[2],..., are algebraically independent, their syzygy ideal does not contain any nontrivial relation:

```
> s1:=Sigma[1](x,y);
  s2:=Sigma[2](x,y);
  SyzygyIdeal([s1,s2]);
```

$$s1 := y + x$$

$$s2 := x\,y$$

$$[0]$$

```
> s1:=Sigma[1](x,y,z);
  s2:=Sigma[2](x,y,z);
  s3:=Sigma[3](x,y,z);
  SyzygyIdeal([s1,s2,s3]);
```

$$s1 := x + y + z$$

$$s2 := x\,y + x\,z + y\,z$$

$$s3 := x\,y\,z$$

$$[0]$$

```
> s1:=Sigma[1](x,y,z,t);
  s2:=Sigma[2](x,y,z,t);
  s3:=Sigma[3](x,y,z,t);
  s4:=Sigma[4](x,y,z,t);
  SyzygyIdeal([s1,s2,s3,s4]);
```

$$s1 := t + x + y + z$$

$$s2 := t\,x + t\,y + t\,z + x\,y + x\,z + y\,z$$

$$s3 := t\,x\,y + t\,x\,z + t\,y\,z + x\,y\,z$$

$$s4 := t\,x\,y\,z$$

$$[0]$$

Thus, this means that any symmetric polynomial f can be uniquely expressed in terms of the elementary symmetric functions because these functions do not satisfy any syzygy relation.

```
>
```

Example 4: This command SyzygyIdeal can be used to find relations among any set of polynomials, that is, not necessarily some G-invariants:

```
> f1:=x^2+y^2;
  f2:=x^3*y-x*y^3;
  f3:=x^2*y^2;
```

$$f1 := x^2 + y^2$$

$$f2 := x^3\,y - x\,y^3$$

$$f3 := x^2\,y^2$$

```
> L:=SyzygyIdeal([f1,f2,f3]);
```

$$L := [\_y_1^{\,2}\,\_y_3 - \_y_2^{\,2} - 4\,\_y_3^{\,2}]$$

```
> L:=map(convert,L,`global`);
```

$$L := [\_y_1^{\,2}\,\_y_3 - \_y_2^{\,2} - 4\,\_y_3^{\,2}]$$

```
> subs({_y[1]=f1,_y[2]=f2,_y[3]=f3},L);
```

$$[(x^2 + y^2)^2\,x^2\,y^2 - (x^3\,y - x\,y^3)^2 - 4\,x^4\,y^4]$$

```
> map(expand,%);
```

$$[0]$$

Here is another example:

```
> f1:=x^2:
  f2:=y^2:
  f3:=x*y:
> L:=SyzygyIdeal([f1,f2,f3]);
```

$$L := [\_y_1 \_y_2 - \_y_3^{\,2}]$$

```
> L:=map(convert,L,`global`);
```

$$L := [\_y_1 \_y_2 - \_y_3^{\,2}]$$

```
> subs({_y[1]=f1,_y[2]=f2,_y[3]=f3},L);
```

$$[0]$$

```
>
>
```

Algorithm used:

- Proposition 3 on page 339 in [1]:
- According to Proposition 3 on page 339 in [1], the ideal I_F is the n-th elimination ideal of an ideal J_F = <f1 - y[1], f2 - y[2], ..., fm - y[m]> in the polynomial ring *k*[x[1], x[2], ..., x[n], y[1], y[2],..., y[m]]. That is, I_F is the intersection of J_F with *k*[y[1], y[2],..., y[m]].

  Step 1: Let F = [f1, f2,..., fm] and let T be any monomial order where any monomial involving one of x[1], x[2], ..., x[n] is greater than all monomials in *k*[y[1], y[2],... y[m]]. For example, we can set T to be the lexicographic order x[1] > x[2] > ...> x[n] > y[1] > y[2] >...> y[m].

  Step 2: Define an ideal J_F = <f1 - y[1], f2 - y[2], ..., fm - y[m]> in the polynomial ring *k*[x[1], x[2], ..., x[n], y[1], y[2],..., y[m]] and compute a Groebner basis G for J_F for the monomial order T.

  Step 3: Find the intersection of G with the ring *k*[y[1], y[2],... y[m]]: This interesection provides a Groebner basis for I_F. That is, the Groebner basis for I_F consists of those polynomials in G, if any, which belong to *k*[y[1], y[2],... y[m]].

See Also: SP:-FiniteGroups, SP:-Reynolds, SP:-isSymmetric, SP:-reduceGinvariants, SP:-isContained

Last modified: June 19, 2008

Function: SP:-AlternatingGroup - returns elements of the alternating group A[n]

Calling Sequence:

AlternatingGroup(n);

Parameters:

- n is a positive integer

Output:

- A list of n x n matrices that represent all elements of the alternating group A[n].

Description:

- Procedure 'AlternatingGroup' returns a list with n!/2 square n x n matrices that represent elements of A[n], a normal subgroup of S[n].
- This procedure has a remember table stored in the library.
- This procedure is used later when computing group invariants of SP:-FiniteGroups with the SP:-Reynolds operator.
- References:
  [1] D. Cox, J. Little, D. O'Shea: "Ideals, varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra", Third Edition, Springer, New York, 2007
  [2] B. Sturmfels: "Algorithms in Invariant Theory", Texts and Monographs in Symbolic Computation, Springer-Verlag/Wien, 1993

Examples:

```
> restart:with(SP):
Remember table of SymmetricGroup has been read and assigned
Remember table of AlternatingGroup has been read and assigned
Remember table of Reynolds has been read and assigned
Remember table of FiniteGroups has been read and assigned
Remember table of generateGinvariants has been read and assigned
```

Example 1: Elements of various alternating groups:

```
> AlternatingGroup(1);
```

$$[[\ 1]]$$

```
> AlternatingGroup(2);
```

$$\left[\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\right]$$

```
> AlternatingGroup(3);
```

$$\left[\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}, \begin{bmatrix}0 & 0 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}, \begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 1\\ 1 & 0 & 0\end{bmatrix}\right]$$

```
> AlternatingGroup(4);
```

$$\left[\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}, \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix}, \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\\ 0 & 1 & 0 & 0\end{bmatrix}, \begin{bmatrix}0 & 1 & 0 & 0\\ 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}, \begin{bmatrix}0 & 0 & 1 & 0\\ 1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\end{bmatrix},\right.$$

$$\begin{bmatrix}0 & 0 & 0 & 1\\ 1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0\end{bmatrix}, \begin{bmatrix}0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}, \begin{bmatrix}0 & 0 & 0 & 1\\ 0 & 1 & 0 & 0\\ 1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix}, \begin{bmatrix}0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\\ 1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\end{bmatrix}, \begin{bmatrix}0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\\ 1 & 0 & 0 & 0\end{bmatrix}, \begin{bmatrix}0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 1 & 0 & 0 & 0\end{bmatrix},$$

$$\left.\begin{bmatrix}0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0\\ 1 & 0 & 0 & 0\end{bmatrix}\right]$$

```
>
```

Example 2: Let's see an action of the permutation group A[3] on a polynomial:

```
> A3:=AlternatingGroup(3):
> f:=x^3-y^2+x*y+z;
```

$$f := x^3 - y^2 + x\,y + z$$

```
> for A in A3 do MatrixAction(A,f,[x,y,z]) end do;
```

$$x^3 - y^2 + x\,y + z$$

$$z^3 - x^2 + x\,z + y$$

$$y^3 - z^2 + y\,z + x$$

```
>
```

Algorithm used:

None

See Also: SP:-Sigma, SP:-FiniteGroups, SP:-SymmetricGroup, SP:-MatrixAction, SP:-Reynolds, Groebner

Last modified: June 19, 2008

Function: SP[] - <description>

Calling Sequence:

lst := dummy(N,M)

Parameters:

- N,M : <input type>

Output:

- lst : <output type>

WARNING:

iff applicable

Description:

- Bullet item list of properties of the function

Examples:

```
> restart:with(SP):
Remember table of SymmetricGroup has been read and assigned
Remember table of AlternatingGroup has been read and assigned
Remember table of Reynolds has been read and assigned
Remember table of FiniteGroups has been read and assigned
Remember table of generateGinvariants has been read and assigned
>
>
```

mandatory test cases and most likly cases a user want to type in

Algorithm used:

The presently implemented algorithm, if possible with certificate.

See Also: SP[dummy]

Last modified: June 19, 2008

Function: SP:-FiniteGroups - returns matrix representations of various finite groups

Calling Sequence:

FiniteGroups(name);
FiniteGroups();

Parameters:

- name is a string of the form 'G' where G is one of the names of known groups

Output:

- A list of matrices that represent all elements of the group G

Description:

- Procedure 'FiniteGroups' when used without any argument, returns a names of known groups.
- Procedure 'FiniteGroups' when used with one argument, returns a list of matrices that represent elements of one of the following groups:

  (1) C2 - cyclic group of 2 x 2 matrices of order 2, hence |C2| = 2 (see Example 13 in Section 7.2 in [1]).
  (2) C4 - a cyclic group of 2 x 2 matrices of order 4, hence |C4| = 4 (see Example 3 and Exercise 16 in Section 7.2 in [1]).
  (3) C8 - a cyclic group of 3 x 3 matrices of order 8, hence |C8| = 8 (see Example 3 in Section 7.2 and Example 6 in Section 7.3 in [1]).
  (4) D6 - a dihedral group of 3 x 3 matrices of order 12, hence |D6| =12. D6 is a the symmetry group of an equilateral triangle (see Example 2.2.6 in [2]).
  (5) D8 - a dihedral group of 2 x 2 matrices of order 16, hence |D8| = 16. D8 is the symmetry group of a regular hexagon in the plane (see Section 2.2 in [2]).
  (6) V4 - the Klein four-group of 2 x 2 matrices of order 4, hence |V4| = 4 (see Example 12 and Exercise 15 in Section 7.2 in [1]).
  (7) C3 - a cyclic group of 2 x 2 matrices of order 3, hence |C3| =3 (see Exercise 7 in Section 7.3 in [1]).
  (8) C6 - a cyclic group of 2 x 2 matrices of order 6, hence |C6|=6 (see Exercise 7 in Section 7.3 in [1]) .
  (9) G8 - a group of diagonal 3 x 3 matrice of order 8, hence |G8|=8 (see Exercise 10 in Section 7.3 in [1]).

- This procedure has a remember table stored in the library. This means that new groups could be

assigned and remembered to this procedure.

- This procedure is used later when computing group invariants of SP:-FiniteGroups with the SP:-Reynolds operator.
- References:
  [1] D. Cox, J. Little, D. O'Shea: "Ideals, varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra", Third Edition, Springer, New York, 2007
  [2] B. Sturmfels: "Algorithms in Invariant Theory", Texts and Monographs in Symbolic Computation, Springer-Verlag/Wien, 1993

Examples:

```
> restart:with(SP):
Remember table of SymmetricGroup has been read and assigned
Remember table of AlternatingGroup has been read and assigned
Remember table of Reynolds has been read and assigned
Remember table of FiniteGroups has been read and assigned
Remember table of generateGinvariants has been read and assigned
>
```

Example 1: Display names of groups for which matrix representations are stored in the remember table of 'FiniteGroups':

```
> op(sort(map(op,[indices(op(4,eval(SP:-FiniteGroups)))])));
```

$$C2, C3, C4, C6, C8, \mathrm{D6}, \mathrm{D8}, G8, V4$$

When a user defines a new group and then assigns this new group entry to 'FiniteGroups', the new group is remembered and upon unloading this package, it is stored in the library. Upon loading again this package, 'FiniteGroups' will know this newly defined group. For example, we can define a new group Gamma3. This is a three dimensional representation of the cyclic group of order 4.

```
> A1:=linalg:-diag(1$3);
  A2:=matrix(3,3,[0,1,0,-1,0,0,0,0,-1]);
  A3:=matrix(3,3,[-1,0,0,0,-1,0,0,0,1]);
  A4:=matrix(3,3,[0,-1,0,1,0,0,0,0,-1]);
  Gamma3:=map(evalm,[A1,A2,A3,A4]);
```

$$A1 := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A2 := \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

$$A3 := \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A4 := \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

$$\Gamma 3 := \left[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}\right]$$

```
> FiniteGroups('Gamma3');
```

```
Group Gamma3 is not known to FiniteGroups. You can make it known by assigning i
t to FiniteGroups('Gamma3').
```

```
> FiniteGroups('Gamma3'):=Gamma3;
```

$$SP{:}\text{-}FiniteGroups(\Gamma 3) := \left[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}\right]$$

```
> FiniteGroups('Gamma3');
```

$$\left[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}\right]$$

Now, we restart Maple and reload the SP package: Notice that the Gamma3 group is now available and need not be defined once more.

```
> restart:
> with(SP);
```

```
Remember table of SymmetricGroup has been read and assigned
Remember table of AlternatingGroup has been read and assigned
Remember table of Reynolds has been read and assigned
Remember table of FiniteGroups has been read and assigned
Remember table of generateGinvariants has been read and assigned
```

[*AlternatingGroup*, *Dpolynom*, *FiniteGroups*, *Hilbert_series*, *MatrixAction*, *ModuleLoad*, *ModuleUnload*, *Molien_series*, *Reynolds*, *SPversion*, *Schur_polynom*, Σ, *SymmetricGroup*, *SyzygyIdeal*, *a_polynom*, *create_partitions*, *generateGinvariants*, *gpolynom*, *hpolynom*, *isContained*, *isGinvariant*, *isSymmetric*, *ispartition*, *load_remember_table*, *maxmindegree*, *permsign*, *powersum*, *reduceGinvariants*, *sigma_to_powersum*]

```
> op(sort(map(op,[indices(op(4,eval(SP:-FiniteGroups)))])));
```

C2, *C3*, *C4*, *C6*, *C8*, D6, D8, *G8*, Γ3, *V4*

```
> FiniteGroups('Gamma3');
```

$$\left[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}\right]$$

```
> FiniteGroups('C2');
```

$$\left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}\right]$$

```
> FiniteGroups('C4');
```

$$\left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\right]$$

```
> FiniteGroups('C8');
```

$$\left[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}\right]$$

```
> FiniteGroups('D6');
```

$$\left[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 \\ \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} \frac{-1}{2} & -\frac{\sqrt{3}}{2} & 0 \\ \frac{\sqrt{3}}{2} & \frac{-1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} \frac{-1}{2} & \frac{\sqrt{3}}{2} & 0 \\ -\frac{\sqrt{3}}{2} & \frac{-1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} \frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ -\frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 \\ -\frac{\sqrt{3}}{2} & \frac{-1}{2} & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} \frac{-1}{2} & -\frac{\sqrt{3}}{2} & 0 \\ -\frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} \frac{-1}{2} & \frac{\sqrt{3}}{2} & 0 \\ \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} \frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ \frac{\sqrt{3}}{2} & \frac{-1}{2} & 0 \\ 0 & 0 & -1 \end{bmatrix}\right]$$

```
> FiniteGroups('D8');
```

$$\left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}, \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}, \begin{bmatrix} -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}, \begin{bmatrix} -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}, \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}, \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix},$$

$$\begin{bmatrix} -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}, \begin{bmatrix} -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}\Bigg]$$

```
> nops(%);
```

$$16$$

```
> FiniteGroups('V4');
```

$$\left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}\right]$$

```
> FiniteGroups('C3');
```

$$\left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & -1 \\ 1 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 1 \\ -1 & 0 \end{bmatrix}\right]$$

```
> FiniteGroups('C6');
```

$$\left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 1 \\ -1 & 0 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} 0 & -1 \\ 1 & -1 \end{bmatrix}, \begin{bmatrix} 1 & -1 \\ 1 & 0 \end{bmatrix}\right]$$

```
> FiniteGroups('G8');
```

$$\left[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}\right]$$

```
>
```

Example 2: Let's see an action of the group G8 on a polynomial:

```
> G:=FiniteGroups('G8');
```

$$G := \left[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}\right]$$

```
> f:=x^3-y^2+x*y+z+z^5;
```

$$f := x^3 - y^2 + x\,y + z + z^5$$

```
> for A in G do MatrixAction(A,f,[x,y,z]) end do;
```

$$x^3 - y^2 + x\,y + z + z^5$$
$$x^3 - y^2 + x\,y - z - z^5$$
$$x^3 - y^2 - x\,y + z + z^5$$
$$x^3 - y^2 - x\,y - z - z^5$$

$$-x^3 - y^2 - x\,y + z + z^5$$
$$-x^3 - y^2 - x\,y - z - z^5$$
$$-x^3 - y^2 + x\,y + z + z^5$$
$$-x^3 - y^2 + x\,y - z - z^5$$

```
>
```

Algorithm used:

None

See Also: SP:-Sigma, SP:-AlternatingGroup, SP:-SymmetricGroup, SP:-MatrixAction, SP:-Reynolds, Groebner

Last modified: June 19, 2008

Function: SP:-generateGinvariants - generates invariants of a finite group using Emmy Noether's Theorem

Calling Sequence:

generateGinvariants[G](L);

Parameters:

- G is a list of n x n matrices that represent a finite group G
- L is a list of all ring indeterminates, that is, L = [x[1], x[2], x[3],..., x[n]], presumably in some polynomial ring *k*[x[1], x[2], x[3],..., x[n]].

Output:

- A list of G-invariants which are usually algebraically related in the ring *k*[x[1], x[2], x[3],..., x[n]]

Description:

- Procedure 'generateGinvariants' generates polynomial invariants of a finite group G of n x n matrices. Each invariant is obtained, according to Emmy Noether's Theorem (see Theorem 5 on page 331 in [1]), by applying the Reynolds operator SP:-Reynolds parametrized by the group G to all monomial terms in n variables in the ring *k*[x[1], x[2], x[3],..., x[n]] entered as a list L. The monomial terms have total degrees ranging from 1 to the order |G| of the group G, that is, they are of the type x^beta, 1 <= |beta| <= |G|.
- Elements of the group G are entered as an index as in generateGinvariants[G](L). Note that Reynolds[G](x^beta1) could yield the same result as Reynolds[G](x^beta2) for beta1 <> beta2. This procedure removes duplicates in the returned list. Also, some monomials when acted on with the Reynolds operator give 0: Thus, zeros are also removed from the output of this procedure. Therefeore, the number of homogeneoues G invariants returned can and usually is smaller than the total number of monomial terms of type x^beta, 1 <= |beta| <= |G|.
- Note also that the G-invariant polynomials returned by this procedure are not necessarily algebraically independent in the ring *k*[x[1], x[2], x[3],..., x[n]]. Relations among the invariants are found with the procedure SP:-SyzygyIdeal. Dependencies in the list of G-invariants can be removed by the procedure SP:-reduceGinvariants.
- To find out which groups are already remembered, check SP:-FiniteGroups.
- References:
  [1] D. Cox, J. Little, D. O'Shea: "Ideals, varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra", Third Edition, Springer, New York, 2007
  [2] B. Sturmfels: "Algorithms in Invariant Theory", Texts and Monographs in Symbolic Computation, Springer-Verlag/Wien, 1993

Examples:

```
> restart:with(SP);
Remember table of SymmetricGroup has been read and assigned
Remember table of AlternatingGroup has been read and assigned
Remember table of Reynolds has been read and assigned
Remember table of FiniteGroups has been read and assigned
Remember table of generateGinvariants has been read and assigned
```

[*AlternatingGroup*, *Dpolynom*, *FiniteGroups*, *Hilbert_series*, *MatrixAction*, *ModuleLoad*, *ModuleUnload*, *Molien_series*, *Reynolds*, *SPversion*, *Schur_polynom*, Σ, *SymmetricGroup*, *SyzygyIdeal*, *a_polynom*, *create_partitions*, *generateGinvariants*, *gpolynom*, *hpolynom*, *isContained*, *isGinvariant*, *isSymmetric*, *ispartition*, *load_remember_table*, *maxmindegree*, *permsign*, *powersum*, *reduceGinvariants*, *sigma_to_powersum*]

```
>
```

Example 1: We will generate invariants of a few finite groups:

```
> S2:=SymmetricGroup(2);
  C4:=FiniteGroups('C4');
  S3:=SymmetricGroup(3);
  Gamma3:=FiniteGroups('Gamma3');
  D6:=FiniteGroups('D6');
```

$$S2 := \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\right]$$

$$C4 := \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\right]$$

$$S3 := \left[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}\right]$$

$$\Gamma 3 := \left[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}\right]$$

$$D6 := \left[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 \\ \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} \frac{-1}{2} & -\frac{\sqrt{3}}{2} & 0 \\ \frac{\sqrt{3}}{2} & \frac{-1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix},\right.$$

$$\begin{bmatrix} \frac{-1}{2} & \frac{\sqrt{3}}{2} & 0 \\ -\frac{\sqrt{3}}{2} & \frac{-1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} \frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ -\frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 \\ -\frac{\sqrt{3}}{2} & \frac{-1}{2} & 0 \\ 0 & 0 & -1 \end{bmatrix},$$

$$\begin{bmatrix} \frac{-1}{2} & -\frac{\sqrt{3}}{2} & 0 \\ -\frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} \frac{-1}{2} & \frac{\sqrt{3}}{2} & 0 \\ \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} \frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ \frac{\sqrt{3}}{2} & \frac{-1}{2} & 0 \\ 0 & 0 & -1 \end{bmatrix}\Bigg]$$

```
> start:=time():
  S2invariants:=generateGinvariants[S2]([x,y]);
  nops(S2invariants);
  time()-start;
  convert(map(isGinvariant[S2],S2invariants,[x,y]),set);
```

$$S2invariants := [x\,y, y + x, x^2 + y^2]$$

$$3$$

$$0.203$$

$$\{true\}$$

```
> start:=time():
  C4invariants:=generateGinvariants[C4]([x,y]);
  nops(C4invariants);
  time()-start;
  convert(map(isGinvariant[C4],C4invariants,[x,y]),set);
```

$$C4invariants := [y^4 + x^4, x^2 + y^2, x\,y^3 - x^3\,y, x^3\,y - x\,y^3, x^2\,y^2]$$

$$5$$

$$0.172$$

$$\{true\}$$

```
> start:=time():
  S3invariants:=generateGinvariants[S3]([x,y,z]);
  nops(S3invariants);
  time()-start;
  convert(map(isGinvariant[S3],S3invariants,[x,y,z]),set);
```

$$S3invariants := [x\,y\,z, z + y + x, x\,y^2\,z^3 + x\,y^3\,z^2 + x^2\,y\,z^3 + x^2\,y^3\,z + x^3\,y\,z^2 + x^3\,y^2\,z,$$
$$y\,z^5 + y^5\,z + x\,z^5 + x\,y^5 + x^5\,z + x^5\,y, y\,z^4 + y^4\,z + x\,z^4 + x\,y^4 + x^4\,z + x^4\,y, x\,y + x\,z + y\,z,$$
$$y^2\,z^3 + y^3\,z^2 + x^2\,z^3 + x^2\,y^3 + x^3\,z^2 + x^3\,y^2, y\,z^3 + y^3\,z + x\,z^3 + x\,y^3 + x^3\,z + x^3\,y,$$
$$x\,y^2\,z^2 + x^2\,y\,z^2 + x^2\,y^2\,z, x^2\,y^2\,z^2, z^2 + y^2 + x^2, y\,z^2 + y^2\,z + x\,z^2 + x\,y^2 + x^2\,z + x^2\,y,$$
$$x\,y\,z^4 + x\,y^4\,z + x^4\,y\,z, x\,y\,z^3 + x\,y^3\,z + x^3\,y\,z, z^3 + y^3 + x^3, y^2\,z^2 + x^2\,z^2 + x^2\,y^2,$$

$y^2 z^4 + y^4 z^2 + x^2 z^4 + x^2 y^4 + x^4 z^2 + x^4 y^2, z^4 + y^4 + x^4, z^5 + y^5 + x^5, z^6 + y^6 + x^6,$
$x y z^2 + x y^2 z + x^2 y z, y^3 z^3 + x^3 z^3 + x^3 y^3]$

$$22$$

$$0.765$$

$$\{true\}$$

```
> start:=time():
  Gamma3invariants:=generateGinvariants[Gamma3]([x,y,z]);
  nops(Gamma3invariants);
  time()-start;
  convert(map(isGinvariant[Gamma3],Gamma3invariants,[x,y,z]),set)
  ;
```

Gamma3invariants :=

$$[x^2 z - y^2 z, y^2 z - x^2 z, y^4 + x^4, x y z, z^2, z^4, x^2 + y^2, x y^3 - x^3 y, x^3 y - x y^3, x^2 y^2, y^2 z^2 + x^2 z^2]$$

$$11$$

$$0.172$$

$$\{true\}$$

```
> start:=time():
  D6invariants:=generateGinvariants[D6]([x,y,z]);
  nops(D6invariants);
  time()-start;
  convert(map(isGinvariant[D6],D6invariants,[x,y,z]),set);
```

D6invariants := $[30 x^4 y^4 z^4 + 76 x^6 y^2 z^4 + x^8 z^4 + 9 y^8 z^4 - 36 x^2 y^6 z^4,$
$30 x^4 y^4 + 76 x^6 y^2 + x^8 + 9 y^8 - 36 x^2 y^6, -7 x^5 y^3 z + 3 x^7 y z - 7 x^3 y^5 z + 3 x y^7 z,$
$6 x^4 y^4 + 4 x^6 y^2 + x^8 + y^8 + 4 x^2 y^6,$
$322 x^4 y^6 z^2 - 42 x^6 y^4 z^2 - 99 x^8 y^2 z^2 + 27 x^{10} z^2 + y^{10} z^2 + 239 x^2 y^8 z^2,$
$30 x^4 y^4 - 36 x^6 y^2 + 9 x^8 + y^8 + 76 x^2 y^6, y^4 z^6 + x^4 z^6 + 2 x^2 y^2 z^6, 3 x^5 y z + 3 x y^5 z - 10 x^3 y^3 z,$
$z^2, z^4, z^6, 14 x^5 y^5 z + 4 x^7 y^3 z - 3 x y^9 z - 3 x^9 y z + 4 x^3 y^7 z,$
$210 x^4 y^4 z^4 + 252 x^6 y^2 z^4 + 27 x^8 z^4 + 43 y^8 z^4 + 28 x^2 y^6 z^4,$
$11 x^6 z^4 + 45 x^2 y^4 z^4 + 9 y^6 z^4 + 15 x^4 y^2 z^4, 9 x^6 z^4 + 15 x^2 y^4 z^4 + 11 y^6 z^4 + 45 x^4 y^2 z^4, x^2 + y^2,$
$-3 x^5 y z^5 + 10 x^3 y^3 z^5 - 3 x y^5 z^5, 6 x^4 y^4 z^2 + 4 x^6 y^2 z^2 + x^8 z^2 + y^8 z^2 + 4 x^2 y^6 z^2,$
$x^6 z^2 - 9 x^2 y^4 z^2 + 3 y^6 z^2 + 21 x^4 y^2 z^2,$
$381 x^4 y^8 - 644 x^6 y^6 + 1029 x^8 y^4 + 3 x^{12} + 162 x^{10} y^2 - 126 x^2 y^{10} + 27 y^{12},$
$210 x^4 y^4 z^4 + 28 x^6 y^2 z^4 + 43 x^8 z^4 + 27 y^8 z^4 + 252 x^2 y^6 z^4,$
$14 x^5 y^5 z + 4 x^7 y^3 z - 3 x y^9 z - 3 x^9 y z + 4 x^3 y^7 z,$
$34 x^4 y^6 + 6 x^6 y^4 - 3 x^8 y^2 + 3 x^{10} + y^{10} + 23 x^2 y^8, x^2 z^{10} + y^2 z^{10},$
$1215 x^4 y^8 + 1764 x^6 y^6 - 945 x^8 y^4 + 81 x^{12} - 162 x^{10} y^2 + 798 x^2 y^{10} + y^{12},$

$322 x^4 y^6 - 42 x^6 y^4 - 99 x^8 y^2 + 27 x^{10} + y^{10} + 239 x^2 y^8$, $15 x^2 y^4 + 11 y^6 + 9 x^6 + 45 x^4 y^2$,
$-14 x^5 y^5 z - 4 x^7 y^3 z + 3 x y^9 z + 3 x^9 y z - 4 x^3 y^7 z$, $x^4 z^4 + y^4 z^4 + 2 x^2 y^2 z^4$,
$6 x^4 y^4 z^4 + 4 x^6 y^2 z^4 + x^8 z^4 + y^8 z^4 + 4 x^2 y^6 z^4$, $210 x^4 y^4 + 252 x^6 y^2 + 27 x^8 + 43 y^8 + 28 x^2 y^6$,
$6 x^4 y^6 z^2 + 34 x^6 y^4 z^2 + 23 x^8 y^2 z^2 + x^{10} z^2 + 3 y^{10} z^2 - 3 x^2 y^8 z^2$,
$3 x^5 y z^3 + 3 x y^5 z^3 - 10 x^3 y^3 z^3$, $210 x^4 y^4 + 28 x^6 y^2 + 43 x^8 + 27 y^8 + 252 x^2 y^6$,
$30 x^4 y^4 z^4 - 36 x^6 y^2 z^4 + 9 x^8 z^4 + y^8 z^4 + 76 x^2 y^6 z^4$,
$210 x^4 y^4 z^2 + 252 x^6 y^2 z^2 + 27 x^8 z^2 + 43 y^8 z^2 + 28 x^2 y^6 z^2$, $y^4 z^2 + x^4 z^2 + 2 x^2 y^2 z^2$,
$-9 x^2 y^4 + 3 y^6 + x^6 + 21 x^4 y^2$, $x^2 z^4 + y^2 z^4$, $x^2 z^8 + y^2 z^8$, $x^2 z^6 + y^2 z^6$, $y^4 z^6 + x^4 z^6 + 2 x^2 y^2 z^6$,
$-945 x^4 y^8 + 1764 x^6 y^6 + 1215 x^8 y^4 + x^{12} + 798 x^{10} y^2 - 162 x^2 y^{10} + 81 y^{12}$,
$30 x^4 y^4 z^2 - 36 x^6 y^2 z^2 + 9 x^8 z^2 + y^8 z^2 + 76 x^2 y^6 z^2$, $y^4 z^2 + x^4 z^2 + 2 x^2 y^2 z^2$, $2 x^2 y^2 + y^4 + x^4$,
$1485 x^4 y^8 + 8316 x^6 y^6 + 13365 x^8 y^4 + 243 x^{12} + 5346 x^{10} y^2 + 66 x^2 y^{10} + 683 y^{12}$,
$210 x^4 y^4 z^2 + 28 x^6 y^2 z^2 + 43 x^8 z^2 + 27 y^8 z^2 + 252 x^2 y^6 z^2$,
$x^6 z^6 - 9 x^2 y^4 z^6 + 3 y^6 z^6 + 21 x^4 y^2 z^6$, $30 x^4 y^4 z^2 + 76 x^6 y^2 z^2 + x^8 z^2 + 9 y^8 z^2 - 36 x^2 y^6 z^2$,
$-42 x^4 y^6 + 322 x^6 y^4 + 239 x^8 y^2 + x^{10} + 27 y^{10} - 99 x^2 y^8$,
$11 x^6 z^6 + 45 x^2 y^4 z^6 + 9 y^6 z^6 + 15 x^4 y^2 z^6$, $3 x^6 z^6 + 21 x^2 y^4 z^6 + y^6 z^6 - 9 x^4 y^2 z^6$,
$1029 x^4 y^8 - 644 x^6 y^6 + 381 x^8 y^4 + 27 x^{12} - 126 x^{10} y^2 + 162 x^2 y^{10} + 3 y^{12}$,
$9 x^6 z^6 + 15 x^2 y^4 z^6 + 11 y^6 z^6 + 45 x^4 y^2 z^6$,
$210 x^4 y^6 + 70 x^6 y^4 + 5 x^8 y^2 + 19 x^{10} + 9 y^{10} + 135 x^2 y^8$,
$9 x^6 z^2 + 15 x^2 y^4 z^2 + 11 y^6 z^2 + 45 x^4 y^2 z^2$,
$13365 x^4 y^8 + 8316 x^6 y^6 + 1485 x^8 y^4 + 683 x^{12} + 66 x^{10} y^2 + 5346 x^2 y^{10} + 243 y^{12}$,
$2 x^2 y^2 z^8 + x^4 z^8 + y^4 z^8$, $7 x^5 y^3 z^3 - 3 x^7 y z^3 + 7 x^3 y^5 z^3 - 3 x y^7 z^3$, z^8, z^{10}, z^{12},
$21 x^2 y^4 + y^6 + 3 x^6 - 9 x^4 y^2$, $-3 x^5 y z - 3 x y^5 z + 10 x^3 y^3 z$, $3 x^5 y z^5 - 10 x^3 y^3 z^5 + 3 x y^5 z^5$,
$2 x^2 y^2 z^8 + x^4 z^8 + y^4 z^8$, $-3 x^5 y z^3 - 3 x y^5 z^3 + 10 x^3 y^3 z^3$,
$11 x^6 z^2 + 45 x^2 y^4 z^2 + 9 y^6 z^2 + 15 x^4 y^2 z^2$,
$34 x^4 y^6 z^2 + 6 x^6 y^4 z^2 - 3 x^8 y^2 z^2 + 3 x^{10} z^2 + y^{10} z^2 + 23 x^2 y^8 z^2$,
$3 x^6 z^4 + 21 x^2 y^4 z^4 + y^6 z^4 - 9 x^4 y^2 z^4$,
$210 x^4 y^6 z^2 + 70 x^6 y^4 z^2 + 5 x^8 y^2 z^2 + 19 x^{10} z^2 + 9 y^{10} z^2 + 135 x^2 y^8 z^2$,
$-345 x^4 y^8 + 1124 x^6 y^6 - 345 x^8 y^4 + 9 x^{12} + 126 x^{10} y^2 + 126 x^2 y^{10} + 9 y^{12}$, $y^2 z^2 + x^2 z^2$,
$-7 x^5 y^3 z^3 + 3 x^7 y z^3 - 7 x^3 y^5 z^3 + 3 x y^7 z^3$, $6 x^4 y^6 + 34 x^6 y^4 + 23 x^8 y^2 + x^{10} + 3 y^{10} - 3 x^2 y^8$,
$70 x^4 y^6 z^2 + 210 x^6 y^4 z^2 + 135 x^8 y^2 z^2 + 9 x^{10} z^2 + 19 y^{10} z^2 + 5 x^2 y^8 z^2$, $2 x^2 y^2 + y^4 + x^4$,
$-42 x^4 y^6 z^2 + 322 x^6 y^4 z^2 + 239 x^8 y^2 z^2 + x^{10} z^2 + 27 y^{10} z^2 - 99 x^2 y^8 z^2$,
$7 x^5 y^3 z - 3 x^7 y z + 7 x^3 y^5 z - 3 x y^7 z$, $3 x^6 z^2 + 21 x^2 y^4 z^2 + y^6 z^2 - 9 x^4 y^2 z^2$,
$x^6 z^4 - 9 x^2 y^4 z^4 + 3 y^6 z^4 + 21 x^4 y^2 z^4$, $45 x^2 y^4 + 9 y^6 + 11 x^6 + 15 x^4 y^2$,

$$x^4 z^4 + y^4 z^4 + 2 x^2 y^2 z^4, 70 x^4 y^6 + 210 x^6 y^4 + 135 x^8 y^2 + 9 x^{10} + 19 y^{10} + 5 x^2 y^8]$$

$$85$$

$$7.532$$

$$\{true\}$$

The above shows that the algorithm based on Emmy Noether's Theorem, although it generates G-invariants, is not efficient since often two or more different monomials may give the same invariant, some monomials may give the trivial invariant zero, and the resulting monomials are usually algebraically dependent.

```
>
>
```

Algorithm used:

- Emmy Noether's Theorem (see Theorem 5 on page 331 in [1]):

  Step1: Make a list T of all monomial terms x^beta in the ring *k*[x[1], x[2], x[3],..., x[n]] such that 1 <= |beta| <= |G| where |beta| is the total order of the monomial x^beta. Here, x^beta = x[1]^beta[1]*x[2]^beta[2]*...*x[n]^beta[n], beta = [beta[1],beta[2],...,beta[n]] and |beta| = add(beta[i],i=1..n).

  Step 2: Apply the Reynolds operator R parametrized by the group elements of a finite group G to each monomial term in the list T from step 1. Call thew new list RT.

  Step 3: Remove all zeros and duplicates in the list RT and return the resulting list.

- Note that this procedure does not remove algebraic dependencies in the returned list. To remove dependencies use procedure SP:-reduceGinvariants.

See Also: SP:-FiniteGroups, SP:-Reynolds, SP:-SymmetricGroup, SP:-reduceGinvariants, SP:-SyzygyIdeal, SP:-MatrixAction

Last modified: June 19, 2008

Function: SP:-gpolynom - returns a g polynomial

Calling Sequence:

gpolynom[k](s);

Parameters:

- The index k must be a positive integer such that 1 <= k <= n where n is the number of variables in the sequence s
- s must be a sequence of n variables

Output:

- A polynomial in n variables specified in the sequence s and additional n variables y[1], y[2], ..., y[k]

Description:

- Procedure gpolynom used as in gpolynom[k](x[k],...x[n]) returns the g[k] polynomial in variables x[k], x[k+1],..., x[n], y[1], y[2],..., y[k] defined in Proposition 5, Section 7.1 in [1]. These polynomials are defined in terms of the complete symmetric polynomials SP:-hpolynom.
- Polynomials gpolynom[k], 1<= k <= n, give a Groebner basis for the ideal <sigma[1] - y[1], ..., sigma[n]-y[n]> in k[x[1],...,x[n],y[1],...,y[n]] for the lex order x[1]>x[2]>...>x[n]>y[1]>y[2]>...>y[n] according to Proposition 5, Section 7.1 in [1]. This ideal is needed in order to determine if an arbitrary polynomial in n variables is symmetric. See the procedure SP:-isSymmetric for more details.
- References:
  [1] D. Cox, J. Little, D. O'Shea: "Ideals, varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra", Third Edition, Springer, New York, 2007
  [2] B. Sturmfels: "Algorithms in Invariant Theory", Texts and Monographs in Symbolic Computation, Springer-Verlag/Wien, 1993

Examples:

```
> restart:with(SP);
Remember table of SymmetricGroup has been read and assigned
Remember table of AlternatingGroup has been read and assigned
Remember table of Reynolds has been read and assigned
Remember table of FiniteGroups has been read and assigned
Remember table of generateGinvariants has been read and assigned
```

[*AlternatingGroup*, *Dpolynom*, *FiniteGroups*, *Hilbert_series*, *MatrixAction*, *ModuleLoad*,

ModuleUnload, *Molien_series*, *Reynolds*, *SPversion*, *Schur_polynom*, Σ, *SymmetricGroup*, *SyzygyIdeal*, *a_polynom*, *create_partitions*, *generateGinvariants*, *gpolynom*, *hpolynom*, *isContained*, *isGinvariant*, *isSymmetric*, *ispartition*, *load_remember_table*, *maxmindegree*, *permsign*, *powersum*, *reduceGinvariants*, *sigma_to_powersum*]

```
>
> gpolynom[2](u[1],u[2],u[3]);
```

$$u_2 u_3 + u_2^2 + u_3^2 - (u_2 + u_3) y_1 + y_2$$

```
> gpolynom[3](x,y,z,t);
```

$$z t^2 + z^2 t + z^3 + t^3 - (z t + z^2 + t^2) y_1 + (z + t) y_2 - y_3$$

```
> gpolynom[0](x,y); #testing an error message
Error, (in SP:-gpolynom[0]) index must be of type posint and it must be no
larger than number of arguments
>
```

Example 1: Displaying various gpolynom[k](u[1],u[2],...,u[n]) when n = 4, 5:

```
> n:=4:
  gpolynom[2](seq(x[i],i=1..n));
```

$$x_2 x_3 + x_2 x_4 + x_3 x_4 + x_2^2 + x_3^2 + x_4^2 - (x_2 + x_3 + x_4) y_1 + y_2$$

```
> n:=5:
  for k from 1 to n do
      g[k]:=gpolynom[k](seq(x[i],i=1..n));
  end do;
```

$$g_1 := x_1 + x_2 + x_3 + x_4 + x_5 - y_1$$

$$g_2 := x_2 x_3 + x_2 x_4 + x_2 x_5 + x_3 x_4 + x_3 x_5 + x_4 x_5 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - (x_2 + x_3 + x_4 + x_5) y_1 + y_2$$

$$g_3 := x_3 x_4 x_5 + x_3 x_4^2 + x_3 x_5^2 + x_3^2 x_4 + x_3^2 x_5 + x_4 x_5^2 + x_4^2 x_5 + x_3^3 + x_4^3 + x_5^3 - (x_3 x_4 + x_3 x_5 + x_4 x_5 + x_3^2 + x_4^2 + x_5^2) y_1 + (x_3 + x_4 + x_5) y_2 - y_3$$

$$g_4 := x_4^2 x_5^2 + x_4 x_5^3 + x_4^3 x_5 + x_4^4 + x_5^4 - (x_4 x_5^2 + x_4^2 x_5 + x_4^3 + x_5^3) y_1 + (x_4 x_5 + x_4^2 + x_5^2) y_2 - (x_4 + x_5) y_3 + y_4$$

$$g_5 := x_5^5 - x_5^4 y_1 + x_5^3 y_2 - x_5^2 y_3 + x_5 y_4 - y_5$$

```
>
```

Example 2: Let's compute a Groebner basis for the ideal J = <sigma[1] - y[1], sigma[2] - y[2], sigma[3] - y[3]> where sigma[1], sigma[2], sigma[3] are elementary symmetric polynomials in three variables x[1], x[2], x[3] in the lex order x[1] > x[2] > x[3] > y[1] > y[2] > y[3]. We do it directly and then we compare with the g polynomials obtained using the command gpolynom:

```
> sigma[1]:=Sigma[1](x[1],x[2],x[3]);
  sigma[2]:=Sigma[2](x[1],x[2],x[3]);
  sigma[3]:=Sigma[3](x[1],x[2],x[3]);
```

$$\sigma_1 := x_1 + x_2 + x_3$$

$$\sigma_2 := x_1 x_2 + x_1 x_3 + x_2 x_3$$

$$\sigma_3 := x_1 x_2 x_3$$

```
> F:=[seq(sigma[i]-y[i],i=1..3)];
```

$$F := [x_1 + x_2 + x_3 - y_1, x_1 x_2 + x_1 x_3 + x_2 x_3 - y_2, x_1 x_2 x_3 - y_3]$$

```
> G:=Groebner:-Basis(F,plex(x[1],x[2],x[3],y[1],y[2],y[3]));
```

$$G := [x_3 y_2 + x_3^3 - y_1 x_3^2 - y_3, y_2 + x_2^2 + x_2 x_3 - x_2 y_1 + x_3^2 - y_1 x_3, x_1 + x_2 + x_3 - y_1]$$

The above basis should consist of polynomials g[1], g[2], and g[3]:

```
> g[1]:=gpolynom[1](x[1],x[2],x[3]);
  g[2]:=gpolynom[2](x[1],x[2],x[3]);
  g[3]:=gpolynom[3](x[1],x[2],x[3]);
```

$$g_1 := x_1 + x_2 + x_3 - y_1$$

$$g_2 := x_2 x_3 + x_2^2 + x_3^2 - (x_2 + x_3) y_1 + y_2$$

$$g_3 := x_3 y_2 + x_3^3 - y_1 x_3^2 - y_3$$

Observe that G[1] = g[3], G[2] = g[2] and G[3] = g[1] as predicted by the Proposition 5.:

```
> simplify(G[1]-g[3]);
  simplify(G[2]-g[2]);
  simplify(G[3]-g[1]);
```

$$0$$

$$0$$

$$0$$

```
>
```

Algorithm used:

None

See Also: SP:-Sigma, SP:-isSymmetric, SP:-hpolynom, FGb, Groebner

Last modified: June 19, 2008

Function: SP:-hpolynom - returns a complete symmetric polynomial

Calling Sequence:

hpolynom[i](s);

Parameters:

- The index i must be a non negative integer
- s must be a sequence of n variables

Output:

- A complete (homogeneous) symmetric polynomial of degree i in n variables.

Description:

- Procedure hpolynom used as in hpolynom[i](u[k],u[k+1],...,u[n]) returns the i-th complete symmetric polynomial in n-k+1 variables u[k],u[k+1],u[k+2],...,u[n] defined as a sum of monomial terms u[k]^ni[k]*u[k+1]^ni[k+1]*...*u[n]^ni[n] where (ni[k],ni[k+1],...,ni[n]) is an integer vector whose components are non negative integers adding up to i. There are choose(n-k+i,i) such vectors hence as many terms in the i-th polynomial (see above Proposition 5 in [1] on page 316 or above Theorem 1.2.7 in [2] page 12).
- Polynomials hpolynom[i] are used to define gpolynom[i] which give a Groebner basis for the ideal <sigma[1] - y[1], ..., sigma[n]-y[n]> in k[x[1],...,x[n],y[1],...,y[n]] for the lex order x[1]>x[2]>...>x[n]>y[1]>y[2]>...>y[n] according to Proposition 5, Section 7.1 in [1]. This ideal is needed in order to determine if an arbitrary polynomial in n variables is symmetric. See the procedure SP:-isSymmetric for more details.
- References:
  [1] D. Cox, J. Little, D. O'Shea: "Ideals, varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra", Third Edition, Springer, New York, 2007
  [2] B. Sturmfels: "Algorithms in Invariant Theory", Texts and Monographs in Symbolic Computation, Springer-Verlag/Wien, 1993

Examples:

```
> restart:with(SP);
Remember table of SymmetricGroup has been read and assigned
Remember table of AlternatingGroup has been read and assigned
Remember table of Reynolds has been read and assigned
Remember table of FiniteGroups has been read and assigned
Remember table of generateGinvariants has been read and assigned
```

[*AlternatingGroup*, *Dpolynom*, *FiniteGroups*, *Hilbert_series*, *MatrixAction*, *ModuleLoad*, *ModuleUnload*, *Molien_series*, *Reynolds*, *SPversion*, *Schur_polynom*, Σ, *SymmetricGroup*, *SyzygyIdeal*, *a_polynom*, *create_partitions*, *generateGinvariants*, *gpolynom*, *hpolynom*, *isContained*, *isGinvariant*, *isSymmetric*, *ispartition*, *load_remember_table*, *maxmindegree*, *permsign*, *powersum*, *reduceGinvariants*, *sigma_to_powersum*]

```
> hpolynom[2](u[1],u[2],u[3]);
```

$$u_1 u_2 + u_1 u_3 + u_2 u_3 + u_1^2 + u_2^2 + u_3^2$$

```
> hpolynom[2](x,y,z,t);
```

$$x y + x z + x t + y z + y t + z t + x^2 + y^2 + z^2 + t^2$$

```
> hpolynom[0](x,y);
```

$$1$$

```
>
```

Example 1: Displaying hpolynom[i](u[1],u[2],...,u[n]) when n = 4 as well as counting the number of terms in each polynomial:

```
> n:=4:k:=1:
> for i from 1 to n do

  [nops(combinat:-choose(n-k+i,i)),hpolynom[i](seq(u[j],j=k..n)),
  nops(hpolynom[i](seq(u[j],j=k..n)))];
  end do;
```

$$[4, u_1 + u_2 + u_3 + u_4, 4]$$

$$[10, u_1 u_2 + u_1 u_3 + u_1 u_4 + u_2 u_3 + u_2 u_4 + u_3 u_4 + u_1^2 + u_2^2 + u_3^2 + u_4^2, 10]$$

$$[20, u_1 u_2 u_3 + u_1 u_2 u_4 + u_1 u_3 u_4 + u_2 u_3 u_4 + u_1 u_2^2 + u_1 u_3^2 + u_1 u_4^2 + u_1^2 u_2 + u_1^2 u_3 + u_1^2 u_4 + u_2 u_3^2 + u_2 u_4^2 + u_2^2 u_3 + u_2^2 u_4 + u_3 u_4^2 + u_3^2 u_4 + u_1^3 + u_2^3 + u_3^3 + u_4^3, 20]$$

$$[35, u_1 u_2 u_3 u_4 + u_1 u_2 u_3^2 + u_1 u_2 u_4^2 + u_1 u_2^2 u_3 + u_1 u_2^2 u_4 + u_1 u_3 u_4^2 + u_1 u_3^2 u_4 + u_1^2 u_2 u_3 + u_1^2 u_2 u_4 + u_1^2 u_3 u_4 + u_2^3 u_4 + u_2^2 u_3^2 + u_2 u_3^3 + u_3^2 u_4^2 + u_2 u_3 u_4^2 + u_2 u_3^2 u_4 + u_2^2 u_3 u_4 + u_2^2 u_4^2 + u_2 u_4^3 + u_2^3 u_3 + u_1^2 u_2^2 + u_1^2 u_3^2 + u_1^2 u_4^2 + u_1 u_2^3 + u_1 u_3^3 + u_1 u_4^3 + u_1^3 u_2 + u_1^3 u_3 + u_1^3 u_4 + u_3 u_4^3 + u_3^3 u_4 + u_1^4 + u_2^4 + u_3^4 + u_4^4, 35]$$

```
>
```

Example 2: Displaying hpolynom[i](u[1],u[2],...,u[n]) when n = 5 as well as counting the number of terms in each polynomial:

```
> n:=5:k:=1:
> for i from 1 to n do
```

```
[nops(combinat:-choose(n-k+i,i)),hpolynom[i](seq(u[j],j=k..n)),
nops(hpolynom[i](seq(u[j],j=k..n)))];
end do;
```

$$[5, u_1 + u_2 + u_3 + u_4 + u_5, 5]$$

$$[15, u_1 u_2 + u_1 u_3 + u_1 u_4 + u_1 u_5 + u_2 u_3 + u_2 u_4 + u_2 u_5 + u_3 u_4 + u_3 u_5 + u_4 u_5 + u_1^2 + u_2^2 + u_3^2 + u_4^2 + u_5^2, 15]$$

$$[35, u_2 u_3 u_4 + u_1 u_2 u_3 + u_1 u_2 u_4 + u_1 u_3 u_4 + u_1 u_2^2 + u_1 u_3^2 + u_1 u_4^2 + u_1^2 u_2 + u_1^2 u_3 + u_1^2 u_4 + u_3 u_4 u_5 + u_3 u_5^2 + u_2 u_3^2 + u_2 u_4^2 + u_2^2 u_3 + u_2^2 u_4 + u_3 u_4^2 + u_3^2 u_4 + u_1^3 + u_2^3 + u_3^3 + u_4^3 + u_2 u_3 u_5 + u_2 u_4 u_5 + u_2 u_5^2 + u_2^2 u_5 + u_3^2 u_5 + u_1 u_2 u_5 + u_1 u_3 u_5 + u_1 u_4 u_5 + u_4 u_5^2 + u_4^2 u_5 + u_1 u_5^2 + u_1^2 u_5 + u_5^3, 35]$$

$$[70, u_3^2 u_5^2 + u_3 u_4^2 u_5 + u_3 u_4 u_5^2 + u_3 u_5^3 + u_1 u_2 u_3 u_4 + u_1 u_2 u_3^2 + u_1 u_2 u_4^2 + u_1 u_2^2 u_3 + u_1 u_2^2 u_4 + u_1 u_3 u_4^2 + u_1 u_3^2 u_4 + u_1^2 u_2 u_3 + u_1^2 u_2 u_4 + u_1^2 u_3 u_4 + u_3^3 u_5 + u_2^3 u_4 + u_2^2 u_3^2 + u_2 u_3^3 + u_3^2 u_4 u_5 + u_3^2 u_4^2 + u_2 u_3 u_4^2 + u_2 u_3^2 u_4 + u_2^2 u_3 u_4 + u_2^2 u_4^2 + u_2 u_4^3 + u_2^3 u_3 + u_1^2 u_2^2 + u_1^2 u_3^2 + u_1^2 u_4^2 + u_1 u_2^3 + u_1 u_3^3 + u_1 u_4^3 + u_1^3 u_2 + u_1^3 u_3 + u_1^3 u_4 + u_3 u_4^3 + u_3^3 u_4 + u_1^4 + u_2^4 + u_3^4 + u_4^4 + u_2 u_3 u_4 u_5 + u_2 u_3 u_5^2 + u_2 u_3^2 u_5 + u_2 u_4 u_5^2 + u_2 u_4^2 u_5 + u_2^2 u_3 u_5 + u_2^2 u_4 u_5 + u_2^2 u_5^2 + u_2 u_5^3 + u_2^3 u_5 + u_1 u_2 u_3 u_5 + u_1 u_2 u_4 u_5 + u_1 u_3 u_4 u_5 + u_1 u_2 u_5^2 + u_4^2 u_5^2 + u_1 u_2^2 u_5 + u_1 u_3 u_5^2 + u_1 u_3^2 u_5 + u_1 u_4 u_5^2 + u_1 u_4^2 u_5 + u_1^2 u_2 u_5 + u_1^2 u_3 u_5 + u_1^2 u_4 u_5 + u_1^2 u_5^2 + u_1 u_5^3 + u_1^3 u_5 + u_4 u_5^3 + u_4^3 u_5 + u_5^4, 70]$$

$$[126, u_3 u_4^4 + u_3 u_4 u_5^3 + u_3^2 u_4 u_5^2 + u_3 u_4^3 u_5 + u_3^3 u_5^2 + u_3^3 u_4^2 + u_3^4 u_4 + u_3^3 u_4 u_5 + u_3 u_4^2 u_5^2 + u_3^2 u_4^3 + u_3^2 u_4^2 u_5 + u_3 u_5^4 + u_1^2 u_2 u_3^2 + u_3^2 u_5^3 + u_3^4 u_5 + u_2 u_3 u_4 u_5^2 + u_2 u_3 u_4^2 u_5 + u_2 u_3^2 u_4 u_5 + u_2^2 u_3 u_4 u_5 + u_2 u_3^2 u_4^2 + u_2 u_3^2 u_5^2 + u_2 u_4^2 u_5^2 + u_2^2 u_3 u_4^2 + u_2^2 u_3 u_5^2 + u_2^2 u_3^2 u_4 + u_2^2 u_3^2 u_5 + u_2^2 u_4 u_5^2 + u_2^2 u_4^2 u_5 + u_1 u_2 u_3 u_4 u_5 + u_1 u_2 u_3 u_4^2 + u_1 u_2 u_3 u_5^2 + u_1 u_2 u_3^2 u_4 + u_1 u_2 u_3^2 u_5 + u_1 u_2 u_4 u_5^2 + u_1 u_2 u_4^2 u_5 + u_1 u_2^2 u_3 u_4 + u_1 u_2^2 u_3 u_5 + u_1 u_2^2 u_4 u_5 + u_1 u_3 u_4 u_5^2 + u_1 u_3 u_4^2 u_5 + u_1 u_3^2 u_4 u_5 + u_1^2 u_2 u_3 u_4 + u_1^2 u_2 u_3 u_5 + u_1^2 u_2 u_4 u_5 + u_1^2 u_3 u_4 u_5 + u_1 u_2^2 u_3^2 + u_1 u_2^2 u_4^2 + u_1 u_2^2 u_5^2 + u_1 u_3^2 u_4^2 + u_1 u_3^2 u_5^2 + u_1 u_4^2 u_5^2 + u_1^2 u_2 u_4^2 + u_1^2 u_2 u_5^2 + u_1^2 u_2^2 u_3 + u_1^2 u_2^2 u_4 + u_1^2 u_2^2 u_5 + u_1^2 u_3 u_4^2 + u_1^2 u_3 u_5^2 + u_1^2 u_3^2 u_4 + u_1^2 u_3^2 u_5 + u_1^2 u_4 u_5^2 + u_1^2 u_4^2 u_5 + u_2 u_3 u_4^3 + u_2 u_3 u_5^3$$

$$+ u_2 u_3^3 u_4 + u_2 u_3^3 u_5 + u_2 u_4 u_5^3 + u_2 u_4^3 u_5 + u_2^3 u_3 u_4 + u_2^3 u_3 u_5 + u_2^3 u_4 u_5 + u_2^2 u_3^3$$
$$+ u_2^2 u_4^3 + u_2^2 u_5^3 + u_2^3 u_3^2 + u_2^3 u_4^2 + u_2^3 u_5^2 + u_2 u_3^4 + u_2 u_4^4 + u_2 u_5^4 + u_2^4 u_3 + u_2^4 u_4$$
$$+ u_2^4 u_5 + u_1 u_2 u_3^3 + u_1 u_2 u_4^3 + u_1 u_2 u_5^3 + u_1 u_2^3 u_3 + u_1 u_2^3 u_4 + u_1 u_2^3 u_5 + u_1 u_3 u_4^3$$
$$+ u_1 u_3 u_5^3 + u_1 u_3^3 u_4 + u_1 u_3^3 u_5 + u_1 u_4 u_5^3 + u_1 u_4^3 u_5 + u_1^3 u_2 u_3 + u_1^3 u_2 u_4 + u_1^3 u_2 u_5$$
$$+ u_1^3 u_3 u_4 + u_1^3 u_3 u_5 + u_1^3 u_4 u_5 + u_4^2 u_5^3 + u_4^3 u_5^2 + u_4 u_5^4 + u_4^4 u_5 + u_1^2 u_2^3 + u_1^2 u_3^3$$
$$+ u_1^2 u_4^3 + u_1^2 u_5^3 + u_1^3 u_2^2 + u_1^3 u_3^2 + u_1^3 u_4^2 + u_1^3 u_5^2 + u_1 u_2^4 + u_1 u_3^4 + u_1 u_4^4 + u_1 u_5^4$$
$$+ u_1^4 u_2 + u_1^4 u_3 + u_1^4 u_4 + u_1^4 u_5 + u_1^5 + u_2^5 + u_3^5 + u_4^5 + u_5^5, 126]$$

```
>
```

Algorithm used:

None

See Also: SP:-Sigma, SP:-isSymmetric, SP:-gpolynom, FGb, Groebner

Last modified: June 19, 2008

Function: SP:-isContained - determines if a polynomial f belongs to a ring *k*[f1, f2,..., fm]

Calling Sequence:

isContained(f, L);
isContained(f, L, r);

Parameters:

- f is a polynomial in the polynomial ring *k*[x[1], x[2], x[3],..., x[n]]
- L = [f1,f2,...,fm] is a list of polynomials in the ring *k*[x[1], x[2], x[3],..., x[n]] that generate a ring *k*[f1, f2,..., fm]
- r - any string like 'r' for the remainder

Output:

- When used with only two arugments as in isContained(f, [f1, f2,..., fm]); it returns true or false depending whether f belongs to *k*[f1, f2,..., fm] or not.
- When used with three arguments as in isContained(f, [f1, f2,..., fm],'r'); it returns false if f does not belong to the ring *k*[f1, f2,..., fm], and it returns a polynomial g in local variables _f1, _f2, ...,_fm which equals f upon substituting _f1 = f1, _f2 = f2,..., _fm = fm

Description:

- Procedure 'isContained' implements results of Proposition 7, page 334, in [1]. Namely, suppose that m polynomials f1, f2,..., fm in *k*[x[1],x[2],x[3],...,x[n]] are given as a list. Then, let f be any polynomial in *k*[x[1], x[2], x[3],..., x[n]]. This procedure tests whether f belongs to the ring k[f1, f2,..., fm] generated by the given m polynomials and, if so, it can write f in terms of f1, f2,..., fm.
- When used with only two arguments as in isContained(f, [f1, f2,..., fm]); it returns true or false depending whether f belongs to *k*[f1, f2,..., fm] or not. If f does belong to the ring *k*[f1, f2,..., fm] then there exists a polynomial g in *k*[_f1, _f2, _f3,..., _fm] such that f = g(f1, f2,... fm). That is, the local variables _f1, _f2,..., _fm are the place holders for the polynomials f1, f2,..., fm.
- When used with three arguments as in isContained(f, [f1, f2,..., fm], r); it returns false in case f does not belong to *k*[f1, f2,..., fm]. If f does belong to *k*[f1, f2,..., fm], the procedure returns a polynomial g in *k*[_f1, _f2, _f3,..., _fm] such that f = g(f1, f2,..., fm). That is, the local variables _f1, _f2,..., _fm are the place holders for the polynomials f1, f2,..., fm.
- The polynomial g, if it exists, is in general is not unique as any polynomial of the form g + h where h belongs to the syzygy ideal of the polynomials f1, f2,..., fm will also give f. This is because the syzygy ideal is generated by relations between the polynomials f1, f2,..., fm, if any. Each such relation is identically zero upon replacing symbols _f1, _f2, ..._fm in terms of the actual polynomials f1, f2, ..., fm. Check the procedure SP:-SyzygyIdeal.

- Care needs to be excercised because variables _f1, _f2, ..., _fm returned by the procedure are defined as local to the procedure. This is so that Maple would not make automatic substitutions for these polynomials should they be defined in the worksheet. In order to be able to replace them with the polynomials f1, f2,..., fm, e.g., to verify that indeed f can be expressed in terms of the polynomials f1, f2, ..., fm, their local attribute needs to be changed to global. See example below.
- In practice, the polynomials f1, f2,..., fm, will often be some elementary invariants of a finite group G, for example, like the elementary symmetric functions are invariant under the group S[n], and they will generate a ring of invariants k[x[1], x[2],..., x[n]]^G. If f does belong to k[x[1], x[2],..., x[n]]^G then f will be expressed in terms of the invariants f1, f2,..., fm assumed to generate k[x[1], x[2],..., x[n]]^G.
- To find out which groups are already remembered, check SP:-FiniteGroups.
- References:
  [1] D. Cox, J. Little, D. O'Shea: "Ideals, varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra", Third Edition, Springer, New York, 2007
  [2] B. Sturmfels: "Algorithms in Invariant Theory", Texts and Monographs in Symbolic Computation, Springer-Verlag/Wien, 1993

Examples:

```
> restart:with(SP);
Remember table of SymmetricGroup has been read and assigned
Remember table of AlternatingGroup has been read and assigned
Remember table of Reynolds has been read and assigned
Remember table of FiniteGroups has been read and assigned
Remember table of generateGinvariants has been read and assigned
```

[*AlternatingGroup*, *Dpolynom*, *FiniteGroups*, *Hilbert_series*, *MatrixAction*, *ModuleLoad*, *ModuleUnload*, *Molien_series*, *Reynolds*, *SPversion*, *Schur_polynom*, Σ, *SymmetricGroup*, *SyzygyIdeal*, *a_polynom*, *create_partitions*, *generateGinvariants*, *gpolynom*, *hpolynom*, *isContained*, *isGinvariant*, *isSymmetric*, *ispartition*, *load_remember_table*, *maxmindegree*, *permsign*, *powersum*, *reduceGinvariants*, *sigma_to_powersum*]

```
>
```

Example 1: Express a polynomial f invariant under the group C4 in terms of three principal (generating) invariants of that group.

```
> f:=x^8+2*x^6*y^2-x^5*y^3+2*x^4*y^4+x^3*y^5+2*x^2*y^6+y^8;
```

$$f := x^8 + 2\,x^6\,y^2 - x^5\,y^3 + 2\,x^4\,y^4 + x^3\,y^5 + 2\,x^2\,y^6 + y^8$$

```
> f1:=x^2+y^2;
  f2:=x^3*y-x*y^3;
  f3:=x^2*y^2;
  C4:=FiniteGroups('C4');
```

$$f1 := x^2 + y^2$$
$$f2 := x^3 y - x y^3$$
$$f3 := x^2 y^2$$
$$C4 := \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\right]$$

Observe that polynomials f1, f2, f3 are invariant under the group C4 defined above. We can check it by applying the Reynolds operator SP:-Reynolds to each polynomial or by using the procedure SP:-isGinvariant:

```
> 'f1'=Reynolds[C4](f1,[x,y]);
  'f2'=Reynolds[C4](f2,[x,y]);
  'f3'=Reynolds[C4](f3,[x,y]);
```

$$f1 = x^2 + y^2$$
$$f2 = x^3 y - x y^3$$
$$f3 = x^2 y^2$$

```
> isGinvariant[C4](f1,[x,y]);
  isGinvariant[C4](f2,[x,y]);
  isGinvariant[C4](f3,[x,y]);
```

$$true$$
$$true$$
$$true$$

Now, the polynomial f is in fact invariant also under the group C4:

```
> isGinvariant[C4](f,[x,y]);
```

$$true$$

We can verify this fact also by using this procedure isContained which can also return an expression for f in terms of the generating invariants f1,f2,f3:

```
> isContained(f,[f1,f2,f3]);
Path set to C:\Maple11\bin.win/libfgbuni.so
    FGb/Maple interface package Version 1.34
    JC Faugere (jcf@calfor.lip6.fr)
    Type ?FGb for documentation
```

$$true$$

Since f does belong to the ring k[f1, f2, f3], we can find now the polynomial g such that f = g(f1,f2,f3):

```
> g:=isContained(f,[f1,f2,f3],'r');
```

$$g := -8\,\_f_3^{\,2} - \_f_3\,\_f_2 - 2\,\_f_2^{\,2} + \_f_1^{\,4}$$

Note that the polynomials (variables) _f1, _f2, and _f3 are local to the procedure isContained. This means that it is impossible to substitute f1, f2, f3 for _f1, _f2, and _f3:

```
> expand(subs({_f[1]=f1,_f[2]=f2,_f[3]=f3},g));
```

$$-8\,\_f_3^{\,2} - \_f_3\,\_f_2 - 2\,\_f_2^{\,2} + \_f_1^{\,4}$$

Thus, in order to be able to substitute for f1, f2, f3 for _f1, _f2, _f3 we need to convert the entire polynomial g to a global polynomial:

```
> g:=convert(g,`global`);
```

$$g := -8\,\_f_3^{\,2} - \_f_3\,\_f_2 - 2\,\_f_2^{\,2} + \_f_1^{\,4}$$

and now we can subsitute these polynomials into the polynomial g:

```
> expand(subs({_f[1]=f1,_f[2]=f2,_f[3]=f3},g));
```

$$x^8 + 2\,x^6\,y^2 - x^5\,y^3 + 2\,x^4\,y^4 + x^3\,y^5 + 2\,x^2\,y^6 + y^8$$

which is the same as the original polynomial f:

```
> 'f' = f;
```

$$f = x^8 + 2\,x^6\,y^2 - x^5\,y^3 + 2\,x^4\,y^4 + x^3\,y^5 + 2\,x^2\,y^6 + y^8$$

```
>
```

Example 2: The polynomial f from example 1 is not invariant under the symmetric group S[2]. Therefore, it cannot be expressed in terms of the two generating invariants of S[2]:

```
> f:=x^8+2*x^6*y^2-x^5*y^3+2*x^4*y^4+x^3*y^5+2*x^2*y^6+y^8;
```

$$f := x^8 + 2\,x^6\,y^2 - x^5\,y^3 + 2\,x^4\,y^4 + x^3\,y^5 + 2\,x^2\,y^6 + y^8$$

```
> S2:=SymmetricGroup(2);
```

$$S2 := \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\right]$$

```
> isGinvariant[S2](f,[x,y]);
```

$$false$$

The two generating invariants for *k*[x,y]^S[2] could be selected to be the two elementary symmetric polynomials sigma[1] and sigma[2]:

```
> sigma[1]:=Sigma[1](x,y);
  sigma[2]:=Sigma[2](x,y);
```

$$\sigma_1 := y + x$$

$$\sigma_2 := x\,y$$

```
> isContained(f,[sigma[1],sigma[2]]);
```

$$false$$

However, if we apply the Reynolds operator parametrized by the group S[2] to the polynomial f, we do get a new S[2] invariant which can be expressed in terms of sigma[1] and sigma[2]:

```
> fnew:=Reynolds[S2](f,[x,y]);
```

$$fnew := x^8 + 2\,x^6\,y^2 + 2\,x^4\,y^4 + 2\,x^2\,y^6 + y^8$$

```
> isGinvariant[S2](fnew,[x,y]);
```

$$true$$

```
> isContained(fnew,[sigma[1],sigma[2]]);
```

$$true$$

```
> g:=isContained(fnew,[sigma[1],sigma[2]],'r');
```

$$g := 8\,\_f_2^{\,4} - 24\,\_f_2^{\,3}\,\_f_1^{\,2} + 22\,\_f_2^{\,2}\,\_f_1^{\,4} - 8\,\_f_2\,\_f_1^{\,6} + \_f_1^{\,8}$$

```
> g:=convert(g,`global`);
```

$$g := 8\,\_f_2^{\,4} - 24\,\_f_2^{\,3}\,\_f_1^{\,2} + 22\,\_f_2^{\,2}\,\_f_1^{\,4} - 8\,\_f_2\,\_f_1^{\,6} + \_f_1^{\,8}$$

```
> expand(subs({_f[1]=sigma[1],_f[2]=sigma[2]},g));
```

$$x^8 + 2\,x^6\,y^2 + 2\,x^4\,y^4 + 2\,x^2\,y^6 + y^8$$

which is the same as the polynomial fnew:

```
> 'fnew' = fnew;
```

$$fnew = x^8 + 2\,x^6\,y^2 + 2\,x^4\,y^4 + 2\,x^2\,y^6 + y^8$$

Thus, the polynomial g gives us an expression of f in terms of sigma[1] and sigma[2].

```
>
```

Example 3: This procedure extends the action of 'isSymmetric' which checks whether the given polynomial is symmetric and if so, it expresses it in terms of elementary symmetric polynomials, to any finite group G. Thus, we should be able to use the procedure 'isGinvariant' also:

```
> f:=x*y+x^4+y^4;
```

$$f := x\,y + x^4 + y^4$$

Clearly, this polynomial is symmetric hence it is invariant under the symmetric group S2:

```
> f:=x*y+x^4+y^4;
  isSymmetric(f);
  isSymmetric(f,r);
```

$$f := x\,y + x^4 + y^4$$

$$true$$

$$\sigma_2 + 2\,\sigma_2^{\,2} - 4\,\sigma_2\,\sigma_1^{\,2} + \sigma_1^{\,4}$$

```
> isContained(f,[sigma[1],sigma[2]]);
  isContained(f,[sigma[1],sigma[2]],'r');
```

$$true$$

$$\_f_2 + 2\,\_f_2^{\,2} - 4\,\_f_2\,\_f_1^{\,2} + \_f_1^{\,4}$$

The latter polynomial in _f1, _f2 is the same as the one above in sigma[1] and sigma[2].

```
>
```

Example 4: We show now another example with 'isContained':

1. First, we need to find matrix form of S2:

```
> S2:=SymmetricGroup(2);
```

$$S2 := \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\right]$$

2. Since |S2| = 2, per Noether's Theorem (check Theorem 5 on page 331 in [1]), k[x,y]^S2 = k[R_S2(x^beta), |beta| <= |S2|=2]. Thus, h[x,y]^S2 is generated by finitely many homogeneous

invariants.

```
> n:=2:
  Gord:=n!:
  x:='x':y:='y':
  mons:=[]:
  for i from 0 to Gord do
  for j from 0 to Gord do
  if i+j<=Gord and i+j>0 then mons:=[op(mons),x^i*y^j] end if;
  end do;
  end do;
  mons;
```

$$[y, y^2, x, x\,y, x^2]$$

```
> Rmons:=map(Reynolds[S2],mons,[x,y]);
```

$$Rmons := \left[\frac{y}{2}+\frac{x}{2}, \frac{y^2}{2}+\frac{x^2}{2}, \frac{y}{2}+\frac{x}{2}, x\,y, \frac{y^2}{2}+\frac{x^2}{2}\right]$$

```
> Rmons:=map(x->x/icontent(x),convert(convert(Rmons,set),list));
```

$$Rmons := [x\,y, x^2+y^2, y+x]$$

```
>
```

3. Thus, the above invariants generate k[x,y]^S2 however, there are too many of them, or, they are on all independent. Observe the following:

```
> isContained(Rmons[1],[Rmons[2],Rmons[3]]);
  isContained(Rmons[2],[Rmons[3],Rmons[1]]);
  isContained(Rmons[3],[Rmons[1],Rmons[2]]);
```

$$true$$

$$true$$

$$false$$

The above shows that the polynomial Rmons[3] does not belong to the ring *k* [Rmons[1],Rmons[2]] whereas polynomial Rmons[1] does belong to the ring *k* [Rmons[2],Rmons[3]] and the polynomial Rmons[2] does belong to the ring *k* [Rmons[3],Rmons[1]]. This means that polynomials xy, x^2+y^2 and x+y are algebraically related. Procedure 'reduceGinvariants' eliminates redundant polynomials, that is, polynomials that are algebraically dependent on the remaining polynomials. Check SP:-reduceGinvariants for more information.

```
> Rmons:=reduceGinvariants[S2](Rmons,[x,y]);
```

$$Rmons := [x\,y, y+x]$$

```
> nops(Rmons);
```

$$2$$

```
>
```

Thus, it appears that there are two elementary invariants: f1 = x+y and f2 = x^2+y^2 that generate

k[x,y]^S2. Of course, polynomials f1 = x+y and f3 = x*y als generate the same ring of invariants of S2. This means that any symmetric polynomial in two variables is expressible in terms of only two invariants that generate *k*[x,y]^S2: x+y and x^2+y^2. Notice that these are the power polynomials s1 and s2.

4. Thus, once we have eliminated redundant invariant monomials, we can now express any symmetric polynomial f in two variables in terms of these two invariants:

```
> f:=x*y+x^4+y^4;
  g:=isContained(f,Rmons,r);
```

$$f := x\,y + x^4 + y^4$$

$$g := \_f_1 - 4\,\_f_1\,\_f_2^{\,2} + 2\,\_f_1^{\,2} + \_f_2^{\,4}$$

```
> g:=convert(g,`global`);
```

$$g := \_f_1 - 4\,\_f_1\,\_f_2^{\,2} + 2\,\_f_1^{\,2} + \_f_2^{\,4}$$

```
> f=expand(subs({seq(_f[i]=Rmons[i],i=1..nops(Rmons))},g));
```

$$x\,y + x^4 + y^4 = x\,y + x^4 + y^4$$

Example 5: Another example of using 'isContained' and the group S[3].

1. First we need to find matrix form of S3:

```
> f:=x*y*z+x^4+y^4+z^4;
  n:=3:
  S||n:=SymmetricGroup(n);
  Gord:=n!;
```

$$f := x\,y\,z + x^4 + y^4 + z^4$$

$$S3 := \left[\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix}, \begin{bmatrix}0&1&0\\1&0&0\\0&0&1\end{bmatrix}, \begin{bmatrix}0&0&1\\1&0&0\\0&1&0\end{bmatrix}, \begin{bmatrix}0&1&0\\0&0&1\\1&0&0\end{bmatrix}, \begin{bmatrix}0&0&1\\0&1&0\\1&0&0\end{bmatrix}\right]$$

$$Gord := 6$$

2. Since |S3| = 6, per Noether's Theorem (check Theorem 5 on page 331 in [1]), *k*[x,y,z]^S3 = *k*[R_S3(x^beta), |beta| <= |S3|=6]. Thus, *k*[x,y,z]^S3 is generated by finitely many homogeneous invariants.

```
> x:='x':y:='y':z:='z':
  mons:=[]:
  for i from 0 to Gord do
  for j from 0 to Gord do
  for k from 0 to Gord do
  if i+j+k<=Gord and i+j+k>0 then mons:=[op(mons),x^i*y^j*z^k]
  end if;
  end do;
  end do;
```

```
end do;
mons:=convert(convert(mons,set),list);
```

$$mons := [y^3 z^3, x y, x z, y z, x, y, z, x^2, y^2, x y z^4, x^2 y z, x^3 y, x y^3, x^2 y^2, x^6, x^4, y^3, x z^2, x^3 z, x^3 y^3, x^4 y^2, y^4, x^2 y^4, x^5, x^3, y^5, y^6, z^4, y z^4, y z^5, x y^2, y x^2, x^2 y^3, x^3 y^2, y x^4, x y^5, y x^5, x y^4, x y z, z^2, z^3, z^5, z^6, y^4 z, y z^2, y^2 z, y^2 z^4, y^2 z^3, y^3 z^2, y^5 z, y^3 z, y^4 z^2, y^2 z^2, y z^3, x z^3, x z^4, x z^5, x y z^2, x y z^3, x y^2 z, x y^2 z^2, x y^2 z^3, x y^3 z, x y^3 z^2, x y^4 z, x^2 z, x^2 z^2, x^2 z^3, x^2 z^4, x^2 y z^2, x^2 y z^3, x^2 y^2 z, x^2 y^2 z^2, x^2 y^3 z, x^3 z^2, x^3 z^3, x^3 y z, x^3 y z^2, x^3 y^2 z, x^4 z, x^4 z^2, x^4 y z, x^5 z]$$

```
> Rmons:=map(Reynolds[S3],mons,[x,y,z]):
> Rmons:=convert(convert(Rmons,set),list):
  Rmons:=map(x->x/icontent(x),Rmons);
```

$$Rmons := [x y + x z + y z, x + y + z, x^5 + y^5 + z^5, x^6 + y^6 + z^6, x^4 y^2 + x^4 z^2 + x^2 y^4 + x^2 z^4 + y^4 z^2 + y^2 z^4, x^2 + y^2 + z^2, x y z^4 + x y^4 z + x^4 y z, x z^2 + x y^2 + y z^2 + y^2 z + y x^2 + x^2 z, y z^5 + y^5 z + x z^5 + x y^5 + x^5 z + y x^5, y^3 + z^3 + x^3, y z^4 + y^4 z + x z^4 + x y^4 + x^4 z + y x^4, x^2 y z + x y^2 z + x y z^2, x^2 y^3 + x^2 z^3 + x^3 y^2 + x^3 z^2 + y^2 z^3 + y^3 z^2, x y^2 z^3 + x y^3 z^2 + x^2 y z^3 + x^2 y^3 z + x^3 y z^2 + x^3 y^2 z, x y^2 z^2 + x^2 y z^2 + x^2 y^2 z, x y z, x^2 y^2 z^2, x^4 + y^4 + z^4, y^3 z^3 + x^3 z^3 + x^3 y^3, x^3 y + x^3 z + x y^3 + x z^3 + y^3 z + y z^3, x^2 y^2 + x^2 z^2 + y^2 z^2, x y z^3 + x y^3 z + x^3 y z]$$

```
> nops(Rmons);
```

$$22$$

3. Thus, the above invariants generate *k*[x,y,z]^S3 however, there are too many of them, or, they are not all algebraically independent.

```
> k:=1:
  h:=Rmons[k]:
  Rmons2:=subsop(k=NULL, Rmons);
  isContained(h,Rmons2);
```

$$Rmons2 := [x + y + z, x^5 + y^5 + z^5, x^6 + y^6 + z^6, x^4 y^2 + x^4 z^2 + x^2 y^4 + x^2 z^4 + y^4 z^2 + y^2 z^4, x^2 + y^2 + z^2, x y z^4 + x y^4 z + x^4 y z, x z^2 + x y^2 + y z^2 + y^2 z + y x^2 + x^2 z, y z^5 + y^5 z + x z^5 + x y^5 + x^5 z + y x^5, y^3 + z^3 + x^3, y z^4 + y^4 z + x z^4 + x y^4 + x^4 z + y x^4, x^2 y z + x y^2 z + x y z^2, x^2 y^3 + x^2 z^3 + x^3 y^2 + x^3 z^2 + y^2 z^3 + y^3 z^2, x y^2 z^3 + x y^3 z^2 + x^2 y z^3 + x^2 y^3 z + x^3 y z^2 + x^3 y^2 z, x y^2 z^2 + x^2 y z^2 + x^2 y^2 z, x y z, x^2 y^2 z^2, x^4 + y^4 + z^4, y^3 z^3 + x^3 z^3 + x^3 y^3, x^3 y + x^3 z + x y^3 + x z^3 + y^3 z + y z^3, x^2 y^2 + x^2 z^2 + y^2 z^2, x y z^3 + x y^3 z + x^3 y z]$$

$$true$$

```
> k:=3:
  h:=Rmons[k]:
  Rmons2:=subsop(k=NULL, Rmons);
```

```
isContained(h,Rmons2);
```

$$Rmons2 := [x\,y + x\,z + y\,z, x + y + z, x^6 + y^6 + z^6, x^4\,y^2 + x^4\,z^2 + x^2\,y^4 + x^2\,z^4 + y^4\,z^2 + y^2\,z^4, x^2 + y^2 + z^2, x\,y\,z^4 + x\,y^4\,z + x^4\,y\,z, x\,z^2 + x\,y^2 + y\,z^2 + y^2\,z + y\,x^2 + x^2\,z, y\,z^5 + y^5\,z + x\,z^5 + x\,y^5 + x^5\,z + y\,x^5, y^3 + z^3 + x^3, y\,z^4 + y^4\,z + x\,z^4 + x\,y^4 + x^4\,z + y\,x^4, x^2\,y\,z + x\,y^2\,z + x\,y\,z^2, x^2\,y^3 + x^2\,z^3 + x^3\,y^2 + x^3\,z^2 + y^2\,z^3 + y^3\,z^2, x\,y^2\,z^3 + x\,y^3\,z^2 + x^2\,y\,z^3 + x^2\,y^3\,z + x^3\,y\,z^2 + x^3\,y^2\,z, x\,y^2\,z^2 + x^2\,y\,z^2 + x^2\,y^2\,z, x\,y\,z, x^2\,y^2\,z^2, x^4 + y^4 + z^4, y^3\,z^3 + x^3\,z^3 + x^3\,y^3, x^3\,y + x^3\,z + x\,y^3 + x\,z^3 + y^3\,z + y\,z^3, x^2\,y^2 + x^2\,z^2 + y^2\,z^2, x\,y\,z^3 + x\,y^3\,z + x^3\,y\,z]$$

$$false$$

```
> k:=6:
  h:=Rmons[k]:
  Rmons2:=subsop(k=NULL, Rmons);
  isContained(h,Rmons2);
```

$$Rmons2 := [x\,y + x\,z + y\,z, x + y + z, x^5 + y^5 + z^5, x^6 + y^6 + z^6, x^4\,y^2 + x^4\,z^2 + x^2\,y^4 + x^2\,z^4 + y^4\,z^2 + y^2\,z^4, x\,y\,z^4 + x\,y^4\,z + x^4\,y\,z, x\,z^2 + x\,y^2 + y\,z^2 + y^2\,z + y\,x^2 + x^2\,z, y\,z^5 + y^5\,z + x\,z^5 + x\,y^5 + x^5\,z + y\,x^5, y^3 + z^3 + x^3, y\,z^4 + y^4\,z + x\,z^4 + x\,y^4 + x^4\,z + y\,x^4, x^2\,y\,z + x\,y^2\,z + x\,y\,z^2, x^2\,y^3 + x^2\,z^3 + x^3\,y^2 + x^3\,z^2 + y^2\,z^3 + y^3\,z^2, x\,y^2\,z^3 + x\,y^3\,z^2 + x^2\,y\,z^3 + x^2\,y^3\,z + x^3\,y\,z^2 + x^3\,y^2\,z, x\,y^2\,z^2 + x^2\,y\,z^2 + x^2\,y^2\,z, x\,y\,z, x^2\,y^2\,z^2, x^4 + y^4 + z^4, y^3\,z^3 + x^3\,z^3 + x^3\,y^3, x^3\,y + x^3\,z + x\,y^3 + x\,z^3 + y^3\,z + y\,z^3, x^2\,y^2 + x^2\,z^2 + y^2\,z^2, x\,y\,z^3 + x\,y^3\,z + x^3\,y\,z]$$

$$true$$

For example, the above shows that polynomials 1, 3, and 6 in the list Rmons are algebraically dependent on the remaining polynomials in Rmons. We can remove all dependent polynomials with the procedure 'reduceGinvariants':

```
> Rmons:=reduceGinvariants[S3](Rmons,[x,y,z]);
```

$$Rmons := [x\,y\,z, x\,y + x\,z + y\,z, x + y + z]$$

```
> nops(Rmons);
```

$$3$$

Thus, it appears that there are three elementary invariants of the symmetric group S[3]. This means that any symmetric polynomial in three variables is expressible in terms of only three invariants that generate k[x,y,z]^S3.

4. Thus, once we have eliminated redundant invariant monomials, we can now express any symmetric polynomial in three variables in terms of these two invariants:

```
> g:=isContained(f,Rmons,r);
```

$$g := \_f_1 + 4\,\_f_1\,\_f_3 + 2\,\_f_2^{\,2} - 4\,\_f_2\,\_f_3^{\,2} + \_f_3^{\,4}$$

```
> g:=convert(g,`global`);
```

$$g := \_f_1 + 4\,\_f_1\,\_f_3 + 2\,\_f_2^{\,2} - 4\,\_f_2\,\_f_3^{\,2} + \_f_3^{\,4}$$

```
> f=expand(subs({seq(_f[i]=Rmons[i],i=1..nops(Rmons))},g));
```

$$x\,y\,z + x^4 + y^4 + z^4 = x\,y\,z + x^4 + y^4 + z^4$$

```
>
```

Algorithm used:

- Proposition 7, page 334, in [1]:

  Step1: Suppose that f, f1, f2,..., fm all belong to a ring *k*[x[1], x[2], x[3],..., x[n]]. Fix a monomial order T in *k*[x[1], x[2], x[3],..., x[n], y[1], y[2], y[3],..., y[m]] such that every monomial involving one of x[1], x[2],... x[n] is greater than all monomials in *k*[y[1], y[2], y[3],..., y[m]]. For example, T could be the lex order plex(x[1], x[2], x[3],..., x[n], y[1], y[2], y[3],..., y[m]).

  Step 2: Compute a Groebner basis G for the ideal <f1 - y1, f2 - y2, ..., fm - ym> for the order T.

  Step 3: Reduce polynomial f modulo the basis G, that is, compute the normal form of f with respect to G: Let g be the remainder of f on division by G.

  Step 4: Polynomial f belongs to *k*[f1, f2,..., fm] if and only if g belongs to *k*[y[1],y[2],..., y[m]]. That is, polynomial g is expressed solely in terms of the auxiliary variables y[1], y[2], ..., y[m].

  Step 5: If f belongs to *k*[f1, f2,..., fm] then f = g(y[1], y[2], ..., y[m]).

- Note that this procedure does not remove algebraic dependencies in the returned list. To remove dependencies use procedure SP:-reduceGinvariants.

See Also: SP:-FiniteGroups, SP:-Reynolds, SP:-SymmetricGroup, SP:-reduceGinvariants, SP:-SyzygyIdeal

Last modified: June 19, 2008

Function: SP:-isGinvariant - checks if a polynomial is invariant under a finite group

Calling Sequence:

isGinvariant[G](f,L);

Parameters:

- G is a list of n x n matrices that represent a finite group G
- f is the polynomial in *k*[x[1], x[2], x[3],..., x[n]]
- L is a list of all ring indeterminates, that is, L = [x[1], x[2], x[3],..., x[n]]

Output:

- True or false depending whether f is invariant under the group G or not.

Description:

- Procedure 'isGinvariant' checks if the given polynomial f is invariant under the given group G in a polynomial ring *k*[x[1], x[2], x[3],..., x[n]]. It returns true/false depending whether f is invariant or not.
- This procedure is invoked as in isGinvariant[G](f,[x[1], x[2], x[3],..., x[n]]) where G is a list of matrices, f is a polynomial in the ring, and the last argument is a set of indeterminates of the ring.
- The action of each element A in G on f is computed using the procedure SP:-MatrixAction. If f(A **x**) = f(**x**) for every A in G then f is invariant under the group G.
- Polynomial f can actually contain fewer indeterminates than in the list [x[1], x[2], x[3],..., x[n]].
- To find out which groups are already remembered, check SP:-FiniteGroups.
- Iinvariants of a finite group G can be generated with the procedure SP:-generateGinvariants. These invariants are usually algebraically dependent in the ring *k*[x[1], x[2], x[3],..., x[n]]. To reduce them to truly algebraically independent, use the procedure SP:-reduceGinvariants.
- References:
  [1] D. Cox, J. Little, D. O'Shea: "Ideals, varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra", Third Edition, Springer, New York, 2007
  [2] B. Sturmfels: "Algorithms in Invariant Theory", Texts and Monographs in Symbolic Computation, Springer-Verlag/Wien, 1993

Examples:

```
> restart:with(SP);
Remember table of SymmetricGroup has been read and assigned
```

```
Remember table of AlternatingGroup has been read and assigned
Remember table of Reynolds has been read and assigned
Remember table of FiniteGroups has been read and assigned
Remember table of generateGinvariants has been read and assigned
```

[*AlternatingGroup*, *Dpolynom*, *FiniteGroups*, *Hilbert_series*, *MatrixAction*, *ModuleLoad*, *ModuleUnload*, *Molien_series*, *Reynolds*, *SPversion*, *Schur_polynom*, Σ, *SymmetricGroup*, *SyzygyIdeal*, *a_polynom*, *create_partitions*, *generateGinvariants*, *gpolynom*, *hpolynom*, *isContained*, *isGinvariant*, *isSymmetric*, *ispartition*, *load_remember_table*, *maxmindegree*, *permsign*, *powersum*, *reduceGinvariants*, *sigma_to_powersum*]

Example 1: Let's check whether some polynomials are G-invariant or not:

```
> C4:=FiniteGroups('C4');
  S2:=SymmetricGroup(2);
  S3:=SymmetricGroup(3);
  A3:=AlternatingGroup(3);
```

$$C4 := \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\right]$$

$$S2 := \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\right]$$

$$S3 := \left[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}\right]$$

$$A3 := \left[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}\right]$$

```
> f1:=x^6+x*y+y^6;
  f2:=x^3*y-x*y^3;
```

$$f1 := x^6 + x\,y + y^6$$

$$f2 := x^3\,y - x\,y^3$$

```
> isGinvariant[S2](f1,[x,y]);  #f1 is S2-invariant
  isGinvariant[C4](f1,[x,y]);  #f1 is not C4-invariant
  isGinvariant[S2](f2,[x,y]);  #f2 is not S2-invariant
  isGinvariant[C4](f2,[x,y]);  #f2 is C4-invariant
```

true

false

false

true

```
> f3:=Sigma[2](x,y,z);
```

$$f3 := x\,y + x\,z + y\,z$$

```
> isGinvariant[S3](f3,[x,y,z]);
  isGinvariant[A3](f3,[x,y,z]);
```

$$true$$

$$true$$

Of course, since f3 is invariant under the group S[3], it is invariant under any subgroup of S[3]. In particular, it is invariant under the alternating group A[3].

```
>
```

Example 2: The Reynolds operator creates G-invariant polynomials. See SP:-Reynolds.

```
> f:=Sigma[2](x,y,z);
```

$$f := x\,y + x\,z + y\,z$$

```
> S3:=SymmetricGroup(3);
```

$$S3 := \left[\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix}, \begin{bmatrix}0&1&0\\1&0&0\\0&0&1\end{bmatrix}, \begin{bmatrix}0&0&1\\1&0&0\\0&1&0\end{bmatrix}, \begin{bmatrix}0&1&0\\0&0&1\\1&0&0\end{bmatrix}, \begin{bmatrix}0&0&1\\0&1&0\\1&0&0\end{bmatrix}\right]$$

```
> g:=Reynolds[S3](f,[x,y,z]);
```

$$g := x\,y + x\,z + y\,z$$

```
> isGinvariant[S3](f,[x,y,z]);
  isGinvariant[S3](g,[x,y,z]);
```

$$true$$

$$true$$

```
>
```

Example 3: Here is another invariant of C2 obtained by applying the Reynolds operator to a polynomial f:

```
> f:=x*y-z^6+x;
```

$$f := x\,y - z^6 + x$$

```
> Gamma3:=FiniteGroups('Gamma3');
```

$$\Gamma 3 := \left[\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}0&1&0\\-1&0&0\\0&0&-1\end{bmatrix}, \begin{bmatrix}-1&0&0\\0&-1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}0&-1&0\\1&0&0\\0&0&-1\end{bmatrix}\right]$$

```
> g:=Reynolds[Gamma3](f,[x,y,z]);
```

$$g := -z^6$$

```
> isGinvariant[Gamma3](f,[x,y,z]);
  isGinvariant[Gamma3](g,[x,y,z]);
```

$$false$$

$$true$$

The above shows that the polynomial f is not invariant under the group Gamma3 whereas the polynomial g obtained by acting with the Reynolds operator parametrized with the group Gamma3 on f is Gamma3-invariant.

```
>
```

Algorithm used:

None

See Also: SP:-Sigma, SP:-FiniteGroups, SP:-AlternatingGroup, SP:-SymmetricGroup, SP:-generateGinvariants, SP:-reduceGinvariants , SP:-MatrixAction

Last modified: June 19, 2008

Function: SP:-isSymmetric- determine if the given polynomial is symmetric

Calling Sequence:

isSymmetric(f);
isSymmetric(f,r);

Parameters:

- f - a multivariate polynomial of type 'polynom'
- r (optional) - of type 'anything'

Output:

- Boolean 'yes' or 'no', depending whether f is symmetric or not. If used with two arguments and the polynomial is found to be symmetric, then a polynomial g in elementary symmetric functions equal to f is returned also.

Description:

- When used only with one argument f, this procedure returns 'yes' or 'no' depending whether f is a symmetric polynomial or f is not a symmetric polynomial, respectively
- When used with two arguments, it returns 'no' if f is not a symmetric polynomial, and it returns 'yes', if f is a symmetric polynomial, and a polynomial f expressed as a polynomial g in terms of elementary symmetric polynomials sigma[k], 1<= k <= multidegree(f).
- Notice that when the polynomial g is returned (see the description of the algorithm below), the elementary symmetric polynomials sigma[k] appearing in g are local variables to the procedure isSymmetric. This so to prevent an automatic substitution of the actual expressions for these polynomials in terms of the original indeterminates should they be defined in the worksheet. One can change their 'local' attribute to the 'global' attribute as shown below.
- In order to compute f as a polynomial g in elementary symmetric polynomials sigma[k] (assuming that f is symmetric), this procedure uses algorithm from Proposition 4 in Section 7.1 from [1]. The algorithm is described below.
- An extension of this procedure to an arbitrray finite group G is accomplished by the procedure SP:-isContained.
- References:
  [1] D. Cox, J. Little, D. O'Shea: "Ideals, varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra", Third Edition, Springer, New York, 2007

Examples:

```
> restart:with(SP);
```

```
Remember table of SymmetricGroup has been read and assigned
Remember table of AlternatingGroup has been read and assigned
Remember table of Reynolds has been read and assigned
Remember table of FiniteGroups has been read and assigned
Remember table of generateGinvariants has been read and assigned
```

[*AlternatingGroup*, *Dpolynom*, *FiniteGroups*, *Hilbert_series*, *MatrixAction*, *ModuleLoad*, *ModuleUnload*, *Molien_series*, *Reynolds*, *SPversion*, *Schur_polynom*, Σ, *SymmetricGroup*, *SyzygyIdeal*, *a_polynom*, *create_partitions*, *generateGinvariants*, *gpolynom*, *hpolynom*, *isContained*, *isGinvariant*, *isSymmetric*, *ispartition*, *load_remember_table*, *maxmindegree*, *permsign*, *powersum*, *reduceGinvariants*, *sigma_to_powersum*]

```
>
```

Example 1. Let's define a few polynomials and check whether they are symmetric or not. If they are symmetric, express each polynomial in terms of elementary symmetric polynomials:

```
> f:=Sigma[3](x[1],x[2],x[3],x[4]);
  isSymmetric(f);
  g:=isSymmetric(f,r);
```

$$f := x_1 x_2 x_3 + x_1 x_2 x_4 + x_1 x_3 x_4 + x_2 x_3 x_4$$

```
Path set to C:\Maple11\bin.win/libfgbuni.so
    FGb/Maple interface package Version 1.34
    JC Faugere (jcf@calfor.lip6.fr)
    Type ?FGb for documentation
```

$$true$$

$$g := \sigma_3$$

```
> f:=Sigma[3](x[1],x[2],x[3],x[4],x[5],x[6]);
  isSymmetric(f);
  g:=isSymmetric(f,r);
```

$$f := x_1 x_2 x_3 + x_1 x_2 x_4 + x_1 x_2 x_5 + x_1 x_2 x_6 + x_1 x_3 x_4 + x_1 x_3 x_5 + x_1 x_3 x_6 + x_1 x_4 x_5 + x_1 x_4 x_6 + x_1 x_5 x_6 + x_2 x_3 x_4 + x_2 x_3 x_5 + x_2 x_3 x_6 + x_2 x_4 x_5 + x_2 x_4 x_6 + x_2 x_5 x_6 + x_3 x_4 x_5 + x_3 x_4 x_6 + x_3 x_5 x_6 + x_4 x_5 x_6$$

$$true$$

$$g := \sigma_3$$

```
> f:=Sigma[2](x[1],x[2],x[3],x[4])+x[1];
  isSymmetric(f);
  g:=isSymmetric(f,r);
```

$$f := x_1 x_2 + x_1 x_3 + x_1 x_4 + x_2 x_3 + x_2 x_4 + x_3 x_4 + x_1$$

$$false$$

$$g := false$$

```
> f:=Sigma[4](x[1],x[2],x[3],x[4],x[5]);
```

```
isSymmetric(f);
g:=isSymmetric(f,r);
```

$$f := x_1\,x_2\,x_3\,x_4 + x_1\,x_2\,x_3\,x_5 + x_1\,x_2\,x_4\,x_5 + x_1\,x_3\,x_4\,x_5 + x_2\,x_3\,x_4\,x_5$$

$$true$$

$$g := \sigma_4$$

```
>
```

Note that sigma[4] returned in the above expression is a local variable after it has been returned by the procedure isSymmetric because it was declared internal to isSymmetric. Thus, even if we define sigma[4] as a global variable, we cannot expand g:

```
> sigma[4]:=Sigma[4](x[1],x[2],x[3],x[4],x[5]); #this is a global
  sigma[4]
```

$$\sigma_4 := x_1\,x_2\,x_3\,x_4 + x_1\,x_2\,x_3\,x_5 + x_1\,x_2\,x_4\,x_5 + x_1\,x_3\,x_4\,x_5 + x_2\,x_3\,x_4\,x_5$$

```
> expand(g); #this is a local sigma[4]
```

$$\sigma_4$$

In particular, this means that g <> sigma[4], the difference g <> sigma[4] won't equal zero!

```
> g - sigma[4];
  eval(%);
```

$$\sigma_4 - x_1\,x_2\,x_3\,x_4 - x_1\,x_2\,x_3\,x_5 - x_1\,x_2\,x_4\,x_5 - x_1\,x_3\,x_4\,x_5 - x_2\,x_3\,x_4\,x_5$$

$$\sigma_4 - x_1\,x_2\,x_3\,x_4 - x_1\,x_2\,x_3\,x_5 - x_1\,x_2\,x_4\,x_5 - x_1\,x_3\,x_4\,x_5 - x_2\,x_3\,x_4\,x_5$$

This is because sigma[4] in the assignment g:=isSymmetric(f,r) above has attribute local whereas sigma[4] in the assignment sigma[4] := Sigma[4](x[1],x[2],x[3],x[4],x[5]) has an attribute global.

However, we can convert an expression that has attribute local to one that has attribute global (and vice versa):

```
> g;
  eval(g);
```

$$\sigma_4$$

$$\sigma_4$$

```
> g:=convert(g,`global`);#this changes attribute of sigma[4] from
  local to global
```

$$g := \sigma_4$$

g can now be evaluated or expanded:

```
> eval(g);
```

$$x_1\,x_2\,x_3\,x_4 + x_1\,x_2\,x_3\,x_5 + x_1\,x_2\,x_4\,x_5 + x_1\,x_3\,x_4\,x_5 + x_2\,x_3\,x_4\,x_5$$

```
> expand(g);
```

$$x_1\,x_2\,x_3\,x_4 + x_1\,x_2\,x_3\,x_5 + x_1\,x_2\,x_4\,x_5 + x_1\,x_3\,x_4\,x_5 + x_2\,x_3\,x_4\,x_5$$

```
> g-sigma[4];
```

$$0$$

```
>
```

Example 2: Suppose we want to express the following symmetric polynomial in terms of the elementary symmetric polynomials and then verify that the conversion was correct:

```
> f:=x[1]^3*x[2]+x[1]^3*x[3]+x[1]*x[2]^3+x[1]*x[3]^3+x[2]^3*x[3]+
  x[2]*x[3]^3;
```

$$f := x_1^3 x_2 + x_1^3 x_3 + x_1 x_2^3 + x_1 x_3^3 + x_2^3 x_3 + x_2 x_3^3$$

```
> isSymmetric(f);
  g:=isSymmetric(f,r);
  eval(g);expand(g);
```

$$true$$

$$g := -\sigma_3 \sigma_1 - 2 \sigma_2^2 + \sigma_1^2 \sigma_2$$

$$-\sigma_3 \sigma_1 - 2 \sigma_2^2 + \sigma_1^2 \sigma_2$$

$$-\sigma_3 \sigma_1 - 2 \sigma_2^2 + \sigma_1^2 \sigma_2$$

In the above expression, sigma[1], sigma[2], and sigma[3] have attribute local because they have been returned by the procedure isSymmetric. This means that even if we define now sigma[1], sigma[2], and sigma[3] as global polynomials:

```
> sigma[1]:=Sigma[1](x[1],x[2],x[3]);
  sigma[2]:=Sigma[2](x[1],x[2],x[3]);
  sigma[3]:=Sigma[3](x[1],x[2],x[3]);
```

$$\sigma_1 := x_1 + x_2 + x_3$$

$$\sigma_2 := x_1 x_2 + x_1 x_3 + x_2 x_3$$

$$\sigma_3 := x_1 x_2 x_3$$

We still won't be able to expand or evaluate polynomial g above:

```
> eval(g);
  expand(g);
```

$$-\sigma_3 \sigma_1 - 2 \sigma_2^2 + \sigma_1^2 \sigma_2$$

$$-\sigma_3 \sigma_1 - 2 \sigma_2^2 + \sigma_1^2 \sigma_2$$

However, let's change now the attribute local to global for sigma[1], sigma[2], and sigma[3] appearing in the definition of g as follows:

```
> g:=convert(g,`global`);
```

$$g := -\sigma_3 \sigma_1 - 2 \sigma_2^2 + \sigma_1^2 \sigma_2$$

It still shows g as the same polynomial in sigma[1], sigma[2] and sigma[3] but now we can expand it:

```
> eval(g);expand(g);
```

$$-x_1 x_2 x_3 (x_1+x_2+x_3) - 2 (x_1 x_2 + x_1 x_3 + x_2 x_3)^2 + (x_1+x_2+x_3)^2 (x_1 x_2 + x_1 x_3 + x_2 x_3)$$

$$x_1^3 x_2 + x_1^3 x_3 + x_1 x_2^3 + x_1 x_3^3 + x_2^3 x_3 + x_2 x_3^3$$

and now we can check that we get back our original polynomial f:

```
> expand(f-g);
```

$$0$$

```
>
```

Example 3: Here are a few examples of more complicated symmetric polynomials:

```
> g:=Sigma[2](x[1],x[2],x[3])^2*Sigma[1](x[1],x[2],x[3]);
```

$$g := (x_1 x_2 + x_1 x_3 + x_2 x_3)^2 (x_1 + x_2 + x_3)$$

```
> isSymmetric(g);
  isSymmetric(g,r);
```

$$true$$

$$\sigma_2^2 \sigma_1$$

```
> h:=expand((f-g)^2+g^2-g);
```

$$\begin{aligned}
h := & -8 x_2^4 x_3^4 x_1 + 28 x_2^4 x_1^5 x_3 + 2 x_2^4 x_1^6 + 4 x_2^5 x_1^5 + 2 x_2^6 x_1^4 + 2 x_1^6 x_3^4 + 4 x_1^5 x_3^5 - x_2^2 x_1^3 \\
& - x_2^3 x_1^2 - x_1^2 x_3^3 - x_2^3 x_3^2 + 8 x_2^3 x_1^6 x_3 + 64 x_2^3 x_1^5 x_3^2 + 28 x_2^5 x_1^4 x_3 + 106 x_2^4 x_1^4 x_3^2 \\
& + 12 x_2^2 x_1^6 x_3^2 + 64 x_2^2 x_1^5 x_3^3 + 160 x_2^3 x_1^4 x_3^3 + 64 x_2^5 x_1^3 x_3^2 + 160 x_2^4 x_1^3 x_3^3 \\
& + 12 x_2^6 x_1^2 x_3^2 + 64 x_2^5 x_1^2 x_3^3 + 8 x_2^6 x_1^3 x_3 + 8 x_1^3 x_3^6 x_2 + 12 x_1^2 x_3^6 x_2^2 - 6 x_1^6 x_2 x_3^2 \\
& + 106 x_2^2 x_1^4 x_3^4 + 160 x_2^3 x_1^3 x_3^4 - 5 x_1 x_2^2 x_3^2 + 106 x_2^4 x_1^2 x_3^4 - x_2^2 x_3^3 + 8 x_1^6 x_2 x_3^3 \\
& + 28 x_1^5 x_2 x_3^4 - 5 x_1^2 x_2 x_3^2 - x_1^3 x_3^2 + 2 x_1^4 x_3^6 + 2 x_2^6 x_3^4 + 4 x_2^5 x_3^5 + 2 x_2^4 x_3^6 - 2 x_1 x_2^3 x_3 \\
& - 2 x_1 x_3^3 x_2 - 5 x_2^2 x_1^2 x_3 + 28 x_1^4 x_2 x_3^5 + 64 x_1^3 x_2^2 x_3^5 + 64 x_1^2 x_2^3 x_3^5 + 8 x_1 x_3^6 x_2^3 \\
& + 8 x_1 x_2^6 x_3^3 + 28 x_1 x_2^5 x_3^4 + x_1^6 x_2^2 + 2 x_1^4 x_2^4 - 2 x_1^6 x_2^3 - 2 x_1^5 x_2^4 + x_1^6 x_3^2 + 2 x_1^4 x_3^4 \\
& - 2 x_1^6 x_3^3 - 2 x_1^5 x_3^4 + x_1^2 x_2^6 - 2 x_1^3 x_2 x_3 + 28 x_1 x_2^4 x_3^5 - 6 x_1^6 x_2^2 x_3 + 2 x_1^3 x_2^2 x_3^3 \\
& - 16 x_1^4 x_2^3 x_3^2 - 20 x_1^5 x_2^2 x_3^2 + 2 x_1^6 x_2 x_3 - 12 x_1^5 x_2^3 x_3 + 2 x_1^3 x_2^4 x_3 + 2 x_1^4 x_2 x_3^3 \\
& - 8 x_1^4 x_2^4 x_3 - 12 x_1^5 x_2 x_3^3 - 16 x_1^4 x_2^2 x_3^3 - 16 x_1^3 x_2^4 x_3^2 - 12 x_1^3 x_2^3 x_3^3 + 2 x_1^4 x_3 x_2^3 \\
& + 2 x_1^3 x_3^2 x_2^3 + 2 x_1^3 x_3^4 x_2 - 2 x_1^4 x_2^5 - 2 x_1^3 x_2^6 + x_1^2 x_3^6 - 2 x_1^4 x_3^5 - 2 x_1^3 x_3^6 + x_2^6 x_3^2 \\
& + 2 x_2^4 x_3^4 - 2 x_2^6 x_3^3 - 2 x_2^5 x_3^4 + x_2^2 x_3^6 - 2 x_2^4 x_3^5 - 2 x_2^3 x_3^6 + 2 x_1^2 x_2^3 x_3^3 + 2 x_1 x_2^6 x_3 \\
& + 2 x_1 x_2^4 x_3^3 - 12 x_1^3 x_2^5 x_3 - 6 x_1^2 x_2^6 x_3 - 20 x_1^2 x_2^5 x_3^2 - 16 x_1^3 x_3^4 x_2^2 - 8 x_1^4 x_3^4 x_2 \\
& - 16 x_1^2 x_2^4 x_3^3 - 6 x_1 x_2^6 x_3^2 - 12 x_1 x_2^5 x_3^3 + 2 x_1 x_3^4 x_2^3 + 2 x_1 x_3^6 x_2 - 12 x_1^3 x_3^5 x_2
\end{aligned}$$

$$-16\,x_1^{\,2}\,x_3^{\,4}\,x_2^{\,3}-20\,x_1^{\,2}\,x_3^{\,5}\,x_2^{\,2}-6\,x_1^{\,2}\,x_3^{\,6}\,x_2-12\,x_1\,x_3^{\,5}\,x_2^{\,3}-6\,x_1\,x_3^{\,6}\,x_2^{\,2}$$

```
> isSymmetric(h,r);
```

$$2\,\sigma_2^{\,4}\,\sigma_1^{\,2}+4\,\sigma_3\,\sigma_1\,\sigma_2^{\,2}-2\,\sigma_3\,\sigma_2\,\sigma_1^{\,3}+\sigma_3^{\,2}\,\sigma_1^{\,2}-\sigma_2^{\,2}\,\sigma_1+\sigma_1^{\,4}\,\sigma_2^{\,2}+4\,\sigma_2^{\,4}\,\sigma_1-2\,\sigma_2^{\,3}\,\sigma_1^{\,3}-4\,\sigma_2^{\,3}\,\sigma_1^{\,2}+4\,\sigma_2^{\,4}+2\,\sigma_3\,\sigma_2^{\,2}\,\sigma_1^{\,2}$$

```
> convert(%,`global`);
```

$$2\,\sigma_2^{\,4}\,\sigma_1^{\,2}+4\,\sigma_3\,\sigma_1\,\sigma_2^{\,2}-2\,\sigma_3\,\sigma_2\,\sigma_1^{\,3}+\sigma_3^{\,2}\,\sigma_1^{\,2}-\sigma_2^{\,2}\,\sigma_1+\sigma_1^{\,4}\,\sigma_2^{\,2}+4\,\sigma_2^{\,4}\,\sigma_1-2\,\sigma_2^{\,3}\,\sigma_1^{\,3}-4\,\sigma_2^{\,3}\,\sigma_1^{\,2}+4\,\sigma_2^{\,4}+2\,\sigma_3\,\sigma_2^{\,2}\,\sigma_1^{\,2}$$

Now we expand the above polynomial in terms of the variables x[1], x[2], x[3] and verify that this is the original polynomial h:

```
> expand(%);
```

$$\begin{aligned}
&-8\,x_2^{\,4}\,x_3^{\,4}\,x_1+28\,x_2^{\,4}\,x_1^{\,5}\,x_3+2\,x_2^{\,4}\,x_1^{\,6}+4\,x_2^{\,5}\,x_1^{\,5}+2\,x_2^{\,6}\,x_1^{\,4}+2\,x_1^{\,6}\,x_3^{\,4}+4\,x_1^{\,5}\,x_3^{\,5}-x_2^{\,2}\,x_1^{\,3}\\
&-x_2^{\,3}\,x_1^{\,2}-x_1^{\,2}\,x_3^{\,3}-x_2^{\,3}\,x_3^{\,2}+8\,x_2^{\,3}\,x_1^{\,6}\,x_3+64\,x_2^{\,3}\,x_1^{\,5}\,x_3^{\,2}+28\,x_2^{\,5}\,x_1^{\,4}\,x_3+106\,x_2^{\,4}\,x_1^{\,4}\,x_3^{\,2}\\
&+12\,x_2^{\,2}\,x_1^{\,6}\,x_3^{\,2}+64\,x_2^{\,2}\,x_1^{\,5}\,x_3^{\,3}+160\,x_2^{\,3}\,x_1^{\,4}\,x_3^{\,3}+64\,x_2^{\,5}\,x_1^{\,3}\,x_3^{\,2}+160\,x_2^{\,4}\,x_1^{\,3}\,x_3^{\,3}\\
&+12\,x_2^{\,6}\,x_1^{\,2}\,x_3^{\,2}+64\,x_2^{\,5}\,x_1^{\,2}\,x_3^{\,3}+8\,x_2^{\,6}\,x_1^{\,3}\,x_3+8\,x_1^{\,3}\,x_3^{\,6}\,x_2+12\,x_1^{\,2}\,x_3^{\,6}\,x_2^{\,2}-6\,x_1^{\,6}\,x_2\,x_3^{\,2}\\
&+106\,x_2^{\,2}\,x_1^{\,4}\,x_3^{\,4}+160\,x_2^{\,3}\,x_1^{\,3}\,x_3^{\,4}-5\,x_1\,x_2^{\,2}\,x_3^{\,2}+106\,x_2^{\,4}\,x_1^{\,2}\,x_3^{\,4}-x_2^{\,2}\,x_3^{\,3}+8\,x_1^{\,6}\,x_2\,x_3^{\,3}\\
&+28\,x_1^{\,5}\,x_2\,x_3^{\,4}-5\,x_1^{\,2}\,x_2\,x_3^{\,2}-x_1^{\,3}\,x_3^{\,2}+2\,x_1^{\,4}\,x_3^{\,6}+2\,x_2^{\,6}\,x_3^{\,4}+4\,x_2^{\,5}\,x_3^{\,5}+2\,x_2^{\,4}\,x_3^{\,6}-2\,x_1\,x_2^{\,3}\,x_3\\
&-2\,x_1\,x_3^{\,3}\,x_2-5\,x_2^{\,2}\,x_1^{\,2}\,x_3+28\,x_1^{\,4}\,x_2\,x_3^{\,5}+64\,x_1^{\,3}\,x_2^{\,2}\,x_3^{\,5}+64\,x_1^{\,2}\,x_2^{\,3}\,x_3^{\,5}+8\,x_1\,x_3^{\,6}\,x_2^{\,3}\\
&+8\,x_1\,x_2^{\,6}\,x_3^{\,3}+28\,x_1\,x_2^{\,5}\,x_3^{\,4}+x_1^{\,6}\,x_2^{\,2}+2\,x_1^{\,4}\,x_2^{\,4}-2\,x_1^{\,6}\,x_2^{\,3}-2\,x_1^{\,5}\,x_2^{\,4}+x_1^{\,6}\,x_3^{\,2}+2\,x_1^{\,4}\,x_3^{\,4}\\
&-2\,x_1^{\,6}\,x_3^{\,3}-2\,x_1^{\,5}\,x_3^{\,4}+x_1^{\,2}\,x_2^{\,6}-2\,x_1^{\,3}\,x_2\,x_3+28\,x_1\,x_2^{\,4}\,x_3^{\,5}-6\,x_1^{\,6}\,x_2^{\,2}\,x_3+2\,x_1^{\,3}\,x_2^{\,2}\,x_3^{\,3}\\
&-16\,x_1^{\,4}\,x_2^{\,3}\,x_3^{\,2}-20\,x_1^{\,5}\,x_2^{\,2}\,x_3^{\,2}+2\,x_1^{\,6}\,x_2\,x_3-12\,x_1^{\,5}\,x_2^{\,3}\,x_3+2\,x_1^{\,3}\,x_2^{\,4}\,x_3+2\,x_1^{\,4}\,x_2\,x_3^{\,3}\\
&-8\,x_1^{\,4}\,x_2^{\,4}\,x_3-12\,x_1^{\,5}\,x_2\,x_3^{\,3}-16\,x_1^{\,4}\,x_2^{\,2}\,x_3^{\,3}-16\,x_1^{\,3}\,x_2^{\,4}\,x_3^{\,2}-12\,x_1^{\,3}\,x_2^{\,3}\,x_3^{\,3}+2\,x_1^{\,4}\,x_3\,x_2^{\,3}\\
&+2\,x_1^{\,3}\,x_3^{\,2}\,x_2^{\,3}+2\,x_1^{\,3}\,x_3^{\,4}\,x_2-2\,x_1^{\,4}\,x_2^{\,5}-2\,x_1^{\,3}\,x_2^{\,6}+x_1^{\,2}\,x_3^{\,6}-2\,x_1^{\,4}\,x_3^{\,5}-2\,x_1^{\,3}\,x_3^{\,6}+x_2^{\,6}\,x_3^{\,2}\\
&+2\,x_2^{\,4}\,x_3^{\,4}-2\,x_2^{\,6}\,x_3^{\,3}-2\,x_2^{\,5}\,x_3^{\,4}+x_2^{\,2}\,x_3^{\,6}-2\,x_2^{\,4}\,x_3^{\,5}-2\,x_2^{\,3}\,x_3^{\,6}+2\,x_1^{\,2}\,x_2^{\,3}\,x_3^{\,3}+2\,x_1\,x_2^{\,6}\,x_3\\
&+2\,x_1\,x_2^{\,4}\,x_3^{\,3}-12\,x_1^{\,3}\,x_2^{\,5}\,x_3-6\,x_1^{\,2}\,x_2^{\,6}\,x_3-20\,x_1^{\,2}\,x_2^{\,5}\,x_3^{\,2}-16\,x_1^{\,3}\,x_3^{\,4}\,x_2^{\,2}-8\,x_1^{\,4}\,x_3^{\,4}\,x_2\\
&-16\,x_1^{\,2}\,x_2^{\,4}\,x_3^{\,3}-6\,x_1\,x_2^{\,6}\,x_3^{\,2}-12\,x_1\,x_2^{\,5}\,x_3^{\,3}+2\,x_1\,x_3^{\,4}\,x_2^{\,3}+2\,x_1\,x_3^{\,6}\,x_2-12\,x_1^{\,3}\,x_3^{\,5}\,x_2\\
&-16\,x_1^{\,2}\,x_3^{\,4}\,x_2^{\,3}-20\,x_1^{\,2}\,x_3^{\,5}\,x_2^{\,2}-6\,x_1^{\,2}\,x_3^{\,6}\,x_2-12\,x_1\,x_3^{\,5}\,x_2^{\,3}-6\,x_1\,x_3^{\,6}\,x_2^{\,2}
\end{aligned}$$

```
> %-h;
```

$$0$$

```
>
```

Algorithm used:

The algorithm used in the procedure isSymmetric is based on Proposition 4, Section 7.1, from [1].

1. Let f be a polynomial in k[x[1],..., x[n]].
2. Fix a monomial order T in the polynomial ring k[x[1],...,x[n],y[1],...,y[n]] in which any monomial involving one of x[1], x[2],..., x[n] is greater than all monomials in k[y[1],..., y[n]]. For example, the lex order x[1] > x[2] > ... > x[n] > y[1] > ... > y[n].
3. Compute a Groebner basis G of the ideal <sigma[1] - y[1], sigma[2] - y[2],..., sigma[n] - y[n]> where sigma[k] is the k-th elementary symmetric function, 1<= k <= n, for the monomial order T. Note that a basis for that ideal is known to be given in terms of g-polynomials (see Proposition 5, Section 7.1 in [1]) which are defined here as SP:-gpolynom. See also SP:-hpolynom.
3. Compute the normal formal g of the polynomial f modulo G. That is, let g be the remainder of the division of f by the list G.
4. Polynomial f is symmetric if and only if g is a polynomial in k[y[1],..., y[n]].
5. If f is symmetric, then f = g(sigma[1], sigma[2], ..., sigma[n]) is the unique expression of f as a polynomial in the elementary symmetric polynomials sigma[1], sigma[2], ... , sigma[n].
6. The Groebner basis G from point 3 is computed using Maple's Groebner package when 1 <= n <= 3 and then using the FGb package when n > 3 for reasons of efficiency.

See Also: SP:-Sigma, SP:-hpolynom, SP:-gpolynom, SP:-isContained

Last modified: June 19, 2008

Function: SP:-MatrixAction - computes an action of a square matrix on a polynomial

Calling Sequence:

MatrixAction(A,f,L);

Parameters:

- A is an n x n matrix
- f is the polynomial in *k*[x[1], x[2], x[3],..., x[n]]
- L is a list of all ring indeterminates, that is, L = [x[1], x[2], x[3],..., x[n]]

Output:

- A polynomial in *k*[x[1], x[2], x[3],..., x[n]] after it has been acted on with the matrix A

Description:

- Procedure 'MatrixAction' defines an action of a square n x n matrix A on a polynomial f in k[x[1], x[2], x[3],..., x[n]] according to the definition g(x) = f(Ax) where x is vector [x[1], x[2], x[3],..., x[n]].
- Polynomial f can actually contain fewer indeterminates than in the list [x[1], x[2], x[3],..., x[n]].
- This procedure is used as in MatrixAction(A,f,L) where L = [x[1], x[2], x[3],..., x[n]] is the list of all indeterminates of the ring *k*[x[1], x[2], x[3],..., x[n]], f is the polynomial in *k*[x[1], x[2], x[3],..., x[n]], and A is an n x n matrix.
- Note, that the procedure only checks whether A is a square matrix of the right size and not whether it is invertible. This procedure is used later when computing group invariants of SP:-FiniteGroups with the SP:-Reynolds operator.
- References:
  [1] D. Cox, J. Little, D. O'Shea: "Ideals, varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra", Third Edition, Springer, New York, 2007
  [2] B. Sturmfels: "Algorithms in Invariant Theory", Texts and Monographs in Symbolic Computation, Springer-Verlag/Wien, 1993

Examples:

```
> restart:with(SP):
Remember table of SymmetricGroup has been read and assigned
Remember table of AlternatingGroup has been read and assigned
Remember table of Reynolds has been read and assigned
Remember table of FiniteGroups has been read and assigned
Remember table of generateGinvariants has been read and assigned
```

Example 1: Acting with a matrix on a polynomial:

```
> A:=matrix(2,2,[1/sqrt(2),-1/sqrt(2),1/sqrt(2),1/sqrt(2)]);
```

$$A := \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}$$

```
> f:=x^2+x*y+y^2;
```

$$f := x^2 + x\,y + y^2$$

```
> f1:=MatrixAction(A,f,[x,y]);
```

$$f1 := \frac{3\,x^2}{2} + \frac{y^2}{2}$$

The above shows that f is not invariant under the action of the matrix A since f <> f1. However, polynomial f1 no longer has the mixed term x*y that f had. Thus, the action of A is a sort of a rotation or a diagonalization. In fact, A is invertible and its order is eight:

```
> evalm(A^2),evalm(A^3),evalm(A^4),evalm(A^5),evalm(A^6),evalm(A^
  7),evalm(A^8);
```

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Thus, matrix A generates a cyclic group of rotations of order 8.

Here are some subsequent actions of powers of A on f:

```
> f2:=MatrixAction(A^2,f,[x,y]);
```

$$f2 := y^2 - x\,y + x^2$$

```
> f3:=MatrixAction(A^3,f,[x,y]);
```

$$f3 := \frac{x^2}{2} + \frac{3\,y^2}{2}$$

```
> f4:=MatrixAction(A^4,f,[x,y]);
```

$$f4 := x^2 + x\,y + y^2$$

The above shows that the polynomial f is invariant under A^4 since f4 = f.

```
>
```

Example 2: Let's see an action under elements of the permutation group S[2] on f.

```
> S2:=SymmetricGroup(2);
```

$$S2 := \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\right]$$

```
> 'f'=f;
  for A in S2 do MatrixAction(A,f,[x,y]) end do;
```

$$f = x^2 + x\,y + y^2$$

$$x^2 + x\,y + y^2$$

$$x^2 + x\,y + y^2$$

```
>
```

Example 3: Let's see an action under elements of the permutation group S[3] on some polynomials:

```
> S3:=SymmetricGroup(3);
```

$$S3 := \left[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}\right]$$

```
> f:=Sigma[2](x,y,z);
```

$$f := x\,y + x\,z + y\,z$$

```
> for A in S3 do MatrixAction(A,f,[x,y,z]) end do;
```

$$x\,y + x\,z + y\,z$$

$$x\,y + x\,z + y\,z$$

$$x\,y + x\,z + y\,z$$

$$x\,y + x\,z + y\,z$$

$$x\,y + x\,z + y\,z$$

$$x\,y + x\,z + y\,z$$

As expected, since the polynomial f is actually sigma[2](x,y,z), it is invariant of S[3].

```
>
```

Example 4: If a polynomial is not invariant under a finite group G, one can easily generate a new invariant polynomial under G by the method of averaging over the group as follows:

```
> f:=y^2+x*y+z+x+z*x;
```

$$f := y^2 + x\,y + z + x + x\,z$$

```
> for i from 1 to nops(S3) do g||i:=MatrixAction(S3[i],f,[x,y,z])
  end do;
```

$$g1 := y^2 + x\,y + z + x + x\,z$$

$$g2 := z^2 + x\,z + y + x + x\,y$$

$$g3 := x^2 + x\,y + z + y + y\,z$$

$$g4 := x^2 + x\,z + y + z + y\,z$$

$$g5 := z^2 + y\,z + x + y + x\,y$$

$$g6 := y^2 + y\,z + x + z + x\,z$$

Thus, the above shows that f is not invariant under the group S[3]. However, let's define

```
> h:=1/6*add(g||i,i=1..6);
```

$$h := \frac{2}{3}x\,y + \frac{2}{3}x\,z + \frac{2}{3}y\,z + \frac{2}{3}x + \frac{2}{3}y + \frac{2}{3}z + \frac{1}{3}x^2 + \frac{1}{3}y^2 + \frac{1}{3}z^2$$

```
> for A in S3 do MatrixAction(A,h,[x,y,z]) end do;
```

$$\frac{2}{3}x\,y + \frac{2}{3}x\,z + \frac{2}{3}y\,z + \frac{2}{3}x + \frac{2}{3}y + \frac{2}{3}z + \frac{1}{3}x^2 + \frac{1}{3}y^2 + \frac{1}{3}z^2$$

$$\frac{2}{3}x\,y + \frac{2}{3}x\,z + \frac{2}{3}y\,z + \frac{2}{3}x + \frac{2}{3}y + \frac{2}{3}z + \frac{1}{3}x^2 + \frac{1}{3}y^2 + \frac{1}{3}z^2$$

$$\frac{2}{3}x\,y + \frac{2}{3}x\,z + \frac{2}{3}y\,z + \frac{2}{3}x + \frac{2}{3}y + \frac{2}{3}z + \frac{1}{3}x^2 + \frac{1}{3}y^2 + \frac{1}{3}z^2$$

$$\frac{2}{3}x\,y + \frac{2}{3}x\,z + \frac{2}{3}y\,z + \frac{2}{3}x + \frac{2}{3}y + \frac{2}{3}z + \frac{1}{3}x^2 + \frac{1}{3}y^2 + \frac{1}{3}z^2$$

$$\frac{2}{3}x\,y + \frac{2}{3}x\,z + \frac{2}{3}y\,z + \frac{2}{3}x + \frac{2}{3}y + \frac{2}{3}z + \frac{1}{3}x^2 + \frac{1}{3}y^2 + \frac{1}{3}z^2$$

$$\frac{2}{3}x\,y + \frac{2}{3}x\,z + \frac{2}{3}y\,z + \frac{2}{3}x + \frac{2}{3}y + \frac{2}{3}z + \frac{1}{3}x^2 + \frac{1}{3}y^2 + \frac{1}{3}z^2$$

The above shows that in fact polynomial h is invariant under S[3] because action of every element of S[3] leaves h invariant. We have in fact created from f an invariant h of S[3] using the averaging of the action of S[3] on f. See also SP:-Reynolds.

```
>
```

Example 5: Matrix A needs to be a square martix of size equal to the number of indeterminates in the ring, but it can be a symbolic matrix:

```
> f:=x^2-x+4;
```

$$f := x^2 - x + 4$$

```
> A:=matrix(2,2,[a,b,c,d]);
```

$$A := \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

```
> MatrixAction(A,f,[x,y]);
```

$$a^2\,x^2 + 2\,a\,x\,b\,y + b^2\,y^2 - a\,x - b\,y + 4$$

```
>
```

Algorithm used:

None

See Also: SP:-Sigma, SP:-FiniteGroups, SP:-AlternatingGroup, SP:-SymmetricGroup, SP:-Reynolds, Groebner

Last modified: June 19, 2008

Function: SP:-powersum - returns a power sum polynomial

Calling Sequence:

powersum[k](s);

Parameters:

- The index k must be a positive integer such that 1 <= k <= n where n is the number of variables in the sequence s
- s must be a sequence of n variables

Output:

- A polynomial in n variables specified in the sequence s

Description:

- Procedure powersum used as in powersum[k](x[1],...x[n]) returns the s[k] power sum polynomial where s[k] = x[1]^k + x[2]^k + ... + x[n]^k. These polynomials are defined above Theorem 8, Section 7.1 in [1].
- According to Theorem 8 in Section 7.1 of [1], if *k* is a field containing rational numbers *Q*, then every symmetric polynomial in *k*[x[1], x[2], ... , x[n]] can be written as a polynomial in the power sums s[1], s[2], ..., s[n].
- References:
  [1] D. Cox, J. Little, D. O'Shea: "Ideals, varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra", Third Edition, Springer, New York, 2007
  [2] B. Sturmfels: "Algorithms in Invariant Theory", Texts and Monographs in Symbolic Computation, Springer-Verlag/Wien, 1993

Examples:

```
> restart:with(SP);
Remember table of SymmetricGroup has been read and assigned
Remember table of AlternatingGroup has been read and assigned
Remember table of Reynolds has been read and assigned
Remember table of FiniteGroups has been read and assigned
Remember table of generateGinvariants has been read and assigned
```

[*AlternatingGroup*, *Dpolynom*, *FiniteGroups*, *Hilbert_series*, *MatrixAction*, *ModuleLoad*, *ModuleUnload*, *Molien_series*, *Reynolds*, *SPversion*, *Schur_polynom*, Σ, *SymmetricGroup*, *SyzygyIdeal*, *a_polynom*, *create_partitions*, *generateGinvariants*, *gpolynom*, *hpolynom*, *isContained*, *isGinvariant*, *isSymmetric*, *ispartition*, *load_remember_table*, *maxmindegree*,

permsign, *powersum*, *reduceGinvariants*, *sigma_to_powersum*]

```
>
```

Example 1: Here are power sums of degrees 1 through 10 in three variables.

```
> for k from 1 to 10 do
  s[k]:=powersum[k](x[1],x[2],x[3]);
  end do;
```

$$s_1 := x_1 + x_2 + x_3$$
$$s_2 := x_1^2 + x_2^2 + x_3^2$$
$$s_3 := x_1^3 + x_2^3 + x_3^3$$
$$s_4 := x_1^4 + x_2^4 + x_3^4$$
$$s_5 := x_1^5 + x_2^5 + x_3^5$$
$$s_6 := x_1^6 + x_2^6 + x_3^6$$
$$s_7 := x_1^7 + x_2^7 + x_3^7$$
$$s_8 := x_1^8 + x_2^8 + x_3^8$$
$$s_9 := x_1^9 + x_2^9 + x_3^9$$
$$s_{10} := x_1^{10} + x_2^{10} + x_3^{10}$$

```
>
```

Since the power functions are symmetric polynomials, we could use procedure isSymmetric to express them in terms of the elementary symmetric polynomials.

```
> for k from 1 to 10 do
   's'[k] = isSymmetric(s[k],r);
  end do;
```

$$x_1 + x_2 + x_3 = \sigma_1$$
$$x_1^2 + x_2^2 + x_3^2 = -2\,\sigma_2 + \sigma_1^2$$
$$x_1^3 + x_2^3 + x_3^3 = 3\,\sigma_3 - 3\,\sigma_2\,\sigma_1 + \sigma_1^3$$
$$x_1^4 + x_2^4 + x_3^4 = 4\,\sigma_3\,\sigma_1 + 2\,\sigma_2^2 + \sigma_1^4 - 4\,\sigma_2\,\sigma_1^2$$
$$x_1^5 + x_2^5 + x_3^5 = -5\,\sigma_3\,\sigma_2 + 5\,\sigma_3\,\sigma_1^2 - 5\,\sigma_2\,\sigma_1^3 + 5\,\sigma_2^2\,\sigma_1 + \sigma_1^5$$
$$x_1^6 + x_2^6 + x_3^6 = 3\,\sigma_3^2 - 12\,\sigma_3\,\sigma_2\,\sigma_1 + 6\,\sigma_3\,\sigma_1^3 - 2\,\sigma_2^3 + 9\,\sigma_2^2\,\sigma_1^2 + \sigma_1^6 - 6\,\sigma_2\,\sigma_1^4$$
$$x_1^7 + x_2^7 + x_3^7 =$$

$$7\,\sigma_3\,\sigma_2^{\,2} + 7\,\sigma_3^{\,2}\,\sigma_1 - 21\,\sigma_3\,\sigma_2\,\sigma_1^{\,2} + 7\,\sigma_3\,\sigma_1^{\,4} - 7\,\sigma_2\,\sigma_1^{\,5} + \sigma_1^{\,7} - 7\,\sigma_2^{\,3}\,\sigma_1 + 14\,\sigma_2^{\,2}\,\sigma_1^{\,3}$$

$$x_1^{\,8} + x_2^{\,8} + x_3^{\,8} = -8\,\sigma_3^{\,2}\,\sigma_2 + 24\,\sigma_3\,\sigma_2^{\,2}\,\sigma_1 + 12\,\sigma_3^{\,2}\,\sigma_1^{\,2} + \sigma_1^{\,8} - 8\,\sigma_2\,\sigma_1^{\,6} + 8\,\sigma_3\,\sigma_1^{\,5} + 20\,\sigma_2^{\,2}\,\sigma_1^{\,4}$$
$$- 32\,\sigma_3\,\sigma_2\,\sigma_1^{\,3} + 2\,\sigma_2^{\,4} - 16\,\sigma_2^{\,3}\,\sigma_1^{\,2}$$

$$x_1^{\,9} + x_2^{\,9} + x_3^{\,9} = 3\,\sigma_3^{\,3} - 9\,\sigma_3\,\sigma_2^{\,3} - 27\,\sigma_3^{\,2}\,\sigma_2\,\sigma_1 - 45\,\sigma_3\,\sigma_2\,\sigma_1^{\,4} + 54\,\sigma_3\,\sigma_2^{\,2}\,\sigma_1^{\,2} + 27\,\sigma_2^{\,2}\,\sigma_1^{\,5}$$
$$- 9\,\sigma_2\,\sigma_1^{\,7} - 30\,\sigma_2^{\,3}\,\sigma_1^{\,3} + 18\,\sigma_3^{\,2}\,\sigma_1^{\,3} + 9\,\sigma_2^{\,4}\,\sigma_1 + 9\,\sigma_3\,\sigma_1^{\,6} + \sigma_1^{\,9}$$

$$x_1^{\,10} + x_2^{\,10} + x_3^{\,10} = 15\,\sigma_3^{\,2}\,\sigma_2^{\,2} + 10\,\sigma_3^{\,3}\,\sigma_1 + \sigma_1^{\,10} - 60\,\sigma_3\,\sigma_2\,\sigma_1^{\,5} + 100\,\sigma_3\,\sigma_2^{\,2}\,\sigma_1^{\,3}$$
$$- 60\,\sigma_3^{\,2}\,\sigma_2\,\sigma_1^{\,2} - 2\,\sigma_2^{\,5} - 40\,\sigma_3\,\sigma_2^{\,3}\,\sigma_1 + 35\,\sigma_2^{\,2}\,\sigma_1^{\,6} - 50\,\sigma_2^{\,3}\,\sigma_1^{\,4} + 25\,\sigma_3^{\,2}\,\sigma_1^{\,4} + 25\,\sigma_2^{\,4}\,\sigma_1^{\,2}$$
$$- 10\,\sigma_2\,\sigma_1^{\,8} + 10\,\sigma_3\,\sigma_1^{\,7}$$

```
>
> restart:with(SP):
Remember table of SymmetricGroup has been read and assigned
Remember table of AlternatingGroup has been read and assigned
Remember table of Reynolds has been read and assigned
Remember table of FiniteGroups has been read and assigned
Remember table of generateGinvariants has been read and assigned
```

Example 2: The so called *Newton identities* give relations between elementary symmetric functions sigma[1], sigma[2], ..., and the power sums s[1], s[2], ...

(i) s[k] - sigma[1]*s[k-1]+ ... + (-1)^(k-1)*sigma[k-1]*s[1] + (-1)^k*k*sigma[k] = 0, for 1 <= k <= n,

(ii) s[k] - sigma[1]*s[k-1+ ... + (-1)^(n-1)*sigma[n-1]*s[k-n+1] + (-1)^n*sigma[n]*sigma[k-n] = 0, for k > n.

Let n = 1, 2, 3, 4, 5. Then, we can verify these identities as follows:

```
> eq1:=k->simplify(powersum[k](ss) +
  add((-1)^j*Sigma[j](ss)*powersum[k-j](ss),j=1..(k-1)) +
  (-1)^k*k*Sigma[k](ss));                    # relation (i)
  eq2:=k->simplify(powersum[k](ss) +
  add((-1)^j*Sigma[j](ss)*powersum[k-j](ss),j=1..(n-1)) +
  (-1)^n*Sigma[n](ss)*powersum[k-n](ss)); # relation (ii)
```

$$eq1 := k \rightarrow \mathrm{simplify}(SP{:}\text{-}powersum_k(ss)$$
$$+ \mathrm{add}((-1)^j\, SP{:}\text{-}\Sigma_j(ss)\, SP{:}\text{-}powersum_{k-j}(ss), j = 1 \,..\, k-1) + (-1)^k\, k\, SP{:}\text{-}\Sigma_k(ss))$$

$$eq2 := k \to \text{simplify}(SP{:}\text{-}powersum_k(ss) + \text{add}((-1)^j\, SP{:}\text{-}\Sigma_j(ss)\, SP{:}\text{-}powersum_{k-j}(ss), j = 1 .. n-1) + (-1)^n\, SP{:}\text{-}\Sigma_n(ss)\, SP{:}\text{-}powersum_{k-n}(ss))$$

```
> n:=1:
  res:=[]:
  ss:=seq(x[i],i=1..n);
  for k from 1 to n+9 do
      if k <= n then res:=[op(res),eq1(k)] else
  res:=[op(res),eq2(k)] end if;
  end do;
  res;
```

$$ss := x_1$$

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

```
> n:=2:
  res:=[]:
  ss:=seq(x[i],i=1..n);
  for k from 1 to n+9 do
      if k <= n then res:=[op(res),eq1(k)] else
  res:=[op(res),eq2(k)] end if;
  end do;
  res;
```

$$ss := x_1, x_2$$

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

```
> n:=3:
  res:=[]:
  ss:=seq(x[i],i=1..n);
  for k from 1 to n+9 do
      if k <= n then res:=[op(res),eq1(k)] else
  res:=[op(res),eq2(k)] end if;
  end do;
  res;
```

$$ss := x_1, x_2, x_3$$

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

```
> n:=4:
  res:=[]:
  ss:=seq(x[i],i=1..n);
```

```
for k from 1 to n+9 do
    if k <= n then res:=[op(res),eq1(k)] else
res:=[op(res),eq2(k)] end if;
end do;
res;
```

$$ss := x_1, x_2, x_3, x_4$$

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

```
> n:=5:
res:=[]:
ss:=seq(x[i],i=1..n);
for k from 1 to n+9 do
    if k <= n then res:=[op(res),eq1(k)] else
res:=[op(res),eq2(k)] end if;
end do;
res;
```

$$ss := x_1, x_2, x_3, x_4, x_5$$

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

```
>
```

Example 3: While the pocedure isSymmetric allows one to express every power sums s[k] in terms of the elementary symmetric functions sigma[j], 1<= j <= k, the Newton identities allow one to express every elementary symmetric function sigma[k] in terms of the power sums s[j], 1 <= j <= k, through the following recursion:

(1) sigma[1] = s[1],

(2) sigma[k] = (-1)^(k-1)*1/k*(s[k] - sigma[1]*s[k-1] + ... + (-1)^(k-1)*sigma[k-1]*s[1]), for any 1 <= k <= n.

```
> Sigma_to_powersum:=k->if k=1 then powersum[1](ss) else
simplify((-1)^(k-1)*1/k*(powersum[k](ss)
+add((-1)^j*Sigma[j](ss)*powersum[k-j](ss),j=1..(k-1)))) end
if;
```

$$Sigma\_to\_powersum := k \rightarrow$$

if $k = 1$ **then** $SP{:}\text{-}powersum_1(ss)$

else $\text{simplify}\left(\frac{(-1)^{(k-1)}\,(SP{:}\text{-}powersum_k(ss) + \text{add}((-1)^j\, SP{:}\text{-}\Sigma_j(ss)\, SP{:}\text{-}powersum_{k-j}(ss), j = 1\,..}{k}\right.$

end if

```
> n:=1:
```

```
ss:=seq(x[i],i=1..n);
Sigma[1](ss)=Sigma_to_powersum(1);
```

$$ss := x_1$$

$$x_1 = x_1$$

```
> n:=2:
ss:=seq(x[i],i=1..n);
for k from 1 to n do
Sigma[k](ss) = Sigma_to_powersum(k);
end do;
```

$$ss := x_1, x_2$$

$$x_1 + x_2 = x_1 + x_2$$

$$x_1\, x_2 = x_1\, x_2$$

```
> n:=3:
ss:=seq(x[i],i=1..n);
for k from 1 to n do
Sigma[k](ss) = Sigma_to_powersum(k);
end do;
```

$$ss := x_1, x_2, x_3$$

$$x_1 + x_2 + x_3 = x_1 + x_2 + x_3$$

$$x_1\, x_2 + x_1\, x_3 + x_2\, x_3 = x_1\, x_2 + x_1\, x_3 + x_2\, x_3$$

$$x_1\, x_2\, x_3 = x_1\, x_2\, x_3$$

```
> n:=4:
ss:=seq(x[i],i=1..n);
for k from 1 to n do
Sigma[k](ss) = Sigma_to_powersum(k);
end do;
```

$$ss := x_1, x_2, x_3, x_4$$

$$x_1 + x_2 + x_3 + x_4 = x_1 + x_2 + x_3 + x_4$$

$$x_1\, x_2 + x_1\, x_3 + x_1\, x_4 + x_2\, x_3 + x_2\, x_4 + x_3\, x_4 = x_1\, x_2 + x_1\, x_3 + x_1\, x_4 + x_2\, x_3 + x_2\, x_4 + x_3\, x_4$$

$$x_1\, x_2\, x_3 + x_1\, x_2\, x_4 + x_1\, x_3\, x_4 + x_2\, x_3\, x_4 = x_1\, x_2\, x_3 + x_1\, x_2\, x_4 + x_1\, x_3\, x_4 + x_2\, x_3\, x_4$$

$$x_1\, x_2\, x_3\, x_4 = x_1\, x_2\, x_3\, x_4$$

```
> n:=5:
ss:=seq(x[i],i=1..n);
for k from 1 to n do
Sigma[k](ss) = Sigma_to_powersum(k);
end do;
```

$$ss := x_1, x_2, x_3, x_4, x_5$$

$$x_1 + x_2 + x_3 + x_4 + x_5 = x_1 + x_2 + x_3 + x_4 + x_5$$

$$x_1 x_2 + x_1 x_3 + x_1 x_4 + x_1 x_5 + x_2 x_3 + x_2 x_4 + x_2 x_5 + x_3 x_4 + x_3 x_5 + x_4 x_5 = x_1 x_2 + x_1 x_3 + x_1 x_4 + x_1 x_5 + x_2 x_3 + x_2 x_4 + x_2 x_5 + x_3 x_4 + x_3 x_5 + x_4 x_5$$

$$x_1 x_2 x_3 + x_1 x_2 x_4 + x_1 x_2 x_5 + x_1 x_3 x_4 + x_1 x_3 x_5 + x_1 x_4 x_5 + x_2 x_3 x_4 + x_2 x_3 x_5 + x_2 x_4 x_5 + x_3 x_4 x_5 = x_1 x_2 x_3 + x_1 x_2 x_4 + x_1 x_2 x_5 + x_1 x_3 x_4 + x_1 x_3 x_5 + x_1 x_4 x_5 + x_2 x_3 x_4 + x_2 x_3 x_5 + x_2 x_4 x_5 + x_3 x_4 x_5$$

$$x_1 x_2 x_3 x_4 + x_1 x_2 x_3 x_5 + x_1 x_2 x_4 x_5 + x_1 x_3 x_4 x_5 + x_2 x_3 x_4 x_5 = x_1 x_2 x_3 x_4 + x_1 x_2 x_3 x_5 + x_1 x_2 x_4 x_5 + x_1 x_3 x_4 x_5 + x_2 x_3 x_4 x_5$$

$$x_1 x_2 x_3 x_4 x_5 = x_1 x_2 x_3 x_4 x_5$$

```
>
```

Example 4: There is, in fact, a procedure in the module called 'sigma_to_powersum' that expresses sigma[k] polynomials in variables x[1], ..., x[n], where 1 <= k <= n, in terms of the power sums s[k]:

```
> n:=7:ss:=seq(x[i],i=1..n);
  for k from 1 to n do
      sigma[k]=sigma_to_powersum[k](ss);
  end do;
```

$$ss := x_1, x_2, x_3, x_4, x_5, x_6, x_7$$

$$\sigma_1 = s_1$$

$$\sigma_2 = -\frac{1}{2} s_2 + \frac{1}{2} s_1^2$$

$$\sigma_3 = \frac{1}{3} s_3 - \frac{1}{2} s_1 s_2 + \frac{1}{6} s_1^3$$

$$\sigma_4 = -\frac{1}{4} s_4 + \frac{1}{3} s_1 s_3 + \frac{1}{8} s_2^2 - \frac{1}{4} s_2 s_1^2 + \frac{1}{24} s_1^4$$

$$\sigma_5 = \frac{1}{5} s_5 - \frac{1}{4} s_1 s_4 - \frac{1}{6} s_3 s_2 + \frac{1}{6} s_3 s_1^2 + \frac{1}{8} s_1 s_2^2 - \frac{1}{12} s_2 s_1^3 + \frac{1}{120} s_1^5$$

$$\sigma_6 = -\frac{1}{6} s_6 + \frac{1}{5} s_1 s_5 + \frac{1}{8} s_4 s_2 - \frac{1}{8} s_4 s_1^2 + \frac{1}{18} s_3^2 - \frac{1}{6} s_3 s_1 s_2 + \frac{1}{18} s_3 s_1^3 - \frac{1}{48} s_2^3 + \frac{1}{16} s_2^2 s_1^2 - \frac{1}{48} s_2 s_1^4 + \frac{1}{720} s_1^6$$

$$\sigma_7 = \frac{1}{8} s_4 s_1 s_2 - \frac{1}{12} s_3 s_2 s_1^2 - \frac{1}{6} s_1 s_6 - \frac{1}{10} s_5 s_2 + \frac{1}{10} s_5 s_1^2 - \frac{1}{12} s_4 s_3 - \frac{1}{24} s_4 s_1^3 + \frac{1}{18} s_1 s_3^2 + \frac{1}{24} s_3 s_2^2 + \frac{1}{72} s_3 s_1^4 - \frac{1}{48} s_1 s_2^3 + \frac{1}{48} s_2^2 s_1^3 - \frac{1}{240} s_2 s_1^5 + \frac{1}{7} s_7 + \frac{1}{5040} s_1^7$$

```
>
```

We can verify that the above is correct by converting each side to polynomials in x[1], x[2],..., x[n]:

```
> n:=4:ss:=seq(x[i],i=1..n);
```

$$ss := x_1, x_2, x_3, x_4$$

```
> for k from 1 to n do

  evalb(Sigma[k](ss)=expand(simplify(subs({seq(s[i]=powersum[i](s
  s),i=1..k)},convert(sigma_to_powersum[k](ss),`global`)))));
  end do;
```

$$true$$
$$true$$
$$true$$
$$true$$

```
>
```

Algorithm used:

None

See Also: SP:-Sigma, SP:-isSymmetric, SP:-hpolynom, SP:-sigma_to_powersum

Last modified: June 19, 2008
Last modified: June 19, 2008

Function: SP:-reduceGinvariants - reduce a list of G invariants

Calling Sequence:

reduceGinvariants[G](F,indets)

Parameters:

- F = [f1,f2,...,fm] is a list of polynomials in the ring *k*[x[1], x[2], x[3],..., x[n]].
- indets = [x[1], x[2],..., x[n]] is a list of indeterminates

Output:

- A list FF with polynomials g1, g2, ..., gs in [x[1], x[2],..., x[n]] where s <= n such that for every i =1,..., s, polynomial g||i does not belong to the ideal I_g||i generated by the remaining (i-1) polynomials.

Description:

- Procedure 'reduceGinvariants' reduces a list F of G-invariants consisting of polynomials f1,f2,...,fm in *k*[x[1], x[2],..., x[n]] where G is a group of n x n matrices. It produces a new list FF with the following property: No polynomial in FF belongs to an ideal in the ring *k*[x[1], x[2],..., x[n]] generated by the remaining polynomials in the list FF.
- To determine whether a polynomial f1 belongs to an ideal I_f1 = <f2, f3, ..., fm> one uses the procedure 'isContained'. If it does not, it is added to the list FF which initially is empty. Then, the same process is applied to the polynomial f2 and the ideal I_f2 = <f1,f3,...,fm>.
- In practice, the polynomials f1, f2,..., fm, will often be some elementary invariants of a finite group G, for example, like the elementary symmetric functions are invariant under the group S[n], and they will generate a ring of invariants k[x[1], x[2],..., x[n]]^G. Then, we will be interested to find a reduced set of generators. Once reduced, there still could be syzygy relation(s) among the polynomials in the output list FF.
- References:
  [1] D. Cox, J. Little, D. O'Shea: "Ideals, varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra", Third Edition, Springer, New York, 2007
  [2] B. Sturmfels: "Algorithms in Invariant Theory", Texts and Monographs in Symbolic Computation, Springer-Verlag/Wien, 1993

Examples:

```
> restart:with(SP);
Remember table of SymmetricGroup has been read and assigned
Remember table of AlternatingGroup has been read and assigned
```

```
Remember table of Reynolds has been read and assigned
Remember table of FiniteGroups has been read and assigned
Remember table of generateGinvariants has been read and assigned
```

[*AlternatingGroup*, *Dpolynom*, *FiniteGroups*, *Hilbert_series*, *MatrixAction*, *ModuleLoad*, *ModuleUnload*, *Molien_series*, *Reynolds*, *SPversion*, *Schur_polynom*, Σ, *SymmetricGroup*, *SyzygyIdeal*, *a_polynom*, *create_partitions*, *generateGinvariants*, *gpolynom*, *hpolynom*, *isContained*, *isGinvariant*, *isSymmetric*, *ispartition*, *load_remember_table*, *maxmindegree*, *permsign*, *powersum*, *reduceGinvariants*, *sigma_to_powersum*]

Example 1: Reducing lists of group invariants that were produced using the procedure 'generateGinvariants' (see SP:-generateGinvariants):

```
> C4:=FiniteGroups('C4');
  C4invariants:=generateGinvariants[C4]([x,y]);
  nops(C4invariants);
  FF:=reduceGinvariants[C4](C4invariants,[x,y]);
  nops(FF);
  I_FF:=SyzygyIdeal(FF);
```

$$C4 := \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\right]$$

$$C4invariants := [x^2 + y^2, y^4 + x^4, x\,y^3 - x^3\,y, x^3\,y - x\,y^3, x^2\,y^2]$$

$$5$$

```
Path set to C:\Maple11\bin.win/libfgbuni.so
    FGb/Maple interface package Version 1.34
    JC Faugere (jcf@calfor.lip6.fr)
    Type ?FGb for documentation
```

$$FF := [x\,y^3 - x^3\,y, x^2\,y^2, x^2 + y^2]$$

$$3$$

$$I\_FF := [\_y_2\,\_y_3^{\,2} - \_y_1^{\,2} - 4\,\_y_2^{\,2}]$$

The above shows that the reduced list of C4 invariants has 3 polynomials (list FF) which satisfy one syzygy relation shown to generate the syzygy ideal I_FF.

```
>
> S2:=SymmetricGroup(2);
  S2invariants:=generateGinvariants[S2]([x,y]);
  nops(S2invariants);
  FF:=reduceGinvariants[S2](S2invariants,[x,y]);
  nops(FF);
  I_FF:=SyzygyIdeal(FF);
```

$$S2 := \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\right]$$

$$S2invariants := [x\,y, x^2 + y^2, y + x]$$

$$3$$

$$FF := [x\,y, y + x]$$

$$2$$

$$I\_FF := [0]$$

The above shows that the reduced list of S2 invariants has 2 polynomials (list FF) which do not satisfy any syzygy relation.

```
>
> S3:=SymmetricGroup(3);
  S3invariants:=generateGinvariants[S3]([x,y,z]);
  nops(S3invariants);
  FF:=reduceGinvariants[S3](S3invariants,[x,y,z]);
  nops(FF);
  I_FF:=SyzygyIdeal(FF);
```

$$S3 := \left[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}\right]$$

$$S3invariants := [z^2 + y^2 + x^2, y^2 z^3 + y^3 z^2 + x^2 z^3 + x^2 y^3 + x^3 z^2 + x^3 y^2, x y^2 z^2 + x^2 y z^2 + x^2 y^2 z,$$
$$z^3 + y^3 + x^3, y z^4 + y^4 z + x z^4 + x y^4 + x^4 z + x^4 y, x y z^4 + x y^4 z + x^4 y z, z^4 + y^4 + x^4,$$
$$y z^2 + y^2 z + x z^2 + x y^2 + x^2 z + x^2 y, x y^2 z^3 + x y^3 z^2 + x^2 y z^3 + x^2 y^3 z + x^3 y z^2 + x^3 y^2 z,$$
$$z^6 + y^6 + x^6, x y z^2 + x y^2 z + x^2 y z, z + y + x, y^3 z^3 + x^3 z^3 + x^3 y^3, x y + x z + y z,$$
$$y^2 z^4 + y^4 z^2 + x^2 z^4 + x^2 y^4 + x^4 z^2 + x^4 y^2, x y z^3 + x y^3 z + x^3 y z, z^5 + y^5 + x^5, x y z, x^2 y^2 z^2,$$
$$y^2 z^2 + x^2 z^2 + x^2 y^2, y z^5 + y^5 z + x z^5 + x y^5 + x^5 z + x^5 y, y z^3 + y^3 z + x z^3 + x y^3 + x^3 z + x^3 y]$$

$$22$$

$$FF := [x\,y\,z, x\,y + x\,z + y\,z, z + y + x]$$

$$3$$

$$I\_FF := [0]$$

The above shows that the reduced list of S3 invariants has 3 polynomials (list FF) which do not satisfy any syzygy relation.

```
>
> Gamma3:=FiniteGroups('Gamma3');
  Gamma3invariants:=generateGinvariants[Gamma3]([x,y,z]);
  nops(Gamma3invariants);
  FF:=reduceGinvariants[Gamma3](Gamma3invariants,[x,y,z]);
  nops(FF);
  I_FF:=SyzygyIdeal(FF);
  nops(I_FF);
```

$$\Gamma 3 := \left[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}\right]$$

$$Gamma3invariants :=$$

$$[x^2+y^2, y^4+x^4, x\,y^3-x^3\,y, x^3\,y-x\,y^3, x^2\,y^2, y^2\,z-x^2\,z, x^2\,z-y^2\,z, y^2\,z^2+x^2\,z^2, x\,y\,z, z^2, z^4]$$

11

$$FF := [x\,y^3-x^3\,y, x^2\,y^2, y^2\,z-x^2\,z, x\,y\,z, x^2+y^2, z^2]$$

6

$$I\_FF := [\_y_2\,\_y_5^2-\_y_1^2-4\,\_y_2^2, \_y_4\,\_y_5^2-\_y_1\,\_y_3-4\,\_y_2\,\_y_4, \_y_5^2\,\_y_6-\_y_3^2-4\,\_y_2\,\_y_6, \_y_2\,\_y_3-\_y_1\,\_y_4, \_y_3\,\_y_4-\_y_1\,\_y_6, \_y_4^2-\_y_2\,\_y_6]$$

6

The above shows that the reduced list of Gamma3 invariants has 6 polynomials (list FF) which satisfy six syzygy relations.

>

>

Algorithm used:

- Proposition 3 on page 339 in [1]:
- According to Proposition 3 on page 339 in [1], the ideal I_F is the n-th elimination ideal of an ideal J_F = <f1 - y[1], f2 - y[2], ..., fm - y[m]> in the polynomial ring *k*[x[1], x[2], ..., x[n], y[1], y[2],..., y[m]]. That is, I_F is the intersection of J_F with *k*[y[1], y[2],..., y[m]].

  Step 1: Let F = [f1, f2,..., fm] and let T be any monomial order where any monomial involving one of x[1], x[2], ..., x[n] is greater than all monomials in *k*[y[1], y[2],... y[m]]. For example, we can set T to be the lexicographic order x[1] > x[2] > ...> x[n] > y[1] > y[2] >...> y[m].

  Step 2: Define an ideal J_F = <f1 - y[1], f2 - y[2], ..., fm - y[m]> in the polynomial ring *k*[x[1], x[2], ..., x[n], y[1], y[2],..., y[m]] and compute a Groebner basis G for J_F for the monomial order T.

  Step 3: Find the intersection of G with the ring *k*[y[1], y[2],... y[m]]: This interesection provides a Groebner basis for I_F. That is, the Groebner basis for I_F consists of those polynomials in G, if any, which belong to *k*[y[1], y[2],... y[m]].

See Also: SP:-FiniteGroups, SP:-generateGinvariants, SP:-SyzygyIdeal, SP:-isContained

Last modified: June 19, 2008

Function: SP:-Reynolds - defines the Reynolds operator

Calling Sequence:

Reynolds[G](f, L);

Parameters:

- G is a list of n x n matrices that represent a finite group G.
- f is the polynomial in *k*[x[1], x[2], x[3],..., x[n]]
- L is a list of all ring indeterminates, that is, L = [x[1], x[2], x[3],..., x[n]]

Output:

- A polynomial in *k*[x[1], x[2], x[3],..., x[n]]^G, that is, a polynomial in the ring of invariants of the group G

Description:

- Procedure 'Reynolds' computes action of the Reynolds operator of a finite group G used as the index of the procedure on a polynomial f in a polynomial ring k[x[1], x[2], x[3],..., x[n]] whose indeterminates are entered as the last list.
- The Reynolds operator R is defined as R[G](f)(**x**) = 1/|G| Sigma f(A**x**) where the summation in Sigma is over all elements A in the group G and f(A**x**) is a polynomial obtained by acting with an n x n matrix A, an element of G, on the polynomial f.
- The action of A in G on f is computed using the procedure SP:-MatrixAction.
- Polynomial f can actually contain fewer indeterminates than in the list [x[1], x[2], x[3],..., x[n]].
- To find out which groups are already remembered, check SP:-FiniteGroups.
- Iinvariants of a finite group G can be generated with the procedure SP:-generateGinvariants. These invariants are usually algebraically dependent in the ring *k*[x[1], x[2], x[3],..., x[n]]. To reduce them to truly algebraically independent, use the procedure SP:-reduceGinvariants.
- References:
  [1] D. Cox, J. Little, D. O'Shea: "Ideals, varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra", Third Edition, Springer, New York, 2007
  [2] B. Sturmfels: "Algorithms in Invariant Theory", Texts and Monographs in Symbolic Computation, Springer-Verlag/Wien, 1993

Examples:

```
> restart:with(SP);
```

```
Remember table of SymmetricGroup has been read and assigned
Remember table of AlternatingGroup has been read and assigned
Remember table of Reynolds has been read and assigned
Remember table of FiniteGroups has been read and assigned
Remember table of generateGinvariants has been read and assigned
```

[*AlternatingGroup*, *Dpolynom*, *FiniteGroups*, *Hilbert_series*, *MatrixAction*, *ModuleLoad*, *ModuleUnload*, *Molien_series*, *Reynolds*, *SPversion*, *Schur_polynom*, Σ, *SymmetricGroup*, *SyzygyIdeal*, *a_polynom*, *create_partitions*, *generateGinvariants*, *gpolynom*, *hpolynom*, *isContained*, *isGinvariant*, *isSymmetric*, *ispartition*, *load_remember_table*, *maxmindegree*, *permsign*, *powersum*, *reduceGinvariants*, *sigma_to_powersum*]

Example 1: Let's compute the action of the Reynolds operator.Acting with a matrix on a polynomial:

```
> f:=Sigma[2](x,y,z);
```

$$f := x\,y + x\,z + y\,z$$

```
> S3:=SymmetricGroup(3);
```

$$S3 := \left[\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix}, \begin{bmatrix}0&1&0\\1&0&0\\0&0&1\end{bmatrix}, \begin{bmatrix}0&0&1\\1&0&0\\0&1&0\end{bmatrix}, \begin{bmatrix}0&1&0\\0&0&1\\1&0&0\end{bmatrix}, \begin{bmatrix}0&0&1\\0&1&0\\1&0&0\end{bmatrix}\right]$$

```
> g:=Reynolds[S3](f,[x,y,z]);
```

$$g := x\,y + x\,z + y\,z$$

The above computed polynomial g is an invariant of the symmetric group S[3]. This can be easily checked as follows:

```
> for A in S3 do
      MatrixAction(A,g,[x,y,z]);
  end do;
```

$$x\,y + x\,z + y\,z$$
$$x\,y + x\,z + y\,z$$
$$x\,y + x\,z + y\,z$$
$$x\,y + x\,z + y\,z$$
$$x\,y + x\,z + y\,z$$
$$x\,y + x\,z + y\,z$$

Example 2: Here is another invariant of C2 obtained by applying the Reynolds operator to a polynomial f:

```
> f:=x*y-z^6+x;
```

$$f := x\,y - z^6 + x$$

```
> G:=FiniteGroups('Gamma3');
```

$$G := \left[\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}0&1&0\\-1&0&0\\0&0&-1\end{bmatrix}, \begin{bmatrix}-1&0&0\\0&-1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}0&-1&0\\1&0&0\\0&0&-1\end{bmatrix}\right]$$

```
> g:=Reynolds[G](f,[x,y,z]);
```

$$g := -z^6$$

```
> for A in G do
      MatrixAction(A,g,[x,y,z]);
  end do;
```

$$-z^6$$

$$-z^6$$

$$-z^6$$

$$-z^6$$

The above shows that the polynomial g = -z^6 is the invariant of teh group Gamma3 which is a three dimensional representation of the cyclic group of order 4.

Of course, if we use another group, for example the symmetric group S[3], we get a different invariant:

```
> g:=Reynolds[S3](f,[x,y,z]);
```

$$g := \frac{1}{3}x\,y-\frac{1}{3}z^6+\frac{1}{3}x+\frac{1}{3}x\,z-\frac{1}{3}y^6+\frac{1}{3}y+\frac{1}{3}z+\frac{1}{3}y\,z-\frac{1}{3}x^6$$

```
> Reynolds[S3](g,[x,y,z]);
```

$$\frac{1}{3}x\,y-\frac{1}{3}z^6+\frac{1}{3}x+\frac{1}{3}x\,z-\frac{1}{3}y^6+\frac{1}{3}y+\frac{1}{3}z+\frac{1}{3}y\,z-\frac{1}{3}x^6$$

```
> for A in S3 do
      MatrixAction(A,g,[x,y,z]);
  end do;
```

$$\frac{1}{3}x\,y-\frac{1}{3}z^6+\frac{1}{3}x+\frac{1}{3}x\,z-\frac{1}{3}y^6+\frac{1}{3}y+\frac{1}{3}z+\frac{1}{3}y\,z-\frac{1}{3}x^6$$

$$\frac{1}{3}x\,y-\frac{1}{3}z^6+\frac{1}{3}x+\frac{1}{3}x\,z-\frac{1}{3}y^6+\frac{1}{3}y+\frac{1}{3}z+\frac{1}{3}y\,z-\frac{1}{3}x^6$$

$$\frac{1}{3}x\,y-\frac{1}{3}z^6+\frac{1}{3}x+\frac{1}{3}x\,z-\frac{1}{3}y^6+\frac{1}{3}y+\frac{1}{3}z+\frac{1}{3}y\,z-\frac{1}{3}x^6$$

$$\frac{1}{3}x\,y-\frac{1}{3}z^6+\frac{1}{3}x+\frac{1}{3}x\,z-\frac{1}{3}y^6+\frac{1}{3}y+\frac{1}{3}z+\frac{1}{3}y\,z-\frac{1}{3}x^6$$

$$\frac{1}{3}x\,y-\frac{1}{3}z^6+\frac{1}{3}x+\frac{1}{3}x\,z-\frac{1}{3}y^6+\frac{1}{3}y+\frac{1}{3}z+\frac{1}{3}y\,z-\frac{1}{3}x^6$$

$$\frac{1}{3}x\,y-\frac{1}{3}z^6+\frac{1}{3}x+\frac{1}{3}x\,z-\frac{1}{3}y^6+\frac{1}{3}y+\frac{1}{3}z+\frac{1}{3}y\,z-\frac{1}{3}x^6$$

```
>
```

Example 3: This is Example 4 on page 331 from [1]:

```
> C4:=FiniteGroups('C4');
```

$$C4 := \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\right]$$

We compute some invariants by acting with the Reynolds operator on some monomials:

```
> g1:=Reynolds[C4](x^2,[x,y]);
  g2:=Reynolds[C4](x*y,[x,y]);
  g3:=Reynolds[C4](x^3*y,[x,y]);
  g4:=Reynolds[C4](x^2*y^2,[x,y]);
```

$$g1 := \frac{x^2}{2} + \frac{y^2}{2}$$

$$g2 := 0$$

$$g3 := \frac{1}{2} x^3 y - \frac{1}{2} y^3 x$$

$$g4 := x^2 y^2$$

Thus, the polynomials g1, g2, g3, g4 belong to *k*[x,y]^C4, the ring of invariants of C4:

```
> g1=Reynolds[C4](g1,[x,y]);
  g2=Reynolds[C4](g2,[x,y]);
  g3=Reynolds[C4](g3,[x,y]);
  g4=Reynolds[C4](g4,[x,y]);
```

$$\frac{x^2}{2} + \frac{y^2}{2} = \frac{x^2}{2} + \frac{y^2}{2}$$

$$0 = 0$$

$$\frac{1}{2} x^3 y - \frac{1}{2} y^3 x = \frac{1}{2} x^3 y - \frac{1}{2} y^3 x$$

$$x^2 y^2 = x^2 y^2$$

```
>
```

Example 4: The Emmy Noether Theorem states as follows:

Theorem (Emmy Noether) Given a finite group G in GL(n,*k*), we have

k[x[1],x[2],...,x[n]]^G = *k*[R[G](x^beta) | |beta| <= |G|].

In particular, *k*[x[1],x[2],...,x[n]]^G is generated by finitely many homogeneous invariants.

Thus, for example, since the order of the group C4 is 4, the ring of invariants k[x,y]^C4 is generated by the following homogeneous polynomials of degree up to and including 4.

First, we generate monomial terms x^beta such that |beta| <= 4:

```
> n:=4:
  x:='x':y:='y':
  mons:=[]:
  for i from 0 to n do
  for j from 0 to n do
  if i+j<=n and i+j>0 then mons:=[op(mons),x^i*y^j] end if;
  end do;
  end do;
  'mons'=mons;
```

$$mons = [y, y^2, y^3, y^4, x, x\,y, x\,y^2, y^3\,x, x^2, x^2\,y, x^2\,y^2, x^3, x^3\,y, x^4]$$

Second, we apply the Reynolds operator to the list of monomials:

```
> generators:=[]:
  for m in mons do
      generators:=[op(generators),Reynolds[C4](m,[x,y])];
  end do;
```

$$generators := [0]$$

$$generators := \left[0, \frac{x^2}{2} + \frac{y^2}{2}\right]$$

$$generators := \left[0, \frac{x^2}{2} + \frac{y^2}{2}, 0\right]$$

$$generators := \left[0, \frac{x^2}{2} + \frac{y^2}{2}, 0, \frac{y^4}{2} + \frac{x^4}{2}\right]$$

$$generators := \left[0, \frac{x^2}{2} + \frac{y^2}{2}, 0, \frac{y^4}{2} + \frac{x^4}{2}, 0\right]$$

$$generators := \left[0, \frac{x^2}{2} + \frac{y^2}{2}, 0, \frac{y^4}{2} + \frac{x^4}{2}, 0, 0\right]$$

$$generators := \left[0, \frac{x^2}{2} + \frac{y^2}{2}, 0, \frac{y^4}{2} + \frac{x^4}{2}, 0, 0, 0\right]$$

$$generators := \left[0, \frac{x^2}{2} + \frac{y^2}{2}, 0, \frac{y^4}{2} + \frac{x^4}{2}, 0, 0, 0, \frac{1}{2}y^3\,x - \frac{1}{2}x^3\,y\right]$$

$$generators := \left[0, \frac{x^2}{2} + \frac{y^2}{2}, 0, \frac{y^4}{2} + \frac{x^4}{2}, 0, 0, 0, \frac{1}{2}y^3\,x - \frac{1}{2}x^3\,y, \frac{x^2}{2} + \frac{y^2}{2}\right]$$

$$generators := \left[0, \frac{x^2}{2} + \frac{y^2}{2}, 0, \frac{y^4}{2} + \frac{x^4}{2}, 0, 0, 0, \frac{1}{2}y^3\,x - \frac{1}{2}x^3\,y, \frac{x^2}{2} + \frac{y^2}{2}, 0\right]$$

$$generators := \left[0, \frac{x^2}{2}+\frac{y^2}{2}, 0, \frac{y^4}{2}+\frac{x^4}{2}, 0, 0, 0, \frac{1}{2}y^3 x-\frac{1}{2}x^3 y, \frac{x^2}{2}+\frac{y^2}{2}, 0, x^2 y^2\right]$$

$$generators := \left[0, \frac{x^2}{2}+\frac{y^2}{2}, 0, \frac{y^4}{2}+\frac{x^4}{2}, 0, 0, 0, \frac{1}{2}y^3 x-\frac{1}{2}x^3 y, \frac{x^2}{2}+\frac{y^2}{2}, 0, x^2 y^2, 0\right]$$

$$generators := \left[0, \frac{x^2}{2}+\frac{y^2}{2}, 0, \frac{y^4}{2}+\frac{x^4}{2}, 0, 0, 0, \frac{1}{2}y^3 x-\frac{1}{2}x^3 y, \frac{x^2}{2}+\frac{y^2}{2}, 0, x^2 y^2, 0, \frac{1}{2}x^3 y-\frac{1}{2}y^3 x\right]$$

$$generators := \left[0, \frac{x^2}{2}+\frac{y^2}{2}, 0, \frac{y^4}{2}+\frac{x^4}{2}, 0, 0, 0, \frac{1}{2}y^3 x-\frac{1}{2}x^3 y, \frac{x^2}{2}+\frac{y^2}{2}, 0, x^2 y^2, 0, \frac{1}{2}x^3 y-\frac{1}{2}y^3 x, \frac{y^4}{2}+\frac{x^4}{2}\right]$$

Finally, we remove the duplicates and the zero polynomial:

```
> generators:=convert(convert(generators,set) minus {0},list):
  generators:=map(m->m/icontent(m),generators);
```

$$generators := [y^3 x - x^3 y, x^2 + y^2, x^3 y - y^3 x, y^4 + x^4, x^2 y^2]$$

```
> nops(generators);
```

$$5$$

Thus, the above shows that the ring of invariants *k*[x,y]^C4 of the group C4 is generated by five polynomials in the list 'generators'. Of course, these generators may be and usually are algebraically dependent.

Example 5: Group C8 has 44 generating invariants that can be computed using the Reynolds operator. There is a built into 'FiniteGroups' a three dimensional repersentation of C8, but we will use a two dimensional repersentations of C8 generated by the following matrix A, that is, C8 is the cyclic group = <A>:

```
> A:=evalm(1/sqrt(2)*matrix(2,2,[1,-1,1,1]));
  C8:=[seq(evalm(A^k),k=1..8)];
```

$$A := \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}$$

$$C8 := \left[\begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}, \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix},$$

$$\left[\begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right]$$

```
> n:=8:
  x:='x':y:='y':
  mons:=[]:
  for i from 0 to n do
  for j from 0 to n do
  if i+j<=n and i+j>0 then mons:=[op(mons),x^i*y^j] end if;
  end do;
  end do;
  'mons'=mons;
```

$$mons = [y, y^2, y^3, y^4, y^5, y^6, y^7, y^8, x, x\,y, x\,y^2, y^3\,x, x\,y^4, x\,y^5, x\,y^6, x\,y^7, x^2, x^2\,y, x^2\,y^2, x^2\,y^3, x^2\,y^4, x^2\,y^5, x^2\,y^6, x^3, x^3\,y, x^3\,y^2, x^3\,y^3, x^3\,y^4, x^3\,y^5, x^4, x^4\,y, x^4\,y^2, x^4\,y^3, x^4\,y^4, x^5, x^5\,y, x^5\,y^2, x^5\,y^3, x^6, x^6\,y, x^6\,y^2, x^7, x^7\,y, x^8]$$

The number of monomials to be acted on is 44:

```
> nops(mons);
```

$$44$$

```
> generators:={}:
  for m in mons do
      generators:={op(generators),Reynolds[C8](m,[x,y])};
  end do:
  generators:=convert(generators minus {0},list):
  generators:=map(m->m/icontent(m),generators);
```

$$generators := [x^2 + y^2, -x\,y^7 + 7\,x^3\,y^5 - 7\,x^5\,y^3 + x^7\,y, x\,y^7 - 7\,x^3\,y^5 + 7\,x^5\,y^3 - x^7\,y, 28\,x^2\,y^6 + 9\,y^8 + 70\,x^4\,y^4 + 28\,x^6\,y^2 + 9\,x^8, x^4 + 2\,x^2\,y^2 + y^4, 12\,x^2\,y^6 + y^8 - 10\,x^4\,y^4 + 12\,x^6\,y^2 + x^8, -4\,x^2\,y^6 + y^8 + 22\,x^4\,y^4 - 4\,x^6\,y^2 + x^8, y^6 + x^6 + 3\,x^2\,y^4 + 3\,x^4\,y^2, x^4 + 2\,x^2\,y^2 + y^4, y^6 + x^6 + 3\,x^2\,y^4 + 3\,x^4\,y^2]$$

```
> nops(generators);
```

$$10$$

```
>
```

Thus, by applying the Noether Theorem, we find ten generators generating the ring of invariants k[x,y]^C8 although only three are needed as these generators are algebraically dependent in k[x,y].

```
>
```

Algorithm used:

None

See Also: SP:-Sigma, SP:-FiniteGroups, SP:-AlternatingGroup, SP:-SymmetricGroup, SP:-generateGinvariants, SP:-reduceGinvariants , SP:-MatrixAction

Last modified: June 19, 2008

Function: SP:-Sigma - creates a symmetric sigma polynomial of degree k in n variables where 1<= k <= n

Calling Sequence:

Sigma[k](x[1],x[2],...,x[n]);

Parameters:

- k - positive integer 1<= k <= n
- n variables, e.g., x[1], x[2],..., x[n]

Output:

- Symmetric polynomial \sigma_k of degree k in n variables

Description:

- This procedure takes no argument (parameter) and it returns information about the current version of the package.

Examples:

```
> restart:with(SP);
Remember table of SymmetricGroup has been read and assigned
Remember table of AlternatingGroup has been read and assigned
Remember table of Reynolds has been read and assigned
Remember table of FiniteGroups has been read and assigned
Remember table of generateGinvariants has been read and assigned
```

[*AlternatingGroup*, *Dpolynom*, *FiniteGroups*, *Hilbert_series*, *MatrixAction*, *ModuleLoad*, *ModuleUnload*, *Molien_series*, *Reynolds*, *SPversion*, *Schur_polynom*, Σ, *SymmetricGroup*, *SyzygyIdeal*, *a_polynom*, *create_partitions*, *generateGinvariants*, *gpolynom*, *hpolynom*, *isContained*, *isGinvariant*, *isSymmetric*, *ispartition*, *load_remember_table*, *maxmindegree*, *permsign*, *powersum*, *reduceGinvariants*, *sigma_to_powersum*]

2. Procedure Sigma used as in Sigma[k](x1,x2,...,xn) returns the elementary symmetric polynomial sigma_k in variables x1,x2,...,xn of degree k:

```
> Sigma[2](x,y,z);
```

$$x\,y + x\,z + y\,z$$

```
> Sigma[4](x,y,z); #if k > n then error is returned
Error, (in SP:-Sigma[4]) index must be no larger than the number of arguments
```

```
> Sigma[1](x[1],x[2],x[3],x[4]);
```

$$x_1 + x_2 + x_3 + x_4$$

```
> Sigma[2](x[1],x[2],x[3],x[4]);
```

$$x_1 x_2 + x_1 x_3 + x_1 x_4 + x_2 x_3 + x_2 x_4 + x_3 x_4$$

```
> Sigma[3](x[1],x[2],x[3],x[4]);
```

$$x_1 x_2 x_3 + x_1 x_2 x_4 + x_1 x_3 x_4 + x_2 x_3 x_4$$

```
> Sigma[4](x[1],x[2],x[3],x[4]);
```

$$x_1 x_2 x_3 x_4$$

Note, that a priori elementary symmetric polynomials sigma_k are not defined

```
> sigma[1];sigma[2];
```

$$\sigma_1$$

$$\sigma_2$$

```
> expand(sigma[1]),expand(sigma[2]);
```

$$\sigma_1, \sigma_2$$

```
>
```

Algorithm used:

None

See Also: SP:-Sigma, SP:-isSymmetric

Last modified: June 19, 2008

Function: SP:-sigma_to_powersum - converts an elementary symmetric polynomial to power sums

Calling Sequence:

sigma_to_powersum[k](s);

Parameters:

- The index k must be a positive integer such that 1 <= k <= n where n is the number of variables in the sequence s
- s must be a sequence of n variables

Output:

- A polynomial in power sums s[j], 1 <= j <= k <= n, in n variables specified in the sequence s

Description:

- Procedure sigma_to_powersum used as in sigma_to_powersum[k](x[1],...x[n]) returns the sigma[k] polynomial in terms of power sum polynomials s[j] where s[j] = x[1]^j + x[2]^j + ... + x[n]^j. These polynomials are defined above Theorem 8, Section 7.1 in [1].
- According to Theorem 8 in Section 7.1 of [1], if *k* is a field containing rational numbers *Q*, then every symmetric polynomial in *k*[x[1], x[2], ... , x[n]] can be written as a polynomial in the power sums s[1], s[2], ..., s[n].
- References:
  [1] D. Cox, J. Little, D. O'Shea: "Ideals, varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra", Third Edition, Springer, New York, 2007
  [2] B. Sturmfels: "Algorithms in Invariant Theory", Texts and Monographs in Symbolic Computation, Springer-Verlag/Wien, 1993

Examples:

```
> restart:with(SP):
Remember table of SymmetricGroup has been read and assigned
Remember table of AlternatingGroup has been read and assigned
Remember table of Reynolds has been read and assigned
Remember table of FiniteGroups has been read and assigned
Remember table of generateGinvariants has been read and assigned
```

Example 1: The so called *Newton identities* give relations between elementary symmetric functions sigma[1], sigma[2], ..., and the power sums s[1], s[2], ...

(i) s[k] - sigma[1]*s[k-1]+ ... + (-1)^(k-1)*sigma[k-1]*s[1] + (-1)^k*k*sigma[k] = 0, for 1 <= k <= n,

(ii) s[k] - sigma[1]*s[k-1+ ... + (-1)^(n-1)*sigma[n-1]*s[k-n+1] + (-1)^n*sigma[n]*sigma[k-n] = 0, for k > n.

Let n = 1, 2, 3, 4, 5. Then, we can verify these identities as follows:

```
> eq1:=k->simplify(powersum[k](ss) +
  add((-1)^j*Sigma[j](ss)*powersum[k-j](ss),j=1..(k-1)) +
  (-1)^k*k*Sigma[k](ss));                    # relation (i)
  eq2:=k->simplify(powersum[k](ss) +
  add((-1)^j*Sigma[j](ss)*powersum[k-j](ss),j=1..(n-1)) +
  (-1)^n*Sigma[n](ss)*powersum[k-n](ss)); # relation (ii)
```

$$eq1 := k \rightarrow \mathrm{simplify}(SP{:}\text{-}powersum_k(ss) + \mathrm{add}((-1)^j\, SP{:}\text{-}\Sigma_j(ss)\, SP{:}\text{-}powersum_{k-j}(ss), j = 1 .. k-1) + (-1)^k\, k\, SP{:}\text{-}\Sigma_k(ss))$$

$$eq2 := k \rightarrow \mathrm{simplify}(SP{:}\text{-}powersum_k(ss) + \mathrm{add}((-1)^j\, SP{:}\text{-}\Sigma_j(ss)\, SP{:}\text{-}powersum_{k-j}(ss), j = 1 .. n-1) + (-1)^n\, SP{:}\text{-}\Sigma_n(ss)\, SP{:}\text{-}powersum_{k-n}(ss))$$

```
> n:=1:
  res:=[]:
  ss:=seq(x[i],i=1..n);
  for k from 1 to n+9 do
      if k <= n then res:=[op(res),eq1(k)] else
  res:=[op(res),eq2(k)] end if;
  end do;
  res;
```

$$ss := x_1$$

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

```
> n:=2:
  res:=[]:
  ss:=seq(x[i],i=1..n);
  for k from 1 to n+9 do
      if k <= n then res:=[op(res),eq1(k)] else
  res:=[op(res),eq2(k)] end if;
```

```
end do;
res;
```

$$ss := x_1, x_2$$

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

```
> n:=3:
res:=[]:
ss:=seq(x[i],i=1..n);
for k from 1 to n+9 do
    if k <= n then res:=[op(res),eq1(k)] else
res:=[op(res),eq2(k)] end if;
end do;
res;
```

$$ss := x_1, x_2, x_3$$

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

```
> n:=4:
res:=[]:
ss:=seq(x[i],i=1..n);
for k from 1 to n+9 do
    if k <= n then res:=[op(res),eq1(k)] else
res:=[op(res),eq2(k)] end if;
end do;
res;
```

$$ss := x_1, x_2, x_3, x_4$$

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

```
> n:=5:
res:=[]:
ss:=seq(x[i],i=1..n);
for k from 1 to n+9 do
    if k <= n then res:=[op(res),eq1(k)] else
res:=[op(res),eq2(k)] end if;
end do;
res;
```

$$ss := x_1, x_2, x_3, x_4, x_5$$

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

```
>
```

Example 2: While the pocedure isSymmetric allows one to express every power sums s[k] in terms of the elementary symmetric functions sigma[j], 1<= j <= k, the Newton identities allow one

to express every elementary symmetric function sigma[k] in terms of the power sums s[j], 1 <= j <= k, through the following recursion:

(1) sigma[1] = s[1],

(2) sigma[k] = (-1)^(k-1)*1/k*(s[k] - sigma[1]*s[k-1] + ... + (-1)^(k-1)*sigma[k-1]*s[1]), for any 1 <= k <= n.

```
> sigma_to_power_sum:=
  k->if k=1 then powersum[1](ss) else
  simplify((-1)^(k-1)*1/k*(powersum[k](ss)
  +add((-1)^j*Sigma[j](ss)*powersum[k-j](ss),j=1..(k-1)))) end
  if;
```

$$sigma\_to\_power\_sum := k \rightarrow$$
$$\textbf{if } k = 1 \textbf{ then } SP{:}\text{-}powersum_1(ss)$$
$$\textbf{else } \mathrm{simplify}\left(\frac{(-1)^{(k-1)}\,(SP{:}\text{-}powersum_k(ss) + \mathrm{add}((-1)^j\, SP{:}\text{-}\Sigma_j(ss)\, SP{:}\text{-}powersum_{k-j}(ss), j = 1\,..}{k}\right.$$
$$\textbf{end if}$$

```
> n:=1:
  ss:=seq(x[i],i=1..n);
  Sigma[1](ss)=sigma_to_power_sum(1);
```

$$ss := x_1$$
$$x_1 = x_1$$

```
> n:=2:
  ss:=seq(x[i],i=1..n);
  for k from 1 to n do
  Sigma[k](ss) = sigma_to_power_sum(k);
  end do;
```

$$ss := x_1, x_2$$
$$x_1 + x_2 = x_1 + x_2$$
$$x_1\, x_2 = x_1\, x_2$$

```
> n:=3:
  ss:=seq(x[i],i=1..n);
  for k from 1 to n do
  Sigma[k](ss) = sigma_to_power_sum(k);
  end do;
```

$$ss := x_1, x_2, x_3$$
$$x_1 + x_2 + x_3 = x_1 + x_2 + x_3$$

$$x_1 x_2 + x_1 x_3 + x_2 x_3 = x_1 x_2 + x_1 x_3 + x_2 x_3$$

$$x_1 x_2 x_3 = x_1 x_2 x_3$$

```
> n:=4:
  ss:=seq(x[i],i=1..n);
  for k from 1 to n do
  Sigma[k](ss) = sigma_to_power_sum(k);
  end do;
```

$$ss := x_1, x_2, x_3, x_4$$

$$x_1 + x_2 + x_3 + x_4 = x_1 + x_2 + x_3 + x_4$$

$$x_1 x_2 + x_1 x_3 + x_1 x_4 + x_2 x_3 + x_2 x_4 + x_3 x_4 = x_1 x_2 + x_1 x_3 + x_1 x_4 + x_2 x_3 + x_2 x_4 + x_3 x_4$$

$$x_1 x_2 x_3 + x_1 x_2 x_4 + x_1 x_3 x_4 + x_2 x_3 x_4 = x_1 x_2 x_3 + x_1 x_2 x_4 + x_1 x_3 x_4 + x_2 x_3 x_4$$

$$x_1 x_2 x_3 x_4 = x_1 x_2 x_3 x_4$$

```
> n:=5:
  ss:=seq(x[i],i=1..n);
  for k from 1 to n do
  Sigma[k](ss) = sigma_to_power_sum(k);
  end do;
```

$$ss := x_1, x_2, x_3, x_4, x_5$$

$$x_1 + x_2 + x_3 + x_4 + x_5 = x_1 + x_2 + x_3 + x_4 + x_5$$

$$x_1 x_2 + x_1 x_3 + x_1 x_4 + x_1 x_5 + x_2 x_3 + x_2 x_4 + x_2 x_5 + x_3 x_4 + x_3 x_5 + x_4 x_5 =$$
$$x_1 x_2 + x_1 x_3 + x_1 x_4 + x_1 x_5 + x_2 x_3 + x_2 x_4 + x_2 x_5 + x_3 x_4 + x_3 x_5 + x_4 x_5$$

$$x_1 x_2 x_3 + x_1 x_2 x_4 + x_1 x_2 x_5 + x_1 x_3 x_4 + x_1 x_3 x_5 + x_1 x_4 x_5 + x_2 x_3 x_4 + x_2 x_3 x_5 + x_2 x_4 x_5$$
$$+ x_3 x_4 x_5 = x_1 x_2 x_3 + x_1 x_2 x_4 + x_1 x_2 x_5 + x_1 x_3 x_4 + x_1 x_3 x_5 + x_1 x_4 x_5 + x_2 x_3 x_4 + x_2 x_3 x_5$$
$$+ x_2 x_4 x_5 + x_3 x_4 x_5$$

$$x_1 x_2 x_3 x_4 + x_1 x_2 x_3 x_5 + x_1 x_2 x_4 x_5 + x_1 x_3 x_4 x_5 + x_2 x_3 x_4 x_5 =$$
$$x_1 x_2 x_3 x_4 + x_1 x_2 x_3 x_5 + x_1 x_2 x_4 x_5 + x_1 x_3 x_4 x_5 + x_2 x_3 x_4 x_5$$

$$x_1 x_2 x_3 x_4 x_5 = x_1 x_2 x_3 x_4 x_5$$

```
>
```

Example 4: A procedure that expresses sigma[k] polynomials in variables x[1], ..., x[n], where 1 <= k <= n, in terms of the power sums s[k] is called sigma_to_powersum:

```
>
> n:=7:ss:=seq(x[i],i=1..n);
  for k from 1 to n do
      sigma[k]=sigma_to_powersum[k](ss);
  end do;
```

$$ss := x_1, x_2, x_3, x_4, x_5, x_6, x_7$$

$$\sigma_1 = s_1$$

$$\sigma_2 = -\frac{1}{2} s_2 + \frac{1}{2} s_1^{\ 2}$$

$$\sigma_3 = \frac{1}{3} s_3 - \frac{1}{2} s_1 s_2 + \frac{1}{6} s_1^{\ 3}$$

$$\sigma_4 = -\frac{1}{4} s_4 + \frac{1}{3} s_1 s_3 + \frac{1}{8} s_2^{\ 2} - \frac{1}{4} s_2 s_1^{\ 2} + \frac{1}{24} s_1^{\ 4}$$

$$\sigma_5 = \frac{1}{5} s_5 - \frac{1}{4} s_1 s_4 - \frac{1}{6} s_3 s_2 + \frac{1}{6} s_3 s_1^{\ 2} + \frac{1}{8} s_1 s_2^{\ 2} - \frac{1}{12} s_2 s_1^{\ 3} + \frac{1}{120} s_1^{\ 5}$$

$$\sigma_6 = -\frac{1}{6} s_6 + \frac{1}{5} s_1 s_5 + \frac{1}{8} s_4 s_2 - \frac{1}{8} s_4 s_1^{\ 2} + \frac{1}{18} s_3^{\ 2} - \frac{1}{6} s_3 s_1 s_2 + \frac{1}{18} s_3 s_1^{\ 3} - \frac{1}{48} s_2^{\ 3} + \frac{1}{16} s_2^{\ 2} s_1^{\ 2} - \frac{1}{48} s_2 s_1^{\ 4} + \frac{1}{720} s_1^{\ 6}$$

$$\sigma_7 = \frac{1}{8} s_4 s_1 s_2 - \frac{1}{12} s_3 s_2 s_1^{\ 2} - \frac{1}{6} s_1 s_6 - \frac{1}{10} s_5 s_2 + \frac{1}{10} s_5 s_1^{\ 2} - \frac{1}{12} s_4 s_3 - \frac{1}{24} s_4 s_1^{\ 3} + \frac{1}{18} s_1 s_3^{\ 2} + \frac{1}{24} s_3 s_2^{\ 2} + \frac{1}{7} s_7 + \frac{1}{5040} s_1^{\ 7} + \frac{1}{72} s_3 s_1^{\ 4} - \frac{1}{48} s_1 s_2^{\ 3} + \frac{1}{48} s_2^{\ 2} s_1^{\ 3} - \frac{1}{240} s_2 s_1^{\ 5}$$

```
>
```

We can verify that the above is correct by converting each side to polynomials in x[1], x[2],..., x[n]:

```
> n:=4:ss:=seq(x[i],i=1..n);
```

$$ss := x_1, x_2, x_3, x_4$$

```
> for k from 1 to n do

  evalb(Sigma[k](ss)=expand(simplify(subs({seq(s[i]=powersum[i](s
  s),i=1..k)},convert(sigma_to_powersum[k](ss),`global`))))); 
  end do;
```

$$true$$
$$true$$
$$true$$
$$true$$

```
>
```

Example 5: Thus, we should be able now to express any symmetric polynomial f in terms of the power sums.

```
> f:=expand((Sigma[3](x[1],x[2],x[3],x[4]))^2-Sigma[2](x[1],x[2],
  x[3],x[4])+3*Sigma[1](x[1],x[2],x[3],x[4])-
```

```
Sigma[1](x[1],x[2],x[3],x[4])*Sigma[4](x[1],x[2],x[3],x[4]));
```

$$f := x_1^2 x_2^2 x_3^2 + 2 x_1^2 x_2^2 x_3 x_4 + 2 x_1^2 x_2 x_3^2 x_4 + 2 x_1 x_2^2 x_3^2 x_4 + x_1^2 x_2^2 x_4^2 + 2 x_1^2 x_2 x_4^2 x_3 + 2 x_1 x_2^2 x_4^2 x_3 + x_1^2 x_3^2 x_4^2 + 2 x_1 x_3^2 x_4^2 x_2 + x_2^2 x_3^2 x_4^2 - x_1 x_2 - x_1 x_3 - x_1 x_4 - x_2 x_3 - x_2 x_4 - x_3 x_4 + 3 x_1 + 3 x_2 + 3 x_3 + 3 x_4 - x_2 x_3 x_4 x_1^2 - x_1 x_3 x_4 x_2^2 - x_1 x_2 x_4 x_3^2 - x_1 x_2 x_3 x_4^2$$

Polynomial f is of course symmetric:

```
> isSymmetric(f);
Path set to C:\Maple11\bin.win/libfgbuni.so
    FGb/Maple interface package Version 1.34
    JC Faugere (jcf@calfor.lip6.fr)
    Type ?FGb for documentation
```

$$true$$

Thus, it can be expressed in terms of elementary symmetric polynomials:

```
> f_sigma:=isSymmetric(f,r);
```

$$f\_sigma := -\sigma_4 \sigma_1 + 3 \sigma_1 - \sigma_2 + \sigma_3^2$$

In order to replace each sigma polynomial in the expression for f, we need to make them 'global' since the procedure isSymmetric returns sigma's which are local to it:

```
> f_sigma:=convert(f_sigma,`global`);
  ss:=op(indets(f));
```

$$f\_sigma := -\sigma_4 \sigma_1 + 3 \sigma_1 - \sigma_2 + \sigma_3^2$$

$$ss := x_1, x_2, x_3, x_4$$

```
> subs({seq(sigma[j]=sigma_to_powersum[j](ss),j=1..3)},f_sigma);
```

$$-\sigma_4 s_1 + 3 s_1 + \frac{1}{2} s_2 - \frac{1}{2} s_1^2 + \left(\frac{1}{3} s_3 - \frac{1}{2} s_1 s_2 + \frac{1}{6} s_1^3\right)^2$$

```
> expand(%);
```

$$-\sigma_4 s_1 + 3 s_1 + \frac{1}{2} s_2 - \frac{1}{2} s_1^2 + \frac{1}{9} s_3^2 - \frac{1}{3} s_3 s_1 s_2 + \frac{1}{9} s_3 s_1^3 + \frac{1}{4} s_1^2 s_2^2 - \frac{1}{6} s_1^4 s_2 + \frac{1}{36} s_1^6$$

```
>
```

Algorithm used:

None

See Also: SP:-Sigma, SP:-powersum, SP:-isSymmetric, SP:-hpolynom, FGb, Groebner

Last modified: June 19, 2008

Function: SP:-SymmetricGroup - returns elements of the symmetric group S[n]

Calling Sequence:

SymmetricGroup(n);

Parameters:

- n is a positive integer

Output:

- A list of matrices n x n that represent all elements of the symmetric group S[n].

Description:

- Procedure 'SymmetricGroup' returns a list with n! square n x n matrices that represent elements of S[n].
- This procedure has a remember table stored in the library.
- This procedure is used later when computing group invariants of SP:-FiniteGroups with the SP:-Reynolds operator.
- References:
  [1] D. Cox, J. Little, D. O'Shea: "Ideals, varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra", Third Edition, Springer, New York, 2007
  [2] B. Sturmfels: "Algorithms in Invariant Theory", Texts and Monographs in Symbolic Computation, Springer-Verlag/Wien, 1993

Examples:

```
> restart:with(SP):
Remember table of SymmetricGroup has been read and assigned
Remember table of AlternatingGroup has been read and assigned
Remember table of Reynolds has been read and assigned
Remember table of FiniteGroups has been read and assigned
Remember table of generateGinvariants has been read and assigned
```

Example 1: Elements of various permutation groups:

```
> SymmetricGroup(1);
```

$$[[1]]$$

```
> SymmetricGroup(2);
```

$$\left[\begin{bmatrix}1&0\\0&1\end{bmatrix},\begin{bmatrix}0&1\\1&0\end{bmatrix}\right]$$

```
> SymmetricGroup(3);
```

$$\left[\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix},\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix},\begin{bmatrix}0&1&0\\1&0&0\\0&0&1\end{bmatrix},\begin{bmatrix}0&0&1\\1&0&0\\0&1&0\end{bmatrix},\begin{bmatrix}0&1&0\\0&0&1\\1&0&0\end{bmatrix},\begin{bmatrix}0&0&1\\0&1&0\\1&0&0\end{bmatrix}\right]$$

```
> SymmetricGroup(4);
```

$$\left[\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix},\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix},\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{bmatrix},\begin{bmatrix}1&0&0&0\\0&0&0&1\\0&1&0&0\\0&0&1&0\end{bmatrix},\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&0&0&1\\0&1&0&0\end{bmatrix},\right.$$

$$\begin{bmatrix}1&0&0&0\\0&0&0&1\\0&0&1&0\\0&1&0&0\end{bmatrix},\begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix},\begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix},\begin{bmatrix}0&0&1&0\\1&0&0&0\\0&1&0&0\\0&0&0&1\end{bmatrix},\begin{bmatrix}0&0&0&1\\1&0&0&0\\0&1&0&0\\0&0&1&0\end{bmatrix},\begin{bmatrix}0&0&1&0\\1&0&0&0\\0&0&0&1\\0&1&0&0\end{bmatrix},$$

$$\begin{bmatrix}0&0&0&1\\1&0&0&0\\0&0&1&0\\0&1&0&0\end{bmatrix},\begin{bmatrix}0&1&0&0\\0&0&1&0\\1&0&0&0\\0&0&0&1\end{bmatrix},\begin{bmatrix}0&1&0&0\\0&0&0&1\\1&0&0&0\\0&0&1&0\end{bmatrix},\begin{bmatrix}0&0&1&0\\0&1&0&0\\1&0&0&0\\0&0&0&1\end{bmatrix},\begin{bmatrix}0&0&0&1\\0&1&0&0\\1&0&0&0\\0&0&1&0\end{bmatrix},\begin{bmatrix}0&0&1&0\\0&0&0&1\\1&0&0&0\\0&1&0&0\end{bmatrix},$$

$$\begin{bmatrix}0&0&0&1\\0&0&1&0\\1&0&0&0\\0&1&0&0\end{bmatrix},\begin{bmatrix}0&1&0&0\\0&0&1&0\\0&0&0&1\\1&0&0&0\end{bmatrix},\begin{bmatrix}0&1&0&0\\0&0&0&1\\0&0&1&0\\1&0&0&0\end{bmatrix},\begin{bmatrix}0&0&1&0\\0&1&0&0\\0&0&0&1\\1&0&0&0\end{bmatrix},\begin{bmatrix}0&0&0&1\\0&1&0&0\\0&0&1&0\\1&0&0&0\end{bmatrix},\begin{bmatrix}0&0&1&0\\0&0&0&1\\0&1&0&0\\1&0&0&0\end{bmatrix},$$

$$\left.\begin{bmatrix}0&0&0&1\\0&0&1&0\\0&1&0&0\\1&0&0&0\end{bmatrix}\right]$$

```
>
```

Example 2: Let's see an action of the permutation group S[3] on a polynomial:

```
> S3:=SymmetricGroup(3):
> f:=x^3-y^2+x*y+z;
```

$$f := x^3 - y^2 + x\,y + z$$

```
> for A in S3 do MatrixAction(A,f,[x,y,z]) end do;
```

$$x^3 - y^2 + x\,y + z$$

$$x^3 - z^2 + x\,z + y$$

$$y^3 - x^2 + x\,y + z$$

$$z^3 - x^2 + x\,z + y$$

$$y^3 - z^2 + y\,z + x$$

$$z^3 - y^2 + y\,z + x$$

```
>
>
```

Algorithm used:

None

See Also: SP:-Sigma, SP:-FiniteGroups, SP:-AlternatingGroup, SP:-MatrixAction, SP:-Reynolds, Groebner

Last modified: June 19, 2008

Help For:

'BIGEBRA 11' - A Maple Package for Clifford and Grassmann Hopf Gebras

Version 1.01 -- designed for Maple 11

(*) Department of Mathematics, Box 5054
Tennessee Technological University
Cookeville, TN 38501 USA
Phone: (931) 372-3569, Fax: (931) 372-6353
E-mail: rablamowicz@tntech.edu
URL: http://math.tntech.edu/rafal/ (home of this package)

(§) Fachbereich Physik, Universit"at Konstanz
Fach M678, 78457 Konstanz, Germany
Phone: +49 7531 883786, FAX: +49 7531 884266
E-mail: Bertfried.Fauser@uni-konstanz.de
URL: http://clifford.physik.uni-konstanz.de/~fauser/

Last revised: December 20, 2007 (BF. & RA)

Calling Sequence:

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

Description:

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

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

Literature:

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

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

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

You can increase the verbosity level of Bigebra setting **infolevel[Bigebra]=3** or higher.

```
> restart:with(Clifford):infolevel(Bigebra)=3:with(Bigebra):
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

To initialize the Clifford coproduct type:

```
> dim_V:=2:
  BI:=linalg[matrix](dim_V,dim_V,[a,b,c,d]);
  make_BI_Id();
```

$$BI := \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

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

$$(Id \,\&t\, Id) + a\,(e1 \,\&t\, e1) + c\,(e2 \,\&t\, e1) + b\,(e1 \,\&t\, e2) + d\,(e2 \,\&t\, e2) + (c\,b - d\,a)\,(e1we2 \,\&t\, e1we2)$$

BI is the dim_V x dim_V matrix of the co-scalarproduct on co-one-vectors, from which the Clifford co-product `&cco` is derived by Rota-Stein co-cliffordization, [2,7,8]. The tensor product `&t` is already defined and ready to use:

```
> &t(e1,&t(e2,e3),e4);  ## associativity, i.e. drop 'parentheses'
```

$$\&t(e1, e2, e3, e4)$$

```
> &t(a*e1+sin(theta)*e3,b*e3-1/x*e1);  ## multilinearity
```

$$a\,b\,(e1 \,\&t\, e3) - \frac{a\,(e1 \,\&t\, e1)}{x} + \sin(\theta)\,b\,(e3 \,\&t\, e3) - \frac{\sin(\theta)\,(e3 \,\&t\, e1)}{x}$$

```
>
```

Alphabetic listing of available procedures in 'BIGEBRA':

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

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

New Types in 'BIGEBRA':

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

See Also: Clifford:-setup, Clifford:-version, Bigebra:-VERSION

Last modified: December 20, 2007/BF/RA.

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

Calling Sequence:

VERSION()

Parameters:

- none.

Output:

- none.

Description:

- VERSION() displays information about the Bigebra package

Examples:

```
> restart:with(Bigebra):

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

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

```
> VERSION();

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

  (§) Department of Mathematics, Box 5054
      Tennessee Technological University
      Cookeville, TN 38505, U.S.A.
      Email: rablamowicz@tntech.edu
      URL:   http://math.tntech.edu/rafal/
  (*) Universit"at Konstanz
      Fachbereich Physik, Fach M678
      78457 Konstanz, Germany
      Email: Bertfried.Fauser@uni-konstanz.de
      URL:   http://clifford.physik.uni-konstanz.de/~fauser/

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

Copyright (c) Rafal Ablamowicz, Bertfried Fauser 1999-2008.
              All rights reserved. See also > ?Bigebra[help]

 BUG-REPORTS to Bertfried Fauser
<==============================================================>
```

[>

[>

See Also: Bigebra helppage

Last modified: December 20, 2007 /BF/RA.

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

Calling Sequence:

t1 = &cco(p1,i)
t1 = &cco(c1)

Parameters:

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

Output:

- t1 : a tensor polynom

Global variables:

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

WARNING:

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

Description:

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

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

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

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

TO DO:

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

Examples:

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

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

```
> dim_V:=2:
  BI:=linalg[matrix](dim_V,dim_V,[a,b,c,d]): #co-scalarproduct
  m1:=make_BI_Id():                          #remember this
  result in m1
```

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

The Clifford co-product of Clifford polynomials needs no slot index:

```
> c1:= &cco(e1);
  c2:= &cco(&t(e1),1); # the same, &t(e1) = e1
  c3:= &cco(&t(e1),2); # the same, the slot is ignored here
```

$$c1 := (Id \,\&t\, e1) - b\,(e1 \,\&t\, e1we2) - d\,(e2 \,\&t\, e1we2) + (e1 \,\&t\, Id) + c\,(e1we2 \,\&t\, e1) + d\,(e1we2 \,\&t\, e2)$$

$$c2 := (Id \,\&t\, e1) - b\,(e1 \,\&t\, e1we2) - d\,(e2 \,\&t\, e1we2) + (e1 \,\&t\, Id) + c\,(e1we2 \,\&t\, e1) + d\,(e1we2 \,\&t\, e2)$$

$$c3 := (Id \,\&t\, e1) - b\,(e1 \,\&t\, e1we2) - d\,(e2 \,\&t\, e1we2) + (e1 \,\&t\, Id) + c\,(e1we2 \,\&t\, e1) + d\,(e1we2 \,\&t\, e2)$$

Reduction to the Grassmann co-product is obtained by letting all parameters of the co-scalarproduct go to zero:

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

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

$$Id \,\&t\, Id$$
$$Id \,\&t\, Id$$
$$(Id \,\&t\, e1) + (e1 \,\&t\, Id)$$
$$(Id \,\&t\, e1) + (e1 \,\&t\, Id)$$

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

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

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

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

$$c4 := (Id \,\&t\, e1) - b\,(e1 \,\&t\, e1we2) - (e2 \,\&t\, e1we2) + (e1 \,\&t\, Id) + b\,(e1we2 \,\&t\, e1) + (e1we2 \,\&t\, e2)$$
$$c5 := (Id \,\&t\, e1) - b\,(e1 \,\&t\, e1we2) - (e2 \,\&t\, e1we2) + (e1 \,\&t\, Id) + b\,(e1we2 \,\&t\, e1) + (e1we2 \,\&t\, e2)$$

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

$$(Id \,\&t\, Id) + (e1 \,\&t\, e1) + b\,(e2 \,\&t\, e1) + b\,(e1 \,\&t\, e2) + (e2 \,\&t\, e2) + (b^2 - 1)\,(e1we2 \,\&t\, e1we2)$$
$$(Id \,\&t\, Id) + (e1 \,\&t\, e1) + b\,(e2 \,\&t\, e1) + b\,(e1 \,\&t\, e2) + (e2 \,\&t\, e2) + (b^2 - 1)\,(e1we2 \,\&t\, e1we2)$$

Reduction to the Grassmann co-product as a test:

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

$$Id \,\&t\, Id$$
$$Id \,\&t\, Id$$
$$(Id \,\&t\, e1) + (e1 \,\&t\, Id)$$
$$(Id \,\&t\, e1) + (e1 \,\&t\, Id)$$

Co-associativitiy of the Clifford co-product:

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

$$-\&t(e1, e1we2, e1we2) + \&t(Id, e1, Id) - \&t(e2, e1, e2) + \&t(e2, e2, e1) + \&t(e1, Id, Id) + \&t(Id, Id, e1) - b\,\&t(Id, e1, e1we2) - \&t(Id, e2, e1we2) + b\,\&t(e1we2, e1, Id) + \&t(e1, e1, e1) + \&t(e1we2, Id, e2) + \&t(e1we2, e2, Id) + \&t(e1we2, e1, e1we2) - b\,\&t(e1, Id, e1we2) - b^2\,\&t(e1we2, e1, e1we2) - \&t(e2, Id, e1we2) - b\,\&t(e1, e1we2, Id) + (b^2 - 1)\,\&t(e1we2, e1we2, e1) + \&t(e1, e2, e2) - \&t(e2, e1we2, Id) + 2\,b\,\&t(e1, e2, e1) + \&t(Id, e1we2, e2) + b^2\,\&t(e1, e1we2, e1we2) + b\,\&t(Id, e1we2, e1) + b\,\&t(e1we2, Id, e1)$$

$$\&t(Id, e1, Id) - \&t(e2, e1, e2) + \&t(e2, e2, e1) + \&t(e1, Id, Id) + \&t(Id, Id, e1) - b\,\&t(Id, e1, e1we2) - \&t(Id, e2, e1we2) + b\,\&t(e1we2, e1, Id) + b^2\,\&t(e1we2, e1we2, e1) + \&t(e1, e1, e1) + (b^2 - 1)\,\&t(e1, e1we2, e1we2) + \&t(e1we2, Id, e2) + \&t(e1we2, e2, Id) + \&t(e1we2, e1, e1we2) - b\,\&t(e1, Id, e1we2) - b^2\,\&t(e1we2, e1, e1we2) - \&t(e2, Id, e1we2) - b\,\&t(e1, e1we2, Id) - \&t(e1we2, e1we2, e1) + \&t(e1, e2, e2) - \&t(e2, e1we2, Id) + 2\,b\,\&t(e1, e2, e1) + \&t(Id, e1we2, e2) + b\,\&t(Id, e1we2, e1) + b\,\&t(e1we2, Id, e1)$$

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

$$0$$

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

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

$$res1 := \&t(Id, e1, e2) - b\,\&t(e1, e1we2, e2) - \&t(e2, e1we2, e2) + \&t(e1, Id, e2) + b\,\&t(e1we2, e1, e2) + \&t(e1we2, e2, e2)$$

$$res2 := \&t(e1, Id, e2) + \&t(e1, e1, e1we2) + b\,\&t(e1, e2, e1we2) + \&t(e1, e2, Id) - \&t(e1, e1we2, e1) - b\,\&t(e1, e1we2, e2)$$

$$res1 - res2 = 0 \text{ is } , false$$

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

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

```
> printf("Worksheet took %f seconds to compute on AMD Athlon
  2700+ 1GB RAM machine\n",time()-bench);
Worksheet took 2.438000 seconds to compute on AMD Athlon 2700+ 1GB RAM machine
>
```

See Also: Bigebra:-`&gco`, Bigebra:-`&t`, Bigebra:-drop_t

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-`&gco_d` - dotted Grassmann co-product for a different filtration

Calling Sequence:

t2 := &gco_d(t1,i)
t2 := &gco_d(c1)

Parameters:

- t1 : tensor polynoms
- i : tensor slot to be acted on
- c1 : Clifford polynom

Output:

- t1 : a tensor polynom

Global:

- for the transition to the regular wedge basis and back to the dotted basis (both represented as eawebw...) the two global matrices F and FT are used.

Description:

- The dotted Grassmann co-product is isomorphic to the regular Grassmann co-product on the dotted wedge basis. The function &gco_d(t1,i) computes this product using the original Grassmann co-product but w.r.t. the **undotted** basis. It is hence the counter part to the function Cliplus:-dwedge which computes the dotted wedge product in the **undotted** basis. The dotted and undotted bases arise from different filtrations of the underlying Grassmann algebra. **As Grassmann algebras they are isomorphic, but they are not isomorphic as Hopf algebras!**
- The function &gco_d needs the Cliplus and it will load it automatically if it was not done previously.
- This functionality is simply gained by wrapping the original Grassmann co-product and using internally the two functions `convert/wedge_to_dwedge` and `convert/dwedge_to_wedge` from the Cliplus package.
- If F and FT are antisymmetric arrays which are mutually transposed to each other (negative of each other) this mapping is an isomorphism (F and FT need not be non-singular !). Hence the dotted Grassmann co-product can be computed in the undotted basis by transforming back and forth.
- From a physical point of view, it is possible to consider the undotted basis as a fermionic time-ordered product and the dotted basis as a fermionic normal-ordered product. Hence, the

dotted Grassmann co-product employs the co-product of normal-ordered fields in the time-ordered basis, see [1].

- It should be noted that the loop tangle 'product \circ coproduct' works out completely differently if one mixes products and co-products for the same algebra with different filtration (ordering).

- **References:**

[1] B. Fauser, On the Hopf algebraic origin of Wick normal ordering, J. Math. Phys.: Gen. Math. 34, 2001:105-115

Examples:

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

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

Define the dimension of the vector space V under consideration to be 3, and define F and FT

```
> dim_V:=3:
  F:=array(antisymmetric,1..dim_V,1..dim_V):
  F=evalm(F);
  FT:=evalm(-1*F);
  w_bas:=cbasis(dim_V); ## the wedge basis
```

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

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

$$w\_bas := [Id, e1, e2, e3, e1we2, e1we3, e2we3, e1we2we3]$$

Now we invoke for the first time the dotted Grassmann co-product which **needs** also load the Cliplus package:

```
> with(Cliplus):
  &gco_d(e1we2);
```

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

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

Note that Cliplus was loaded. As the Grassmann co-product, &gco_d can act on tensors in the i-th slot. For example, let's act on the second tensor slot of the tensor &t(e1,e2we3,e3) occupied by e2we3:

```
> &gco_d(&t(e1,e2we3,e3),2);
```

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

$$+ \&t(e1, e2we3, Id, e3)$$

Now let us show how the co-product acts on dotted and undotted elements

```
> w_p1:=e1we2;                #selection of an element in undotted
  basis
  w_c1:=&gco_d(w_p1);         #action of &gco_d on undotted element
  e1we2
  d_p1:=dwedge[F](e1,e2);     #transformation of e1we2 to dotted
  basis
  d_p2:=convert(w_p1,wedge_to_dwedge,F); #another way to
  accomplish the same transformation
  d_c1:=&gco_d(d_p1);         #action of &gco_d on the image of
  e1we2 in dotted basis
```

$$w\_p1 := e1we2$$

$$w\_c1 := (Id \,\&t\, e1we2) + F_{1,2}\,(Id \,\&t\, Id) + (e1 \,\&t\, e2) - (e2 \,\&t\, e1) + (e1we2 \,\&t\, Id)$$

$$d\_p1 := e1we2 + F_{1,2}\,Id$$

$$d\_p2 := e1we2 + F_{1,2}\,Id$$

$$d\_c1 := (Id \,\&t\, e1we2) + 2\,F_{1,2}\,(Id \,\&t\, Id) + (e1 \,\&t\, e2) - (e2 \,\&t\, e1) + (e1we2 \,\&t\, Id)$$

The following examples compose the dotted co-product with dotted and undotted wedge (acting on a wedge basis!!). First, let's show a dotted basis:

```
> Grassmann_basis:=cbasis(3);
  #Grassmann un-dotted basis
  dotted_basis:=map(convert,Grassmann_basis,wedge_to_dwedge,F);
  #dotted basis
```

$$Grassmann\_basis := [Id, e1, e2, e3, e1we2, e1we3, e2we3, e1we2we3]$$

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

We will use the following notation for the dotted basis, e.g., e1We2 = e1we2+F[1,2]*Id, etc.:

```
> S:={e1we2+F[1,2]*Id=e1We2,e1we3+F[1,3]*Id=e1We3,e2we3+F[2,3]*Id
  =e2We3,
  e1we2we3+F[2,3]*e1-F[1,3]*e2+F[1,2]*e3=e1We2We3}:
  subs(S,dotted_basis); #dotted basis in shorter (dotted wedge)
  notation
```

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

Then, we compose dotted co-product with undotted and dotted wedge:

```
> for i in dotted_basis do
  d_p1:=&gco_d(i):
  drop_t(&map(d_p1,1,dwedge[F]));
```

```
`action_dwedge_o_&gco_d` =
2^maxgrade(%)*subs(S,%/2^maxgrade(%));
d_p2:=convert(%%,dwedge_to_wedge,-F);
print(`****************`);
od;
```

$$d\_p1 := Id \,\&t\, Id$$

$$Id$$

$$action\_dwedge\_o\_\&gco\_d = Id$$

$$d\_p2 := Id$$

$$**********$$

$$d\_p1 := (Id \,\&t\, e1) + (e1 \,\&t\, Id)$$

$$2\, e1$$

$$action\_dwedge\_o\_\&gco\_d = 2\, e1$$

$$d\_p2 := 2\, e1$$

$$**********$$

$$d\_p1 := (Id \,\&t\, e2) + (e2 \,\&t\, Id)$$

$$2\, e2$$

$$action\_dwedge\_o\_\&gco\_d = 2\, e2$$

$$d\_p2 := 2\, e2$$

$$**********$$

$$d\_p1 := (Id \,\&t\, e3) + (e3 \,\&t\, Id)$$

$$2\, e3$$

$$action\_dwedge\_o\_\&gco\_d = 2\, e3$$

$$d\_p2 := 2\, e3$$

$$**********$$

$$d\_p1 := (Id \,\&t\, e1we2) + 2\, F_{1,2}\, (Id \,\&t\, Id) + (e1 \,\&t\, e2) - (e2 \,\&t\, e1) + (e1we2 \,\&t\, Id)$$

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

$$action\_dwedge\_o\_\&gco\_d = 4\, e1We2$$

$$d\_p2 := 4\, e1we2$$

$$**********$$

$$d\_p1 := (Id \,\&t\, e1we3) + 2\, F_{1,3}\, (Id \,\&t\, Id) + (e1 \,\&t\, e3) - (e3 \,\&t\, e1) + (e1we3 \,\&t\, Id)$$

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

$$action\_dwedge\_o\_\&gco\_d = 4\, e1We3$$

$$d\_p2 := 4\, e1we3$$

$$**********$$

$$d\_p1 := (Id \,\&t\, e2we3) + 2\, F_{2,3}\, (Id \,\&t\, Id) + (e2 \,\&t\, e3) - (e3 \,\&t\, e2) + (e2we3 \,\&t\, Id)$$

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

$$action\_dwedge\_o\_\&gco\_d = 4\ e2We3$$

$$d\_p2 := 4\ e2we3$$

****

$$d\_p1 := (Id\ \&t\ e1we2we3) + 2\ F_{2,3}\ (Id\ \&t\ e1) - 2\ F_{1,3}\ (Id\ \&t\ e2) + 2\ F_{1,2}\ (Id\ \&t\ e3)$$
$$+ (e1\ \&t\ e2we3) + 2\ F_{2,3}\ (e1\ \&t\ Id) - (e2\ \&t\ e1we3) - 2\ F_{1,3}\ (e2\ \&t\ Id) + (e1we2\ \&t\ e3)$$
$$+ (e3\ \&t\ e1we2) + 2\ F_{1,2}\ (e3\ \&t\ Id) - (e1we3\ \&t\ e2) + (e2we3\ \&t\ e1) + (e1we2we3\ \&t\ Id)$$

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

$$action\_dwedge\_o\_\&gco\_d = 8\ e1We2We3$$

$$d\_p2 := 8\ e1we2we3$$

****

Thus, the above shows that **(wedge o Grassmann co-product)(x) = 2^(grade of x) * x** for any x in the Grassmann basis (Grassmann_basis) shown above and that the same is true, namely, **(dotted wedge o dotted Grassmann co-product)(y) = 2^(grade of y) * y** for any y in the dotted wedge basis (dotted_basis) shown above.

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM\n",time()-bench);
```

```
Worksheet took 3.814000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
```

```
>
```

See Also: Bigebra:-`&gco`, Bigebra:-`&cco`, Bigebra:-`&t`, Bigebra:-drop_t, Bigebra:-`&map`, Cliplus:-dwedge

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-`&gco` - Grassmann co-product

Calling Sequence:

t1 := &gco(t2,i)
t1 := &gco(c1)

Parameters:

- t2 : a tensorpolynom (an element of `type/tensorpolynom`) of rank not less than i in each factor
  i : the slot number (first slot is from the left is 1) on which the co-product acts
- c1 is a Clifford polynom (an element of one of these types: `type/clibasmon`, `type/climon`, `type/clipolynom`)

Output:

- t1 : is a tensorpolynom.

WARNING:

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

Description:

- A Grassmann algebra leads naturally to a multi-vector space /\V. This space has a dual which we call (\/V)^* = \/V^* . There is a natural pairing between one-vectors and co-one-vectors which can be extended to a graded scalar valued (target/domain is the ring k) action of co-multi-vectors on multivectors called pairing: < A, B> : \/V^* x /\V --> k. Let us denote the Grassmann product of co-multi-vectors by \/V^*, i.e using a &v (vee)-product. One obtains by categorical duality (i.e. by reversing arrows in commutative diagrams) the coproduct Δ . For two-vectors this reads:

$$
\begin{aligned}
< a1 \vee a2 \mid b > &= < a1 \,\&t\, a2 \mid \Delta(b) > \\
&= < a1 \,\&t\, a2 \mid \sum_i (b)\_(1i) \,\&t\, (b)\_(2i) > \\
&= \sum_i < a1 \mid (b)\_(1i) > \; < a2 \mid (b)\_(2i) >
\end{aligned}
$$

  Since the co-vectors a1 and a2 are arbitrary co-multi-vectors, this defines the coproduct on an arbitrary multi-vector Grassmann element b in /\V. If a1 is a co-one-multivector, this turns out to be the Laplace row expansion of the pairing, see [8,3]. The same consideration can be done for columns, i.e. moving the wedge /\ from right to left in the pairing to generate a Grassmann co-product on co-multi-vectors. Since we denote currently vectors and co-vectors by the *same* vector symbol `e`, the user has to take care of the fact in which slot of a tensor a vector or co-vector resides, see EV.

- Expanded in our basis, the above formalism leads to a combinatorial formula using split-sums and shuffles which are internally computed in BIGEBRA, [4]. From this construction, one concludes that the Grassmann co-product enjoys the following properties:
- The Grassmann co-product is associative, (Δ &t Id) Δ = (Id &t Δ) Δ.
- The Grassmann co-product is graded co-commutative Δ = τΔ, where τ is the graded switch.
- The Grassmann co-product is linear.
- Together with the Grassmann wedge product one proves this structure to be a Grassmann Hopf gebra, which posesses an antipode.
- The Grassmann Hopf gebra is an bi-augmented bi-connected Hopf algebra, which is also called a non interacting Hopf gebra [3].

Examples:

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

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

Some examples of Grassmann co-products of Clifford polynomials:

```
> &gco(e1);
  &gco(&t(e1),1); # the same, since
  e1 = drop_t( &t(e1) );
```

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

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

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

$$e1 = e1$$

```
> &gco(e1we2);
  &gco(a*e3);
  &gco(e1we2+a*e3);
```

$$(Id\ \&t\ e1we2) + (e1\ \&t\ e2) - (e2\ \&t\ e1) + (e1we2\ \&t\ Id)$$

$$a\,(Id\ \&t\ e3) + a\,(e3\ \&t\ Id)$$

$$(Id\ \&t\ e1we2) + (e1\ \&t\ e2) - (e2\ \&t\ e1) + (e1we2\ \&t\ Id) + a\,(Id\ \&t\ e3) + a\,(e3\ \&t\ Id)$$

Acting on tensor slots:

```
> &gco(&t(e1,e2),1);
  &gco(&t(e1,e2),2);
```

$$\&t(Id, e1, e2) + \&t(e1, Id, e2)$$

$$\&t(e1, Id, e2) + \&t(e1, e2, Id)$$

```
> &gco(Id);
```

```
&gco(%,1);
```

$$Id \,\&t\, Id$$

$$\&t(Id, Id, Id)$$

```
> &gco(a*&t(e1,e2)+b*&t(e3,e4),1);
```

$$a\,\&t(Id, e1, e2) + a\,\&t(e1, Id, e2) + b\,\&t(Id, e3, e4) + b\,\&t(e3, Id, e4)$$

Checking co-associativity:

```
> &gco(&gco(&t(e1we2),1),1);
  &gco(&gco(&t(e1we2),1),2);
  evalb(%-%%=0);
```

$$\&t(Id, Id, e1we2) + \&t(Id, e1, e2) + \&t(e1, Id, e2) - \&t(Id, e2, e1) - \&t(e2, Id, e1) + \&t(Id, e1we2, Id) + \&t(e1, e2, Id) - \&t(e2, e1, Id) + \&t(e1we2, Id, Id)$$

$$\&t(Id, Id, e1we2) + \&t(Id, e1, e2) + \&t(e1, Id, e2) - \&t(Id, e2, e1) - \&t(e2, Id, e1) + \&t(Id, e1we2, Id) + \&t(e1, e2, Id) - \&t(e2, e1, Id) + \&t(e1we2, Id, Id)$$

$$true$$

Checking graded co-commutativity:

```
> g1:=&gco(e1we2+e1we2we3);
  g2:=gswitch(g1,1);
```

$$g1 := (Id \,\&t\, e1we2) + (e1 \,\&t\, e2) - (e2 \,\&t\, e1) + (e1we2 \,\&t\, Id) + (Id \,\&t\, e1we2we3) + (e1 \,\&t\, e2we3) - (e2 \,\&t\, e1we3) + (e1we2 \,\&t\, e3) + (e3 \,\&t\, e1we2) - (e1we3 \,\&t\, e2) + (e2we3 \,\&t\, e1) + (e1we2we3 \,\&t\, Id)$$

$$g2 := (Id \,\&t\, e1we2) + (e1 \,\&t\, e2) - (e2 \,\&t\, e1) + (e1we2 \,\&t\, Id) + (Id \,\&t\, e1we2we3) + (e1 \,\&t\, e2we3) - (e2 \,\&t\, e1we3) + (e1we2 \,\&t\, e3) + (e3 \,\&t\, e1we2) - (e1we3 \,\&t\, e2) + (e2we3 \,\&t\, e1) + (e1we2we3 \,\&t\, Id)$$

```
> evalb(%-%%=0);
```

$$true$$

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

```
> res1:=&gco(&t(e1,e2),1);
  res2:=&gco(&t(e1,e2),2);
  printf("res1 - res2 =0 is %s !",evalb(tcollect(res1-res2)=0));
```

$$res1 := \&t(Id, e1, e2) + \&t(e1, Id, e2)$$

$$res2 := \&t(e1, Id, e2) + \&t(e1, e2, Id)$$

```
res1 - res2 =0 is false !
```

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

```
> &gco(&t(e1,e2),3);
```

```
Error, (in Bigebra:-&gco) invalid subscript selector
```

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM\n",time()-bench);
```

```
Worksheet took 0.064000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
```

```
>
```

See Also: Bigebra:-`&cco`, Bigebra:-`&t`, Bigebra:-drop_t, Bigebra:-`&map`

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-`&gco_pl` - Graßmann-Plücker co-product for hyperplanes

Calling Sequence:

t1 := &gco_pl(t2,i)
t1 := &gco_pl(c1)

Parameters:

- t2 : a tensorpolynom (an element of `type/tensorpolynom`) of rank not less than i in each factor
  i : the slot number (first slot is from the left is 1) on which the co-product acts
- c1 is a Clifford polynom (an element of one of these types: `type/clibasmon`, `type/climon`, `type/clipolynom`)

Output:

- t1 : is a tensorpolynom.

WARNING:

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

Description:

- In analogy with the Grassmann co-product &gco, which is dual to the wedge, the Grassmann-Plücker co-product is dual to the meet or vee product &v denoted \/. Using a natural pairing and the same construction as for the Grassmann co-product we obtain a co-product which acts on hyperplanes. These hyperplanes are parameterized in Plücker coordinates which is the origin of the name Grassmann-Plücker co-product.
- Note that due to the Plücker coordinatization we need dim_V to be set to the dimension of the generating space V.
- The Grassmann-Plücker co-product is expected to behave quite similarly to the Grassmann product itself. However, due to the Plücker coordinatization a further duality is involved which yields some awkward signs and some unconventional outcomes.

Examples:

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

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

Some examples of Grassmann-Plücker co-products of Clifford monoms *describing hyperplanes*:

```
> dim_V:=3:
  &gco_pl(e1);
  &gco_pl(&t(e1),1); # the same
```

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

$$(e1\ \&t\ e1we2we3) - (e1we2\ \&t\ e1we3) + (e1we3\ \&t\ e1we2) + (e1we2we3\ \&t\ e1)$$

$$(e1\ \&t\ e1we2we3) - (e1we2\ \&t\ e1we3) + (e1we3\ \&t\ e1we2) + (e1we2we3\ \&t\ e1)$$

Note that the volume element behaves now as the unit under the meet (vee-product) and we obtain thus

```
> &gco_pl(e1we2we3);
```

$$e1we2we3\ \&t\ e1we2we3$$

while we find for hyperplanes (i.e. dim_V-1 vectors) the usual behavior:

```
> &gco_pl(e1we2);
  &gco_pl(e1we3);
  &gco_pl(e2we3);
```

$$(e1we2\ \&t\ e1we2we3) + (e1we2we3\ \&t\ e1we2)$$

$$(e1we3\ \&t\ e1we2we3) + (e1we2we3\ \&t\ e1we3)$$

$$(e2we3\ \&t\ e1we2we3) + (e1we2we3\ \&t\ e2we3)$$

Checking co-associativity:

```
> &gco_pl(&gco_pl(&t(e1we2),1),1);
  &gco_pl(&gco_pl(&t(e1we2),1),2);
  evalb(%-%%=0);
```

$$\&t(e1we2, e1we2we3, e1we2we3) + \&t(e1we2we3, e1we2, e1we2we3) + \&t(e1we2we3, e1we2we3, e1we2)$$

$$\&t(e1we2, e1we2we3, e1we2we3) + \&t(e1we2we3, e1we2, e1we2we3) + \&t(e1we2we3, e1we2we3, e1we2)$$

$$true$$

Checking **ungraded** co-commutativity:

```
> g1:=&gco_pl(e1we2+e1we2we3);
  g2:=switch(g1,1);
  evalb(%-%%=0);
```

$$g1 := (e1we2\ \&t\ e1we2we3) + (e1we2we3\ \&t\ e1we2) + (e1we2we3\ \&t\ e1we2we3)$$

$$g2 := (e1we2\ \&t\ e1we2we3) + (e1we2we3\ \&t\ e1we2) + (e1we2we3\ \&t\ e1we2we3)$$

$$true$$

To show that the Grassmann-Plücker co-product is indeed the dual of the meet, we compute the loop tangle \/ \circ \Delta which should evaluate to 2 to the power of the grade of the input element. In dimension 3 we find:

```
> dim_V:=3:bas:=cbasis(dim_V);
```

```
out1:=[seq(&gco_pl(bas[i]),i=1..nops(bas))]:
u:=proc(x) &map(x,1,`&v`) end:
map(drop_t@u,out1);
```

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

$$[8\,Id, 4\,e1, 4\,e2, 4\,e3, 2\,e1we2, 2\,e1we3, 2\,e2we3, e1we2we3]$$

Note, that the grade is taken w.r.t. the Plücker coordinatization. In this setting, the volume element is the scalar and has grade zero, bi-vectors (i.e. dim_V-1 vectors) have grade one and pick up a factor two etc., while the identity is of Plücker grade 3 (i.e. = dim_V-0) and gets a factor 2^3=8. To show the same relation for the ordinary Grassmann product, we compute the same loop tangle with Grassmann co-product and wedge:

```
> bas:=cbasis(dim_V);
  out1:=[seq(&gco(bas[i]),i=1..nops(bas))]:
  u:=proc(x) &map(x,1,`&w`) end:
  map(drop_t@u,out1);
```

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

$$[Id, 2\,e1, 2\,e2, 2\,e3, 4\,e1we2, 4\,e1we3, 4\,e2we3, 8\,e1we2we3]$$

We may note, that we find not two equivalent versions of the Grassmann-Plücker co-product as e.g. the meet and &v (vee), but that there is a left and right version which differ in an overall sign depending on the parity of the dimension dim_V and the parity of the monomial x on which it is calculated.

We will now proceed to show that this is in full analogy to the convolution unit:

```
> dim_V:=2:bas:=cbasis(dim_V):
  U:=proc(x) linop(x,U) end:
  X:=add(_X[i]*bas[i],i=1..2^dim_V);
  lhs_eq:=&map(tcollect(mapop(&gco_pl(X),2,U)),1,`&v`);
```

$$X := \_X_1\, Id + \_X_2\, e1 + \_X_3\, e2 + \_X_4\, e1we2$$

$$\begin{aligned} lhs\_eq := {} & \_X_1\, U_{2,3}\, \&t(0) + \_X_1\, U_{4,4}\, \&t(Id) + \_X_1\, U_{4,3}\, \&t(e1) + \_X_1\, U_{3,4}\, \&t(0) \\ & + \_X_1\, U_{1,4}\, \&t(0) - \_X_1\, U_{2,2}\, \&t(Id) + \_X_4\, U_{3,4}\, \&t(e2) - \_X_3\, U_{3,3}\, \&t(e2) + \_X_4\, U_{1,4}\, \&t(Id) \\ & - \_X_2\, U_{1,2}\, \&t(Id) - \_X_1\, U_{3,3}\, \&t(Id) + \_X_3\, U_{3,4}\, \&t(0) - \_X_2\, U_{3,4}\, \&t(Id) + \_X_1\, U_{2,1}\, \&t(e1) \\ & - \_X_3\, U_{1,3}\, \&t(Id) + \_X_3\, U_{4,4}\, \&t(e2) - \_X_1\, U_{4,2}\, \&t(e2) + \_X_1\, U_{1,3}\, \&t(0) - \_X_2\, U_{2,2}\, \&t(e1) \\ & + \_X_4\, U_{2,4}\, \&t(e1) - \_X_1\, U_{1,2}\, \&t(0) - \_X_2\, U_{4,2}\, \&t(e1we2) + \_X_1\, U_{1,1}\, \&t(Id) \\ & - \_X_3\, U_{4,3}\, \&t(e1we2) + \_X_3\, U_{2,4}\, \&t(Id) + \_X_3\, U_{1,4}\, \&t(0) - \_X_2\, U_{3,2}\, \&t(e2) \\ & + \_X_1\, U_{2,4}\, \&t(0) - \_X_3\, U_{2,3}\, \&t(e1) + \_X_1\, U_{3,1}\, \&t(e2) + \_X_2\, U_{2,4}\, \&t(0) - \_X_1\, U_{3,2}\, \&t(0) \\ & + \_X_1\, U_{4,1}\, \&t(e1we2) + \_X_2\, U_{4,4}\, \&t(e1) + \_X_4\, U_{4,4}\, \&t(e1we2) + \_X_2\, U_{1,4}\, \&t(0) \end{aligned}$$

```
> rhs_eq:=X; ## i.e. the identity mapping
```

$$rhs\_eq := \_X_1\, Id + \_X_2\, e1 + \_X_3\, e2 + \_X_4\, e1we2$$

```
> sol:=tsolve1(tcollect(drop_t(lhs_eq)-X),[seq(seq(U[i,j],i=1..2^
  dim_V),j=1..2^dim_V)],[seq(_X[i],i=1..2^dim_V)]);
```

$$sol := [\{U_{2,4}=0, U_{3,4}=0, U_{4,1}=0, U_{1,4}=0, U_{4,4}=1, U_{2,2}=0, U_{3,2}=0, U_{4,2}=0, U_{1,3}=0, U_{1,1}=0, U_{2,1}=0, U_{3,1}=0, U_{1,2}=0, U_{2,3}=0, U_{3,3}=0, U_{4,3}=0\}]$$

```
> matU:=linalg[matrix](2^dim_V,2^dim_V,(i,j)->U[i,j]):
  matU:=subs(sol[1],evalm(matU));
```

$$matU := \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This outcome shows that we have a unit in this convolution algebra, however, the unit is the projection onto the highest grade element i.e. the lowest Plücker grade, i.e. the volume element.

A further step is to look for the new antipode of this algebra.

```
> S:=proc(x) linop(x,S) end:
  lhs_eq:=clicollect(drop_t(&map(tcollect(mapop(&gco_pl(X),2,S)),
  1,`&v`)));
```

$$\begin{aligned} lhs\_eq := &-(-\_X_4\, S_{1,4} - \_X_1\, S_{4,4} + \_X_2\, S_{3,4} - \_X_1\, S_{1,1} + \_X_1\, S_{3,3} + \_X_1\, S_{2,2} - \_X_3\, S_{2,4} \\ &+ \_X_2\, S_{1,2} + \_X_3\, S_{1,3})\, Id + (-\_X_2\, S_{4,2} + \_X_4\, S_{4,4} + \_X_1\, S_{4,1} - \_X_3\, S_{4,3})\, e1we2 \\ &+ (\_X_4\, S_{2,4} - \_X_3\, S_{2,3} + \_X_1\, S_{4,3} - \_X_2\, S_{2,2} + \_X_1\, S_{2,1} + \_X_2\, S_{4,4})\, e1 \\ &+ (-\_X_2\, S_{3,2} - \_X_3\, S_{3,3} + \_X_1\, S_{3,1} + \_X_3\, S_{4,4} + \_X_4\, S_{3,4} - \_X_1\, S_{4,2})\, e2 \end{aligned}$$

```
> rhs_eq:=subs(sol[1],linop(X,U));
```

$$rhs\_eq := \_X_4\, e1we2$$

```
> sol:=tsolve1(tcollect(drop_t(lhs_eq)-rhs_eq),[seq(seq(S[i,j],i=
  1..2^dim_V),j=1..2^dim_V)],[seq(_X[i],i=1..2^dim_V)]);
```

$$sol := [\{S_{3,4}=0, S_{2,4}=0, S_{4,1}=0, S_{1,4}=0, S_{4,2}=0, S_{1,3}=0, S_{3,3}=1, S_{4,3}=0, S_{1,1}=1, S_{1,2}=0, S_{3,2}=0, S_{2,3}=0, S_{2,2}=1, S_{4,4}=1, S_{2,1}=0, S_{3,1}=0\}]$$

```
> matS:=linalg[matrix](2^dim_V,2^dim_V,(i,j)->S[i,j]):
  matS:=subs(sol[1],evalm(matS));
```

$$matS := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

As a further generalization, we are now able to define a **Clifford-Plücker product.** We will restrict ourselves to the case where we contract a Plücker 1-vector (hyperplane) with an arbitrary polynom X as defined above. We turn to dim_V = 3, so that hyperplanes are e1we2, e1we3 and e2we3.

```
> Gamma[pl]:=proc(x,u)
    local a;
    a:=&t(x,&gco_pl(u));
    contract(a,1,scalarpart@LC,B)+&map(&t(x,u),1,`&v`);
  end:
```

```
> dim_V:=3:
  clicollect(simplify(drop_t(Gamma[pl](e1we2,e1we2))));
  clicollect(simplify(drop_t(Gamma[pl](e1we2,e1we2we3))));
  clicollect(simplify(drop_t(Gamma[pl](e1we2,e2we3))));
  clicollect(simplify(drop_t(Gamma[pl](e1we2,a*e1we2+b*e2we3+c*e1
  we3+x*e1+y*e2+z*e3+t*Id))));
```

$$(B_{2,1}\,B_{1,2} - B_{2,2}\,B_{1,1})\,e1we2we3$$

$$e1we2$$

$$-(-B_{2,2}\,B_{1,3} + B_{2,3}\,B_{1,2})\,e1we2we3 - e2$$

$$\begin{aligned} z\,Id &- (-z\,B_{2,2}\,B_{1,3} + z\,B_{2,3}\,B_{1,2} - x\,B_{2,2}\,B_{1,1} + x\,B_{2,1}\,B_{1,2})\,e1we3 \\ &- (-z\,B_{2,3}\,B_{1,1} + z\,B_{2,1}\,B_{1,3} + y\,B_{2,1}\,B_{1,2} - y\,B_{2,2}\,B_{1,1})\,e2we3 \\ &- (-x\,B_{2,1}\,B_{1,3} + y\,B_{2,3}\,B_{1,2} + x\,B_{2,3}\,B_{1,1} - y\,B_{2,2}\,B_{1,3})\,e1we2 \\ &+ (a\,B_{2,1}\,B_{1,2} + b\,B_{2,2}\,B_{1,3} - b\,B_{2,3}\,B_{1,2} + c\,B_{2,1}\,B_{1,3} - c\,B_{2,3}\,B_{1,1} - a\,B_{2,2}\,B_{1,1})\,e1we2we3 \\ &- (t\,B_{2,3}\,B_{1,2} + c - t\,B_{2,2}\,B_{1,3})\,e1 - (t\,B_{2,1}\,B_{1,3} - t\,B_{2,3}\,B_{1,1} + b)\,e2 \\ &+ t\,(B_{2,1}\,B_{1,2} - B_{2,2}\,B_{1,1})\,e3 \end{aligned}$$

This very brief example shows that the bilinear form of the Clifford-Plücker product is given by the matrix of minors of the original bilinear form B which we have used in the contraction of the hyperplanes with the arbitrary monomial.

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM\n",time()-bench);
```

```
Worksheet took 2.764000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
```

```
>
```

See Also: Bigebra:-`&gco`, Bigebra:-`&cco`, Bigebra:-`&t`, Bigebra:-drop_t, Bigebra:-`&map`

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-`&map` - maps a product (2 -> 1 map) onto a tensor polynom

Calling Sequence:

t2 := &map(t1,i,pr)

Parameters:

- t1 : a tensor polynomial
- i : the i-th tensor slot to act in the pair (i,i+1).
- pr : a product, i.e. a Clifford polynomial-valued function of two Clifford polynomials i.e. a 2-> 1 map.

Output:

- t2 : the 'product' tensor polynom

Description:

```
>
```

Examples:

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

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

Mapping wedge products:

```
> &map(&t(e1,e2)-a*&t(e3we4,e1)-x*&t(e1,e1-e2),1,wedge);
  drop_t(%); # convert into a Clifford polynom
  &map(&t(e1,e2,e3,e4),1,wedge);  # first pair (1,2)
  &map(&t(e1,e2,e3,e4),3,wedge);  # last  pair (3,4)
  &map(&t(e1,e2,e3,e4),4,wedge);  # ==> ERROR <== 4+1=5 is not
  available
```

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

$$\&t(e1we2) - a\,\&t(e1we3we4) - x\,\&t(0) + x\,\&t(e1we2)$$

$$e1we2 - a\,e1we3we4 + x\,e1we2$$

$$\&t(e1we2, e3, e4)$$

$$\&t(e1, e2, e3we4)$$

```
Error, (in Bigebra:-&map) invalid subscript selector
```

Multiply back again by the Clifford/Grassmann co-product, this gives a loop in tangle notation:

```
> &gco(&t(e1),1);&map(%,1,wedge);
  &gco(&t(e1),1);&map(%,1,cmul);    # equivalent
```

```
&gco(&t(e1we2),1);&map(%,1,wedge);
&gco(&t(e1we2),1);&map(%,1,cmul); # different
```

$$(Id \,\&t\, e1) + (e1 \,\&t\, Id)$$

$$2\,\&t(e1)$$

$$(Id \,\&t\, e1) + (e1 \,\&t\, Id)$$

$$2\,\&t(e1)$$

$$(Id \,\&t\, e1we2) + (e1 \,\&t\, e2) - (e2 \,\&t\, e1) + (e1we2 \,\&t\, Id)$$

$$4\,\&t(e1we2)$$

$$(Id \,\&t\, e1we2) + (e1 \,\&t\, e2) - (e2 \,\&t\, e1) + (e1we2 \,\&t\, Id)$$

$$4\,\&t(e1we2) + B_{1,2}\,\&t(Id) - B_{2,1}\,\&t(Id)$$

One can show, that wedging back the Grassmann co-product yields 2^grade of the homogeneous multi-vectors, extended by the multilinearity. The Clifford case works out differently, there one finds 2^dim_V (for non degenerate scalar or co-scalar products). Here, dim_V is the dimension of the vector space (V,B) that yields Cl(V,B). For more information the global variable dim_V, see CLIFFORD_ENV.

```
> dim_V:=2:B:=linalg[matrix](dim_V,dim_V,[a,b,c,d]):
  BI:=linalg[matrix](dim_V,dim_V,[u,z,t,v]):make_BI_Id():
> c1:=&cco(&t(e1),1);&map(%,1,wedge):drop_t(%);      # as above
  c2:=&gco(&t(e1we2),1);&map(%,1,cmul):drop_t(%);    # depends on
  B
  c3:=&cco(&t(Id+e1+e2+e1we2),1);&map(%,1,wedge):drop_t(%);   #
  depends on BI
  c4:=&cco(&t(e1we2),1);&map(%,1,cmul):drop_t(%);    # depends on
  B and BI
```

$$c1 := (Id \,\&t\, e1) - z\,(e1 \,\&t\, e1we2) - v\,(e2 \,\&t\, e1we2) + (e1 \,\&t\, Id) + t\,(e1we2 \,\&t\, e1) + v\,(e1we2 \,\&t\, e2)$$

$$2\,e1$$

$$c2 := (Id \,\&t\, e1we2) + (e1 \,\&t\, e2) - (e2 \,\&t\, e1) + (e1we2 \,\&t\, Id)$$

$$4\,e1we2 + b\,Id - c\,Id$$

$$c3 := (Id \,\&t\, Id) + u\,(e1 \,\&t\, e1) + t\,(e2 \,\&t\, e1) + z\,(e1 \,\&t\, e2) + v\,(e2 \,\&t\, e2) + (t\,z - v\,u)\,(e1we2 \,\&t\, e1we2) + (Id \,\&t\, e1) - z\,(e1 \,\&t\, e1we2) - v\,(e2 \,\&t\, e1we2) + (e1 \,\&t\, Id) + t\,(e1we2 \,\&t\, e1) + v\,(e1we2 \,\&t\, e2) + (Id \,\&t\, e2) + u\,(e1 \,\&t\, e1we2) + t\,(e2 \,\&t\, e1we2) + (e2 \,\&t\, Id) - u\,(e1we2 \,\&t\, e1) - z\,(e1we2 \,\&t\, e2) + t\,(e1we2 \,\&t\, e1we2) + (e1 \,\&t\, e2) + (Id \,\&t\, e1we2) + (e1we2 \,\&t\, Id) - (e2 \,\&t\, e1) - z\,(e1we2 \,\&t\, e1we2)$$

$$2\,e1 + Id + 2\,e2 - t\,e1we2 + z\,e1we2 + 4\,e1we2$$

$$c4 := t\,(e1we2 \,\&t\, e1we2) + (e1 \,\&t\, e2) + (Id \,\&t\, e1we2) - (e2 \,\&t\, e1) + (e1we2 \,\&t\, Id)$$

$$-z\,(e1we2\ \&t\ e1we2)$$

$$t\,(Id\,c\,b - Id\,d\,a + c\,e1we2 - b\,e1we2) + b\,Id + 4\,e1we2 - c\,Id$$

$$-z\,(Id\,c\,b - Id\,d\,a + c\,e1we2 - b\,e1we2)$$

However, we can also map e.g. the meet

```
> &map(&t(e1,e1we2,e1,e2),2,meet);
```

$$\&t(e1, e1, e2)$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM\n",time()-bench);
Worksheet took 0.546000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
>
```

See Also: Bigebra help page, Bigebra:-meet, Bigebra:-`&v`, CLIFFORD help

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-`&t` - tensor product

Calling Sequence:

- t0 = &t(c1,c2,...cn) [or c1 &t c2 &t ... &t cn, **not recommended**]

Parameters:

- c1, c2, ..., cn - expressions of `type/clipolynom`

Output:

- t0 : is a tensor polynom.

Description:

- At the time of loading, BIGEBRA initializes the tensor product &t. The operator &t is defined using Maple's define facility. Define was patched for two reasons:
  - It had bugs.
  - Scalars had to be changed to be of `type/cliscalar`.
- We recommend to use the **prefix form** &t(c1,..,cn) of &t since the infix form c1 &t c2 .. &t cn can cause problems if no parentheses are used (see below and define).
- The tensor product &t is only available in its operator (ampersand) form. It is defined as an associative (flat) Maple operator which is multilinear w.r.t. to cliscalars. This allows the user to define Grassmannians, spinor modules, Clifford modules, algebraic varieties etc.
- In the expression &t(c1,c2,...,cn), one calls the place 'i', where the parameter 'ci' resides, the i-th slot or place of the tensor product.
- The 'product' rule of a tensor product is simply concatenation of the slots. Associativity allows one to drop parentheses (watch out, parentheses are needed when using the infix form). Associativity is called 'flat' in Maple:
  (c1) &t (c2) &t ... &t (cn) = &t(c1,c2,...,cn)
  &t(c1,...,c2) &t &t(c3,...,c4) = &t(c1,...,c2,c3,...,c4)
- The tensor product is linear in each factor w.r.t. cliscalars.
  &t(c1,..,a*ci+b*ci`,..,cn) = a* &t(c1,..,ci,..,pc) + b* &t(c1,..,ci`,..,cn)
  where ci,ci` are clifford polynoms and a,b are cliscalars.
- This tensor product is a prototype in the sense that one can regard it as a graded tensor product also. The difference comes from the functions which are applied to the tensor product, as e.g. the switch and graded switch .
- **TO DO:** In Maple 6 a new version of &t will allow to specify the type of scalars, i.e. the domain (ring) the tensor product is built over.

Examples:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

Infix form (**not recommended**, see below) and concatenation (i.e. **associativity**):

```
> e1 &t e2;
  e1 &t e2we3 &t e3we4;
  &t(e1,e2) &t &t(e3,e1we2); ## the middle &t acts as 'product'.
  &t(e1,&t(e2,&t(e1we2,e3)));
```

$$e1 \ \&t \ e2$$

$$\&t(e1, e2we3, e3we4)$$

$$\&t(e1, e2, e3, e1we2)$$

$$\&t(e1, e2, e1we2, e3)$$

The infix form of &t is peculiar, since it has a BUG !! See this and watch out. It is recommended **not** to use the infix form!

```
> &t(a*e1,b*e2); # correct
```

$$a\, b\, (e1 \ \&t \ e2)$$

```
> a*e1 &t b*e2;  # BUG, see second tensor slot !
```

$$a\, b\, (e1 \ \&t \ 1)\, e2$$

```
> (a*e1) &t (b*e2); # correct
```

$$a\, b\, (e1 \ \&t \ e2)$$

The reason for this **BUG** is that the infix form gets confused what kind of object it has to treat. `&t` does not know what kind of type it treats and causes an error. Putting in parentheses is **necessary**. We recommend to use the **prefix form**.

Linearity in each slot:

```
> &t(e1,e2+e3,e1we2);
```

$$\&t(e1, e2, e1we2) + \&t(e1, e3, e1we2)$$

```
> &t(sin(theta)*e2+cos(theta)*e1we2,e3);
```

$$\sin(\theta)\, (e2 \ \&t \ e3) + \cos(\theta)\, (e1we2 \ \&t \ e3)$$

```
> &t(a*e1+b*e2,c*e3,d*e4,f*e5+g*e6);
```

$$a\, c\, d\, f\, \&t(e1, e3, e4, e5) + a\, c\, d\, g\, \&t(e1, e3, e4, e6) + b\, c\, d\, f\, \&t(e2, e3, e4, e5) + b\, c\, d\, g\, \&t(e2, e3, e4, e6)$$

```
> type(a,cliscalar);
  type(exp(x),cliscalar);
  type(a*e1,cliscalar);
```

true

true

false

```
> printf("Worksheet took %f seconds on Intel Pentium M 2.13 GHz
  2GB RAM WinXP Prof\n",time()-bench);
Worksheet took 0.031000 seconds on Intel Pentium M 2.13 GHz 2GB RAM WinXP Prof
>
```

A cliscalar is anything which is not of any of these types: `type/clipolynom`, `type/climon`, `type/clibasmon`, or `type/tensorpolynom`.

See Also: Bigebra:-&map, Bigebra:-mapop, Bigebra:-mapop2, Bigebra:-contract, Bigebra:-peek, Bigebra:-poke, Bigebra:-switch, Bigebra:-`type/tensorbasmonom`, Bigebra:-`type/tensormonom`, Bigebra:-`type/tensorpolynom`

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-`&v` -- the vee (meet) product.
Bigebra:-meet -- the meet product.

Calling Sequence:

c3 := &v(c1,c2) [or c1 &v c2, not recommended]
c3 := meet(c1,c2) -- synonym.

Parameters:

- c1,c2 - expressions of `type/clipolynom`

Output:

- c3 - expression of `type/clipolynom`

Global variables:

- dim_V - dimension of the vector space (V,B) that is defined in CLIFFORD as a global variable.

Description:

- The pair of operations wedge (i.e. join) and ***meet*** acting on Grassmann multi-vectors make up, together with the duality operator, the ***Grassmann Cayley algebra***. This algebra is of tremendous importance in geometrical applications like robotics, visual perception, camera calibration. However, incidence geometries have their own well developed mathematical theory, see e.g. P. Dembowski, Finite Geometries, Springer Verlag, New York, 1968.

- To avoid confusion we should point out that the notion of a meet is not unique in literature. Let A be a homogeneous decomposable multivector called an ***extensor***. Every such extensor spans a linear subspace of the space over which it was constructed. The span of A is called the ***support*** of A, denoted as *supp A*. Meet and join can be defined in set theoretic terms on the support of extensors. Let A, B denote extensors, one defines:

  A **cup** B := {x in V | x in supp A ***or*** in supp B} i.e. the set theoretic union
  A **cap** B := {x in V | x in supp A ***and*** in supp B} i.e. the set theoretic intersection

  The operators **cup** and **cap** are the same as in set theory. Under these operations every set is an idempotent: A **cup** A = A and B **cap** B = B. Moreover, one finds **cup** *o* **cup** = Id and **cap** *o* **cap** = Id for these operators. Including the set theoretic operation of taking the complement, (A -> |A with A **cup** |A = whole space, where we have used, in lack of an over bar, the Grassmann notation of a preceding bar), this constitutes the structure of an ortho-modular lattice. Boolean logic is based on this construction. The two operations of meet and join are related by de Morgan laws:

  |(A **cup** B) = (|A) **cap** (|B)
  |(A **cap** B) = (|A) **cup** (|B).

In terms of logic we have: **cup** = **and**, **cap** = **or**, | = **not**.

In CLIFFORD and Bigebra packages, the meet and join are defined in the following way:

The wedge product of two extensors A and B is an extensor C which has as support the disjoint union of the supports of A and B. However, extensors having the same support are isomorphic (interchangeable). ***We define the join to be this wedge operation.*** The meet is usually defined using a symmetric correlation in the projective space **P**^dim(V). It needs thus a theorem to show that the meet is independent from its construction. Grassmann defined the meet, which he called ***regressive product***, in [A2]**, 1862,** §5, No. 94 page 61ff. The regressive product was already present in [A1], chapter 3, §125ff. Grassmann edited in 1877 a reprint with annotations where he gave some explanations on his presentation. A careful reading shows that the regressive product was present already in 1844. The Ergänzung is not explicit in [A1], but Grassmann discusses the grade of the complement |A which he calls there 'Ergänzzahlen' (A1 §133)) using the so called 'Ergänzung' (Grassmann A2, §4, No. 89 page 57), which we defined already above as |, of an extensor A to be |A. In analogy to de Morgan laws (which he most likely did not knew) as:

| (A ∨ B) := (|A) ∧ (|B).

[Grassmann used no sign for products, having over 16 of them working, many at the same time and their type had to be deduced by context. He used furthermore no parentheses which makes his writings cumbersome to read. The ∧ sign mutated from an (uppercase) Lambda used by Burali-Forti and Marcolongo to be the wedge of Bourbaki.]

The usage of the Ergänzung points out clearly that the meet depends on the dimension of the space. We will see below, that this definition of the meet is computationally very ineffective.

Alfred Lotze, (Über eine neue Begründung der regressiven Multiplikation extensiver Größen in einem Hauptgebiet n-ter Stufe, Jahresbericht der DMV, 57:102-110,1955) defined a *universal formula* for the regressive product of r-factors. He showed that if one considers the n-1 dimensional space as a space of co-vectors then the original *wedge product* becomes by the same formula the regressive product of the co-vectors, pointing out the fact that a symmetric correlation is needed for this purpose. That is: (n-1)-multi-vectors are not co-vectors, but may be seen as reciprocal vectors. In [4], G.-C. Rota and coworkers gave a definition of the meet in terms of a Peano algebra which is essentially the same construction. However, they used the notion of Hopf algebra which allows to write down the formulas in a comprehensible way.

The Grassmann wedge product has as logical counterpart in the exclusive or ***xor*** , the Ergänzung is **not** w.r.t. the chosen volume form of the space V the Grassmann algebra is build over. The meaning of the meet follows from his duality relation.

- In Bigebra, the meet and &v (vee) products are implemented as follows (**note** the order of factors in the bracket!):

meet(c1,c2) := [c2_(1),c1] c2_(2)
&v(c1,c2) := c1_(1) [c2,c1_(2)] ,

where the bracket [,] is a scalar valued alternating multilinear volume form and the co-products are given in Sweedler notation. It can be shown (and is tested below) that both forms represent the same operation.

- The Hopf algebraic definition of the meet gives us a great deal of ***computational benefits*** as we will show below in some benchmarks. However it works exactly as the Grassmann regressive product.
- Grassmann introduced the so called *stereometric product*, which, being context sensitive, switches between the wedge and the &v (vee-) product. Using polymorphism this could be implemented, and the user can easily program such a wrapper function. We found it peculiar to implement it using the same notation for basis elements for vectors and co-vectors.
- The meet as defined here is independent of the assigned scalar product *B* or the assigned co-scalarproduct *BI* . In fact it can be shown that the vee-product is SL_n invariant. If one is interested in projective geometry, the invariants derived from meet and join are GL_n invariants.
- The meet product is related to the notion of a Hopf algebraic integral [3]. As a remarkable fact, in any Clifford Hopf gebra over dim V = 2 one is not able to find an non zero integral. The notion of meet has thus to be reconsidered in the deformed case.

Examples:

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

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

Infix form (**not recommended**, see help page on &t). Note that we have not assigned a scalar- or co-scalarproduct.

```
> dim_V:=2:
  e1 &v e2;
  e1 &v e1we2, e1we2 &v e2, e1we2 &v e1we2;
```

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

$$-Id$$

$$e1, e2, e1we2$$

First of all let us check that both versions of the meet compute indeed identically:

```
> for i from 1 to 6 do
  dim_V:=i:
  bas:=cbasis(dim_V);
```

```
X:=add(_X[i]*bas[i],i=1..2^dim_V):
Y:=add(_Y[i]*bas[i],i=1..2^dim_V):
bench:=time():
printf("In dimension %d the equation `meet(X,Y)=&v(X,Y)` is %a
and took %f seconds to verify on Intel Pentium M 2.13 GHz 2GB
RAM WinXP
Prof\n",dim_V,evalb(0=simplify(meet(X,Y)-&v(X,Y))),time()-bench
);
od:
```

```
In dimension 1 the equation `meet(X,Y)=&v(X,Y)` is true and took 0.125000 secon
ds to verify on Intel Pentium M 2.13 GHz 2GB RAM WinXP Prof
In dimension 2 the equation `meet(X,Y)=&v(X,Y)` is true and took 0.359000 secon
ds to verify on Intel Pentium M 2.13 GHz 2GB RAM WinXP Prof
In dimension 3 the equation `meet(X,Y)=&v(X,Y)` is true and took 1.078000 secon
ds to verify on Intel Pentium M 2.13 GHz 2GB RAM WinXP Prof
In dimension 4 the equation `meet(X,Y)=&v(X,Y)` is true and took 5.750000 secon
ds to verify on Intel Pentium M 2.13 GHz 2GB RAM WinXP Prof
In dimension 5 the equation `meet(X,Y)=&v(X,Y)` is true and took 31.984000 seco
nds to verify on Intel Pentium M 2.13 GHz 2GB RAM WinXP Prof
In dimension 6 the equation `meet(X,Y)=&v(X,Y)` is true and took 185.278000 sec
onds to verify on Intel Pentium M 2.13 GHz 2GB RAM WinXP Prof
```

```
>
```

The following **example** will show, that the meet and the join are **exterior** products on their own right and cannot be distinguished. This makes it unnecessary to use the \/ (vee) sign for the ordinary wedge product as Rota promoted to stress the analogy with set theoretic operators. We will see that the join of points is the meet of (hyper) planes and the meet of points is the join of (hyper) planes. To demonstrate this, we compute the meet for a Grassmann basis. We check associativity, unit, and show that this product is an exterior product on its own right on reciprocal (sometimes called wrongly dual) vectors (i.e. hyperplanes). The reciprocal meet is then defined to be the meet w.r.t. hyper-planes. Then it is shown that this reciprocal meet is indeed the wedge (join of points) with which we started. [To give a crude reciprocal meet we use Grassmann's Ergänzung, but a combinatorial evaluation is also possible but proved to be too long for this help page.]

We present our demonstration in dimension 3. Define the n-1 (i.e. 2-) vectors A(i). These multi-vectors are the images of a covector basis under dualization, see [4,3] and should be called reciprocal vectors. Their definition involves a symmetric correlation. The 'meet' or `&v` (vee) product of vectors acts as an exterior (or wedge) product on these reciprocal vectors. This is an immediate consequence of categorical duality and is related to the Plücker coordinatization of hyper-planes.

While we show here explicitly how to define a a Meet and Join for hyperplanes, there is an generic Grassmann co-product &gco_pl in the package which could be used with some benefits for the performance, but would probably obscure out aim here.

```
> dim_V:=3:
  # A(i),A(ij) etc are new basis elements
```

```
#
# ==> define the hyperplane basis A(i), A(ij) etc
A:=proc(x)
  local T;
  T:=table([123=-Id,
            31=-e2,23=-e1,12=-e3,
            13= e2,32= e1,21= e3,
            3=e1we2,2=-e1we3,1=e2we3,
            0=e1we2we3]);
  RETURN(T[x]);
end:
#
# w2A is a translation procedure which turns the output
# into the new A basis of reciprocal vectos (plane vectors)
#
# ==> w2A  (wedge basis to hyperplane basis A(i)
w2A:=proc(x)
     local bas,y;
     bas:={Id=-'A(123)',
          e1=-'A(23)', e2= 'A(13)',e3= -'A(12)',
          e1we2='A(3)',e1we3=-'A(2)',e2we3='A(1)',
          e1we2we3='A(0)'};
     RETURN(subs(bas,Clifford:-reorder(x)));
end:
#
# &V (uppercase) is a wrapper function to make the usage of
the
#     A(i) basis more comfortable
#
# ==> &V can act on the hyperplane basis A(i) seen as wedge
multivectors
#     and yields A(jk) hyperplane 2-vectors
#     [same goes for the Meet (formerly meet)]
`&V`:=proc(x,y)
    w2A(&v(eval(x),eval(y)));
end:
`Meet`:=proc(x,y)
    w2A(meet(eval(x),eval(y)));
end:
```

After these preliminary definitions we can directly show the meet to be the '***wedge product of hyperplanes***'. First of all we check some elementary properties of the meet acting on hyperplanes.

```
> A(0),w2A(A(0)); # The 'scalar' w.r.t. the &v product
  A(1),w2A(A(1));
```

```
A(2),w2A(A(2));
A(3),w2A(A(3)); # (reciprocal) vectors
```

$$e1we2we3, \mathrm{A}(0)$$
$$e2we3, \mathrm{A}(1)$$
$$-e1we3, \mathrm{A}(2)$$
$$e1we2, \mathrm{A}(3)$$

```
> &V(A(0),A(1)),&V(A(1),A(0));        #  shows A0 to be the
identity
Meet(A(0),A(1)),Meet(A(1),A(0));    #  synonym but internally
computed differently
```

$$\mathrm{A}(1), \mathrm{A}(1)$$
$$\mathrm{A}(1), \mathrm{A}(1)$$

Now we produce reciprocal bi-vectors (bi-hyperplanes to be precise) A(ij) and the volume element A(123)

```
> &V(A(1),A(2)),&V(A(2),A(3)),&V(A(3),A(1));  # BI-HYPERPLANES
&V(A(1),&V(A(2),A(3))),                       # VOLUME ELEMENT
EVALUATES TO -ID
&v(A(1),eval(&V(A(2),A(3))));                 # eval is needed
here to apply A(23)
```

$$\mathrm{A}(12), \mathrm{A}(23), -\mathrm{A}(13)$$
$$\mathrm{A}(123), -Id$$

There are no higher multi-hyperplanes (reciprocal multi-vectors) and the following expressions evaluate to zero:

```
> &V(A(1),A(123)),&V(A(12),A(23));
```

$$0, 0$$

The bracket for co-vectors can be defined using the fact that -Id is the volume in the space of hyperplanes as the projection onto -Id. Hence we can define the reciprocal meet RMeet of reciprocal vectors. This is also a demonstration how to extend the features of the CLIFFORD/Bigebra packages:

```
> B:=linalg[diag](1$dim_V):     ###  internally used for
Grassmann Erg"anzung
`&RMeet`:=proc(x,y)                        ###  function
co-meet
    local yy,res,lst,var_i,v1,v2;
    option `Copyright (c) Ablamowicz, Fauser 2000/02. All
```

```
rights reserved.`;
    ## crude version of the Grassmann co-product on the
'multivector plane space' \/A
    yy:=&t(e1we2we3,&gco(eval(cmul(e3we2we1,y))),e1we2we3);
    yy:=&map(tcollect(&map(switch(yy,3),3,cmul)),1,cmul);
##
    if type(yy,tensorbasmonom) or type(yy,tensormonom)then
      lst:=[yy];
    else
      lst:=[op(yy)];
    fi;
    res:=0;
    for var_i in lst do
      v1,v2:=peek(var_i,1);
      res := res - scalarpart(&v(eval(x),v1))*drop_t(op(v2));
    od;
    res;
end:
```

To exemplify out claim, let us define the two mutually reciprocal basis sets of points, joined points (i.e., lines) and point space volume and the hyperplanes bi-hyperplanes (i.e., lines) and the volume of the hyperplane multi-vector space -Id.

```
> bas:=cbasis(3);
bas_A:=[A(0),A(1),A(2),A(3),A(12),A(13),A(23),A(123)];
```

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

$$bas\_A := [e1we2we3, e2we3, -e1we3, e1we2, -e3, e2, -e1, -Id]$$

For easy comparison, we compute the multiplication table of the RMeet product. This multiplication table is a tensor of rank three. To be able to display this tensor as rank two array, we put the resulting multivectors (in Grassmann basis) into the array. The numerical matrices m_[ij]^k are then obtained by setting one basis element to 1 and all other to zero (i.e. by acting with the dual multivectors on this scheme.)

```
> Mul_tab_RMeet:=linalg[matrix](2^dim_V,2^dim_V,(i,j)-> 0):
for i from 1 to 2^dim_V do
for j from 1 to 2^dim_V do
  Mul_tab_RMeet[i,j]:=reorder(&RMeet(bas[i],bas[j]));
od:od:
evalm(Mul_tab_RMeet);
```

$$\begin{bmatrix} -Id, & -e1, & -e2, & -e3, & -e1we2, & -e1we3, & -e2we3, & -e1we2we3 \\ -e1, & 0, & -e1we2, & -e1we3, & 0, & 0, & -e1we2we3, & 0 \\ -e2, & e1we2, & 0, & -e2we3, & 0, & e1we2we3, & 0, & 0 \\ -e3, & e1we3, & e2we3, & 0, & -e1we2we3, & 0, & 0, & 0 \\ -e1we2, & 0, & 0, & -e1we2we3, & 0, & 0, & 0, & 0 \\ -e1we3, & 0, & e1we2we3, & 0, & 0, & 0, & 0, & 0 \\ -e2we3, & -e1we2we3, & 0, & 0, & 0, & 0, & 0, & 0 \\ -e1we2we3, & 0, & 0, & 0, & 0, & 0, & 0, & 0 \end{bmatrix}$$

Our final **goal** is to show, that the above defined multiplication for RMeet (the **meet of hyperplanes) is equivalent to the wedge product of points**. We compute therefore the multiplication table for the wedge also:

```
> Mul_tab_wedge:=linalg[matrix](2^dim_V,2^dim_V,(i,j)-> 0):
  for i from 1 to 2^dim_V do
  for j from 1 to 2^dim_V do
    Mul_tab_wedge[i,j]:=&w(bas[i],bas[j]);
  od:od:
  evalm(Mul_tab_wedge);
```

$$\begin{bmatrix} Id, & e1, & e2, & e3, & e1we2, & e1we3, & e2we3, & e1we2we3 \\ e1, & 0, & e1we2, & e1we3, & 0, & 0, & e1we2we3, & 0 \\ e2, & -e1we2, & 0, & e2we3, & 0, & -e1we2we3, & 0, & 0 \\ e3, & -e1we3, & -e2we3, & 0, & e1we2we3, & 0, & 0, & 0 \\ e1we2, & 0, & 0, & e1we2we3, & 0, & 0, & 0, & 0 \\ e1we3, & 0, & -e1we2we3, & 0, & 0, & 0, & 0, & 0 \\ e2we3, & e1we2we3, & 0, & 0, & 0, & 0, & 0, & 0 \\ e1we2we3, & 0, & 0, & 0, & 0, & 0, & 0, & 0 \end{bmatrix}$$

The final check is to add both matrices which gives zero. This shows that up to a sign (which is irrelevant in projective plane geometry) the products are the same. Or, as operator equation:

RMeet(x,y) = -wedge(x,y)

The sign belongs to the fact that in three dimensions we find that the volume element squares to negative identity, which means that we would reach the original wedge after a second turn in our argumentation. However, we resist to demonstrate this explicitly here.

```
> evalm(Mul_tab_RMeet+Mul_tab_wedge);
```

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Finally we will provide some ***benchmarks*** which shall show how efficient the two alternate definitions of the meet are. One, as adopted recently by Hestenes and followers, is based on the Grassmann's Ergänzung and the other is based on Hopf algebra methods as employed in Bigebra and given by Lotze and Rota.

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM\n",time()-bench);
Worksheet took 189.513000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RA
M
> restart:bench:=time():with(Clifford):with(Bigebra):
> dim_V:=3:B:=linalg[diag](1$dim_V):
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
```

We use dimension 3 and a orthogonal, normalized Euclidean bilinear form (the polar form of a quadratic form).

As a **Benchmark** we compute 100 times a certain meet (this is not a good idea, since some functions may remember its results, e.g. the wedge product from the CLIFFORD package, but it gives nevertheless a feeling what is going on).

The Hopf algebraic case needs:

```
> s:=time():
  for i from 1 to 100 do
    &v(e1we2,e2we3);
  od:
  printf("This took us %f seconds",time()-s);
  &v(e1we2,e2we3);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
This took us 1.437000 seconds
```

$$-e2$$

```
> restart:bench:=time():with(Clifford):with(Bigebra):   # reload
  everything to be fair
  dim_V:=3:B:=linalg[diag](1$dim_V):
  s:=time():
  for i from 1 to 100 do
```

```
cmul(e3we2we1,wedge(cmul(e1we2we3,e1we2),cmul(e1we2we3,e2we3)))
;
od:
printf("This took us %f seconds",time()-s);
cmul(e3we2we1,wedge(cmul(e1we2we3,e1we2),cmul(e1we2we3,e2we3)))
;
```

```
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
This took us 0.687000 seconds
```

$$e2$$

Since we compute the Clifford product using a very fast Hopf algebraic function cmulRS, this works out faster. However, we could speed up &v by directly employing the Hopf algebraic routines and avoiding wrapper functions as `peek`. Furthermore we have not computed the inverse of the Ergänzung but introduced simply e3we2we1 which is (e1we2we3)^(-1) in our case.

Now let us go for a non-orthogonal but still symmetric bilinear form (a polar form of a quadratic form or a symmetric correlation) and check what happens there:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
  dim_V:=3:B:=linalg[matrix](dim_V,dim_V,(i,j)->if i<=j then
  g[i,j] else g[j,i] fi);
```

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

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

Compute the Bigebra meet &v (vee product):

```
> s:=time():
     clicollect(&v(e1we2,e2we3));
  printf("This took us %f seconds",time()-s);
```

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

$$-e2$$

```
This took us 0.015000 seconds
```

We reload once more the package to be fair and compute the meet using the Ergänzung:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
  dim_V:=3:B:=linalg[matrix](dim_V,dim_V,(i,j)->if i<=j then
  g[i,j] else g[j,i] fi);
  s:=time():
  clicollect(cmul(e3we2we1,wedge(cmul(e1we2we3,e1we2),cmul(e1we2w
  e3,e2we3))));
  printf("This took us %f seconds",time()-s);
```

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

```
elp]
```

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

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

$$(g_{1,3}^{4} g_{2,2}^{2} + g_{1,1}^{2} g_{2,3}^{4} + g_{1,2}^{4} g_{3,3}^{2} + g_{2,2}^{2} g_{1,1}^{2} g_{3,3}^{2} + 4 g_{1,3}^{2} g_{2,3}^{2} g_{1,2}^{2} - 2 g_{2,2}^{2} g_{1,1} g_{3,3} g_{1,3}^{2} - 2 g_{2,2} g_{1,1}^{2} g_{2,3}^{2} g_{3,3} + 2 g_{2,2} g_{1,1} g_{2,3}^{2} g_{1,3}^{2} - 2 g_{2,2} g_{1,2}^{2} g_{3,3}^{2} g_{1,1} + 2 g_{2,2} g_{1,2}^{2} g_{3,3} g_{1,3}^{2} - 4 g_{2,2} g_{1,3}^{3} g_{2,3} g_{1,2} + 4 g_{2,2} g_{1,3} g_{2,3} g_{1,2} g_{1,1} g_{3,3} + 2 g_{1,1} g_{2,3}^{2} g_{3,3} g_{1,2}^{2} - 4 g_{1,2}^{3} g_{3,3} g_{2,3} g_{1,3} - 4 g_{1,2} g_{1,1} g_{2,3}^{3} g_{1,3}) e2$$

```
This took us 0.250000 seconds
```

This shows already a difference (approx. a factor 6, which varies from computation to computation due to garbage collection overhead) which would further *increase* if the dimension were higher. Thus, the **computational efficiency** of the meet has been demonstrated.

Moreover, we can go beyond the possibilities of the Ergänzungs method since we can compute the meet in the presence of a non-symmetric bilinear form (which cannot be derived from a quadratic form by polarization) using Hopf algebra methods. Our meet works **independently of the assigned bilinear form** while the Eränzungs method needs an orthogonal non-degenerate bilinear form (which is the polar form of the symmetric correlation, i.e. a quadratic form).

Let us use an arbitrary bilinear form in 3 dimensions:

```
> B:=linalg[matrix](dim_V,dim_V,(i,j)->b[i,j]);
```

$$B := \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$$

The Hopf algebraic meet remains to be

```
> &v(e1we2,e2we3);
```

$$-e2$$

while the 'meet' computed using the Ergänzung does not even yield a homogeneous multi-vector, but a Clifford polynomial:

```
> clicollect(simplify(cmul(e3we2we1,wedge(cmul(e1we2we3,e1we2),
                                cmul(e1we2we3,e2we3)))));
```

$$(-2 b_{2,2} b_{3,1} b_{2,3} b_{1,2} b_{1,3} - b_{2,2} b_{3,2} b_{2,1} b_{1,3}^{2} + b_{3,1} b_{2,2}^{2} b_{1,3}^{2} - b_{3,1}^{2} b_{2,2}^{2} b_{1,3} - b_{2,2} b_{1,1} b_{3,2} b_{2,3} b_{3,1} - b_{2,2}^{2} b_{1,1} b_{3,3} b_{1,3} + b_{3,1} b_{2,3}^{2} b_{1,2}^{2} + b_{2,2} b_{2,1} b_{1,2} b_{3,3} b_{1,3}$$

$$+ b_{1,2} b_{3,2} b_{2,1} b_{1,3} b_{2,3} - b_{1,2} b_{1,1} b_{3,2} b_{2,3}^{2} + b_{2,2}^{2} b_{1,1} b_{3,3} b_{3,1} + b_{1,1} b_{3,2}^{2} b_{2,3} b_{2,1}$$
$$+ b_{3,2} b_{2,1}^{2} b_{1,2} b_{3,3} - b_{2,2} b_{2,1} b_{1,2} b_{3,3} b_{3,1} - b_{3,2} b_{3,1} b_{2,3} b_{1,2} b_{2,1} - b_{3,2}^{2} b_{2,1}^{2} b_{1,3}$$
$$+ b_{2,2} b_{3,1}^{2} b_{2,3} b_{1,2} + b_{1,2} b_{2,2} b_{1,1} b_{3,3} b_{2,3} - b_{2,1} b_{1,2}^{2} b_{3,3} b_{2,3} + 2 b_{2,2} b_{3,2} b_{2,1} b_{1,3} b_{3,1}$$
$$+ b_{2,2} b_{1,1} b_{3,2} b_{2,3} b_{1,3} - b_{3,2} b_{2,2} b_{1,1} b_{3,3} b_{2,1}) e1we2we3 + (-b_{3,2} b_{3,1}^{2} b_{2,3} b_{1,2} b_{2,2}$$
$$+ b_{3,2} b_{2,1} b_{1,2} b_{3,3} b_{3,1} b_{2,2} - b_{3,2} b_{2,2}^{2} b_{1,1} b_{3,3} b_{3,1} - b_{3,1} b_{2,3}^{2} b_{1,2}^{2} b_{3,2}$$
$$+ b_{3,1} b_{2,3} b_{1,2}^{2} b_{3,3} b_{2,2} + b_{2,1} b_{1,2}^{2} b_{3,3} b_{3,2} b_{2,3} - b_{2,1} b_{1,2}^{2} b_{3,3}^{2} b_{2,2} + b_{1,2} b_{1,1} b_{3,2}^{2} b_{2,3}^{2}$$
$$- b_{1,2} b_{3,2}^{2} b_{2,1} b_{1,3} b_{2,3} + b_{1,2} b_{3,2} b_{2,1} b_{1,3} b_{3,3} b_{2,2} - 2 b_{1,2} b_{1,1} b_{3,2} b_{2,3} b_{3,3} b_{2,2}$$
$$+ b_{1,2} b_{2,2}^{2} b_{1,1} b_{3,3}^{2} + b_{1,2} b_{3,1} b_{2,2} b_{1,3} b_{3,2} b_{2,3} - b_{1,2} b_{3,1} b_{2,2}^{2} b_{1,3} b_{3,3}$$
$$+ b_{3,1} b_{2,3} b_{1,2} b_{3,2}^{2} b_{2,1} + b_{3,2}^{3} b_{2,1}^{2} b_{1,3} - b_{2,2}^{2} b_{1,1} b_{3,3}^{2} b_{2,1} + b_{2,2}^{2} b_{1,1} b_{3,3} b_{2,3} b_{3,1}$$
$$+ b_{3,1} b_{2,2}^{2} b_{1,3} b_{2,1} b_{3,3} - b_{3,1}^{2} b_{2,2}^{2} b_{1,3} b_{2,3} + b_{2,2} b_{3,1}^{2} b_{2,3}^{2} b_{1,2} + b_{2,2} b_{2,1}^{2} b_{1,2} b_{3,3}^{2}$$
$$- 2 b_{2,2} b_{3,1} b_{2,3} b_{1,2} b_{2,1} b_{3,3} - 2 b_{3,2}^{2} b_{2,1} b_{1,3} b_{3,1} b_{2,2} - b_{2,1}^{2} b_{1,2} b_{3,3} b_{3,2}^{2}$$
$$+ b_{1,1} b_{3,2}^{2} b_{2,3} b_{3,1} b_{2,2} - b_{1,1} b_{3,2}^{3} b_{2,3} b_{2,1} + b_{2,2} b_{1,1} b_{3,3} b_{3,2}^{2} b_{2,1} + b_{3,2} b_{3,1}^{2} b_{2,2}^{2} b_{1,3}$$
$$- b_{2,2} b_{3,2} b_{2,1}^{2} b_{1,3} b_{3,3} + b_{2,2} b_{3,2} b_{2,1} b_{1,3} b_{2,3} b_{3,1} + b_{2,2} b_{1,1} b_{3,2} b_{2,3} b_{2,1} b_{3,3}$$
$$- b_{2,2} b_{1,1} b_{3,2} b_{2,3}^{2} b_{3,1}) e1 + (-b_{3,2} b_{3,1}^{2} b_{2,2} b_{1,3} b_{1,2} - b_{3,2} b_{2,1} b_{1,2}^{2} b_{3,3} b_{3,1}$$
$$+ b_{3,2} b_{2,2} b_{1,1} b_{3,3} b_{3,1} b_{1,2} + b_{1,1}^{2} b_{3,2}^{3} b_{2,3} + b_{3,1} b_{2,3} b_{1,2}^{2} b_{3,2} b_{1,3} - b_{3,1} b_{2,3} b_{1,2}^{3} b_{3,3}$$
$$- 2 b_{3,2} b_{2,1} b_{1,3} b_{3,3} b_{1,2}^{2} + b_{1,1} b_{3,2} b_{2,3} b_{3,3} b_{1,2}^{2} - b_{2,2} b_{1,1} b_{3,3}^{2} b_{1,2}^{2}$$
$$+ b_{3,1} b_{2,2} b_{1,3} b_{3,3} b_{1,2}^{2} + b_{1,2} b_{3,2}^{2} b_{2,1} b_{1,3}^{2} - b_{1,2} b_{1,1} b_{3,2}^{2} b_{2,3} b_{1,3}$$
$$+ b_{1,2} b_{2,2} b_{1,1} b_{3,3} b_{3,2} b_{1,3} - b_{1,2} b_{3,1} b_{2,2} b_{1,3}^{2} b_{3,2} - 2 b_{3,1} b_{2,3} b_{1,2} b_{3,2}^{2} b_{1,1}$$
$$+ b_{2,1} b_{1,2}^{3} b_{3,3}^{2} + b_{2,2}^{2} b_{1,1}^{2} b_{3,3}^{2} + b_{3,1}^{2} b_{2,2}^{2} b_{1,3}^{2} - 2 b_{2,2}^{2} b_{1,1} b_{3,3} b_{1,3} b_{3,1}$$
$$+ b_{2,2} b_{3,1} b_{2,3} b_{1,2} b_{1,1} b_{3,3} + b_{3,2}^{2} b_{2,1} b_{1,3} b_{3,1} b_{1,2} - b_{3,2}^{3} b_{2,1} b_{1,3} b_{1,1}$$
$$+ b_{2,1} b_{1,2} b_{3,3} b_{3,2}^{2} b_{1,1} - b_{2,2} b_{1,1}^{2} b_{3,3} b_{3,2}^{2} + b_{3,1} b_{2,2} b_{1,3} b_{3,2}^{2} b_{1,1} + b_{3,2} b_{3,1}^{2} b_{2,3} b_{1,2}^{2}$$
$$- b_{2,2} b_{3,1}^{2} b_{2,3} b_{1,2} b_{1,3} + b_{2,2} b_{3,2} b_{2,1} b_{1,3} b_{1,1} b_{3,3} - b_{2,2} b_{3,2} b_{2,1} b_{1,3}^{2} b_{3,1}$$
$$- b_{2,2} b_{2,1} b_{1,2} b_{3,3}^{2} b_{1,1} + b_{2,2} b_{2,1} b_{1,2} b_{3,3} b_{1,3} b_{3,1} - b_{2,2} b_{1,1}^{2} b_{3,2} b_{2,3} b_{3,3}$$
$$+ b_{2,2} b_{1,1} b_{3,2} b_{2,3} b_{1,3} b_{3,1}) e2 - (-b_{3,2} b_{3,1} b_{2,2} b_{1,3} b_{2,1} b_{1,2} + b_{3,2} b_{3,1} b_{2,2}^{2} b_{1,3} b_{1,1}$$
$$- b_{3,2} b_{3,1} b_{2,3} b_{1,2} b_{2,2} b_{1,1} + 2 b_{3,2} b_{2,1} b_{1,2} b_{3,3} b_{2,2} b_{1,1} + 2 b_{3,1} b_{2,3} b_{1,2}^{2} b_{2,2} b_{1,3}$$
$$- b_{3,2} b_{2,1} b_{1,3} b_{2,3} b_{1,2}^{2} - b_{2,1} b_{1,2}^{2} b_{3,3} b_{2,2} b_{1,3} + b_{2,1} b_{1,2}^{3} b_{3,3} b_{2,3} + b_{1,1} b_{3,2} b_{2,3}^{2} b_{1,2}^{2}$$

$$- b_{2,2}\, b_{1,1}\, b_{3,3}\, b_{2,3}\, b_{1,2}^{2} + b_{1,2}\, b_{3,2}\, b_{2,1}\, b_{1,3}^{2}\, b_{2,2} - b_{1,2}\, b_{1,1}\, b_{3,2}\, b_{2,3}\, b_{2,2}\, b_{1,3}$$
$$+ b_{1,2}\, b_{2,2}^{2}\, b_{1,1}\, b_{3,3}\, b_{1,3} - b_{1,2}\, b_{3,1}\, b_{2,2}^{2}\, b_{1,3}^{2} + b_{3,2}^{2}\, b_{2,1}^{2}\, b_{1,3}\, b_{1,2} - b_{3,2}^{2}\, b_{2,1}\, b_{1,3}\, b_{2,2}\, b_{1,1}$$
$$+ b_{2,2}^{2}\, b_{1,1}^{2}\, b_{3,3}\, b_{2,3} - b_{2,2}^{2}\, b_{1,1}\, b_{3,3}\, b_{1,3}\, b_{2,1} - b_{3,1}\, b_{2,2}^{2}\, b_{1,3}\, b_{1,1}\, b_{2,3} + b_{3,1}\, b_{2,2}^{2}\, b_{1,3}^{2}\, b_{2,1}$$
$$- b_{2,2}\, b_{3,2}\, b_{2,1}^{2}\, b_{1,3}^{2} - b_{2,2}\, b_{1,1}^{2}\, b_{3,2}\, b_{2,3}^{2} - b_{2,2}\, b_{3,1}\, b_{2,3}\, b_{1,2}\, b_{1,3}\, b_{2,1}$$
$$+ b_{2,2}\, b_{3,1}\, b_{2,3}^{2}\, b_{1,2}\, b_{1,1} - b_{3,1}\, b_{2,3}^{2}\, b_{1,2}^{3} - b_{1,1}\, b_{3,2}^{2}\, b_{2,3}\, b_{2,1}\, b_{1,2} + b_{1,1}^{2}\, b_{3,2}^{2}\, b_{2,3}\, b_{2,2}$$
$$- b_{3,2}\, b_{2,1}^{2}\, b_{1,2}^{2}\, b_{3,3} - b_{3,2}\, b_{2,2}^{2}\, b_{1,1}^{2}\, b_{3,3} + b_{3,2}\, b_{3,1}\, b_{2,3}\, b_{1,2}^{2}\, b_{2,1}$$
$$+ 2\, b_{2,2}\, b_{3,2}\, b_{2,1}\, b_{1,3}\, b_{1,1}\, b_{2,3} - b_{2,2}\, b_{2,1}\, b_{1,2}\, b_{3,3}\, b_{1,1}\, b_{2,3} + b_{2,2}\, b_{2,1}^{2}\, b_{1,2}\, b_{3,3}\, b_{1,3})\, e3$$

```
>
```

See Also: Bigebra:-`&map`, Bigebra:-peek, Bigebra:-poke, Bigebra:-switch

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-bracket - the bracket of Peano space (i.e. invariant theory)

Calling Sequence:

sc := bracket(c1,c2,c3,...)

Parameters:

- ci : Clifford polynomials.

Output:

- sc : a cliscalar

Global variables:

- dim_V : the dimension of the Peano space

Description:

- Let V be a k-linear space of finite dimension n. Let lower case x_i denote elements of V, which we will also call *letters*. We define *bracket* [.,...,] as an alternating multilinear scalar-valued map as follows:

  (i) [.,...,] : V x V ... x V --> k where V x V ... x V has n factors,

  (ii) [x_1, x_2,..., x_n] = sign(p) [x_p(1), x_p(2),..., x_p(n)],

  (iii) [x_1,..., \alpha*x_r + \beta y_r,..., x_n] = \alpha*[x_1,..., x_r,..., x_n] + \beta*[x_1,..., y_r,..., x_n].

  The sign is due to the permutation p on the arguments of the bracket. The pair P = (V,[.,...,]) is called a *Peano space*. See [1] and references therein.

- A *standard Peano* space is a Peano space over a k-linear space V of dimension n whose bracket has the additional property that for every vector x in V there exist vectors x_2,..., x_n such that

  [x, x_2,..., x_n] <> 0

- In such a space the length of the bracket, i.e., the number of entries, equals the dimension of the space, and conversely.

- In a standard Peano space, the bracket encodes linear independence: Suppose x_r = sum a_i*x_i, where the summation is over 1 <= i <=n, i <> r. Then

  [x_1,..., x_r,..., x_n] = [x_1,..., sum a_i*x_i,..., x_n] = sum a_i*[[x_1,..., x_i,..., x_n] = 0

due to the alternating nature of the bracket. Thus, a *basis* of V is any set of n vectors {e_1,..., e_n} whose bracket does not vanish. Furthermore, two vectors x and y in V are linearly independent if an only if there exist n - 2 vectors x_3,..., x_n such that [x, y, x_3,..., x_n] <> 0.

- A basis {e_1,e_2,...,e_n} for V is called *unimodular* (or, linearly ordered and normalized) if for the ordered list e_1, e_2,..., e_n, the bracket [e_1, e_2,..., e_n] = 1. The group which maps two linearly ordered bases onto another is GL_n, while the group which maps two unimodular bases is SL_n.

- For more information on how the bracket is used, see the Bigebra:-meet and Bigebra:-`&v` products.

Examples:

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

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

```
>
```

Set an arbitrary bilinear form to show that the bracket does not depend on this setting:

```
> dim_V:=4:
  n:=dim_V:
  B:=linalg[matrix](dim_V,dim_V,(i,j)->b[i,j]);
```

$$B := \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} \\ b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} \end{bmatrix}$$

```
> bracket(e1,e2,e3,e4),bracket(e2,e1,e3,e4),bracket(e3,e2,e4,e1),
  bracket(e4,e3,e1,e2);
```

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

$$1, -1, 1, -1$$

```
> bracket(e1,e1,e3,e4);
```

$$0$$

```
> x:=add(x||i*e||i,i=1..n);
  y:=add(y||i*e||i,i=1..n);
```

$$x := x1\ e1 + x2\ e2 + x3\ e3 + x4\ e4$$

$$y := y1\ e1 + y2\ e2 + y3\ e3 + y4\ e4$$

Projecting components out of the vector x:

```
> bracket(x,e2,e3,e4);
  bracket(e1,x,e3,e4);
```

```
bracket(e1,e2,x,e4);
bracket(e1,e2,e3,x);
```

$$x1$$
$$x2$$
$$x3$$
$$x4$$

Example 1: Plucker coordinates of projective lines in P3

Let L be a line in the 3-dimensional projective space P^3(k) = P^3. [2] Using the bracket we can find homogeneous coordinates for L. Suppose the line L is projectively parameterized using two linearly independent vectors x and y in k^4 (or, equivalently, two distinct points x <> y in P^3) defined above. Then, we can compute the following six polynomials in k[x1,x2,x3,x4,y1,y2,y3,y4]:

```
> w12:=bracket(x,y,e1,e2);
  w13:=bracket(x,y,e1,e3);
  w14:=bracket(x,y,e1,e4);
  w23:=bracket(x,y,e2,e3);
  w24:=bracket(x,y,e2,e4);
  w34:=bracket(x,y,e3,e4);
```

$$w12 := -x4\, y3 + x3\, y4$$
$$w13 := x4\, y2 - x2\, y4$$
$$w14 := -x3\, y2 + x2\, y3$$
$$w23 := x1\, y4 - x4\, y1$$
$$w24 := x3\, y1 - x1\, y3$$
$$w34 := -x2\, y1 + x1\, y2$$

These six polynomials define a vector \omega(x,y) = (w12, w13, w14, w23, w24, w34) in k^6 whose components are the *Plucker coordinates* of the line L. It is well known that omega(x,y) gives a unique point in P^5 which depends only on L. Hence, every line L in P^3 determines a well-defined point \omega(L) in P^5.

It is also well known that the Plucker coordinates of L satisfy the following relation:

```
> simplify(w12*w34-w13*w24+w14*w23);
```

$$0$$

This relation like other below can be found using the elimination theory applied to Groebner bases. [2] This requirese that the above six defining equations be written as six polynomials f1, f2, f3, f4, f5, f6 in the polynomial ring R = R[x1,x2,x3,x4,y1,y2,y3,y4,w12,w13,w14,w23,w24,w34]. Then, one computes a Groebner basis G for the ideal I = <f1, f2, f3, f4, f5, f6> in R and later for the 8th elimination ideal I8 = I \cap R[w12,w13,w14,w23,w24,w34]. It is well known that the Groebner basis G8 for I8 is obtained

from G by keeping only those polynomials from G which belong to R[w12,w13,w14,w23,w24,w34]. We need to use an elimination order lexdeg to find G: For example, we could use the lex order x1>x2>x3>x4>y1>y2>y3>y4>w12>w13>w14>w23>w24>w34 or another, more efficient, elimination order.

```
> with(Groebner):
> w12,w13,w14,w23,w24,w34:='w12','w13','w14','w23','w24','w34':
> f1:=w12-bracket(x,y,e1,e2);
  f2:=w13-bracket(x,y,e1,e3);
  f3:=w14-bracket(x,y,e1,e4);
  f4:=w23-bracket(x,y,e2,e3);
  f5:=w24-bracket(x,y,e2,e4);
  f6:=w34-bracket(x,y,e3,e4);
```

$$f1 := w12 + x4\, y3 - x3\, y4$$
$$f2 := w13 - x4\, y2 + x2\, y4$$
$$f3 := w14 + x3\, y2 - x2\, y3$$
$$f4 := w23 - x1\, y4 + x4\, y1$$
$$f5 := w24 - x3\, y1 + x1\, y3$$
$$f6 := w34 + x2\, y1 - x1\, y2$$

```
> F:=[seq(f||i,i=1..6)];
```

$$F := [w12 + x4\, y3 - x3\, y4, w13 - x4\, y2 + x2\, y4, w14 + x3\, y2 - x2\, y3, w23 - x1\, y4 + x4\, y1, w24 - x3\, y1 + x1\, y3, w34 + x2\, y1 - x1\, y2]$$

```
> G:=Basis(F,plex(x1,x2,x3,x4,y1,y2,y3,y4,w12,w13,w14,w23,w24,w34
  ));
  G1:=Basis(F,lexdeg([x1,x2,x3,x4,y1,y2,y3,y4],[w12,w13,w14,w23,w
  24,w34]));
  nops(G);
  g:=G[1];
```

$$G := [w23\, w14 + w34\, w12 - w13\, w24, -w23\, w14\, y2 + y2\, w13\, w24 + w34\, y3\, w13 + w34\, w14\, y4, w13\, y3 + w14\, y4 + w12\, y2, w34\, y3 + w14\, y1 + w24\, y2, w13\, y1 - w34\, y4 + w23\, y2, -w23\, y3 + w12\, y1 - w24\, y4, -w12 - x4\, y3 + x3\, y4, -w23\, x2\, w14 + x2\, w13\, w24 + w34\, w14\, x4 + w34\, w13\, x3, w14\, x4 + w12\, x2 + w13\, x3, w13 - x4\, y2 + x2\, y4, -w14 - x3\, y2 + x2\, y3, w24\, x2 + w34\, x3 + w14\, x1, -w34\, x4 + w23\, x2 + w13\, x1, -w24\, x4 - w23\, x3 + w12\, x1, -w23 + x1\, y4 - x4\, y1, w24 - x3\, y1 + x1\, y3, -w34 - x2\, y1 + x1\, y2]$$

$$G1 := [w23\, w14 + w34\, w12 - w13\, w24, w13\, y3 + w14\, y4 + w12\, y2, w34\, y3 + w14\, y1 + w24\, y2, w13\, y1 - w34\, y4 + w23\, y2, -w23\, y3 + w12\, y1 - w24\, y4,$$

$w14\,x4 + w12\,x2 + w13\,x3,\ w24\,x2 + w34\,x3 + w14\,x1,\ -w34\,x4 + w23\,x2 + w13\,x1,$
$-w24\,x4 - w23\,x3 + w12\,x1,\ w12 + x4\,y3 - x3\,y4,\ -w13 + x4\,y2 - x2\,y4,\ w14 + x3\,y2 - x2\,y3,$
$w23 - x1\,y4 + x4\,y1,\ x3\,y1 - x1\,y3 - w24,\ w34 + x2\,y1 - x1\,y2]$

17

$$g := w23\,w14 + w34\,w12 - w13\,w24$$

The above shows that the Groebner basis G contains 17 polynomials of which only one, the first one in the above list, generates I8 = <g>. We could use the auxiliary package RJgrobner to isolate that one polynomial. Type ?RJgrobner for more help. Polynomial g provides a *syzygy relation* between the new variables w12, w13, w14, w23, w24, w34.

```
> with(RJgrobner);
```

[*Beziercubic*, *Gbasis*, *GbasisL*, *GreatestCommonDivisor*, *IntersectionOfIdeals*, *LeastCommonMultiple*, *ProductOfIdeals*, *QuotientOfIdeals*, *RJversion*, *RadicalMembership*, *SINGULARlink*, *Spoly*, *SumOfIdeals*, *completelyreducedGbasis*, *condition*, *condition2*, *condition3*, *curvature*, *homogenize*, *implicitBeziercubic*, *maxdegree*, *maxmindegree*, *minimalGbasis*, *reducedGbasis*, *reducepol*, *reduction*]

```
> GbasisL(G,[x1,x2,x3,x4,y1,y2,y3,y4]);
```

$$[w23\,w14 + w34\,w12 - w13\,w24]$$

If we now let L vary over all possible projective lines in P^3, the points \omega(L) will belong to a nonsingular quadric V(z12*z34-z13*z24+z14*z23) in P^5 where by (z12, z13, z14, z23, z24, z34) we mean the homogeneous coordinates in P^5. It is well known that the points in this quadric are in bijective correspondence with the set of lines in P^3. [1]

```
>
> dim_V:=5:
  n:=dim_V:
  B:=linalg[matrix](dim_V,dim_V,(i,j)->b[i,j]);
```

$$B := \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} & b_{3,5} \\ b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} & b_{4,5} \\ b_{5,1} & b_{5,2} & b_{5,3} & b_{5,4} & b_{5,5} \end{bmatrix}$$

```
> bracket(e1,e2,e3,e4,e5),bracket(e2,e1,e3,e4,e5),bracket(e3,e2,e
  4,e5,e1),bracket(e4,e3,e5,e1,e2);
```

1, -1, -1, -1

```
> bracket(e1,e1,e3,e4,5);
```

0

```
> x:=add(x||i*e||i,i=1..n);
  y:=add(y||i*e||i,i=1..n);
```

$$x := x1\,e1 + x2\,e2 + x3\,e3 + x4\,e4 + x5\,e5$$

$$y := y1\ e1 + y2\ e2 + y3\ e3 + y4\ e4 + y5\ e5$$

Projecting components out of the vector x:

```
> bracket(x,e2,e3,e4,e5);
  bracket(e1,x,e3,e4,e5);
  bracket(e1,e2,x,e4,e5);
  bracket(e1,e2,e3,x,e5);
  bracket(e1,e2,e3,e4,x);
```

$$x1$$
$$x2$$
$$x3$$
$$x4$$
$$x5$$

Example 2: Plucker coordinates for projective lines in P^4

Let L be a line in the 4-dimensional projective space P^4(k) = P^4. [1, problem 17, p. 412] Using the bracket we can find homogeneous coordinates for L. Suppose the line L is projectively parameterized using two linearly independent vectors x and y in k^5 (or, equivalently, two distinct points x <> y in P^4) defined above. Then, we can compute the following ten polynomials in k[x1,x2,x3,x4,x5,y1,y2,y3,y4,y5,w123,w124,w125,w134,w135,w145,w234,w235,w245,w345]:

```
> f1:=w345-bracket(x,y,e3,e4,e5);
  f2:=w245-bracket(x,y,e2,e4,e5);
  f3:=w235-bracket(x,y,e2,e3,e5);
  f4:=w234-bracket(x,y,e2,e3,e4);
  f5:=w145-bracket(x,y,e1,e4,e5);
  f6:=w135-bracket(x,y,e1,e3,e5);
  f7:=w134-bracket(x,y,e1,e3,e4);
  f8:=w125-bracket(x,y,e1,e2,e5);
  f9:=w124-bracket(x,y,e1,e2,e4);
  f10:=w123-bracket(x,y,e1,e2,e3);
```

$$f1 := w345 + x2\ y1 - x1\ y2$$
$$f2 := w245 + x1\ y3 - x3\ y1$$
$$f3 := w235 + x4\ y1 - x1\ y4$$
$$f4 := w234 + x1\ y5 - x5\ y1$$
$$f5 := w145 + x3\ y2 - x2\ y3$$
$$f6 := w135 + x2\ y4 - x4\ y2$$
$$f7 := w134 + x5\ y2 - x2\ y5$$
$$f8 := w125 + x4\ y3 - x3\ y4$$
$$f9 := w124 - x5\ y3 + x3\ y5$$

$$f10 := w123 + x5\,y4 - x4\,y5$$

Thus, these ten polynomials define a vector \omega(x,y) = (w123, w124,w125,w134,w135,w145,w234,w235,w245,w345) in k^10 whose components are the *Plucker coordinates* of the line L. It is well known that omega(x,y) gives a unique point in P^9 which depends only on L. Hence, every line L in P^4 determines a well-defined point \omega(L) in P^9

Next, we proceed to finding the syzygy relations between these ten Plucker coordinates of L.

```
> F:=[seq(f||i,i=1..10)];
```

$$F := [w345 + x2\,y1 - x1\,y2,\ w245 + x1\,y3 - x3\,y1,\ w235 + x4\,y1 - x1\,y4,\ w234 + x1\,y5 - x5\,y1,\ w145 + x3\,y2 - x2\,y3,\ w135 + x2\,y4 - x4\,y2,\ w134 + x5\,y2 - x2\,y5,\ w125 + x4\,y3 - x3\,y4,\ w124 - x5\,y3 + x3\,y5,\ w123 + x5\,y4 - x4\,y5]$$

We will use FGb package (type ?FGb for help) to compute a Groebner basis for the ideal I = <f1,f2,...,f10> because it is faster than Maple's Groebner package. We will also use the elimination monomial order by separating the entire list of variables into two disjoint lists.

```
> with(FGb);
Path set to C:\Maple11\bin.win/libfgbuni.so
    FGb/Maple interface package Version 1.34
    JC Faugere (jcf@calfor.lip6.fr)
    Type ?FGb for documentation
```

$$[ModuleUnload, cpu, date, estime\_size, fgb\_debug, fgb\_exit, fgb\_gbasis, fgb\_gbasis\_elim, fgb\_gbasis\_lm, fgb\_hilbert, fgb\_interface, fgb\_minpoly\_sqfr, fgb\_nop, fgb\_normalForm, fgb\_radical2, fgb\_radical2\_quiet, fgb\_rr, fgb\_rr\_crit, fgb\_rr\_sqfr, pack\_fgb\_init]$$

```
> G:=fgb_gbasis(F,0,[x1,x2,x3,x4,x5,y1,y2,y3,y4,y5],[w123,w124,w1
  25,w134,w135,w145,w234,w235,w245,w345]);
  nops(G);
```

$$G := [w345 + x2\,y1 - x1\,y2,\ x3\,y1 - x1\,y3 - w245,\ w235 + x4\,y1 - x1\,y4,\ x5\,y1 - x1\,y5 - w234,\ w145 + x3\,y2 - x2\,y3,\ x4\,y2 - x2\,y4 - w135,\ w134 + x5\,y2 - x2\,y5,\ w125 + x4\,y3 - x3\,y4,\ x5\,y3 - x3\,y5 - w124,\ w123 + x5\,y4 - x4\,y5,\ x1\,w123 + x4\,w234 + x5\,w235,\ x1\,w124 - x3\,w234 + x5\,w245,\ x1\,w125 - x3\,w235 - x4\,w245,\ x1\,w134 + x2\,w234 + x5\,w345,\ x1\,w135 + x2\,w235 - x4\,w345,\ x1\,w145 + x2\,w245 + x3\,w345,\ x2\,w123 - x4\,w134 - x5\,w135,\ x2\,w124 + x3\,w134 - x5\,w145,\ x2\,w125 + x3\,w135 + x4\,w145,\ x3\,w123 + x4\,w124 + x5\,w125,\ y1\,w123 + y4\,w234 + y5\,w235,\ y1\,w124 - y3\,w234 + y5\,w245,\ y1\,w125 - y3\,w235 - y4\,w245,\ y1\,w134 + y2\,w234 + y5\,w345,\ y1\,w135 + y2\,w235 - y4\,w345,\ y1\,w145 + y2\,w245 + y3\,w345,\ y2\,w123 - y4\,w134 - y5\,w135,\ y2\,w124 + y3\,w134 - y5\,w145,\ y2\,w125 + y3\,w135 + y4\,w145,\ y3\,w123 + y4\,w124 + y5\,w125,\ w125\,w134 - w124\,w135 + w123\,w145,\ w125\,w234 - w124\,w235 + w123\,w245,\ w135\,w234 - w134\,w235 + w123\,w345,\ w145\,w234 - w134\,w245 + w124\,w345,$$

$w145\ w235 - w135\ w245 + w125\ w345]$

35

```
> G10:=GbasisL(G,[x1,x2,x3,x4,x5,y1,y2,y3,y4,y5]);
  nops(G10);
```

$G10 := [w125\ w134 - w124\ w135 + w123\ w145, w125\ w234 - w124\ w235 + w123\ w245,$
$w135\ w234 - w134\ w235 + w123\ w345, w145\ w234 - w134\ w245 + w124\ w345,$
$w145\ w235 - w135\ w245 + w125\ w345]$

5

Thus, we have found five syzygy relations between the ten coordinates:

```
> s1:=G10[1];
  s2:=G10[2];
  s3:=G10[3];
  s4:=G10[4];
  s5:=G10[5];
```

$s1 := w125\ w134 - w124\ w135 + w123\ w145$

$s2 := w125\ w234 - w124\ w235 + w123\ w245$

$s3 := w135\ w234 - w134\ w235 + w123\ w345$

$s4 := w145\ w234 - w134\ w245 + w124\ w345$

$s5 := w145\ w235 - w135\ w245 + w125\ w345$

Like for the lines in P^3, the point \omega(x,y) is unique and it belongs to a variety V contained in P^4. Furthermore, it can be shown that there exists a bijection between the set of projective lines L in P^4 that sends each line L to \omega(L) in V, and the set of points of the variety V. [2]

Thus, we see that bracket gives Plucker coordinates for linear varieties L in P^n while the set \omega(L) can be given a structure of a projective variety defined through one, when n = 3, five, when n = 4, or more, when n > 4, syzygy relations. These projective varieties are called *Grassmannians*. [2]

```
>
```

Example 3: Some more properties of 'bracket'

```
> bracket(&w(e1,e2,e3,e4)),bracket(&w(e1,e2),&w(e3,e4)),
  bracket(e1,&w(e2,e3),e4),bracket(e1,e2,e3,e4);
  bracket(e1),bracket(e2we3),bracket(e2we3we4);bracket(e1,e2);
```

0, 0, 0, 0

0, 0, 0

0

```
> &t(e1we3we4,e2+e3,e2+e3,e4);
  contract(%,1,bracket);
```

$\&t(e1we3we4, e2, e2, e4) + \&t(e1we3we4, e2, e3, e4) + \&t(e1we3we4, e3, e2, e4)$

$+ \&t(e1we3we4, e3, e3, e4)$

$$0$$

```
> printf("Worksheet took %f seconds on Intel Pentium M 2.13 GHz
  2GB RAM WinXP Prof\n",time()-bench);
Worksheet took 4.736000 seconds on Intel Pentium M 2.13 GHz 2GB RAM WinXP Prof
>
```

References:

[1] B. Fauser, "A Treatise on Quantum Clifford Algebras", Habilitation Thesis, Universitat Konstanz, Fachbereich Physik, Fach M 678, 78457 Konstanz, January 25, 2002 [arXiv:math.QA/0202059 v1 7 Feb 2002]

[2] D. Cox, J. Little, D. and O'Shea: "Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra", Springer, 1997.


```
>
```

Algorithm:

See Also: Bigebra:-contract, Bigebra:-`&v`, Bigebra:-meet, Bigebra help page, Bigebra:-pairing, Bigebra:-EV

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-contract - contraction of adjacent slots of a tensor, lowers the tensor rank by two.

Calling Sequence:

t2 := contract(t1,i,f)

Parameters:

- t1 : a tensor polynomial.
  i : the i-th tensor slot to act on the pair (i,i+1).
  f : a scalar valued function of two Clifford polynomials; f : /\V x /\V --> k.

Output:

- t2 : the contracted tensor polynom.

Description:

- The contraction is needed to contract tensor polynoms by arbitrary 2->0 mappings 'f'. Depending on the type of the mapping f, the contraction can be a contraction in the sense of tensor calculus, i.e. like a sum over co- and contravariant indices, or like a (co-) scalar product acting on two indices of the same valence (co- or contravariant). For more information see [3]. The contraction can be seen as cup tangle, i.e. 2->0 but on yet unoriented tangles.
- Possible scalar-valued functions of the Bigebra package are, bracket@wedge, or bracket with two arguments, pairing, EV, scalarpart@cmul, etc...
- A more detailed help page is planed in the future.

Examples:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

Set the dimension and a symmetric bilinear form

```
> dim_V:=4:B:=linalg[matrix](dim_V,dim_V,(i,j)->abs(i-5/j));
```

$$B := \begin{bmatrix} 4 & \frac{3}{2} & \frac{2}{3} & \frac{1}{4} \\ 3 & \frac{1}{2} & \frac{1}{3} & \frac{3}{4} \\ 2 & \frac{1}{2} & \frac{4}{3} & \frac{7}{4} \\ 1 & \frac{3}{2} & \frac{7}{3} & \frac{11}{4} \end{bmatrix}$$

```
> contract(&t(e1,e2),1,scalarpart@cmul);
  contract(&t(e1,e2,e3),2,scalarpart@cmul);
```

```
contract(&t(a*e1,e2-b*e4,e3),1,scalarpart@cmul);
contract(&t(e1),2,scalarpart@cmul);  # ERROR -> only rank one
tensor
```

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

$$\frac{3}{2}$$

$$\frac{1}{3}\,\&t(e1)$$

$$\frac{3}{2}\,a\,\&t(e3)-\frac{1}{4}\,a\,b\,\&t(e3)$$

```
Error, (in Bigebra:-contract) Tensor with two args at least needed
```

```
> contract(&t(e1we2,e3we4),1,bracket@wedge);
contract(&t(e1,e2we3we4,y*e1-e3),2,bracket@wedge);
contract(&t(e1we2,e3we4)-&t(a*e1we3,e2we4-b*e4,e3),1,bracket@we
dge);
contract(&t(e1,e2,e3,e4),2,bracket@wedge);
contract(&t(e1,e2,e3,e4),5,pairing);  # ==> ERROR <== no 5th
and 5th tensor slot
```

$$1$$

$$-y\,\&t(e1)$$

$$1+a\,\&t(e3)$$

$$0$$

```
Error, (in Bigebra:-peek) improper op or subscript selector
```

```
> contract(&t(e1we2,e3we4),1,pairing);
contract(&t(e1,e2we3we4,y*e1-e3),2,pairing);
contract(&t(e1we2,e3we4)-&t(a*e1we3,e2we4-b*e4,e3),1,pairing);
contract(&t(e1,e2,e3,e4),2,pairing);
```

$$\frac{-5}{12}$$

$$0$$

$$-\frac{5}{12}+\frac{5}{2}\,a\,\&t(e3)$$

$$\frac{1}{3}\,(e1\ \&t\ e4)$$

```
> contract(&t(e1we2+a*e3we4,e3we4-y*e1we2),1,EV);
contract(&t(e1,e2we3we4,y*e1-e3),2,scalarpart@meet);
contract(&t(e1we2,e3we4)-&t(a*e1we3,e2we4-b*e4,e3),1,scalarpart
@LC);
```

```
contract(&t(e1,Id,Id,e4),2,scalarpart@wedge);
```

$$-y + a$$

$$y \,\&t(e1)$$

$$-\frac{5}{12} + \frac{5}{2} a \,\&t(e3)$$

$$e1 \,\&t\, e4$$

```
> printf("Worksheet to %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM WinXP Prof",time()-bench);
```

```
Worksheet to 0.343000 seconds to compute on AMD Athlon 2700+ 1GB RAM WinXP Prof
```

```
>
```

See Also: Bigebra help page, Bigebra:-meet, Bigebra:-`&v`, Bigebra:-pairing, Bigebra:-EV, CLIFFORD help page

Last modified: December 20, 2007 /BF/RA.

Function: Define:-define and Define:-definemore -- (partially) patched version of Maple 6's define.

Calling Sequence:

define(op,prop1,prop2,...)

Parameters:

- op : the name of an operator (or function) which is defined by 'define', e.g. `&t`, i.e. the BIGEBRA tensor product.
  prop.i : properties like 'multilinear', 'flat' (which we have tested) and functional programming.
  new : an option domain may be specified to use k-linearity over a ring k *domain={type}*

Output:

- none.

Warning:

- multilinearity works **only** in the associative (Maple's flat option) case.

Description:

- Maple's define has bugs so it cannot be used for our purpose. For this reason, Bigebra installs a patch to 'define' which allows to use the 'flat' (***associativity***) and 'multilinear' properties. The name of such an 'defined' operator is usually composed by an &(ampersand) followed by some name, i.e. &t for the tensor product, but see the tensor product help page for some peculiarities of the &(ampersand) operators, which may case errors too if used in its ***infix form***.
- A second reason for patching define is that the multilinear option handles constants not in an appropriate way. The multilinear option considers beside numbers (integers, floats, ...) any other algebraic expression as 'non constant', i.e. keeps it inside of the user-defined &(ampersand) function/operator. Since we want to consider e.g. modules, this behavior is not convenient. In fact, a tensor product defined by define (after having fixed the errors) is only a Z-tensorproduct or an R-tensorproduct, if floats are used. Integer, real or complex **variables** have to be added to the 'constants' list painfully. This was inconvenient.
- The user can define with our 'define' new associative multilinear operators over a *(common!)* domain which he specifies by overwriting the type cliscalar of the CLIFFORD package. Any element which has type 'cliscalar' is then considered as a constant and put in front or the multilinear function. In this way, modules, varieties, polynomial rings in several variables (the option orderless was not tested, but is quite trivial an should work) etc. can be 'defined'. That is one defines a tensor product over the 'cliscalars'.
- In further upcoming versions of BIGEBRA (for Maple 6) the user can supply an arbitrary domain. From Maple 7 on, local types are possible and BIGEBRA will allow then to have

separate domains for any user defined &(ampersand) operator.

Examples:

```
> restart:
  libname:=libname[2..-1]; # make sure we are not using the
  patched define from <Cliffordlib> libary
```

$$libname := \text{"C:\Maple11/lib", "C:\Maple11/SPlib", "C:\laylinalg"}$$

BUG in multilinear option:

Define &r as a multilinear function

```
> define(`&r`,multilinear);
```

Check some cases which work. Note that `false` and `true` compute like constants.

```
> &r(-e1);                    # but see flat,multilinear below
  &r(2*e1,3*e2+5*e3);
  &r(a*e1,e2-e3,e4+e5);       # note a is not handled here
  constants:=constants,a;     # add a to the constants
  &r(a*e1,e2-e3,e4+e5);       # now a is put in front
  &r(false*e1,true*e2);       # this 'works' since false,true are
  'constants'
```

$$-\&r(e1)$$

$$6\,(e1\ \&r\ e2) + 10\,(e1\ \&r\ e3)$$

$$\&r(a\ e1, e2, e4) + \&r(a\ e1, e2, e5) - \&r(a\ e1, e3, e4) - \&r(a\ e1, e3, e5)$$

$$constants := false, \gamma, \infty, true, Catalan, FAIL, \pi, a$$

$$a\ \&r(e1, e2, e4) + a\ \&r(e1, e2, e5) - a\ \&r(e1, e3, e4) - a\ \&r(e1, e3, e5)$$

$$false\ true\,(e1\ \&r\ e2)$$

```
> e:=exp(1);evalf(e);
  &r(e*e1,2*e1);
```

$$e := \mathbf{e}$$

$$2.718281828$$

$$2\,\mathbf{e}\,(e1\ \&r\ e1)$$

But see this (loops until out of memory or to many levels of recursion):

```
> &r(2.5*e1);  # bug loops infinitely
```

$$2.5\ \&r(1.000000000\ e1)$$

Let's see what happens if we use negative elements

```
> &r(-e1,-e2,e3); # &r(e1,e2,e3) expected
```

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

```
> &r(-e1,-e2);    # &r(e1,e2) expected
```

$$e1\ \&r\ e2$$

```
> &r(&r(-e1));     # -&r(&r(e1)) expected
```

$$-\&r(\&r(e1))$$

This renders the multilinear function to be useless, even the treatment of constants is poor, since any multilinear function has to be defined over some ring which has to be specified, but cannot be so in Maple's define.

Let us check the flat option:

```
> restart:
  libname:=libname[2..-1]; # make sure we are not using the
  patched define from <Cliffordlib> libary
  define(`&r`,flat);
```

$$libname := \text{"C:\Maple11/lib", "C:\Maple11/SPlib", "C:\laylinalg"}$$

```
> &r(&r(e1,e2),e3);
  &r(&r(e1,e2),&r(e2,e3));
```

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

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

```
> &r(-e1,&r(e2));
  &r(&r(e1));
```

$$(-e1)\ \&r\ e2$$

$$\&r(e1)$$

Old Maple 5 bug partially fixed :

```
> &r(-e1);
```

$$\&r(-e1)$$

```
> restart:
  libname:=libname[2..-1]; # make sure we are not using the
  patched define from <Cliffordlib> libary
  define(`&r`,flat,multilinear);
  &r(-&r(e1),-&r(e2));   # bug, does not work out anything
```

$$libname := \text{"C:\Maple11/lib", "C:\Maple11/SPlib", "C:\laylinalg"}$$

$$e1\ \&r\ e2$$

A good deal of Maple 5 bugs are resolved in Maple 6 we check for :

```
> &r(&r(e1,-e2)); # expected -(e1 &r e2)
```

$$-(e1\ \&r\ e2)$$

```
> &r(&r(-e1));     # expected -&r(e1)
```

$$-\&r(e1)$$

```
> &r(-e1);         # expected -&r(e1)
```

$$-\&r(e1)$$

Now we load the Bigebra package which uses a patched 'define':

```
> restart:bench:=time():with(Clifford):with(Bigebra):
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> define(`&r`,multilinear);
> &r(2.5*e1);    # works now
  &r(&r(-e1));   # OK now
```

$$2.5\ \&r(e1)$$

$$-\&r(\&r(e1))$$

Note, that flat, multilinear works, but ***we still cannot define non associative*** operators! This will probably be fixed in a new version of Bigebra. ***However***, the tensor product (already defined in Bigebra) is associative i.e. ***flat and multilinear*** hence it works out correctly:

```
> &t(2.5*e1);
  &t(&t(e1));
  &t(&t(2.5*e1));
  &t(2.5*e1,e2);
```

$$2.5\ \&t(e1)$$

$$\&t(e1)$$

$$2.5\ \&t(e1)$$

$$2.5\ (e1\ \&t\ e2)$$

We repeat the above examples which did not work out there in some cases (and were wrongly computed in maple 5):

```
> &t(-e1,-e2);     # (-e1) &r (-e2) formerly
```

$$e1\ \&t\ e2$$

```
> &t(&t(-e1));     # formerly Error, too many levels of recursion
```

$$-\&t(e1)$$

```
> &t(-&t(e1),-&t(e2));   # formerly Error, too many levels of
  recursion
```

$$e1\ \&t\ e2$$

No additional tensor slots are created now:

```
> &t(&t(e1,-e2)); # formerly -&t(1,e1,e2)
```

$$-(e1\ \&t\ e2)$$

```
> &t(&t(-e1));     # formerly -&t(1,1,e1)
  &t(&t(e1));
```

$$-\&t(e1)$$

$$\&t(e1)$$

```
> &t(-e1);         # formerly -(1 &t e1)
```

$$-\&t(e1)$$

This shows examples, but can ***not prove*** `&t` to be a flat multilinear operator. However, Bigebra's tensor product was tested in numerous huge worksheets (running for days using over 700MB RAM) and produced mathematically reasonable and afterwards (manually) provable output, so we are convinced to have a safe device.

User defined types:

The user may use the type cliscalar to define tensor products i.e. associative (i.e. flat) multilinear functions himself. For this purpose, he has to re-define the type 'cliscalar' to check for the ring he wants to use. Below we define tenor products T1 and T2 over the polynomial ring in x and the Laurent polynomials in the variables x,y,z. It is also possible to redefine a global variable _scalartypes from CLIFFORD. See Cliff5[CLIFFORD_ENV] for more information.

Example 1: Unprotect the type cliscalar which was protected by CLIFFORD. Define a new operator &T1 to be associative and multilinear. Re-define the scalars to be polynomials over the integers in the variable x, i.e. belong to Z[[x]].

```
> unprotect(`type/cliscalar`): # type/cliscalar was protected by
  CLIFFORD
  `type/mydomain`:=proc(expr) type(expr,polynom(integer,x)) end:
  define(`&T1`,flat,multilinear,domain='mydomain');
```

Check some types to be in our domain (Z[[x]] = `cliscalar`):

```
> type(2*x+3*x^2+5,mydomain);  # true, since in Z[[x]]
  type(x/2,mydomain);          # false, no fractions allowed
  type(2*y^2+3*y-1,mydomain);  # false, polynomial variable is
  not x
```

$$true$$
$$false$$
$$false$$

Some examples of tensor products:

```
> collect(&T1(3*x^2-x+5,x-x^4),`&T1`);
  collect(&T1(3*y*x^2-x+5,x*y-x^4),`&T1`);
```

$$(3\,x^3 - 3\,x^6 - x^2 + x^5 + 5\,x - 5\,x^4)\,(1\ \&T1\ 1)$$
$$(x^5 - 5\,x^4)\,(1\ \&T1\ 1) + 3\,x^3\,(y\ \&T1\ y) - 3\,x^6\,(y\ \&T1\ 1) + (-x^2 + 5\,x)\,(1\ \&T1\ y)$$

A second example, using algebraic functions as domain, i.e. define a variety:

```
> unprotect(`type/cliscalar`):  # was done already above
  `type/mypolydom`:=proc(expr) type(expr,
  radalgfun(rational,[x,y,z])) end:
> define(`&T2`,flat,multilinear,domain='mypolydom');
> type(x/(1-x*y),mypolydom); # true, belons to the ring
  k[[x,y,x^(-1),y^(-1)]]
  type(sin(x),mypolydom);    # false, transcendental
  type(a^2,mypolydom);       # false, belons to the polynomial
  ring k[[a]]
  type(y^3,mypolydom);       # true, belons to the polynomial
  ring k[[y]]
```

$$true$$
$$false$$
$$false$$
$$true$$

```
> &T2(x/(1-x*y),sin(x)*b,a^2+y^3*b);
```

$$\frac{x\,\&\mathrm{T2}(1,\sin(x)\,b,a^2)}{1-x\,y}+\frac{x\,y^3\,\&\mathrm{T2}(1,\sin(x)\,b,b)}{1-x\,y}$$

Example 2: Symmetric tensor products:

```
> unprotect(`type/cliscalar`):  # was done already above
  `type/myinteger`:=proc(expr) type(expr, integer) end:
> if assigned(`&S`) then unassign(`&S`) fi:
  define(`&S`,flat,multilinear,orderless,domain='myinteger');
> &S(x1,x2), &S(x2,x1);
  &S(x1+x2,2*x2+x1+x3);
```

$$x1\ \&\mathrm{S}\ x2,\ x1\ \&\mathrm{S}\ x2$$
$$3\,(x1\ \&\mathrm{S}\ x2)+2\,(x2\ \&\mathrm{S}\ x2)+(x1\ \&\mathrm{S}\ x1)+(x1\ \&\mathrm{S}\ x3)+(x2\ \&\mathrm{S}\ x3)$$

```
> &S(x1,x2,&S(x1,x2));
```

$$\&\mathrm{S}(x1,x1,x2,x2)$$

Now we define a (crude) symmetric product on these monomials:

```
> `&s`:=proc()
     local lst, st, res, i, fun;
     if nargs=0 then RETURN(0) fi;
     if nargs=1 then RETURN(`'&s'`(args)); fi;
  ### WARNING: note that `I` is no longer of type `^`
     fun:=proc(a1) local k; if type(a1,`^`) then
  seq(op(1,a1),k=1..op(2,a1)) else a1 fi end:
     lst:=sort(map(fun,[args]),address);
     st:={op(lst)};
     res:=[]:
     for i from 1 to nops(st) do
       res:=[op(res),st[i]^(nops(select(has,lst,st[i])))]; ;
     od:
     RETURN(`&S`(op(sort(res,address))));
  end:
> &s(x1^2,x1,x2^2);
  &s(x1,x3,x1^2,x1,x2^3);
  &s(2*x1+x2,x2,x1+x2);
```

$$x2^2\ \&\mathrm{S}\ x1^3$$
$$\&\mathrm{S}(x3,x2^3,x1^4)$$

$$2\ \&S(x1, x1, x2^3) + 3\ \&S(x1, x2, x2^3) + \&S(x2, x2, x2^3)$$

This allows us to multiply symmetric functions by applying the multiplication to monoms:

Let s123=x1+x2+x3 be a symmetric polynomial we compute s123 * s123:

```
> eval(subs(`&S`=`&s`,&S(x1+x2+x3,x1+x2+x3)));
```

$$\&S(x1^2) + 2\ (x1\ \&S\ x2) + 2\ (x1\ \&S\ x3) + \&S(x2^2) + 2\ (x2\ \&S\ x3) + \&S(x3^2)$$

Or compute s1^2 &s s123

```
> eval(subs(`&S`=`&s`,&S(x1^2,x1+x2+x3)));
```

$$\&S(x1^3) + (x2\ \&S\ x1^2) + (x3\ \&S\ x1^2)$$

The results are well known from the theory of symmetric functions:

s123 * s123 = s1^2 + s2^2 + s3^2 + 2* s12 + 2* s13 + 2*s23
s1^2 * s123 = s1^3 + s(1^2)2 + s(1^2)3

As a further device these results should be expanded in a canonical basis of symmetric polynomials, which can be done by the user!

NOTE: This device has to be ***used with great care***, since it was not well tested! Certain definitions of types may interfere with the abilities of define.

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM WinXP Prof",time()-bench);
Worksheet took 0.016000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
WinXP Prof
>
```

See Also: define (by Maple), Bigebra help, Bigebra:-`&t`, `type/cliscalar`

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-drop_t - drops the tensor symbol &t from tensors of rank one

Calling Sequence:

c1 := drop_t(p1)

Parameters:

- p1 : is a tensorpolynom, generally of rank one (but drop_t works on general elements)

Output:

- c1 : a Clifford polynomial or an expression sequence if the tensor is of rank > 1

Description:

- For computational reasons it was convenient to stay with expressions like &t(Id) or &t(e1we2), even if those elements are Clifford polynomials. To get rid of these tensor symbol &t, use drop_t.
- Drop_t is a helper function, that was useful in some worksheets, it may be replaced in later versions of Bigebra.
- Drop_t was especially useful during solving equations in Clifford polynomials if one searches for ***elements*** fulfilling some tangle equation having one outgoing line (n->1 maps). If equations have to be solved for ***operators*** acting inside of the tangle on arbitrary elements going through the tangle, one should use tsolve1.

Examples:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

Drop_t on homogenous tensors:

```
> drop_t(&t(e1we2));
  drop_t(&t(Id));
  drop_t(&t(e1)+&t(e2)+&t(e3));
```

$$e1we2$$

$$Id$$

$$e1 + e2 + e3$$

```
> drop_t(a*&t(e1));
  drop_t(&t(a*e2));
  drop_t(x*&t(a*e1we2+b+e2-c*e1we2we3)-y*&t(e4));
```

$$a\ e1$$

$$a\ e2$$

$$x\,(a\ e1we2 + b + e2 - c\ e1we2we3) - y\ e4$$

Abuse of drop_t:

```
> drop_t(&t(a*e2we3,e1we4));
  drop_t(&t(a*e1+b*e2,e3));
```

$$a\ e2we3$$

$$a\ e1 + b\ e2$$

```
> printf("Worksheet to %f seconds on Intel Pentium M 2.13 GHz 2GB
  RAM WinXP Prof",time()-bench);
Worksheet to 0.016000 seconds on AMD Athlon 2700+ 1GB RAM WinXP Prof
>
```

NOTE that drop_t acts like a projection on the first tensor slot and that the other slots are lost, ***no error message*** is delivered to the user, watch out. If you need access to other tensor slots, see Bigebra:-peek.

See Also: Bigebra:-`&t`, Bigebra:-`type/tensorpolynom`, Bigebra:-peek

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-EV - the evaluation map

Calling Sequence:

sc := EV(c1,c2)

Parameters:

- c1,c2 : are Clifford polynomials

Output:

- sc : is a scalar.

Description:

- The evaluation maps, EV : \/V x /\V --> k, or EV : /\V x \/V --> k, is the defined via the action of a dual element, called also linear form, on a multi-vector polynomial. The dual element is a co-multivector polynomial. However, since we have not yet supplied different bases in CLIFFORD/Bigebra, co-element are represented also as Clifford polynomials . *The user has to trace which elements in which slot of a tensor are considered be co-elements!*
- The evaluation is free of any bilinear form *B*. The **canonical** co-basis is given by the Grassmann basis \/V.
- We do not distinguish between the action of co-elements on elements and the action of elements on co-elements.

Examples:

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

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

The evaluation on homogeneous decomposable elements of the same grade:

```
> EV(Id,Id);
  EV(e1,e2),EV(e2,e1),EV(ei,ei),EV(ei,ej),EV(e1,e1);
  EV(e1we2,e1we2),EV(e1we3,e1we2);
  EV(e1we2we3,e1we2we3);
```

$$1$$

$$0, 0, 1, 0, 1$$

$$1, 0$$

$$1$$

NOTE that differently named symbolic indices are considered to be different! Bigebra is not yet well tested and designed to handle symbolic indices.

On homogeneous decomposable elements of different grades, EV yields:

```
> EV(0,Id),EV(e1we2,0),EV(Id,ei);
  EV(e1we2,e2),EV(e2,e1we2);
```

$$0, 0, 0$$

$$0, 0$$

Evaluating inhomogeneous elements:

```
> EV(a*Id+b*e1,Id+e3);
  EV(a*Id+b*e1,e1+e1we2);
  EV(a*Id-b*e1-e1we2+d*e2we3we4,Id+e2we3-4*sin(x)*e1we2);
```

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

$$a$$

$$b$$

$$a + 4\sin(x)$$

Use contract to map the evaluation onto adjacent tensor slots.

```
> contract(&t(e1,e2we3+e3we1,e3we1,e2),2,EV);
```

$$e1 \,\&t\, e2$$

```
> printf("worksheet tok %f seconds on Intel Pentium M 2.13 GHz
  2GB RAM WinXP Prof\n",time()-bench);
```

```
worksheet tok 0.095000 seconds on Intel Pentium M 2.13 GHz 2GB RAM WinXP Prof
```

```
>
```

See Also: Bigebra:-`&t`, Bigebra:-`type/tensorpolynom`, Bigebra:-contract, Bigebra:-pairing, Bigebra help page

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-gantipode - the antipode map for Grassmann Hopf gebra

Calling Sequence:

t1 := gantipode(t1,i)
c1 := gantipode(c1)

Parameters:

- t2 : a tensor polynom
- c2 : a Clifford polynom

Output:

- t1 : a tensor polynom
- c1 : a Clifford polynom

Description:

- The Grassmann antipode is the convolutive inverse of the identity map. It fulfills the antipode axioms (e.g. Sweedler)

  S(x_(1)) x_(2) = eta\circ\epsilon(x) = x_(1) S(x_(2)).

- The Grassmann antipode is closely related to the grade involution of the Grassmann algebra, see examples below. This involution constitutes a Z_2 grading which is also present in Clifford algebras.
- The Grassmann antipode can be obtained by direct computation (e.g. using tsolev1) from the unital Grassmann bi-convolution.

Examples:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

On a Grassmann basis we compute the antipode map, which shows it to be equivalent to the grade involution, this is not an accident but can be proved:

```
> dim_V:=4:bas:=cbasis(dim_V);
  Sbas:=map(gantipode,bas);
  Cbas:=map(gradeinv,bas);
  printf("It is `%a` that the two lists Sbas and Cbas contain the
  same elements",evalb({0}={op(zip((i,j)->i-j,Sbas,Cbas))}));
```

bas := [*Id*, *e1*, *e2*, *e3*, *e4*, *e1we2*, *e1we3*, *e1we4*, *e2we3*, *e2we4*, *e3we4*, *e1we2we3*, *e1we2we4*,

$e1we3we4, e2we3we4, e1we2we3we4]$

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

$Sbas := [Id, -e1, -e2, -e3, -e4, e1we2, e1we3, e1we4, e2we3, e2we4, e3we4, -e1we2we3, -e1we2we4, -e1we3we4, -e2we3we4, e1we2we3we4]$

$Cbas := [Id, -e1, -e2, -e3, -e4, e1we2, e1we3, e1we4, e2we3, e2we4, e3we4, -e1we2we3, -e1we2we4, -e1we3we4, -e2we3we4, e1we2we3we4]$

```
It is `true` that the two lists Sbas and Cbas contain the same elements
```

Now we give some examples where the Grassmann antipode acts on slots of tensor products:

```
> gantipode(&t(e1,e2we3,e4),2);
  gantipode(&t(e1,e2we3,e4),3);
```

$$\&t(e1, e2we3, e4)$$

$$-\&t(e1, e2we3, e4)$$

On special elements we find:

```
> gantipode(0),gantipode(&t(e1,0,e3),2);
```

$$0, 0$$

On inhomogeneous elements we find:

```
> gantipode(&t(a*Id-b*e1-e1we2,d*e2we3we4,Id+e2we3-4*sin(x)*e1we2
  ),1);
```

$$a\,d\,\&t(Id, e2we3we4, Id) + a\,d\,\&t(Id, e2we3we4, e2we3) - 4\,a\,d\,\sin(x)\,\&t(Id, e2we3we4, e1we2) + b\,d\,\&t(e1, e2we3we4, Id) + b\,d\,\&t(e1, e2we3we4, e2we3) - 4\,b\,d\,\sin(x)\,\&t(e1, e2we3we4, e1we2) - d\,\&t(e1we2, e2we3we4, Id) - d\,\&t(e1we2, e2we3we4, e2we3) + 4\,d\,\sin(x)\,\&t(e1we2, e2we3we4, e1we2)$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM WinXP Prof",time()-bench);
Worksheet took 0.109000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
WinXP Prof
>
```

See Also: Bigebra:-`&t`, Bigebra:-`type/tensorpolynom`, Bigebra help page

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-gco_unit - Grassmann co-unit

Calling Sequence:

p2 := gco_unit(p1,i)

Parameters:

- p1 : a tensor polynom
- i : the i-th slot to act on

Output:

- p2 : a tensor polynom

Description:

- The Grassmann co-unit is essentially the projection onto the identity component in the i-th slot of a tensor polynom. It acts like the augmentation linear form.
- It can be shown that Grassmann algebra is an augmented connected algebra, which has important consequences in topology, but also for computational means, see [3].
- The co-unit, denoted by ε, can be constructed by categorical duality from the axioms of a unit, and fulfills these relations:

  (*) $$(\varepsilon \,\backslash\text{otime}\, \mathrm{Id})\,\Delta = \mathrm{Id} = (\mathrm{Id}\,\backslash\text{otimes}\, \varepsilon)\,\Delta$$

  which are the dual relations of a algebra unit. Δ is the Graßmann co-product and Id the identity mapping.
- Note that the co-unit lowers the rank of a tensor by one and maps Clifford polynoms onto scalars.

Examples:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

The co-unit acts on tensors:

```
> dim_V:=2:bas:=cbasis(dim_V);
  map(gco_unit,bas);
  gco_unit(&t(Id,e1),1);
  gco_unit(&t(Id,e1),2);
  gco_unit(&t(0,e1),1);
```

$$bas := [Id, e1, e2, e1we2]$$

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

$$[Id, 0, 0, 0]$$

$$\&t(e1)$$

$$0$$

$$0$$

Check that it is a co-unit in dim:=3:

```
> dim_V:=3:
  bas:=cbasis(dim_V):
  X:=add(_x[i]*bas[i],i=1..2^dim_V);
```

$$X := \_x_1\, Id + \_x_2\, e1 + \_x_3\, e2 + \_x_4\, e3 + \_x_5\, e1we2 + \_x_6\, e1we3 + \_x_7\, e2we3 + \_x_8\, e1we2we3$$

This is an arbitrary element, we have to compute the lhs and rhs of the defining equation (*) and have to compare the output with the identity mapping (i.e. the arbitrary element). We have to use tcollect in an intermediate step to expand the output of &gco.

```
> LHS:=gco_unit(tcollect(&gco(X)),1);
  RHS:=gco_unit(tcollect(&gco(X)),2);
```

$$LHS := \_x_1\, \&t(Id) + \_x_2\, \&t(e1) + \_x_3\, \&t(e2) + \_x_4\, \&t(e3) + \_x_5\, \&t(e1we2) + \_x_6\, \&t(e1we3) + \_x_7\, \&t(e2we3) + \_x_8\, \&t(e1we2we3)$$

$$RHS := \_x_1\, \&t(Id) + \_x_2\, \&t(e1) + \_x_3\, \&t(e2) + \_x_4\, \&t(e3) + \_x_5\, \&t(e1we2) + \_x_6\, \&t(e1we3) + \_x_7\, \&t(e2we3) + \_x_8\, \&t(e1we2we3)$$

Now we check if the equations (*) are true:

```
> printf("The left equality sign of (*) is %a",
  evalb(X-drop_t(LHS)=0));
  printf("The right equality sign of (*) is %s",
  evalb(X-drop_t(RHS)=0));
The left equality sign of (*) is true
The right equality sign of (*) is true
```

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM WinXP Prof",time()-bench);
Worksheet took 0.156000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
WinXP Prof
>
```

See Also: Bigebra:-`&t`, Bigebra:-tcollect, Bigebra:-contract, Bigebra help page

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-gswitch - graded switch of tensor slots

Calling Sequence:

p1 = gswitch(p2,i)

Parameters:

- p2 : a tensorpolynom which is of not less than rank i in each factor
  i : the slot number (first slot from the left is 1) of the pair (i,i+1) on which the switch acts

Output:

- p1 : a tensorpolynom

Global variables:

- _CLIENV[_QDEF_PREFACTOR]

Description:

- Given a tensor polynomial the graded switch swaps two adjacent slots in a tensor product. In switching the factors, it takes account of the sign of the permutation. Denote the grade of a homogenous Grassmann element A by |A|. The graded switch of two homogenous elements is related to the (ungraded) switch as follows:

  $$\tau'(A \,\&t\, B) = (-1)^{(|A|\,|B|)}\, \tau(A \,\&t\, B) = (-1)^{(|A|\,|B|)}\, (B \,\&t\, A).$$

  The action is extended by linearity to arbitrary inhomogeneous elements.
- The graded switch is the natural switch for the Grassmann Hopf gebra. If this switch is used in the crossed products, the co-product becomes an algebra homomorphism while the wedge product becomes a co-gebra homomorphism.
- The switch of an antipodal convolution algebra can be derived [3,7]. It happens to be the graded switch in the case of the Grassmann Hopf gebra.
- The graded switch makes the Grassmann co-gebra graded co-commutative.

Examples:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

Graded switch/swap of tensor factors:

```
> &t(e1,e2);
  gswitch(%,1);
```

$$e1 \,\&t\, e2$$

$$-(e2 \,\&t\, e1)$$

```
> &t(Id,Id);
  gswitch(%,1);
```

$$Id \,\&t\, Id$$

$$Id \,\&t\, Id$$

```
> &t(e1,e1we2);
  gswitch(%,1);
```

$$e1 \,\&t\, e1we2$$

$$e1we2 \,\&t\, e1$$

```
> p1:=&t(e1,e2);
  gsw:=&map(gswitch(p1,1),1,wedge);
  aswaa:=gantipode(&map(gantipode(gantipode(p1,1),2),1,wedge),1);
```

$$p1 := e1 \,\&t\, e2$$

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

$$gsw := \&t(e1we2)$$

$$aswaa := \&t(e1we2)$$

The graded switch and the antipode of a Grassmann Hopf algebra are related as follows:

```
> for i from 1 to 4 do
  dim_V:=i:bas:=cbasis(dim_V):
  X:=add(_X[i]*bas[i],i=1..2^dim_V):
  Y:=add(_Y[i]*bas[i],i=1..2^dim_V):
  p1:=&t(X,Y);
  gsw:=&map(gswitch(p1,1),1,wedge);
  awaa:=gantipode(&map(gantipode(gantipode(p1,1),2),1,wedge),1);
  printf("gsw=%a\n",gsw);
  printf("awaa=%a\n",awaa);
  printf("#####\nIn dimension %d the equation `gsw=awaa` is
  `%a`\n#####\n",dim_V,evalb(0=simplify(drop_t(gsw-awaa))));
  od:
gsw=_X[1]*_Y[1]*`&t`(Id)+_X[1]*_Y[2]*`&t`(e1)+_X[2]*_Y[1]*`&t`(e1)-_X[2]*_Y[2]*
`&t`(0)
awaa=_X[1]*_Y[1]*`&t`(Id)+_X[1]*_Y[2]*`&t`(e1)+_X[2]*_Y[1]*`&t`(e1)+_X[2]*_Y[2]
*`&t`(0)
#####
In dimension 1 the equation `gsw=awaa` is `true`
#####
gsw=_X[1]*_Y[1]*`&t`(Id)+_X[1]*_Y[2]*`&t`(e1)+_X[1]*_Y[3]*`&t`(e2)+_X[1]*_Y[4]*
`&t`(e1we2)+_X[2]*_Y[1]*`&t`(e1)-_X[2]*_Y[2]*`&t`(0)+_X[2]*_Y[3]*`&t`(e1we2)+_X
[2]*_Y[4]*`&t`(0)+_X[3]*_Y[1]*`&t`(e2)-_X[3]*_Y[2]*`&t`(e1we2)-_X[3]*_Y[3]*`&t`
(0)+_X[3]*_Y[4]*`&t`(0)+_X[4]*_Y[1]*`&t`(e1we2)+_X[4]*_Y[2]*`&t`(0)+_X[4]*_Y[3]
*`&t`(0)+_X[4]*_Y[4]*`&t`(0)
awaa=_X[1]*_Y[1]*`&t`(Id)+_X[1]*_Y[2]*`&t`(e1)+_X[1]*_Y[3]*`&t`(e2)+_X[1]*_Y[4]
*`&t`(e1we2)+_X[2]*_Y[1]*`&t`(e1)+_X[2]*_Y[2]*`&t`(0)+_X[2]*_Y[3]*`&t`(e1we2)-_
X[2]*_Y[4]*`&t`(0)+_X[3]*_Y[1]*`&t`(e2)-_X[3]*_Y[2]*`&t`(e1we2)+_X[3]*_Y[3]*`&t
```

```
`(0)-_X[3]*_Y[4]*`&t`(0)+_X[4]*_Y[1]*`&t`(e1we2)-_X[4]*_Y[2]*`&t`(0)-_X[4]*_Y[3
]*`&t`(0)+_X[4]*_Y[4]*`&t`(0)
#####
In dimension 2 the equation `gsw=awaa` is `true`
#####
gsw=_X[7]*_Y[7]*`&t`(0)+_X[2]*_Y[1]*`&t`(e1)+_X[7]*_Y[2]*`&t`(e1we2we3)+_X[5]*_
Y[4]*`&t`(e1we2we3)-_X[8]*_Y[3]*`&t`(0)+_X[6]*_Y[8]*`&t`(0)+_X[4]*_Y[7]*`&t`(0)
+_X[2]*_Y[7]*`&t`(e1we2we3)+_X[7]*_Y[1]*`&t`(e2we3)+_X[1]*_Y[4]*`&t`(e3)-_X[6]*
_Y[3]*`&t`(e1we2we3)+_X[8]*_Y[5]*`&t`(0)+_X[6]*_Y[1]*`&t`(e1we3)+_X[6]*_Y[2]*`&
t`(0)-_X[2]*_Y[8]*`&t`(0)+_X[1]*_Y[7]*`&t`(e2we3)-_X[3]*_Y[6]*`&t`(e1we2we3)+_X
[2]*_Y[5]*`&t`(0)+_X[1]*_Y[1]*`&t`(Id)+_X[5]*_Y[6]*`&t`(0)+_X[4]*_Y[6]*`&t`(0)+
_X[8]*_Y[7]*`&t`(0)+_X[7]*_Y[5]*`&t`(0)+_X[1]*_Y[2]*`&t`(e1)+_X[7]*_Y[6]*`&t`(0
)+_X[7]*_Y[8]*`&t`(0)+_X[3]*_Y[7]*`&t`(0)-_X[8]*_Y[8]*`&t`(0)+_X[5]*_Y[2]*`&t`(
0)+_X[5]*_Y[7]*`&t`(0)-_X[4]*_Y[2]*`&t`(e1we3)+_X[8]*_Y[6]*`&t`(0)+_X[6]*_Y[4]*
`&t`(0)+_X[1]*_Y[8]*`&t`(e1we2we3)-_X[2]*_Y[2]*`&t`(0)+_X[1]*_Y[3]*`&t`(e2)+_X[
5]*_Y[8]*`&t`(0)+_X[4]*_Y[5]*`&t`(e1we2we3)+_X[7]*_Y[3]*`&t`(0)+_X[6]*_Y[5]*`&t
`(0)+_X[5]*_Y[1]*`&t`(e1we2)+_X[5]*_Y[5]*`&t`(0)+_X[1]*_Y[6]*`&t`(e1we3)+_X[3]*
_Y[5]*`&t`(0)+_X[2]*_Y[4]*`&t`(e1we3)+_X[2]*_Y[6]*`&t`(0)+_X[5]*_Y[3]*`&t`(0)+_
X[7]*_Y[4]*`&t`(0)+_X[3]*_Y[1]*`&t`(e2)-_X[3]*_Y[2]*`&t`(e1we2)+_X[2]*_Y[3]*`&t
`(e1we2)-_X[3]*_Y[3]*`&t`(0)-_X[4]*_Y[4]*`&t`(0)+_X[3]*_Y[4]*`&t`(e2we3)+_X[6]*
_Y[7]*`&t`(0)-_X[8]*_Y[4]*`&t`(0)+_X[4]*_Y[1]*`&t`(e3)-_X[4]*_Y[8]*`&t`(0)-_X[3
]*_Y[8]*`&t`(0)-_X[4]*_Y[3]*`&t`(e2we3)-_X[8]*_Y[2]*`&t`(0)+_X[8]*_Y[1]*`&t`(e1
we2we3)+_X[6]*_Y[6]*`&t`(0)+_X[1]*_Y[5]*`&t`(e1we2)
awaa=_X[7]*_Y[7]*`&t`(0)+_X[2]*_Y[1]*`&t`(e1)+_X[7]*_Y[2]*`&t`(e1we2we3)+_X[5]*
_Y[4]*`&t`(e1we2we3)+_X[8]*_Y[3]*`&t`(0)-_X[6]*_Y[8]*`&t`(0)-_X[4]*_Y[7]*`&t`(0
)+_X[2]*_Y[7]*`&t`(e1we2we3)+_X[7]*_Y[1]*`&t`(e2we3)+_X[1]*_Y[4]*`&t`(e3)-_X[6]
*_Y[3]*`&t`(e1we2we3)-_X[8]*_Y[5]*`&t`(0)+_X[6]*_Y[1]*`&t`(e1we3)-_X[6]*_Y[2]*`
&t`(0)+_X[2]*_Y[8]*`&t`(0)+_X[1]*_Y[7]*`&t`(e2we3)-_X[3]*_Y[6]*`&t`(e1we2we3)-_
X[2]*_Y[5]*`&t`(0)+_X[1]*_Y[1]*`&t`(Id)+_X[5]*_Y[6]*`&t`(0)-_X[4]*_Y[6]*`&t`(0)
-_X[8]*_Y[7]*`&t`(0)+_X[7]*_Y[5]*`&t`(0)+_X[1]*_Y[2]*`&t`(e1)+_X[7]*_Y[6]*`&t`(
0)-_X[7]*_Y[8]*`&t`(0)-_X[3]*_Y[7]*`&t`(0)+_X[8]*_Y[8]*`&t`(0)-_X[5]*_Y[2]*`&t`
(0)+_X[5]*_Y[7]*`&t`(0)-_X[4]*_Y[2]*`&t`(e1we3)-_X[8]*_Y[6]*`&t`(0)-_X[6]*_Y[4]
*`&t`(0)+_X[1]*_Y[8]*`&t`(e1we2we3)+_X[2]*_Y[2]*`&t`(0)+_X[1]*_Y[3]*`&t`(e2)-_X
[5]*_Y[8]*`&t`(0)+_X[4]*_Y[5]*`&t`(e1we2we3)-_X[7]*_Y[3]*`&t`(0)+_X[6]*_Y[5]*`&
t`(0)+_X[5]*_Y[1]*`&t`(e1we2)+_X[5]*_Y[5]*`&t`(0)+_X[1]*_Y[6]*`&t`(e1we3)-_X[3]
*_Y[5]*`&t`(0)+_X[2]*_Y[4]*`&t`(e1we3)-_X[2]*_Y[6]*`&t`(0)-_X[5]*_Y[3]*`&t`(0)-
_X[7]*_Y[4]*`&t`(0)+_X[3]*_Y[1]*`&t`(e2)-_X[3]*_Y[2]*`&t`(e1we2)+_X[2]*_Y[3]*`&
t`(e1we2)+_X[3]*_Y[3]*`&t`(0)+_X[4]*_Y[4]*`&t`(0)+_X[3]*_Y[4]*`&t`(e2we3)+_X[6]
*_Y[7]*`&t`(0)+_X[8]*_Y[4]*`&t`(0)+_X[4]*_Y[1]*`&t`(e3)+_X[4]*_Y[8]*`&t`(0)+_X[
3]*_Y[8]*`&t`(0)-_X[4]*_Y[3]*`&t`(e2we3)+_X[8]*_Y[2]*`&t`(0)+_X[8]*_Y[1]*`&t`(e
1we2we3)+_X[6]*_Y[6]*`&t`(0)+_X[1]*_Y[5]*`&t`(e1we2)
#####
In dimension 3 the equation `gsw=awaa` is `true`
#####
gsw=-_X[14]*_Y[15]*`&t`(0)+_X[8]*_Y[16]*`&t`(0)+_X[7]*_Y[7]*`&t`(0)+_X[2]*_Y[1]
*`&t`(e1)-_X[13]*_Y[13]*`&t`(0)+_X[5]*_Y[16]*`&t`(0)+_X[10]*_Y[15]*`&t`(0)+_X[6
]*_Y[15]*`&t`(0)+_X[6]*_Y[8]*`&t`(0)-_X[12]*_Y[4]*`&t`(0)+_X[4]*_Y[7]*`&t`(0)+_
X[1]*_Y[4]*`&t`(e3)+_X[9]*_Y[11]*`&t`(0)+_X[8]*_Y[5]*`&t`(0)+_X[1]*_Y[16]*`&t`(
e1we2we3we4)+_X[5]*_Y[10]*`&t`(0)-_X[10]*_Y[4]*`&t`(e2we3we4)+_X[3]*_Y[6]*`&t`(
0)+_X[6]*_Y[2]*`&t`(0)+_X[2]*_Y[8]*`&t`(0)+_X[14]*_Y[11]*`&t`(0)+_X[1]*_Y[13]*`
&t`(e1we2we4)+_X[1]*_Y[1]*`&t`(Id)+_X[1]*_Y[9]*`&t`(e2we3)+_X[16]*_Y[6]*`&t`(0)
+_X[5]*_Y[6]*`&t`(e1we2we4)+_X[11]*_Y[16]*`&t`(0)+_X[14]*_Y[7]*`&t`(0)+_X[13]*_
Y[6]*`&t`(0)+_X[11]*_Y[14]*`&t`(0)+_X[8]*_Y[7]*`&t`(0)-_X[14]*_Y[13]*`&t`(0)-_X
[4]*_Y[8]*`&t`(e1we3we4)+_X[11]*_Y[9]*`&t`(0)+_X[6]*_Y[10]*`&t`(0)+_X[9]*_Y[7]*
`&t`(0)+_X[1]*_Y[2]*`&t`(e1)-_X[14]*_Y[2]*`&t`(0)+_X[15]*_Y[9]*`&t`(0)+_X[8]*_Y
[9]*`&t`(e1we2we3we4)+_X[2]*_Y[10]*`&t`(e1we2we4)+_X[3]*_Y[5]*`&t`(e2we4)+_X[5]
*_Y[1]*`&t`(e4)+_X[6]*_Y[12]*`&t`(0)+_X[13]*_Y[10]*`&t`(0)+_X[7]*_Y[6]*`&t`(0)+
_X[6]*_Y[3]*`&t`(0)+_X[16]*_Y[14]*`&t`(0)+_X[8]*_Y[14]*`&t`(0)-_X[15]*_Y[12]*`&
t`(0)+_X[1]*_Y[7]*`&t`(e1we3)+_X[9]*_Y[3]*`&t`(0)-_X[8]*_Y[4]*`&t`(e1we3we4)+_X
```

```
[9]*_Y[1]*`&t`(e2we3)+_X[7]*_Y[8]*`&t`(0)+_X[8]*_Y[8]*`&t`(0)-_X[4]*_Y[2]*`&t`(
e1we3)+_X[8]*_Y[6]*`&t`(0)+_X[3]*_Y[16]*`&t`(0)-_X[2]*_Y[2]*`&t`(0)-_X[3]*_Y[13
]*`&t`(0)+_X[14]*_Y[16]*`&t`(0)+_X[15]*_Y[8]*`&t`(0)+_X[13]*_Y[8]*`&t`(0)+_X[1]
*_Y[3]*`&t`(e2)+_X[6]*_Y[4]*`&t`(e1we2we3)-_X[5]*_Y[2]*`&t`(e1we4)+_X[10]*_Y[6]
*`&t`(0)-_X[12]*_Y[14]*`&t`(0)+_X[5]*_Y[8]*`&t`(0)+_X[6]*_Y[16]*`&t`(0)+_X[15]*
_Y[16]*`&t`(0)+_X[11]*_Y[10]*`&t`(0)-_X[5]*_Y[5]*`&t`(0)-_X[13]*_Y[2]*`&t`(0)-_
X[13]*_Y[14]*`&t`(0)+_X[11]*_Y[8]*`&t`(0)-_X[13]*_Y[12]*`&t`(0)+_X[10]*_Y[16]*`
&t`(0)+_X[5]*_Y[9]*`&t`(e2we3we4)+_X[6]*_Y[9]*`&t`(0)+_X[7]*_Y[14]*`&t`(0)+_X[1
]*_Y[10]*`&t`(e2we4)+_X[2]*_Y[4]*`&t`(e1we3)+_X[12]*_Y[8]*`&t`(0)+_X[9]*_Y[10]*
`&t`(0)+_X[2]*_Y[6]*`&t`(0)+_X[7]*_Y[4]*`&t`(0)+_X[15]*_Y[6]*`&t`(0)+_X[11]*_Y[
1]*`&t`(e3we4)+_X[11]*_Y[11]*`&t`(0)-_X[3]*_Y[7]*`&t`(e1we2we3)+_X[15]*_Y[7]*`&
t`(0)+_X[2]*_Y[7]*`&t`(0)-_X[15]*_Y[4]*`&t`(0)+_X[3]*_Y[1]*`&t`(e2)-_X[3]*_Y[2]
*`&t`(e1we2)+_X[12]*_Y[7]*`&t`(0)+_X[4]*_Y[5]*`&t`(e3we4)+_X[5]*_Y[11]*`&t`(0)+
_X[2]*_Y[3]*`&t`(e1we2)+_X[1]*_Y[5]*`&t`(e4)-_X[3]*_Y[3]*`&t`(0)-_X[4]*_Y[4]*`&
t`(0)+_X[3]*_Y[4]*`&t`(e2we3)+_X[16]*_Y[10]*`&t`(0)+_X[16]*_Y[1]*`&t`(e1we2we3w
e4)+_X[14]*_Y[8]*`&t`(0)-_X[2]*_Y[14]*`&t`(0)+_X[16]*_Y[12]*`&t`(0)+_X[1]*_Y[6]
*`&t`(e1we2)+_X[6]*_Y[7]*`&t`(0)+_X[4]*_Y[9]*`&t`(0)-_X[15]*_Y[3]*`&t`(0)+_X[8]
*_Y[10]*`&t`(0)+_X[16]*_Y[4]*`&t`(0)+_X[11]*_Y[3]*`&t`(e2we3we4)+_X[9]*_Y[2]*`&
t`(e1we2we3)+_X[14]*_Y[9]*`&t`(0)+_X[4]*_Y[1]*`&t`(e3)-_X[8]*_Y[3]*`&t`(e1we2we
4)-_X[14]*_Y[5]*`&t`(0)-_X[13]*_Y[5]*`&t`(0)+_X[7]*_Y[15]*`&t`(0)-_X[4]*_Y[3]*`
&t`(e2we3)+_X[8]*_Y[2]*`&t`(0)+_X[6]*_Y[14]*`&t`(0)+_X[9]*_Y[16]*`&t`(0)-_X[4]*
_Y[15]*`&t`(0)+_X[16]*_Y[15]*`&t`(0)+_X[10]*_Y[8]*`&t`(0)+_X[3]*_Y[10]*`&t`(0)+
_X[12]*_Y[5]*`&t`(e1we2we3we4)+_X[10]*_Y[5]*`&t`(0)+_X[1]*_Y[14]*`&t`(e1we3we4)
-_X[7]*_Y[3]*`&t`(e1we2we3)+_X[13]*_Y[9]*`&t`(0)+_X[9]*_Y[15]*`&t`(0)+_X[7]*_Y[
1]*`&t`(e1we3)+_X[4]*_Y[13]*`&t`(e1we2we3we4)+_X[16]*_Y[2]*`&t`(0)+_X[9]*_Y[6]*
`&t`(0)+_X[16]*_Y[16]*`&t`(0)+_X[11]*_Y[12]*`&t`(0)+_X[6]*_Y[6]*`&t`(0)+_X[1]*_
Y[15]*`&t`(e2we3we4)-_X[5]*_Y[12]*`&t`(e1we2we3we4)+_X[11]*_Y[6]*`&t`(e1we2we3w
e4)+_X[15]*_Y[11]*`&t`(0)+_X[8]*_Y[12]*`&t`(0)+_X[6]*_Y[13]*`&t`(0)-_X[15]*_Y[5
]*`&t`(0)+_X[2]*_Y[9]*`&t`(e1we2we3)+_X[14]*_Y[1]*`&t`(e1we3we4)-_X[15]*_Y[14]*
`&t`(0)+_X[11]*_Y[5]*`&t`(0)+_X[4]*_Y[16]*`&t`(0)+_X[9]*_Y[5]*`&t`(e2we3we4)+_X
[14]*_Y[6]*`&t`(0)+_X[2]*_Y[11]*`&t`(e1we3we4)+_X[16]*_Y[7]*`&t`(0)+_X[12]*_Y[1
0]*`&t`(0)-_X[2]*_Y[12]*`&t`(0)+_X[2]*_Y[16]*`&t`(0)-_X[5]*_Y[3]*`&t`(e2we4)+_X
[13]*_Y[16]*`&t`(0)+_X[10]*_Y[1]*`&t`(e2we4)+_X[12]*_Y[9]*`&t`(0)+_X[2]*_Y[15]*
`&t`(e1we2we3we4)+_X[2]*_Y[5]*`&t`(e1we4)-_X[14]*_Y[4]*`&t`(0)+_X[12]*_Y[6]*`&t
`(0)-_X[5]*_Y[13]*`&t`(0)-_X[15]*_Y[2]*`&t`(e1we2we3we4)-_X[13]*_Y[15]*`&t`(0)+
_X[11]*_Y[13]*`&t`(0)+_X[4]*_Y[11]*`&t`(0)+_X[13]*_Y[11]*`&t`(0)+_X[16]*_Y[3]*`
&t`(0)+_X[16]*_Y[13]*`&t`(0)+_X[7]*_Y[11]*`&t`(0)+_X[9]*_Y[14]*`&t`(0)-_X[14]*_
Y[12]*`&t`(0)+_X[15]*_Y[1]*`&t`(e2we3we4)-_X[13]*_Y[3]*`&t`(0)+_X[10]*_Y[12]*`&
t`(0)-_X[4]*_Y[10]*`&t`(e2we3we4)+_X[9]*_Y[9]*`&t`(0)+_X[16]*_Y[11]*`&t`(0)+_X[
8]*_Y[1]*`&t`(e1we4)+_X[1]*_Y[11]*`&t`(e3we4)+_X[9]*_Y[13]*`&t`(0)+_X[14]*_Y[3]
*`&t`(e1we2we3we4)+_X[6]*_Y[5]*`&t`(e1we2we4)-_X[15]*_Y[15]*`&t`(0)+_X[10]*_Y[9
]*`&t`(0)-_X[5]*_Y[4]*`&t`(e3we4)-_X[5]*_Y[15]*`&t`(0)-_X[12]*_Y[13]*`&t`(0)+_X
[12]*_Y[16]*`&t`(0)+_X[15]*_Y[10]*`&t`(0)+_X[16]*_Y[5]*`&t`(0)+_X[6]*_Y[1]*`&t`
(e1we2)+_X[6]*_Y[11]*`&t`(e1we2we3we4)+_X[10]*_Y[13]*`&t`(0)+_X[4]*_Y[6]*`&t`(e
1we2we3)+_X[3]*_Y[9]*`&t`(0)-_X[4]*_Y[14]*`&t`(0)+_X[10]*_Y[14]*`&t`(0)-_X[12]*
_Y[3]*`&t`(0)+_X[14]*_Y[10]*`&t`(0)+_X[7]*_Y[12]*`&t`(0)-_X[3]*_Y[8]*`&t`(e1we2
we4)+_X[9]*_Y[8]*`&t`(e1we2we3we4)+_X[10]*_Y[3]*`&t`(0)+_X[12]*_Y[11]*`&t`(0)+_
X[1]*_Y[8]*`&t`(e1we4)+_X[7]*_Y[9]*`&t`(0)+_X[16]*_Y[9]*`&t`(0)+_X[11]*_Y[4]*`&
t`(0)+_X[7]*_Y[2]*`&t`(0)+_X[8]*_Y[13]*`&t`(0)-_X[15]*_Y[13]*`&t`(0)+_X[8]*_Y[1
5]*`&t`(0)+_X[7]*_Y[16]*`&t`(0)+_X[3]*_Y[11]*`&t`(e2we3we4)+_X[7]*_Y[5]*`&t`(e1
we3we4)+_X[11]*_Y[15]*`&t`(0)-_X[4]*_Y[12]*`&t`(0)+_X[8]*_Y[11]*`&t`(0)-_X[3]*_
Y[12]*`&t`(0)+_X[7]*_Y[13]*`&t`(0)-_X[10]*_Y[7]*`&t`(e1we2we3we4)-_X[3]*_Y[15]*
`&t`(0)-_X[5]*_Y[14]*`&t`(0)+_X[1]*_Y[12]*`&t`(e1we2we3)+_X[13]*_Y[7]*`&t`(0)-_
X[2]*_Y[13]*`&t`(0)+_X[10]*_Y[2]*`&t`(e1we2we4)-_X[13]*_Y[4]*`&t`(e1we2we3we4)+
_X[5]*_Y[7]*`&t`(e1we3we4)-_X[14]*_Y[14]*`&t`(0)-_X[12]*_Y[2]*`&t`(0)+_X[9]*_Y[
4]*`&t`(0)+_X[10]*_Y[11]*`&t`(0)-_X[12]*_Y[12]*`&t`(0)+_X[10]*_Y[10]*`&t`(0)-_X
[3]*_Y[14]*`&t`(e1we2we3we4)-_X[7]*_Y[10]*`&t`(e1we2we3we4)+_X[12]*_Y[1]*`&t`(e
1we2we3)+_X[9]*_Y[12]*`&t`(0)+_X[13]*_Y[1]*`&t`(e1we2we4)-_X[12]*_Y[15]*`&t`(0)
+_X[11]*_Y[7]*`&t`(0)+_X[16]*_Y[8]*`&t`(0)+_X[11]*_Y[2]*`&t`(e1we3we4)
```

```
awaa=_X[14]*_Y[15]*`&t`(0)+_X[8]*_Y[16]*`&t`(0)+_X[7]*_Y[7]*`&t`(0)+_X[2]*_Y[1]
*`&t`(e1)+_X[13]*_Y[13]*`&t`(0)-_X[5]*_Y[16]*`&t`(0)-_X[10]*_Y[15]*`&t`(0)-_X[6
]*_Y[15]*`&t`(0)+_X[6]*_Y[8]*`&t`(0)+_X[12]*_Y[4]*`&t`(0)-_X[4]*_Y[7]*`&t`(0)+_
X[1]*_Y[4]*`&t`(e3)+_X[9]*_Y[11]*`&t`(0)-_X[8]*_Y[5]*`&t`(0)+_X[1]*_Y[16]*`&t`(
e1we2we3we4)-_X[5]*_Y[10]*`&t`(0)-_X[10]*_Y[4]*`&t`(e2we3we4)-_X[3]*_Y[6]*`&t`(
0)-_X[6]*_Y[2]*`&t`(0)-_X[2]*_Y[8]*`&t`(0)-_X[14]*_Y[11]*`&t`(0)+_X[1]*_Y[13]*`
&t`(e1we2we4)+_X[1]*_Y[1]*`&t`(Id)+_X[1]*_Y[9]*`&t`(e2we3)+_X[16]*_Y[6]*`&t`(0)
+_X[5]*_Y[6]*`&t`(e1we2we4)+_X[11]*_Y[16]*`&t`(0)-_X[14]*_Y[7]*`&t`(0)-_X[13]*_
Y[6]*`&t`(0)-_X[11]*_Y[14]*`&t`(0)+_X[8]*_Y[7]*`&t`(0)+_X[14]*_Y[13]*`&t`(0)-_X
[4]*_Y[8]*`&t`(e1we3we4)+_X[11]*_Y[9]*`&t`(0)+_X[6]*_Y[10]*`&t`(0)+_X[9]*_Y[7]*
`&t`(0)+_X[1]*_Y[2]*`&t`(e1)+_X[14]*_Y[2]*`&t`(0)-_X[15]*_Y[9]*`&t`(0)+_X[8]*_Y
[9]*`&t`(e1we2we3we4)+_X[2]*_Y[10]*`&t`(e1we2we4)+_X[3]*_Y[5]*`&t`(e2we4)+_X[5]
*_Y[1]*`&t`(e4)-_X[6]*_Y[12]*`&t`(0)-_X[13]*_Y[10]*`&t`(0)+_X[7]*_Y[6]*`&t`(0)-
_X[6]*_Y[3]*`&t`(0)-_X[16]*_Y[14]*`&t`(0)-_X[8]*_Y[14]*`&t`(0)+_X[15]*_Y[12]*`&
t`(0)+_X[1]*_Y[7]*`&t`(e1we3)-_X[9]*_Y[3]*`&t`(0)-_X[8]*_Y[4]*`&t`(e1we3we4)+_X
[9]*_Y[1]*`&t`(e2we3)+_X[7]*_Y[8]*`&t`(0)+_X[8]*_Y[8]*`&t`(0)-_X[4]*_Y[2]*`&t`(
e1we3)+_X[8]*_Y[6]*`&t`(0)-_X[3]*_Y[16]*`&t`(0)+_X[2]*_Y[2]*`&t`(0)+_X[3]*_Y[13
]*`&t`(0)-_X[14]*_Y[16]*`&t`(0)-_X[15]*_Y[8]*`&t`(0)-_X[13]*_Y[8]*`&t`(0)+_X[1]
*_Y[3]*`&t`(e2)+_X[6]*_Y[4]*`&t`(e1we2we3)-_X[5]*_Y[2]*`&t`(e1we4)+_X[10]*_Y[6]
*`&t`(0)+_X[12]*_Y[14]*`&t`(0)-_X[5]*_Y[8]*`&t`(0)+_X[6]*_Y[16]*`&t`(0)-_X[15]*
_Y[16]*`&t`(0)+_X[11]*_Y[10]*`&t`(0)+_X[5]*_Y[5]*`&t`(0)+_X[13]*_Y[2]*`&t`(0)+_
X[13]*_Y[14]*`&t`(0)+_X[11]*_Y[8]*`&t`(0)+_X[13]*_Y[12]*`&t`(0)+_X[10]*_Y[16]*`
&t`(0)+_X[5]*_Y[9]*`&t`(e2we3we4)+_X[6]*_Y[9]*`&t`(0)-_X[7]*_Y[14]*`&t`(0)+_X[1
]*_Y[10]*`&t`(e2we4)+_X[2]*_Y[4]*`&t`(e1we3)-_X[12]*_Y[8]*`&t`(0)+_X[9]*_Y[10]*
`&t`(0)-_X[2]*_Y[6]*`&t`(0)-_X[7]*_Y[4]*`&t`(0)-_X[15]*_Y[6]*`&t`(0)+_X[11]*_Y[
1]*`&t`(e3we4)+_X[11]*_Y[11]*`&t`(0)-_X[3]*_Y[7]*`&t`(e1we2we3)-_X[15]*_Y[7]*`&
t`(0)-_X[2]*_Y[7]*`&t`(0)+_X[15]*_Y[4]*`&t`(0)+_X[3]*_Y[1]*`&t`(e2)-_X[3]*_Y[2]
*`&t`(e1we2)-_X[12]*_Y[7]*`&t`(0)+_X[4]*_Y[5]*`&t`(e3we4)-_X[5]*_Y[11]*`&t`(0)+
_X[2]*_Y[3]*`&t`(e1we2)+_X[1]*_Y[5]*`&t`(e4)+_X[3]*_Y[3]*`&t`(0)+_X[4]*_Y[4]*`&
t`(0)+_X[3]*_Y[4]*`&t`(e2we3)+_X[16]*_Y[10]*`&t`(0)+_X[16]*_Y[1]*`&t`(e1we2we3w
e4)-_X[14]*_Y[8]*`&t`(0)+_X[2]*_Y[14]*`&t`(0)-_X[16]*_Y[12]*`&t`(0)+_X[1]*_Y[6]
*`&t`(e1we2)+_X[6]*_Y[7]*`&t`(0)-_X[4]*_Y[9]*`&t`(0)+_X[15]*_Y[3]*`&t`(0)+_X[8]
*_Y[10]*`&t`(0)-_X[16]*_Y[4]*`&t`(0)+_X[11]*_Y[3]*`&t`(e2we3we4)+_X[9]*_Y[2]*`&
t`(e1we2we3)-_X[14]*_Y[9]*`&t`(0)+_X[4]*_Y[1]*`&t`(e3)-_X[8]*_Y[3]*`&t`(e1we2we
4)+_X[14]*_Y[5]*`&t`(0)+_X[13]*_Y[5]*`&t`(0)-_X[7]*_Y[15]*`&t`(0)-_X[4]*_Y[3]*`
&t`(e2we3)-_X[8]*_Y[2]*`&t`(0)-_X[6]*_Y[14]*`&t`(0)+_X[9]*_Y[16]*`&t`(0)+_X[4]*
_Y[15]*`&t`(0)-_X[16]*_Y[15]*`&t`(0)+_X[10]*_Y[8]*`&t`(0)-_X[3]*_Y[10]*`&t`(0)+
_X[12]*_Y[5]*`&t`(e1we2we3we4)-_X[10]*_Y[5]*`&t`(0)+_X[1]*_Y[14]*`&t`(e1we3we4)
-_X[7]*_Y[3]*`&t`(e1we2we3)-_X[13]*_Y[9]*`&t`(0)-_X[9]*_Y[15]*`&t`(0)+_X[7]*_Y[
1]*`&t`(e1we3)+_X[4]*_Y[13]*`&t`(e1we2we3we4)-_X[16]*_Y[2]*`&t`(0)+_X[9]*_Y[6]*
`&t`(0)+_X[16]*_Y[16]*`&t`(0)-_X[11]*_Y[12]*`&t`(0)+_X[6]*_Y[6]*`&t`(0)+_X[1]*_
Y[15]*`&t`(e2we3we4)-_X[5]*_Y[12]*`&t`(e1we2we3we4)+_X[11]*_Y[6]*`&t`(e1we2we3w
e4)-_X[15]*_Y[11]*`&t`(0)-_X[8]*_Y[12]*`&t`(0)-_X[6]*_Y[13]*`&t`(0)+_X[15]*_Y[5
]*`&t`(0)+_X[2]*_Y[9]*`&t`(e1we2we3)+_X[14]*_Y[1]*`&t`(e1we3we4)+_X[15]*_Y[14]*
`&t`(0)-_X[11]*_Y[5]*`&t`(0)-_X[4]*_Y[16]*`&t`(0)+_X[9]*_Y[5]*`&t`(e2we3we4)-_X
[14]*_Y[6]*`&t`(0)+_X[2]*_Y[11]*`&t`(e1we3we4)+_X[16]*_Y[7]*`&t`(0)-_X[12]*_Y[1
0]*`&t`(0)+_X[2]*_Y[12]*`&t`(0)-_X[2]*_Y[16]*`&t`(0)-_X[5]*_Y[3]*`&t`(e2we4)-_X
[13]*_Y[16]*`&t`(0)+_X[10]*_Y[1]*`&t`(e2we4)-_X[12]*_Y[9]*`&t`(0)+_X[2]*_Y[15]*
`&t`(e1we2we3we4)+_X[2]*_Y[5]*`&t`(e1we4)+_X[14]*_Y[4]*`&t`(0)-_X[12]*_Y[6]*`&t
`(0)+_X[5]*_Y[13]*`&t`(0)-_X[15]*_Y[2]*`&t`(e1we2we3we4)+_X[13]*_Y[15]*`&t`(0)-
_X[11]*_Y[13]*`&t`(0)-_X[4]*_Y[11]*`&t`(0)-_X[13]*_Y[11]*`&t`(0)-_X[16]*_Y[3]*`
&t`(0)-_X[16]*_Y[13]*`&t`(0)+_X[7]*_Y[11]*`&t`(0)-_X[9]*_Y[14]*`&t`(0)+_X[14]*_
Y[12]*`&t`(0)+_X[15]*_Y[1]*`&t`(e2we3we4)+_X[13]*_Y[3]*`&t`(0)-_X[10]*_Y[12]*`&
t`(0)-_X[4]*_Y[10]*`&t`(e2we3we4)+_X[9]*_Y[9]*`&t`(0)+_X[16]*_Y[11]*`&t`(0)+_X[
8]*_Y[1]*`&t`(e1we4)+_X[1]*_Y[11]*`&t`(e3we4)-_X[9]*_Y[13]*`&t`(0)+_X[14]*_Y[3]
*`&t`(e1we2we3we4)+_X[6]*_Y[5]*`&t`(e1we2we4)+_X[15]*_Y[15]*`&t`(0)+_X[10]*_Y[9
]*`&t`(0)-_X[5]*_Y[4]*`&t`(e3we4)+_X[5]*_Y[15]*`&t`(0)+_X[12]*_Y[13]*`&t`(0)-_X
[12]*_Y[16]*`&t`(0)-_X[15]*_Y[10]*`&t`(0)-_X[16]*_Y[5]*`&t`(0)+_X[6]*_Y[1]*`&t`
(e1we2)+_X[6]*_Y[11]*`&t`(e1we2we3we4)-_X[10]*_Y[13]*`&t`(0)+_X[4]*_Y[6]*`&t`(e
```

```
1we2we3)-_X[3]*_Y[9]*`&t`(0)+_X[4]*_Y[14]*`&t`(0)-_X[10]*_Y[14]*`&t`(0)+_X[12]*
_Y[3]*`&t`(0)-_X[14]*_Y[10]*`&t`(0)-_X[7]*_Y[12]*`&t`(0)-_X[3]*_Y[8]*`&t`(e1we2
we4)+_X[9]*_Y[8]*`&t`(e1we2we3we4)-_X[10]*_Y[3]*`&t`(0)-_X[12]*_Y[11]*`&t`(0)+_
X[1]*_Y[8]*`&t`(e1we4)+_X[7]*_Y[9]*`&t`(0)+_X[16]*_Y[9]*`&t`(0)-_X[11]*_Y[4]*`&
t`(0)-_X[7]*_Y[2]*`&t`(0)-_X[8]*_Y[13]*`&t`(0)+_X[15]*_Y[13]*`&t`(0)-_X[8]*_Y[1
5]*`&t`(0)+_X[7]*_Y[16]*`&t`(0)+_X[3]*_Y[11]*`&t`(e2we3we4)+_X[7]*_Y[5]*`&t`(e1
we3we4)-_X[11]*_Y[15]*`&t`(0)+_X[4]*_Y[12]*`&t`(0)+_X[8]*_Y[11]*`&t`(0)+_X[3]*_
Y[12]*`&t`(0)-_X[7]*_Y[13]*`&t`(0)-_X[10]*_Y[7]*`&t`(e1we2we3we4)+_X[3]*_Y[15]*
`&t`(0)+_X[5]*_Y[14]*`&t`(0)+_X[1]*_Y[12]*`&t`(e1we2we3)-_X[13]*_Y[7]*`&t`(0)+_
X[2]*_Y[13]*`&t`(0)+_X[10]*_Y[2]*`&t`(e1we2we4)-_X[13]*_Y[4]*`&t`(e1we2we3we4)+
_X[5]*_Y[7]*`&t`(e1we3we4)+_X[14]*_Y[14]*`&t`(0)+_X[12]*_Y[2]*`&t`(0)-_X[9]*_Y[
4]*`&t`(0)+_X[10]*_Y[11]*`&t`(0)+_X[12]*_Y[12]*`&t`(0)+_X[10]*_Y[10]*`&t`(0)-_X
[3]*_Y[14]*`&t`(e1we2we3we4)-_X[7]*_Y[10]*`&t`(e1we2we3we4)+_X[12]*_Y[1]*`&t`(e
1we2we3)-_X[9]*_Y[12]*`&t`(0)+_X[13]*_Y[1]*`&t`(e1we2we4)+_X[12]*_Y[15]*`&t`(0)
+_X[11]*_Y[7]*`&t`(0)+_X[16]*_Y[8]*`&t`(0)+_X[11]*_Y[2]*`&t`(e1we3we4)
#####
In dimension 4 the equation `gsw=awaa` is `true`
#####
```

However, be aware that the reversion is the Clifford reversion which introduces possibly additional terms! If one thinks about a Grassmann reversion one would have to set B to be a diagonal bilinear form (their values do not matter and could even be zero), since only in this setting one has identities like e1 &c e2 = e1 &w e2 etc. i.e. the Clifford and Grassmann bases coincide (since only off diagonal B[i,j] terms occur in reordering).

```
> reversion(e2we1);
  subs(B[2,1]=-F[1,2],B[1,2]=F[1,2],%);   ## antisymmetric part
  reversion(reversion(e1we2));            ## reversion is
  involutive !!
```

$$e1we2 + B_{1,2}\, Id - B_{2,1}\, Id$$

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

$$e1we2$$

```
> B:=linalg[diag](1$2);
  reversion(e2we1); ## works out (as expected?) in this case
```

$$B := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$e1we2$$

The graded switch is involutive:

```
> &t(e1,e2);
  gswitch(%,1);
  gswitch(%,1);
```

$$e1 \;\&t\; e2$$

$$-(e2 \;\&t\; e1)$$

$$e1 \;\&t\; e2$$

```
> &t(e1,a*e2+b*e2we3,e1we4-sin(x)*e5);
  gswitch(%,1);
  gswitch(%%,2);
```

$$a\, \&t(e1, e2, e1we4) - a \sin(x)\, \&t(e1, e2, e5) + b\, \&t(e1, e2we3, e1we4)$$

$$- b \sin(x)\ \&t(e1, e2we3, e5)$$
$$-a\ \&t(e2, e1, e1we4) + a \sin(x)\ \&t(e2, e1, e5) + b\ \&t(e2we3, e1, e1we4)$$
$$- b \sin(x)\ \&t(e2we3, e1, e5)$$
$$a\ \&t(e1, e1we4, e2) + a \sin(x)\ \&t(e1, e5, e2) + b\ \&t(e1, e1we4, e2we3)$$
$$- b \sin(x)\ \&t(e1, e5, e2we3)$$

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

```
> gswitch(&t(e1,e2),3);
Error, (in Bigebra:-gswitch) invalid subscript selector
```

```
> printf("worksheet to %f seconds on Intel Pentium M 2.13 GHz 2GB
  RAM RAM WinXP Prof",time()-bench);
worksheet to 2.545000 seconds on Intel Pentium M 2.13 GHz 2GB RAM RAM WinXP Pro
f
>
```

See Also: Bigebra:-`&t`, Bigebra:-switch, Bigebra:-`&gco`

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-init - initialization routine (automatically invoked)

Calling Sequence:

init()

Parameters:

- none.

Output:

- none.

Global variables:

- _CLIENV[_SILENT], _CLIENV[_QDEF_PREFACTOR], _fakenow

Description:

- Every Maple module can supply an initialization routine (see ?module) which is automatically executed at loading time. In the Bigebra module, this mechanism is taken to define some functions which need to reside in the top-level namespace. Furthermore, the tesnor product &t is predefined using the define facility of Maple, which has been patched for serval reasons . Define has further to be changed in order to treat cliscalars as constants.
  Note: define is now able to get a new option domain='{type}' which makes the herewith defined operator {type}-multilinear. I.e. the tensor product can be over integer, polynoms with integer coefficients, etc..
- The function &gco_d needs the Cliplus package which has to be loaded by the user.
- Bigebra can be made to be more verbose if **infolevel[Bigebra]:=3** or higher. By default, the package is very calm.
- Bigebra can simply be used to define user supplied multilinear function, see define.
- As any Maple package, with(Bigebra) returns a list of functions which got loaded in the table Bigebra:-...].

Examples:

```
> restart:with(Clifford):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*]

Suppress startup message:

```
> restart:with(Clifford):with(Bigebra):
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

The tensor product is now ready for use:

```
> &t(a*e1,&t(b*e2+c*e3,e4));
```

$$a\ b\ \&t(e1, e2, e4) + a\ c\ \&t(e1, e3, e4)$$

```
>
```

See Also: Bigebra:-`&t`, Bigebra:-`type/tensorpolynom`

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-linop - define a linear operator on /\V using a matrix
Bigebra:-linop2 - define a linear operator on /\V x /\V using a matrix

Calling Sequence:

c2:= linop(c1,name)

Parameters:

- c1 : a Clifford polynom
- name : the kernel name of matrix entries e.g. R <-> (R[i,j])

Output:

- c2 : a Clifford polynom.

Description:

- The linop command can be used to define linear operators, i.e. elements of End /\V, using matrices. The basis assumed is the standard Grassmann basis of CLIFFORD.
- The main purpose of these two functions is to handle operators which need a long computation time, e.g. the antipode of a Clifford Hopf gebra or the switch (needs linop2) of a Clifford Hopf gebra.
- To be able to map an operator one has to define a wrapper function, see below.
- The linop2 command is similar to linop, but it acts on the space /\V x /\V, i.e. it creates operators from End (/\V x /\V). Examples of such operators are the switch, graded switch, Clifford switch, etc.

Examples:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
  dim_V:=3:
```

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

Let the name of the operator be R, its matrix elements be R[i,j], we have:

```
> linop(Id,R);
  linop(e1,R);
  linop(e1we2,R);
```

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

$$R_{1,1}\, Id + R_{2,1}\, e1 + R_{3,1}\, e2 + R_{4,1}\, e3 + R_{5,1}\, e1we2 + R_{6,1}\, e1we3 + R_{7,1}\, e2we3 + R_{8,1}\, e1we2we3$$

$$R_{1,2}\, Id + R_{2,2}\, e1 + R_{3,2}\, e2 + R_{4,2}\, e3 + R_{5,2}\, e1we2 + R_{6,2}\, e1we3 + R_{7,2}\, e2we3$$

$$+ R_{8,2}\, e1we2we3$$

$$R_{1,5}\, Id + R_{2,5}\, e1 + R_{3,5}\, e2 + R_{4,5}\, e3 + R_{5,5}\, e1we2 + R_{6,5}\, e1we3 + R_{7,5}\, e2we3 + R_{8,5}\, e1we2we3$$

Define an operator (function) R which can be mapped on tensor slots. Then map it on some examples

```
> R:=proc(x) linop(x,R) end:
  mapop(&t(e1,e2we3),1,R);
  mapop(&t(e1,e2we3),2,R);
```

$$R_{1,2}\,(Id \,\&t\, e2we3) + R_{2,2}\,(e1 \,\&t\, e2we3) + R_{3,2}\,(e2 \,\&t\, e2we3) + R_{4,2}\,(e3 \,\&t\, e2we3) + R_{5,2}\,(e1we2 \,\&t\, e2we3) + R_{6,2}\,(e1we3 \,\&t\, e2we3) + R_{7,2}\,(e2we3 \,\&t\, e2we3) + R_{8,2}\,(e1we2we3 \,\&t\, e2we3)$$

$$R_{1,7}\,(e1 \,\&t\, Id) + R_{2,7}\,(e1 \,\&t\, e1) + R_{3,7}\,(e1 \,\&t\, e2) + R_{4,7}\,(e1 \,\&t\, e3) + R_{5,7}\,(e1 \,\&t\, e1we2) + R_{6,7}\,(e1 \,\&t\, e1we3) + R_{7,7}\,(e1 \,\&t\, e2we3) + R_{8,7}\,(e1 \,\&t\, e1we2we3)$$

Define the antipode of the Grassmann algebra over V having dim_V=2 via a matrix and apply it to a 2^2 basis:

```
> dim_V:=2:
  for i from 1 to 4 do for j from 1 to 4 do
   if i<>j then S[i,j]:=0 else if i=2 or i=3 then S[i,j]:=-1 else
  S[i,j]:=1 fi fi
  od:od:
  matS:=linalg[matrix](4,4,(i,j)->S[i,j]);
  operS:=proc(x) linop(x,S) end:
  bas:=cbasis(2);
  map(eval@operS,bas);
  map(gantipode,bas);
```

$$matS := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$bas := [Id, e1, e2, e1we2]$$

$$[Id, -e1, -e2, e1we2]$$

$$[Id, -e1, -e2, e1we2]$$

Such an operator is mapped onto tensor slots using mapop, some examples are:

```
> mapop(&t(e1,e1we2,e2),2,operS);
  mapop(&t(e1,e1we2,e2),3,operS);
```

$$\&t(e1, e1we2, e2)$$

$$-\&t(e1, e1we2, e2)$$

Linop2 is the counterpart for operators acting on V^ &t V^, in our case the vector space dimension is 2 (dim_V=2) and dim V^ = 2^2 = 4 so the product is a 16 times 16 matrix:

```
> bas:=cbasis(dim_V):
  GSW:=op2mat2(gswitch,1);
  `V^2_bas`:=[seq(seq(&t(bas[i],bas[j]),i=1..2^dim_V),j=1..2^dim_
  V)];
  `V^2_GSW_bas`:=convert(evalm(GSW &* `V^2_bas`),list);
  `V^2_gs_bas`:=map(gswitch,`V^2_bas`,1);
  printf("Are the two lists equal ?  %a
  \n",op({seq(is(`V^2_GSW_bas`[i]=`V^2_gs_bas`[i]),i=1..4^dim_V)}
  ));
```

$$GSW := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$V^2\_bas := [Id \,\&t\, Id, e1 \,\&t\, Id, e2 \,\&t\, Id, e1we2 \,\&t\, Id, Id \,\&t\, e1, e1 \,\&t\, e1, e2 \,\&t\, e1, e1we2 \,\&t\, e1, Id \,\&t\, e2, e1 \,\&t\, e2, e2 \,\&t\, e2, e1we2 \,\&t\, e2, Id \,\&t\, e1we2, e1 \,\&t\, e1we2, e2 \,\&t\, e1we2, e1we2 \,\&t\, e1we2]$

$V^2\_GSW\_bas := [Id \,\&t\, Id, Id \,\&t\, e1, Id \,\&t\, e2, Id \,\&t\, e1we2, e1 \,\&t\, Id, -(e1 \,\&t\, e1), -(e1 \,\&t\, e2), e1 \,\&t\, e1we2, e2 \,\&t\, Id, -(e2 \,\&t\, e1), -(e2 \,\&t\, e2), e2 \,\&t\, e1we2, e1we2 \,\&t\, Id, e1we2 \,\&t\, e1, e1we2 \,\&t\, e2, e1we2 \,\&t\, e1we2]$

$V^2\_gs\_bas := [Id \,\&t\, Id, Id \,\&t\, e1, Id \,\&t\, e2, Id \,\&t\, e1we2, e1 \,\&t\, Id, -(e1 \,\&t\, e1), -(e1 \,\&t\, e2), e1 \,\&t\, e1we2, e2 \,\&t\, Id, -(e2 \,\&t\, e1), -(e2 \,\&t\, e2), e2 \,\&t\, e1we2, e1we2 \,\&t\, Id, e1we2 \,\&t\, e1, e1we2 \,\&t\, e2, e1we2 \,\&t\, e1we2]$

```
Are the two lists equal ?  true
```

This is the matrix representation of the graded switch of the Grassmann Hopf gebra, as can be checked by comparing the two lists of basis vectors. We define, using this matrix, the operator 'gs' and compare some elements with the BIGEBRA built in procedure gswitch.

```
> gs:=proc(x) linop2(x,GSW) end:
```

```
gs(&t(e1,e2)),gswitch(&t(e1,e2),1);
gs(&t(e1we2,e2)),gswitch(&t(e1we2,e2),1);
```

$$-(e2\ \&t\ e1),\ -(e2\ \&t\ e1)$$

$$e2\ \&t\ e1we2,\ e2\ \&t\ e1we2$$

Using mapop2, we can map 2->2 tensor operators onto two adjacent tensor slots i,i+1 of a tensor.

```
> mapop2(&t(e1,e2,e3,e4),1,gs);
```

$$-\&t(e2, e1, e3, e4)$$

```
> mapop2(&t(e1,e1we2,e2,e1we2),1,gs);
mapop2(&t(e1,e1we2,e2,e1we2),2,gs);
mapop2(&t(e1,e1we2,e2,e1we2),3,gs);
```

$$\&t(e1we2, e1, e2, e1we2)$$

$$\&t(e1, e2, e1we2, e1we2)$$

$$\&t(e1, e1we2, e1we2, e2)$$

```
> printf("The worksheet took %f seconds to compute on Intel
  Pentium M 2.13 GHz 2GB RAM WinXP Prof",time()-bench);
The worksheet took 0.438000 seconds to compute on Intel Pentium M 2.13 GHz 2GB
RAM WinXP Prof
>
```

NOTE: If the entries of the tensor polynom are out of the bound of the matrix, this function may go into an **endless loop**! E.g. mapop2(&t(e5,e6),1,gs); in our example, since dim_V was 2.

See Also: Bigebra:-mapop, Bigebra:-mapop2, Bigebra:-EV, Bigebra:-pairing, Bigebra:-op2mat, Bigebra:-list2mat2, Bigebra help page

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-linop - define a linear operator on /\V using a matrix
Bigebra:-linop2 - define a linear operator on /\V x /\V using a matrix

Calling Sequence:

c2:= linop(c1,name)

Parameters:

- c1 : a Clifford polynom
- name : the kernel name of matrix entries e.g. R <-> (R[i,j])

Output:

- c2 : a Clifford polynom.

Description:

- The linop command can be used to define linear operators, i.e. elements of End /\V, using matrices. The basis assumed is the standard Grassmann basis of CLIFFORD.
- The main purpose of these two functions is to handle operators which need a long computation time, e.g. the antipode of a Clifford Hopf gebra or the switch (needs linop2) of a Clifford Hopf gebra.
- To be able to map an operator one has to define a wrapper function, see below.
- The linop2 command is similar to linop, but it acts on the space /\V x /\V, i.e. it creates operators from End (/\V x /\V). Examples of such operators are the switch, graded switch, Clifford switch, etc.

Examples:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
  dim_V:=3:
```

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

Let the name of the operator be R, its matrix elements be R[i,j], we have:

```
> linop(Id,R);
  linop(e1,R);
  linop(e1we2,R);
```

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

$$R_{1,1}\, Id + R_{2,1}\, e1 + R_{3,1}\, e2 + R_{4,1}\, e3 + R_{5,1}\, e1we2 + R_{6,1}\, e1we3 + R_{7,1}\, e2we3 + R_{8,1}\, e1we2we3$$

$$R_{1,2}\, Id + R_{2,2}\, e1 + R_{3,2}\, e2 + R_{4,2}\, e3 + R_{5,2}\, e1we2 + R_{6,2}\, e1we3 + R_{7,2}\, e2we3$$

$$+ R_{8,2}\, e1we2we3$$

$$R_{1,5}\, Id + R_{2,5}\, e1 + R_{3,5}\, e2 + R_{4,5}\, e3 + R_{5,5}\, e1we2 + R_{6,5}\, e1we3 + R_{7,5}\, e2we3 + R_{8,5}\, e1we2we3$$

Define an operator (function) R which can be mapped on tensor slots. Then map it on some examples

```
> R:=proc(x) linop(x,R) end:
  mapop(&t(e1,e2we3),1,R);
  mapop(&t(e1,e2we3),2,R);
```

$$R_{1,2}\,(Id \,\&t\, e2we3) + R_{2,2}\,(e1 \,\&t\, e2we3) + R_{3,2}\,(e2 \,\&t\, e2we3) + R_{4,2}\,(e3 \,\&t\, e2we3) + R_{5,2}\,(e1we2 \,\&t\, e2we3) + R_{6,2}\,(e1we3 \,\&t\, e2we3) + R_{7,2}\,(e2we3 \,\&t\, e2we3) + R_{8,2}\,(e1we2we3 \,\&t\, e2we3)$$

$$R_{1,7}\,(e1 \,\&t\, Id) + R_{2,7}\,(e1 \,\&t\, e1) + R_{3,7}\,(e1 \,\&t\, e2) + R_{4,7}\,(e1 \,\&t\, e3) + R_{5,7}\,(e1 \,\&t\, e1we2) + R_{6,7}\,(e1 \,\&t\, e1we3) + R_{7,7}\,(e1 \,\&t\, e2we3) + R_{8,7}\,(e1 \,\&t\, e1we2we3)$$

Define the antipode of the Grassmann algebra over V having dim_V=2 via a matrix and apply it to a 2^2 basis:

```
> dim_V:=2:
  for i from 1 to 4 do for j from 1 to 4 do
   if i<>j then S[i,j]:=0 else if i=2 or i=3 then S[i,j]:=-1 else
  S[i,j]:=1 fi fi
  od:od:
  matS:=linalg[matrix](4,4,(i,j)->S[i,j]);
  operS:=proc(x) linop(x,S) end:
  bas:=cbasis(2);
  map(eval@operS,bas);
  map(gantipode,bas);
```

$$matS := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$bas := [Id, e1, e2, e1we2]$$

$$[Id, -e1, -e2, e1we2]$$

$$[Id, -e1, -e2, e1we2]$$

Such an operator is mapped onto tensor slots using mapop, some examples are:

```
> mapop(&t(e1,e1we2,e2),2,operS);
  mapop(&t(e1,e1we2,e2),3,operS);
```

$$\&t(e1, e1we2, e2)$$

$$-\&t(e1, e1we2, e2)$$

Linop2 is the counterpart for operators acting on V^ &t V^, in our case the vector space dimension is 2 (dim_V=2) and dim V^ = 2^2 = 4 so the product is a 16 times 16 matrix:

```
> bas:=cbasis(dim_V):
  GSW:=op2mat2(gswitch,1);
  `V^2_bas`:=[seq(seq(&t(bas[i],bas[j]),i=1..2^dim_V),j=1..2^dim_
  V)];
  `V^2_GSW_bas`:=convert(evalm(GSW &* `V^2_bas`),list);
  `V^2_gs_bas`:=map(gswitch,`V^2_bas`,1);
  printf("Are the two lists equal ?  %a
  \n",op({seq(is(`V^2_GSW_bas`[i]=`V^2_gs_bas`[i]),i=1..4^dim_V)}
  ));
```

$$GSW := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$V^2\_bas := [Id \,\&t\, Id, e1 \,\&t\, Id, e2 \,\&t\, Id, e1we2 \,\&t\, Id, Id \,\&t\, e1, e1 \,\&t\, e1, e2 \,\&t\, e1, e1we2 \,\&t\, e1, Id \,\&t\, e2, e1 \,\&t\, e2, e2 \,\&t\, e2, e1we2 \,\&t\, e2, Id \,\&t\, e1we2, e1 \,\&t\, e1we2, e2 \,\&t\, e1we2, e1we2 \,\&t\, e1we2]$

$V^2\_GSW\_bas := [Id \,\&t\, Id, Id \,\&t\, e1, Id \,\&t\, e2, Id \,\&t\, e1we2, e1 \,\&t\, Id, -(e1 \,\&t\, e1), -(e1 \,\&t\, e2), e1 \,\&t\, e1we2, e2 \,\&t\, Id, -(e2 \,\&t\, e1), -(e2 \,\&t\, e2), e2 \,\&t\, e1we2, e1we2 \,\&t\, Id, e1we2 \,\&t\, e1, e1we2 \,\&t\, e2, e1we2 \,\&t\, e1we2]$

$V^2\_gs\_bas := [Id \,\&t\, Id, Id \,\&t\, e1, Id \,\&t\, e2, Id \,\&t\, e1we2, e1 \,\&t\, Id, -(e1 \,\&t\, e1), -(e1 \,\&t\, e2), e1 \,\&t\, e1we2, e2 \,\&t\, Id, -(e2 \,\&t\, e1), -(e2 \,\&t\, e2), e2 \,\&t\, e1we2, e1we2 \,\&t\, Id, e1we2 \,\&t\, e1, e1we2 \,\&t\, e2, e1we2 \,\&t\, e1we2]$

```
Are the two lists equal ?  true
```

This is the matrix representation of the graded switch of the Grassmann Hopf gebra, as can be checked by comaping the two lists of basis vectors. We define, using this matrix, the operator 'gs' and compare some elements with the BIGEBRA built in procedure gswitch.

```
> gs:=proc(x) linop2(x,GSW) end:
```

```
gs(&t(e1,e2)),gswitch(&t(e1,e2),1);
gs(&t(e1we2,e2)),gswitch(&t(e1we2,e2),1);
```

$$-(e2\ \&t\ e1),\ -(e2\ \&t\ e1)$$

$$e2\ \&t\ e1we2,\ e2\ \&t\ e1we2$$

Using mapop2, we can map 2->2 tensor operators onto two adjacent tensor slots i,i+1 of a tensor.

```
> mapop2(&t(e1,e2,e3,e4),1,gs);
```

$$-\&t(e2, e1, e3, e4)$$

```
> mapop2(&t(e1,e1we2,e2,e1we2),1,gs);
mapop2(&t(e1,e1we2,e2,e1we2),2,gs);
mapop2(&t(e1,e1we2,e2,e1we2),3,gs);
```

$$\&t(e1we2, e1, e2, e1we2)$$

$$\&t(e1, e2, e1we2, e1we2)$$

$$\&t(e1, e1we2, e1we2, e2)$$

```
> printf("The worksheet took %f seconds to compute on Intel
  Pentium M 2.13 GHz 2GB RAM WinXP Prof",time()-bench);
The worksheet took 0.453000 seconds to compute on Intel Pentium M 2.13 GHz 2GB
RAM WinXP Prof
>
```

NOTE: If the entries of the tensor polynom are out of the bound of the matrix, this function may go into an **endless loop**! E.g. mapop2(&t(e5,e6),1,gs); in our example, since dim_V was 2.

See Also: Bigebra:-mapop, Bigebra:-mapop2, Bigebra:-EV, Bigebra:-pairing, Bigebra:-op2mat, Bigebra:-list2mat2, Bigebra help page

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-lists2mat - derive a matrix of an operator whose action is given as two lists of source and target elements: op: x --> op(x) = y

Bigebra:-lists2mat2 - derive a matrix of an operator from source and target element in V^ x V^ ---> V^ x V^

Calling Sequence:

m1:= lists2mat(list1,list2)

m1:= lists2mat2(list1,list2)

Parameters:

- list1,list2 : two lists of elements related by the action of an operator list2[i] := operator(list[i]) for all i.

Output:

- m1 : a 2^dim_V x 2^dim_V matrix (a 4^dim_V x 4^dim_V matrix)

Global parameters:

- dim_V

Description:

- The lists2mat command is useful to derive matrix forms of linear operators. It can be used together with linop (linop2) to move from an operator description to matrix form and back. The derived matrices are regarded as elements of End V^ (End V^ &t V^) . The basis used is assumed to be the standard Grassmann basis of Clifford or the following sum^{2^dim_V}_{i,j=1} &t(b[i],b[j]), where the b[i] are Grassmann bases of V^.
- The main purpose of these two functions is to get a matrix form of heavily used operators needed in long computation time, e.g. the antipode of a Clifford Hopf gebra or the switch (may be derived from lists2mat2). The functionality of linalg is then available to those matrices.

Examples:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
  dim_V:=2:
```

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

Let the name of the operator be R, its matrix elements be R[i,j], we have:

```
> linop(Id,R);
  linop(e1,R);
  linop(e1we2,R);
```

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

$$R_{1,1}\, Id + R_{2,1}\, e1 + R_{3,1}\, e2 + R_{4,1}\, e1we2$$
$$R_{1,2}\, Id + R_{2,2}\, e1 + R_{3,2}\, e2 + R_{4,2}\, e1we2$$
$$R_{1,4}\, Id + R_{2,4}\, e1 + R_{3,4}\, e2 + R_{4,4}\, e1we2$$

Get the matrix form of linop(x,R). First we define an operator R from which we derive the matrix form, but we show in a second line that parameters may be passed to op2mat which allows us to use linop directly:

```
> sbas:=cbasis(dim_V):          # define a spanning set (source)
  tbas:=map(linop,sbas,R);      # define the target elements
  lists2mat(sbas,tbas);         #
```

$$tbas := [R_{1,1}\, Id + R_{2,1}\, e1 + R_{3,1}\, e2 + R_{4,1}\, e1we2,\ R_{1,2}\, Id + R_{2,2}\, e1 + R_{3,2}\, e2 + R_{4,2}\, e1we2,\ R_{1,3}\, Id + R_{2,3}\, e1 + R_{3,3}\, e2 + R_{4,3}\, e1we2,\ R_{1,4}\, Id + R_{2,4}\, e1 + R_{3,4}\, e2 + R_{4,4}\, e1we2]$$

$$\begin{bmatrix} R_{1,1} & R_{1,2} & R_{1,3} & R_{1,4} \\ R_{2,1} & R_{2,2} & R_{2,3} & R_{2,4} \\ R_{3,1} & R_{3,2} & R_{3,3} & R_{3,4} \\ R_{4,1} & R_{4,2} & R_{4,3} & R_{4,4} \end{bmatrix}$$

Derive the matrix of the Grassmann antipode, in dim_V=2 we get a 4x4 matrix:

```
> abas:=map(gantipode,sbas,1);:
  matS:=lists2mat(sbas,abas);
  map(linop,sbas,matS);
  map(gantipode,sbas,1);
```

$$abas := [Id, -e1, -e2, e1we2]$$

$$matS := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[Id, -e1, -e2, e1we2]$$
$$[Id, -e1, -e2, e1we2]$$

We checked using linop that this is the same operator as defined abstractly. Hence our indexing is compatible.

lists2mat2 is the counterpart for operators acting on V^ &t V^, in our case the vector space dimension is 2 (dim_V=2) and dim V^ = 2^2 = 4 so the output is a 16 times 16 matrix:

We check the function by an examples and will see that it is compatible in indexing with linop2:

```
> `V^2_sbas`:=[seq(seq(&t(sbas[i],sbas[j]),i=1..2^dim_V),j=1..2^d
  im_V)];
  `V^2_tbas`:=map(gswitch,`V^2_sbas`,1);                    # act
```

```
using the operator
GSW:=lists2mat2(`V^2_sbas`,`V^2_tbas`);
```

$V^2\_sbas := [Id \,\&t\, Id, e1 \,\&t\, Id, e2 \,\&t\, Id, e1we2 \,\&t\, Id, Id \,\&t\, e1, e1 \,\&t\, e1, e2 \,\&t\, e1, e1we2 \,\&t\, e1, Id \,\&t\, e2, e1 \,\&t\, e2, e2 \,\&t\, e2, e1we2 \,\&t\, e2, Id \,\&t\, e1we2, e1 \,\&t\, e1we2, e2 \,\&t\, e1we2, e1we2 \,\&t\, e1we2]$

$V^2\_tbas := [Id \,\&t\, Id, Id \,\&t\, e1, Id \,\&t\, e2, Id \,\&t\, e1we2, e1 \,\&t\, Id, -(e1 \,\&t\, e1), -(e1 \,\&t\, e2), e1 \,\&t\, e1we2, e2 \,\&t\, Id, -(e2 \,\&t\, e1), -(e2 \,\&t\, e2), e2 \,\&t\, e1we2, e1we2 \,\&t\, Id, e1we2 \,\&t\, e1, e1we2 \,\&t\, e2, e1we2 \,\&t\, e1we2]$

$$GSW := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

For more examples see op2mat . Which can easily be converted to the lists2mat case.

```
> printf("The worksheet took %f seconds to compute on Intel
  Pentium M 2.13 GHz 2GB RAM WinXP Prof",time()-bench);
The worksheet took 0.453000 seconds to compute on Intel Pentium M 2.13 GHz 2GB
RAM WinXP Prof
>
```

NOTE: If the entries of the tensor polynom are out of the bound of the matrix, this function may go into an **endless loop**! E.g. mapop2(&t(e5,e6),1,gs); in our example, since dim_V was 2.

See Also: Bigebra:-mapop, Bigebra:-mapop2, Bigebra:-EV, Bigebra:-pairing, Bigebra:-linop, Bigebra:-op2mat, Bigebra:-help

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-make_BI_Id - initialization of the Clifford co-product

Calling Sequence:

- make_BI_Id();

Parameters:

- none.

Global variables:

- BI_Id - set by make_BI_Id
- dim_V - dimension of B, as described in CLIFFORD (see CLIFFORD_ENV)
- BI - dim_V x dim_V matrix of the Clifford co-scalarproduct w.r.t. a co-basis

Description:

- Like the Clifford product, a Clifford co-product needs a bilinear form to be defined on the base space of the Grassmann algebra. In the case of the co-product this form is tied to co-one-vectors, so it is called co-scalarproduct. Since we deal with finite dimensional spaces, the dimension of the co-vector space is dim_V as for the vector space used by CLIFFORD. Hence we use the ***global variable dim_V***, which has to be assigned (caution: CLIFFORD sets this variable to 9 by default which may result in a very long initialization). The matrix of the Clifford co-product is stored in BI. BI can be assigned freely, without any restrictions or relations to the matrix of the Clifford scalar product B. BI can be singular or nonsymmetric or even zero, in which case the Clifford co-product reduces to the Grassmann co-product.
- The make_BI_Id() function is needed to initialize the Clifford co-product which is calculated using the Rota-sausage tangle. In Sweedler notation this reads:

  Δ_(&c)(x) = (wedge &t wedge)(Id &t BI_Id &t Id)Δ(x)

  where Δ_(&c) is the Clifford co-product, Δ is the Grassmann co-product, BI_Id is a two tensor, BI_Id_(1) &t BI_Id_(2) in Sweedler notation. The function make_BI_Id sets this global variable BI_Id which is essentially the cap tangle in the Rota-sausage tangle. Note, that unlike in the Grassmann case, the Clifford co-product of the identity is not the tensor product of two identity elements. That is, the unit of the Clifford product is not Clifford co-algebra homomorphism. Clifford algebras are not connected, see Milnore and Moore.
- BI_Id represents the 'cup' tangle in the co-cliffordization.
- BI_Id is not a tensor polynom but a data structure to speed up internal computations for the clifford co-product. In later versions of BIGEBRA this variable may be incorporated directly into the Clifford co-product and will be dropped in the package.

Examples:

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

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

```
> dim_V:=2:
  BI:=linalg[matrix](dim_V,dim_V,[a,b,c,d]); #co-scalarproduct
```

$$BI := \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

```
> assigned(BI_Id);
```

$$false$$

The Clifford co-product does not yet work

```
> &cco(&t(e1,e2),1);
```

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

$$BI\_Id_{1_1}\, BI\_Id_{1_2}\, BI\_Id_{1_3}\, \&t(Id, e1, e2) + BI\_Id_{1_1}\, BI\_Id_{1_2}\, BI\_Id_{1_3}\, \&t(e1, Id, e2)$$

```
> make_BI_Id();
```

$$(Id\ \&t\ Id) + a\,(e1\ \&t\ e1) + c\,(e2\ \&t\ e1) + b\,(e1\ \&t\ e2) + d\,(e2\ \&t\ e2) + (c\,b - d\,a)\,(e1we2\ \&t\ e1we2)$$

```
> assigned(BI_Id);
```

$$true$$

We can compose from BI_Id the cup tangle as a tensor of rank 2. Therefore one has to know, that the data structure of BI_Id is a list of lists, where the inner list contains a triple of a [prefactor, first tensor slot, second tensor slot]. Adding this up yields:

```
> BI_Id;
  cup:=add(BI_Id[i][1]*&t(BI_Id[i][2],BI_Id[i][3]),i=1..nops(BI_Id));  # this is returend by make_BI_Id()
```

$$[[1, Id, Id], [a, e1, e1], [c, e2, e1], [b, e1, e2], [d, e2, e2], [c\,b - d\,a, e1we2, e1we2]]$$

$$cup := (Id\ \&t\ Id) + a\,(e1\ \&t\ e1) + c\,(e2\ \&t\ e1) + b\,(e1\ \&t\ e2) + d\,(e2\ \&t\ e2) + (c\,b - d\,a)\,(e1we2\ \&t\ e1we2)$$

Let us check that BI_Id is the Clifford co-product of Id:

```
> &cco(Id);
```

$$(Id\ \&t\ Id) + a\,(e1\ \&t\ e1) + c\,(e2\ \&t\ e1) + b\,(e1\ \&t\ e2) + d\,(e2\ \&t\ e2) + (c\,b - d\,a)\,(e1we2\ \&t\ e1we2)$$

Further examples are:

```
> &cco(e1);
  &cco(e1we2);
```

$$(Id\ \&t\ e1) - b\,(e1\ \&t\ e1we2) - d\,(e2\ \&t\ e1we2) + (e1\ \&t\ Id) + c\,(e1we2\ \&t\ e1) + d\,(e1we2\ \&t\ e2)$$

$$(e1\ \&t\ e2) + (Id\ \&t\ e1we2) + c\,(e1we2\ \&t\ e1we2) - (e2\ \&t\ e1) - b\,(e1we2\ \&t\ e1we2) + (e1we2\ \&t\ Id)$$

```
> bas:=cbasis(dim_V):
  &cco(add(a[i]*bas[i],i=1..2^dim_V));
```

$$a_1\, d\,(e2\ \&t\ e2) + a_1\, b\,(e1\ \&t\ e2) + a_1\, c\,(e2\ \&t\ e1) + a_4\, c\,(e1we2\ \&t\ e1we2)$$
$$- a_3\, b\,(e1we2\ \&t\ e2) - a_3\, a\,(e1we2\ \&t\ e1) + a_3\, a\,(e1\ \&t\ e1we2) + a_3\, c\,(e2\ \&t\ e1we2)$$
$$+ a_2\, c\,(e1we2\ \&t\ e1) + a_1\,(e1we2\ \&t\ e1we2)\, c\, b - a_2\, d\,(e2\ \&t\ e1we2)$$
$$- a_1\,(e1we2\ \&t\ e1we2)\, d\, a + a_1\,(Id\ \&t\ Id) - a_4\, b\,(e1we2\ \&t\ e1we2) + a_1\, a\,(e1\ \&t\ e1)$$
$$+ a_3\,(Id\ \&t\ e2) + a_3\,(e2\ \&t\ Id) + a_2\, d\,(e1we2\ \&t\ e2) - a_2\, b\,(e1\ \&t\ e1we2) + a_4\,(e1\ \&t\ e2)$$
$$+ a_4\,(Id\ \&t\ e1we2) - a_4\,(e2\ \&t\ e1) + a_4\,(e1we2\ \&t\ Id) + a_2\,(Id\ \&t\ e1) + a_2\,(e1\ \&t\ Id)$$

```
> printf("Worksheet to %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM WinXP Prof",time()-bench);
Worksheet to 0.529000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM Wi
nXP Prof
>
```

See Also: Bigebra:-`&cco`, Bigebra:-`&gco`, Bigebra:-`&t`

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-mapop - maps a linear operator from End V^ onto tensor slots
Bigebra:-mapop2 - maps a linear operator from End (V^ &t V^) onto two tensor slots

Calling Sequence:

t2 := mapop(t1,i,End1)
t2 := mapop2(t1,i,End2)

Parameters:

- t1 : a tensor polynom
- i : the i-th tensor slot
- End1 : a (linear) operator / an endomorphism End /\V
- End2 : a (linear) operator / an endomorphisms End /\V &t /\V

Output:

- t2 : is a tensor polynom

Description:

- The mapop device allows to define operators and map them onto a certain place in a tensor polynom. Linear operators can be defined using the linop device. Arbitrary not necessarily linear operators can be defined as functions of one or two tensor slots, however, due to the multilinearity of &t they will be nearly linear.

Examples:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
  dim_V:=3:
```

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

We define a linear operator, connected to the integral of a Grassmann Hopf gebra:

```
> integral:=proc(a1) Id*coeff(a1,&w(seq(cat(e,i),i=1..dim_V)))
  end:
```

Now we can apply the integral to any tensor we want:

```
> mapop(&t(e1,e1,e1we2we3),2,integral);
  mapop(&t(e1,e1,e1we2we3),3,integral);
```

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

$$0$$

$$\&t(e1, e1, Id)$$

We can define an operator which projects out the even grade elements:

```
> evengrades:=proc(a1) if gradeinv(a1) = a1 then a1 else 0*Id fi
  end:
```

And apply this to an tensor having multiple grades, but note that the even grades are only projected out in the i-th slot.

```
> mapop(&t(Id,Id+e1+e3+e2we3+e1we2+e1we2we3,e2),2,evengrades);
```

$$\&t(Id, Id, e2) + \&t(Id, e2we3, e2) + \&t(Id, e1we2, e2)$$

As an example for mapop2, we define a function of two tensor elements which compares neighbors and returns Id &t Id if they agree. To exemplify the mechanism at work, we postpone evaluation and evaluate the expanded expression only afterwards.

```
> same_neighbours:=proc(a1) local x,y; x,y:=op(a1); if x=y then
  &t(Id,Id) else 0 fi end:
> same_neighbours(&t(e1,e1)),same_neighbours(&t(e1,e2));;
```

$$Id \;\&t\; Id, 0$$

```
> 'mapop2'(&t(e1,e2+e3,e1+e2,e3),2,same_neighbours);
  eval(%);
```

$$Bigebra{:-}mapop2(\&t(e1, e2, e1, e3) + \&t(e1, e2, e2, e3) + \&t(e1, e3, e1, e3) + \&t(e1, e3, e2, e3), 2, same\_neighbours)$$

$$\&t(e1, Id, Id, e3)$$

The mapop functionality is effectively combined with the linop and linop2 functions which define linear operators in terms of matrix elements w.r.t. the standard Grassmann basis.

```
> matS:=linalg[matrix](2^3,2^3,(i,j)->if i=j+1 or i+1=j then 1
  elif i=j then 2 else 0 fi);
```

$$matS := \begin{bmatrix} 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 \end{bmatrix}$$

```
> S:=proc(x) linop(x,matS) end:
> tcollect(mapop(&t(Id,a*e1+b*e2+c*e3,e1we2),2,S));
```

$$(2\,a + b)\,\&t(Id, e1, e1we2) + (b + 2\,c)\,\&t(Id, e3, e1we2) + a\,\&t(Id, Id, e1we2) + (a + 2\,b + c)\,\&t(Id, e2, e1we2) + c\,\&t(Id, e1we2, e1we2)$$

```
> printf("Worksheet to %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM WinXP Prof",time()-bench);
Worksheet to 0.094000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM Wi
nXP Prof
>
```

For a corresponding example using linop2 and mapop2 see the help page for linop.

See Also: Bigebra:-linop, Bigebra:-tcollect, Bigebra:-contract, Bigebra help page

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-mapop - maps a linear operator from End V^ onto tensor slots
Bigebra:-mapop2 - maps a linear operator from End (V^ &t V^) onto two tensor slots

Calling Sequence:

t2 := mapop(t1,i,End1)
t2 := mapop2(t1,i,End2)

Parameters:

- t1 : a tensor polynom
- i : the i-th tensor slot
- End1 : a (linear) operator / an endomorphism End /\V
- End2 : a (linear) operator / an endomorphisms End /\V &t /\V

Output:

- t2 : is a tensor polynom

Description:

- The mapop device allows to define operators and map them onto a certain place in a tensor polynom. Linear operators can be defined using the linop device. Arbitrary not necessarily linear operators can be defined as functions of one or two tensor slots, however, due to the multilinearity of &t they will be nearly linear.

Examples:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
  dim_V:=3:
```

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

We define a linear operator, connected to the integral of a Grassmann Hopf gebra:

```
> integral:=proc(a1) Id*coeff(a1,&w(seq(cat(e,i),i=1..dim_V)))
  end:
```

Now we can apply the integral to any tensor we want:

```
> mapop(&t(e1,e1,e1we2we3),2,integral);
  mapop(&t(e1,e1,e1we2we3),3,integral);
```

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

$$0$$

$$\&t(e1, e1, Id)$$

We can define an operator which projects out the even grade elements:

```
> evengrades:=proc(a1) if gradeinv(a1) = a1 then a1 else 0*Id fi
  end:
```

And apply this to an tensor having multiple grades, but note that the even grades are only projected out in the i-th slot.

```
> mapop(&t(Id,Id+e1+e3+e2we3+e1we2+e1we2we3,e2),2,evengrades);
```

$$\&t(Id, Id, e2) + \&t(Id, e2we3, e2) + \&t(Id, e1we2, e2)$$

As an example for mapop2, we define a function of two tensor elements which compares neighbors and returns Id &t Id if they agree. To exemplify the mechanism at work, we postpone evaluation and evaluate the expanded expression only afterwards.

```
> same_neighbours:=proc(a1) local x,y; x,y:=op(a1); if x=y then
  &t(Id,Id) else 0 fi end:
> same_neighbours(&t(e1,e1)),same_neighbours(&t(e1,e2));;
```

$$Id \,\&t\, Id, 0$$

```
> 'mapop2'(&t(e1,e2+e3,e1+e2,e3),2,same_neighbours);
  eval(%);
```

$$Bigebra{:}\text{-}mapop2(\&t(e1, e2, e1, e3) + \&t(e1, e2, e2, e3) + \&t(e1, e3, e1, e3) + \&t(e1, e3, e2, e3), 2, same\_neighbours)$$

$$\&t(e1, Id, Id, e3)$$

The mapop functionality is effectively combined with the linop and linop2 functions which define linear operators in terms of matrix elements w.r.t. the standard Grassmann basis.

```
> matS:=linalg[matrix](2^3,2^3,(i,j)->if i=j+1 or i+1=j then 1
  elif i=j then 2 else 0 fi);
```

$$matS := \begin{bmatrix} 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 \end{bmatrix}$$

```
> S:=proc(x) linop(x,matS) end:
> tcollect(mapop(&t(Id,a*e1+b*e2+c*e3,e1we2),2,S));
```

$$(2\,a + b)\,\&t(Id, e1, e1we2) + (a + 2\,b + c)\,\&t(Id, e2, e1we2) + a\,\&t(Id, Id, e1we2) + (b + 2\,c)\,\&t(Id, e3, e1we2) + c\,\&t(Id, e1we2, e1we2)$$

```
> printf("Worksheet to %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM WinXP Prof",time()-bench);
Worksheet to 0.108000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM Wi
nXP Prof
>
```

For a corresponding example using linop2 and mapop2 see the help page for linop.

See Also: Bigebra:-linop, Bigebra:-tcollect, Bigebra:-contract, Bigebra help page

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-`&v` -- the vee (meet) product.
Bigebra:-meet -- the meet product.

Calling Sequence:

c3 := &v(c1,c2) [or c1 &v c2, not recommended]
c3 := meet(c1,c2) -- synonym.

Parameters:

- c1,c2 - expressions of `type/clipolynom`

Output:

- c3 - expression of `type/clipolynom`

Global variables:

- dim_V - dimension of the vector space (V,B) that is defined in CLIFFORD as a global variable.

Description:

- The pair of operations wedge (i.e. join) and ***meet*** acting on Grassmann multi-vectors make up, together with the duality operator, the ***Grassmann Cayley algebra***. This algebra is of tremendous importance in geometrical applications like robotics, visual perception, camera calibration. However, incidence geometries have their own well developed mathematical theory, see e.g. P. Dembowski, Finite Geometries, Springer Verlag, New York, 1968.

- To avoid confusion we should point out that the notion of a meet is not unique in literature. Let A be a homogeneous decomposable multivector called an ***extensor***. Every such extensor spans a linear subspace of the space over which it was constructed. The span of A is called the ***support*** of A, denoted as *supp A*. Meet and join can be defined in set theoretic terms on the support of extensors. Let A, B denote extensors, one defines:

  A **cup** B := {x in V | x in supp A ***or*** in supp B} i.e. the set theoretic union
  A **cap** B := {x in V | x in supp A ***and*** in supp B} i.e. the set theoretic intersection

  The operators **cup** and **cap** are the same as in the set theory. Under these operations every set is an idempotent: A **cup** A = A and B **cap** B = B. Moreover, one finds **cup** *o* **cup** = Id and **cap** *o* **cap** = Id for these operators. Including the set theoretic operation of taking the complement, (A -> |A with A **cup** |A = whole space, where we have used, in lack of an over bar, the Grassmann notation of a preceding bar), this constitutes the structure of an ortho-modular lattice. Boolean logic is based on this construction. The two operations of meet and join are related by de Morgan laws:

  |(A **cup** B) = (|A) **cap** (|B)
  |(A **cap** B) = (|A) **cup** (|B).

In terms of logic we have: **cup** = **and**, **cap** = **or**, | = **not**.

In CLIFFORD and Bigebra packages, the meet and join are defined in the following way:

The wedge product of two extensors A and B is an extensor C whose support the disjoint union of the supports of A and B. However, extensors having the same support are isomorphic (interchangeable). ***We define the join to be this wedge operation.*** The meet is usually defined using a symmetric correlation in the projective space **P**^dim(V). It needs thus a theorem to show that the meet is independent from its construction. Grassmann defined the meet, which he called ***regressive product***, in [A2]**, 1862,** §5, No. 94 page 61ff. The regressive product was already present in [A1], chapter 3, §125ff. Grassmann edited in 1877 a reprint with annotations where he gave some explanations on his presentation. A careful reading shows that the regressive product was present already in 1844. The Ergänzung is not explicit in [A1], but Grassmann discusses the grade of the complement |A which he calls there 'Ergänzzahlen' (A1 §133)) using the so called 'Ergänzung' (Grassmann A2, §4, No. 89 page 57), which we defined already above as |, of an extensor A to be |A. In analogy to de Morgan laws (which he most likely did not know) as:

| (A ∨ B) := (|A) ∧ (|B).

[Grassmann used no sign for products, having over 16 of them working, many at the same time and their type had to be deduced from the context. Furthermore, he used no parentheses which makes his writings cumbersome to read. The ∧ sign mutated from an (uppercase) Lambda used by Burali-Forti and Marcolongo to be the wedge of Bourbaki.]

The usage of the Ergänzung points out clearly that the meet depends on the dimension of the space. We will see below, that this definition of the meet is computationally very ineffective.

Alfred Lotze (Über eine neue Begründung der regressiven Multiplikation extensiver Größen in einem Hauptgebiet n-ter Stufe, Jahresbericht der DMV, 57:102-110,1955) defined a *universal formula* for the regressive product of r-factors. He showed that if one considers the n-1 dimensional space as a space of co-vectors, then the original *wedge product* becomes by the same formula the regressive product of the co-vectors, pointing out the fact that a symmetric correlation is needed for this purpose. That is: (n-1)-multi-vectors are not co-vectors, but may be seen as reciprocal vectors. In [4], G.-C. Rota and coworkers gave a definition of the meet in terms of a Peano algebra which is essentially the same construction. However, they used the notion of Hopf algebra which allows one to write these formulas in a comprehensive way.

The Grassmann wedge product has as logical counterpart in the exclusive or ***xor*** , the Ergänzung is **not** w.r.t. the chosen volume form of the space V the Grassmann algebra is build over. The meaning of the meet follows from his duality relation.

- In Bigebra, the meet and &v (vee) products are implemented as follows (**note** the order of factors

in the bracket!):

meet(c1,c2) := [c2_(1),c1] c2_(2)
&v(c1,c2) := c1_(1) [c2,c1_(2)] ,

where the bracket [,] is a scalar-valued alternating multilinear volume form and the co-products are given in Sweedler notation. It can be shown (and is tested below) that both forms represent the same operation.

- The Hopf algebraic definition of the meet gives us a great deal of ***computational benefits*** as we will show below in some benchmarks. However it works exactly as the Grassmann regressive product.
- Grassmann introduced the so called *stereometric product*, which, being context sensitive, switches between the wedge and the &v (vee-) product. Using polymorphism, this could be implemented, and the user can easily program such a wrapper function. We found it peculiar to implement it using the same notation for basis elements for vectors and co-vectors.
- The meet as defined here is independent of the assigned scalar product *B* or the assigned co-scalar product *BI*. In fact, it can be shown that the vee-product is SL_n invariant. If one is interested in projective geometry, the invariants derived from meet and join are GL_n invariants.
- The meet product is related to the notion of a Hopf algebraic integral [3]. As a remarkable fact, in any Clifford Hopf gebra over dim V = 2 one is not able to find a non zero integral. The notion of meet has thus to be reconsidered in the deformed case.

Examples:

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

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

Infix form (**not recommended**, see help page on &t). Note that we have not assigned a scalar- or co-scalarproduct.

```
> dim_V:=2:
  e1 &v e2;
  e1 &v e1we2, e1we2 &v e2, e1we2 &v e1we2;
```

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

$$-Id$$

$$e1, e2, e1we2$$

First of all let us check that both versions of the meet compute indeed identically:

```
> for i from 1 to 5 do
  dim_V:=i:
```

```
bas:=cbasis(dim_V);
X:=add(_X[i]*bas[i],i=1..2^dim_V):
Y:=add(_Y[i]*bas[i],i=1..2^dim_V):
bench:=time():
printf("In dimension %d the equation `meet(X,Y)=&v(X,Y)` is %a
and took %f seconds to verify on Intel Pentium M 2.13 GHz 2GB
RAM WinXP
Prof\n",dim_V,evalb(0=simplify(meet(X,Y)-&v(X,Y))),time()-bench
);
od:
```

```
In dimension 1 the equation `meet(X,Y)=&v(X,Y)` is true and took 0.125000 secon
ds to verify on Intel Pentium M 2.13 GHz 2GB RAM WinXP Prof
In dimension 2 the equation `meet(X,Y)=&v(X,Y)` is true and took 0.329000 secon
ds to verify on Intel Pentium M 2.13 GHz 2GB RAM WinXP Prof
In dimension 3 the equation `meet(X,Y)=&v(X,Y)` is true and took 1.078000 secon
ds to verify on Intel Pentium M 2.13 GHz 2GB RAM WinXP Prof
In dimension 4 the equation `meet(X,Y)=&v(X,Y)` is true and took 5.703000 secon
ds to verify on Intel Pentium M 2.13 GHz 2GB RAM WinXP Prof
In dimension 5 the equation `meet(X,Y)=&v(X,Y)` is true and took 33.252000 seco
nds to verify on Intel Pentium M 2.13 GHz 2GB RAM WinXP Prof
```

```
>
```

The following **example** will show, that the meet and the join are **exterior** products on their own right and cannot be distinguished. This makes it unnecessary to use the \/ (vee) sign for the ordinary wedge product as Rota promoted to stress the analogy with set theoretic operators. We will see that the join of points is the meet of (hyper) planes and the meet of points is the join of (hyper) planes. To demonstrate this, we compute the meet for a Grassmann basis. We check associativity, unit, and show that this product is an exterior product on its own right on reciprocal (sometimes called wrongly dual) vectors (i.e. hyperplanes). The reciprocal meet is then defined to be the meet w.r.t. hyper-planes. Then it is shown that this reciprocal meet is indeed the wedge (join of points) with which we started. [To give a crude reciprocal meet we use Grassmann's Ergänzung, but a combinatorial evaluation is also possible but proved to be too long for this help page.]

We present our demonstration in dimension 3. Define the n-1 (i.e. 2-) vectors A(i). These multi-vectors are the images of a covector basis under dualization, see [4,3] and should be called reciprocal vectors. Their definition involves a symmetric correlation. The 'meet' or `&v` (vee) product of vectors acts as an exterior (or wedge) product on these reciprocal vectors. This is an immediate consequence of categorical duality and is related to the Plücker coordinatization of hyper-planes.

While we show here explicitly how to define a a Meet and Join for hyperplanes, there is an generic Grassmann co-product &gco_pl in the package which could be used with some benefits for the performance, but would probably obscure out aim here.

```
> dim_V:=3:
  # A(i),A(ij) etc are new basis elements
```

```
#
# ==> define the hyperplane basis A(i), A(ij) etc
A:=proc(x)
  local T;
  T:=table([123=-Id,
            31=-e2,23=-e1,12=-e3,
            13= e2,32= e1,21= e3,
            3=e1we2,2=-e1we3,1=e2we3,
            0=e1we2we3]);
  RETURN(T[x]);
end:
#
# w2A is a translation procedure which turns the output
# into the new A basis of reciprocal vectos (plane vectors)
#
# ==> w2A  (wedge basis to hyperplane basis A(i)
w2A:=proc(x)
     local bas,y;
     bas:={Id=-'A(123)',
          e1=-'A(23)', e2= 'A(13)',e3= -'A(12)',
          e1we2='A(3)',e1we3=-'A(2)',e2we3='A(1)',
          e1we2we3='A(0)'};
     RETURN(subs(bas,Clifford:-reorder(x)));
end:
#
# &V (uppercase) is a wrapper function to make the usage of
the
#     A(i) basis more comfortable
#
# ==> &V can act on the hyperplane basis A(i) seen as wedge
multivectors
#     and yields A(jk) hyperplane 2-vectors
#     [same goes for the Meet (formerly meet)]
`&V`:=proc(x,y)
    w2A(&v(eval(x),eval(y)));
end:
`Meet`:=proc(x,y)
    w2A(meet(eval(x),eval(y)));
end:
```

After these preliminary definitions we can directly show the meet to be the '***wedge product of hyperplanes***'. First of all we check some elementary properties of the meet acting on hyperplanes.

```
> A(0),w2A(A(0)); # The 'scalar' w.r.t. the &v product
  A(1),w2A(A(1));
```

```
A(2),w2A(A(2));
A(3),w2A(A(3)); # (reciprocal) vectors
```

$$e1we2we3, \mathrm{A}(0)$$
$$e2we3, \mathrm{A}(1)$$
$$-e1we3, \mathrm{A}(2)$$
$$e1we2, \mathrm{A}(3)$$

```
> &V(A(0),A(1)),&V(A(1),A(0));        #  shows A0 to be the
identity
Meet(A(0),A(1)),Meet(A(1),A(0));    #  synonym but internally
computed differently
```

$$\mathrm{A}(1), \mathrm{A}(1)$$
$$\mathrm{A}(1), \mathrm{A}(1)$$

Now we produce reciprocal bi-vectors (bi-hyperplanes to be precise) A(ij) and the volume element A(123)

```
> &V(A(1),A(2)),&V(A(2),A(3)),&V(A(3),A(1));  # BI-HYPERPLANES
&V(A(1),&V(A(2),A(3))),                       # VOLUME ELEMENT
EVALUATES TO -ID
&v(A(1),eval(&V(A(2),A(3))));                 # eval is needed
here to apply A(23)
```

$$\mathrm{A}(12), \mathrm{A}(23), -\mathrm{A}(13)$$
$$\mathrm{A}(123), -Id$$

There are no higher multi-hyperplanes (reciprocal multi-vectors) and the following expressions evaluate to zero:

```
> &V(A(1),A(123)),&V(A(12),A(23));
```

$$0, 0$$

The bracket for co-vectors can be defined using the fact that -Id is the volume in the space of hyperplanes as the projection onto -Id. Hence we can define the reciprocal meet RMeet of reciprocal vectors. This is also a demonstration how to extend the features of the CLIFFORD/Bigebra packages:

```
> B:=linalg[diag](1$dim_V):     ###  internally used for
Grassmann Erg"anzung
`&RMeet`:=proc(x,y)                        ###  function
co-meet
    local yy,res,lst,var_i,v1,v2;
    option `Copyright (c) Ablamowicz, Fauser 2000/02. All
```

```
rights reserved.`;
    ## crude version of the Grassmann co-product on the
'multivector plane space' \/A
    yy:=&t(e1we2we3,&gco(eval(cmul(e3we2we1,y))),e1we2we3);
    yy:=&map(tcollect(&map(switch(yy,3),3,cmul)),1,cmul);
##
    if type(yy,tensorbasmonom) or type(yy,tensormonom)then
      lst:=[yy];
    else
      lst:=[op(yy)];
    fi;
    res:=0;
    for var_i in lst do
      v1,v2:=peek(var_i,1);
      res := res - scalarpart(&v(eval(x),v1))*drop_t(op(v2));
    od;
    res;
end:
```

To exemplify out claim, let us define the two mutually reciprocal basis sets of points, joined points (i.e., lines) and point space volume and the hyperplanes bi-hyperplanes (i.e., lines) and the volume of the hyperplane multi-vector space -Id.

```
> bas:=cbasis(3);
bas_A:=[A(0),A(1),A(2),A(3),A(12),A(13),A(23),A(123)];
```

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

$$bas\_A := [e1we2we3, e2we3, -e1we3, e1we2, -e3, e2, -e1, -Id]$$

For easy comparison, we compute the multiplication table of the RMeet product. This multiplication table is a tensor of rank three. To be able to display this tensor as rank two array, we put the resulting multivectors (in Grassmann basis) into the array. The numerical matrices m_[ij]^k are then obtained by setting one basis element to 1 and all other to zero (i.e. by acting with the dual multivectors on this scheme.)

```
> Mul_tab_RMeet:=linalg[matrix](2^dim_V,2^dim_V,(i,j)-> 0):
for i from 1 to 2^dim_V do
for j from 1 to 2^dim_V do
  Mul_tab_RMeet[i,j]:=reorder(&RMeet(bas[i],bas[j]));
od:od:
evalm(Mul_tab_RMeet);
```

$$\begin{bmatrix} -Id, & -e1, & -e2, & -e3, & -e1we2, & -e1we3, & -e2we3, & -e1we2we3 \\ -e1, & 0, & -e1we2, & -e1we3, & 0, & 0, & -e1we2we3, & 0 \\ -e2, & e1we2, & 0, & -e2we3, & 0, & e1we2we3, & 0, & 0 \\ -e3, & e1we3, & e2we3, & 0, & -e1we2we3, & 0, & 0, & 0 \\ -e1we2, & 0, & 0, & -e1we2we3, & 0, & 0, & 0, & 0 \\ -e1we3, & 0, & e1we2we3, & 0, & 0, & 0, & 0, & 0 \\ -e2we3, & -e1we2we3, & 0, & 0, & 0, & 0, & 0, & 0 \\ -e1we2we3, & 0, & 0, & 0, & 0, & 0, & 0, & 0 \end{bmatrix}$$

Our final **goal** is to show, that the above defined multiplication for RMeet (the **meet of hyperplanes) is equivalent to the wedge product of points**. We compute therefore the multiplication table for the wedge also:

```
> Mul_tab_wedge:=linalg[matrix](2^dim_V,2^dim_V,(i,j)-> 0):
  for i from 1 to 2^dim_V do
  for j from 1 to 2^dim_V do
    Mul_tab_wedge[i,j]:=&w(bas[i],bas[j]);
  od:od:
  evalm(Mul_tab_wedge);
```

$$\begin{bmatrix} Id, & e1, & e2, & e3, & e1we2, & e1we3, & e2we3, & e1we2we3 \\ e1, & 0, & e1we2, & e1we3, & 0, & 0, & e1we2we3, & 0 \\ e2, & -e1we2, & 0, & e2we3, & 0, & -e1we2we3, & 0, & 0 \\ e3, & -e1we3, & -e2we3, & 0, & e1we2we3, & 0, & 0, & 0 \\ e1we2, & 0, & 0, & e1we2we3, & 0, & 0, & 0, & 0 \\ e1we3, & 0, & -e1we2we3, & 0, & 0, & 0, & 0, & 0 \\ e2we3, & e1we2we3, & 0, & 0, & 0, & 0, & 0, & 0 \\ e1we2we3, & 0, & 0, & 0, & 0, & 0, & 0, & 0 \end{bmatrix}$$

The final check is to add both matrices which gives zero. This shows that up to a sign (which is irrelevant in projective plane geometry) the products are the same. Or, as operator equation:

RMeet(x,y) = -wedge(x,y)

The sign belongs to the fact that in three dimensions we find that the volume element squares to negative identity, which means that we would reach the original wedge after a second turn in our argumentation. However, we resist to demonstrate this explicitly here.

```
> evalm(Mul_tab_RMeet+Mul_tab_wedge);
```

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Finally we will provide some ***benchmarks*** which shall show how efficient the two alternate definitions of the meet are. One, as adopted recently by Hestenes and followers, is based on the Grassmann's Ergänzung and the other is based on Hopf algebra methods as employed in Bigebra and given by Lotze and Rota.

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM\n",time()-bench);
```

```
Worksheet took 37.831000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
```

```
> restart:bench:=time():with(Clifford):with(Bigebra):
> dim_V:=3:B:=linalg[diag](1$dim_V):
```

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

We use dimension 3 and a orthogonal, normalized Euclidean bilinear form (the polar form of a quadratic form).

As a **Benchmark** we compute 100 times a certain meet (this is not a good idea, since some functions may remember its results, e.g. the wedge product from the CLIFFORD package, but it gives nevertheless a feeling what is going on).

The Hopf algebraic case needs:

```
> s:=time():
  for i from 1 to 100 do
    &v(e1we2,e2we3);
  od:
  printf("This took us %f seconds",time()-s);
  &v(e1we2,e2we3);
```

```
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
This took us 1.406000 seconds
```

$$-e2$$

```
> restart:bench:=time():with(Clifford):with(Bigebra):   # reload
  everything to be fair
  dim_V:=3:B:=linalg[diag](1$dim_V):
  s:=time():
  for i from 1 to 100 do

  cmul(e3we2we1,wedge(cmul(e1we2we3,e1we2),cmul(e1we2we3,e2we3)))
```

```
;
od:
printf("This took us %f seconds",time()-s);
cmul(e3we2we1,wedge(cmul(e1we2we3,e1we2),cmul(e1we2we3,e2we3)))
;
```

```
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
This took us 0.625000 seconds
```

$$e2$$

Since we compute the Clifford product using a very fast Hopf algebraic function cmulRS, this works out faster. However, we could speed up &v by directly employing the Hopf algebraic routines and avoiding wrapper functions as `peek`. Furthermore we have not computed the inverse of the Ergänzung but introduced simply e3we2we1 which is (e1we2we3)^(-1) in our case.

Now let us go for a non-orthogonal but still symmetric bilinear form (a polar form of a quadratic form or a symmetric correlation) and check what happens there:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
  dim_V:=3:B:=linalg[matrix](dim_V,dim_V,(i,j)->if i<=j then
  g[i,j] else g[j,i] fi);
```

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

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

Compute the Bigebra meet &v (vee product):

```
> s:=time():
      clicollect(&v(e1we2,e2we3));
  printf("This took us %f seconds",time()-s);
```

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

$$-e2$$

```
This took us 0.015000 seconds
```

We reload once more the package to be fair and compute the meet using the Ergänzung:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
  dim_V:=3:B:=linalg[matrix](dim_V,dim_V,(i,j)->if i<=j then
  g[i,j] else g[j,i] fi);
  s:=time():
  clicollect(cmul(e3we2we1,wedge(cmul(e1we2we3,e1we2),cmul(e1we2w
  e3,e2we3))));
  printf("This took us %f seconds",time()-s);
```

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

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

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

$$\begin{aligned}(&g_{1,1}^2 g_{2,3}^4 + g_{1,2}^4 g_{3,3}^2 + g_{1,3}^4 g_{2,2}^2 + g_{2,2}^2 g_{1,1}^2 g_{3,3}^2 + 4 g_{1,3}^2 g_{2,3}^2 g_{1,2}^2 \\ &- 2 g_{2,2}^2 g_{1,1} g_{3,3} g_{1,3}^2 - 2 g_{2,2} g_{1,1}^2 g_{2,3}^2 g_{3,3} + 2 g_{2,2} g_{1,1} g_{2,3}^2 g_{1,3}^2 - 2 g_{2,2} g_{1,2}^2 g_{3,3}^2 g_{1,1} \\ &+ 2 g_{2,2} g_{1,2}^2 g_{3,3} g_{1,3}^2 - 4 g_{2,2} g_{1,3}^3 g_{2,3} g_{1,2} + 4 g_{2,2} g_{1,3} g_{2,3} g_{1,2} g_{1,1} g_{3,3} \\ &+ 2 g_{1,1} g_{2,3}^2 g_{3,3} g_{1,2}^2 - 4 g_{1,2}^3 g_{3,3} g_{2,3} g_{1,3} - 4 g_{1,2} g_{1,1} g_{2,3}^3 g_{1,3}) \, e2\end{aligned}$$

```
This took us 0.312000 seconds
```

This shows already a difference (approx. a factor 6, which varies from computation to computation due to garbage collection overhead) which would further *increase* if the dimension were higher. Thus, the **computational efficiency** of the meet has been demonstrated.

Moreover, we can go beyond the possibilities of the Ergänzungs method since we can compute the meet in the presence of a non-symmetric bilinear form (which cannot be derived from a quadratic form by polarization) using Hopf algebra methods. Our meet works **independently of the assigned bilinear form** while the Eränzungs method needs an orthogonal non-degenerate bilinear form (which is the polar form of the symmetric correlation, i.e. a quadratic form).

Let us use an arbitrary bilinear form in 3 dimensions:

```
> B:=linalg[matrix](dim_V,dim_V,(i,j)->b[i,j]);
```

$$B := \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$$

The Hopf algebraic meet remains to be

```
> &v(e1we2,e2we3);
```

$$-e2$$

while the 'meet' computed using the Ergänzung does not even yield a homogeneous multi-vector, but a Clifford polynomial:

```
> clicollect(simplify(cmul(e3we2we1,wedge(cmul(e1we2we3,e1we2),cm
  ul(e1we2we3,e2we3)))));
```

$$\begin{aligned}(&-2 b_{2,2} b_{3,1} b_{2,3} b_{1,2} b_{1,3} + b_{2,2} b_{3,1}^2 b_{2,3} b_{1,2} + b_{1,2} b_{2,2} b_{1,1} b_{3,3} b_{2,3} \\ &+ b_{2,2} b_{2,1} b_{1,2} b_{3,3} b_{1,3} - b_{3,2} b_{2,2} b_{1,1} b_{3,3} b_{2,1} - b_{3,2} b_{3,1} b_{2,3} b_{1,2} b_{2,1} + b_{1,1} b_{3,2}^2 b_{2,3} b_{2,1} \\ &- b_{3,1}^2 b_{2,2}^2 b_{1,3} + b_{3,2} b_{2,1}^2 b_{1,2} b_{3,3} + b_{2,2} b_{1,1} b_{3,2} b_{2,3} b_{1,3} - b_{2,2} b_{1,1} b_{3,2} b_{2,3} b_{3,1}\end{aligned}$$

$$- b_{1,2}\, b_{1,1}\, b_{3,2}\, b_{2,3}^{2} - b_{2,1}\, b_{1,2}^{2}\, b_{3,3}\, b_{2,3} + b_{3,1}\, b_{2,3}^{2}\, b_{1,2}^{2} + b_{2,2}^{2}\, b_{1,1}\, b_{3,3}\, b_{3,1}$$
$$+ b_{3,1}\, b_{2,2}^{2}\, b_{1,3}^{2} - b_{2,2}\, b_{3,2}\, b_{2,1}\, b_{1,3}^{2} - b_{2,2}^{2}\, b_{1,1}\, b_{3,3}\, b_{1,3} + b_{1,2}\, b_{3,2}\, b_{2,1}\, b_{1,3}\, b_{2,3}$$
$$- b_{2,2}\, b_{2,1}\, b_{1,2}\, b_{3,3}\, b_{3,1} + 2\, b_{2,2}\, b_{3,2}\, b_{2,1}\, b_{1,3}\, b_{3,1} - b_{3,2}^{2}\, b_{2,1}^{2}\, b_{1,3})\, e1we2we3 + ($$
$$b_{1,1}\, b_{3,2}^{2}\, b_{2,3}\, b_{3,1}\, b_{2,2} - b_{1,1}\, b_{3,2}^{3}\, b_{2,3}\, b_{2,1} + b_{2,2}\, b_{1,1}\, b_{3,3}\, b_{3,2}^{2}\, b_{2,1} + b_{3,2}\, b_{3,1}^{2}\, b_{2,2}^{2}\, b_{1,3}$$
$$- b_{3,2}\, b_{3,1}^{2}\, b_{2,3}\, b_{1,2}\, b_{2,2} + b_{3,2}\, b_{2,1}\, b_{1,2}\, b_{3,3}\, b_{3,1}\, b_{2,2} - b_{3,2}\, b_{2,2}^{2}\, b_{1,1}\, b_{3,3}\, b_{3,1}$$
$$- b_{2,2}^{2}\, b_{1,1}\, b_{3,3}^{2}\, b_{2,1} + b_{2,2}^{2}\, b_{1,1}\, b_{3,3}\, b_{2,3}\, b_{3,1} + b_{3,1}\, b_{2,2}^{2}\, b_{1,3}\, b_{2,1}\, b_{3,3} - b_{3,1}^{2}\, b_{2,2}^{2}\, b_{1,3}\, b_{2,3}$$
$$- 2\, b_{2,2}\, b_{3,1}\, b_{2,3}\, b_{1,2}\, b_{2,1}\, b_{3,3} - b_{3,1}\, b_{2,3}^{2}\, b_{1,2}^{2}\, b_{3,2} + b_{3,1}\, b_{2,3}\, b_{1,2}^{2}\, b_{3,3}\, b_{2,2}$$
$$+ b_{2,1}\, b_{1,2}^{2}\, b_{3,3}\, b_{3,2}\, b_{2,3} - b_{2,1}\, b_{1,2}^{2}\, b_{3,3}^{2}\, b_{2,2} + b_{3,2}^{3}\, b_{2,1}^{2}\, b_{1,3} + b_{2,2}\, b_{2,1}^{2}\, b_{1,2}\, b_{3,3}^{2}$$
$$- b_{2,1}^{2}\, b_{1,2}\, b_{3,3}\, b_{3,2}^{2} - b_{2,2}\, b_{3,2}\, b_{2,1}^{2}\, b_{1,3}\, b_{3,3} + b_{2,2}\, b_{3,2}\, b_{2,1}\, b_{1,3}\, b_{2,3}\, b_{3,1}$$
$$+ b_{2,2}\, b_{1,1}\, b_{3,2}\, b_{2,3}\, b_{2,1}\, b_{3,3} - b_{2,2}\, b_{1,1}\, b_{3,2}\, b_{2,3}^{2}\, b_{3,1} + b_{2,2}\, b_{3,1}^{2}\, b_{2,3}^{2}\, b_{1,2}$$
$$+ b_{1,2}\, b_{3,1}\, b_{2,2}\, b_{1,3}\, b_{3,2}\, b_{2,3} - b_{1,2}\, b_{3,1}\, b_{2,2}^{2}\, b_{1,3}\, b_{3,3} - b_{1,2}\, b_{3,2}^{2}\, b_{2,1}\, b_{1,3}\, b_{2,3}$$
$$+ b_{1,2}\, b_{3,2}\, b_{2,1}\, b_{1,3}\, b_{3,3}\, b_{2,2} - 2\, b_{1,2}\, b_{1,1}\, b_{3,2}\, b_{2,3}\, b_{3,3}\, b_{2,2} + b_{1,2}\, b_{1,1}\, b_{3,2}^{2}\, b_{2,3}^{2}$$
$$+ b_{1,2}\, b_{2,2}^{2}\, b_{1,1}\, b_{3,3}^{2} + b_{3,1}\, b_{2,3}\, b_{1,2}\, b_{3,2}^{2}\, b_{2,1} - 2\, b_{3,2}^{2}\, b_{2,1}\, b_{1,3}\, b_{3,1}\, b_{2,2})\, e1 + ($$
$$-b_{2,2}\, b_{1,1}^{2}\, b_{3,3}\, b_{3,2}^{2} + b_{3,1}\, b_{2,2}\, b_{1,3}\, b_{3,2}^{2}\, b_{1,1} + b_{3,2}\, b_{3,1}^{2}\, b_{2,3}\, b_{1,2}^{2} - b_{3,2}\, b_{2,1}\, b_{1,2}^{2}\, b_{3,3}\, b_{3,1}$$
$$+ b_{3,2}\, b_{2,2}\, b_{1,1}\, b_{3,3}\, b_{3,1}\, b_{1,2} - 2\, b_{2,2}^{2}\, b_{1,1}\, b_{3,3}\, b_{1,3}\, b_{3,1} + b_{2,2}\, b_{3,1}\, b_{2,3}\, b_{1,2}\, b_{1,1}\, b_{3,3}$$
$$- b_{2,2}\, b_{3,1}^{2}\, b_{2,3}\, b_{1,2}\, b_{1,3} - b_{3,2}\, b_{3,1}^{2}\, b_{2,2}\, b_{1,3}\, b_{1,2} + b_{3,1}\, b_{2,2}\, b_{1,3}\, b_{3,3}\, b_{1,2}^{2}$$
$$- 2\, b_{3,2}\, b_{2,1}\, b_{1,3}\, b_{3,3}\, b_{1,2}^{2} + b_{1,1}\, b_{3,2}\, b_{2,3}\, b_{3,3}\, b_{1,2}^{2} - b_{2,2}\, b_{1,1}\, b_{3,3}^{2}\, b_{1,2}^{2} + b_{2,2}^{2}\, b_{1,1}^{2}\, b_{3,3}^{2}$$
$$+ b_{3,1}^{2}\, b_{2,2}^{2}\, b_{1,3}^{2} + b_{2,1}\, b_{1,2}^{3}\, b_{3,3}^{2} + b_{1,1}^{2}\, b_{3,2}^{3}\, b_{2,3} + b_{2,2}\, b_{3,2}\, b_{2,1}\, b_{1,3}\, b_{1,1}\, b_{3,3}$$
$$+ b_{2,1}\, b_{1,2}\, b_{3,3}\, b_{3,2}^{2}\, b_{1,1} - b_{2,2}\, b_{3,2}\, b_{2,1}\, b_{1,3}^{2}\, b_{3,1} - b_{2,2}\, b_{2,1}\, b_{1,2}\, b_{3,3}^{2}\, b_{1,1}$$
$$+ b_{2,2}\, b_{2,1}\, b_{1,2}\, b_{3,3}\, b_{1,3}\, b_{3,1} - b_{2,2}\, b_{1,1}^{2}\, b_{3,2}\, b_{2,3}\, b_{3,3} + b_{2,2}\, b_{1,1}\, b_{3,2}\, b_{2,3}\, b_{1,3}\, b_{3,1}$$
$$+ b_{3,1}\, b_{2,3}\, b_{1,2}^{2}\, b_{3,2}\, b_{1,3} - b_{3,1}\, b_{2,3}\, b_{1,2}^{3}\, b_{3,3} - b_{1,2}\, b_{3,1}\, b_{2,2}\, b_{1,3}^{2}\, b_{3,2}$$
$$- b_{1,2}\, b_{1,1}\, b_{3,2}^{2}\, b_{2,3}\, b_{1,3} + b_{1,2}\, b_{2,2}\, b_{1,1}\, b_{3,3}\, b_{3,2}\, b_{1,3} + b_{1,2}\, b_{3,2}^{2}\, b_{2,1}\, b_{1,3}^{2}$$
$$- 2\, b_{3,1}\, b_{2,3}\, b_{1,2}\, b_{3,2}^{2}\, b_{1,1} + b_{3,2}^{2}\, b_{2,1}\, b_{1,3}\, b_{3,1}\, b_{1,2} - b_{3,2}^{3}\, b_{2,1}\, b_{1,3}\, b_{1,1})\, e2 - ($$
$$-b_{1,1}\, b_{3,2}^{2}\, b_{2,3}\, b_{2,1}\, b_{1,2} + b_{1,1}^{2}\, b_{3,2}^{2}\, b_{2,3}\, b_{2,2} - b_{3,2}\, b_{2,1}^{2}\, b_{1,2}^{2}\, b_{3,3} - b_{3,2}\, b_{2,2}^{2}\, b_{1,1}^{2}\, b_{3,3}$$
$$+ b_{3,2}\, b_{3,1}\, b_{2,3}\, b_{1,2}^{2}\, b_{2,1} - b_{3,2}\, b_{3,1}\, b_{2,3}\, b_{1,2}\, b_{2,2}\, b_{1,1} + 2\, b_{3,2}\, b_{2,1}\, b_{1,2}\, b_{3,3}\, b_{2,2}\, b_{1,1}$$
$$- b_{3,2}\, b_{3,1}\, b_{2,2}\, b_{1,3}\, b_{2,1}\, b_{1,2} + b_{2,2}^{2}\, b_{1,1}^{2}\, b_{3,3}\, b_{2,3} - b_{2,2}^{2}\, b_{1,1}\, b_{3,3}\, b_{1,3}\, b_{2,1}$$

$$- b_{3,1} b_{2,2}^{2} b_{1,3} b_{1,1} b_{2,3} + b_{3,1} b_{2,2}^{2} b_{1,3}^{2} b_{2,1} + b_{2,2} b_{3,1} b_{2,3}^{2} b_{1,2} b_{1,1}$$
$$+ 2 b_{2,2} b_{3,2} b_{2,1} b_{1,3} b_{1,1} b_{2,3} - b_{2,2} b_{3,2} b_{2,1}^{2} b_{1,3}^{2} + b_{3,2} b_{3,1} b_{2,2}^{2} b_{1,3} b_{1,1}$$
$$- b_{3,2} b_{2,1} b_{1,3} b_{2,3} b_{1,2}^{2} - b_{2,1} b_{1,2}^{2} b_{3,3} b_{2,2} b_{1,3} + b_{2,1} b_{1,2}^{3} b_{3,3} b_{2,3} + b_{1,1} b_{3,2} b_{2,3}^{2} b_{1,2}^{2}$$
$$- b_{2,2} b_{1,1} b_{3,3} b_{2,3} b_{1,2}^{2} - b_{3,1} b_{2,3}^{2} b_{1,2}^{3} - b_{2,2} b_{1,1}^{2} b_{3,2} b_{2,3}^{2} - b_{2,2} b_{2,1} b_{1,2} b_{3,3} b_{1,1} b_{2,3}$$
$$+ b_{2,2} b_{2,1}^{2} b_{1,2} b_{3,3} b_{1,3} - b_{2,2} b_{3,1} b_{2,3} b_{1,2} b_{1,3} b_{2,1} + 2 b_{3,1} b_{2,3} b_{1,2}^{2} b_{2,2} b_{1,3}$$
$$+ b_{1,2} b_{3,2} b_{2,1} b_{1,3}^{2} b_{2,2} - b_{1,2} b_{1,1} b_{3,2} b_{2,3} b_{2,2} b_{1,3} + b_{1,2} b_{2,2}^{2} b_{1,1} b_{3,3} b_{1,3}$$
$$- b_{1,2} b_{3,1} b_{2,2}^{2} b_{1,3}^{2} + b_{3,2}^{2} b_{2,1}^{2} b_{1,3} b_{1,2} - b_{3,2}^{2} b_{2,1} b_{1,3} b_{2,2} b_{1,1}) e3$$

```
>
>
```

See Also: Bigebra:-`&map`, Bigebra:-peek, Bigebra:-poke, Bigebra:-switch

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-op2mat - derive a matrix of an operator acting in V^ which is possibly unfaithful and not irreducible

Bigebra:-op2mat2 - derive a matrix of an operator acting in V^ &t V^ which is possibly unfaithful and not irreducible

Calling Sequence:

m1:= op2mat(fkt)

m1:= op2mat2(fkt)

Parameters:

- fkt : an linear operator acting on V^ (or V^ &t V^)

Output:

- m1 : a 2^dim_V x 2^dim_V matrix (a 4^dim_V x 4^dim_V matrix)

Global parameters:

- dim_V

Description:

- The op2mat command is useful to derive matrix forms of linear operators. It can be used together with linop (linop2) to move from an operator description to matrix form and back. The derived matrices are regarded as elements of End V^ (End V^ &t V^) . The basis used is assumed to be the standard Grassmann basis of Clifford or the following sum^{2^dim_V}_{i,j=1} &t(b[i],b[j]), where the b[i] are Grassmann bases of V^.
- The main purpose of these two functions is to get a matrix form of heavily used operators needed in long computation time, e.g. the antipode of a Clifford Hopf gebra or the switch (may be derived from op2mat2). The functionality of linalg is then available to those matrices.

Examples:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
  dim_V:=2:
```

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

Let the name of the operator be R, its matrix elements be R[i,j], we have:

```
> linop(Id,R);
  linop(e1,R);
  linop(e1we2,R);
```

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

$$R_{1,1}\, Id + R_{2,1}\, e1 + R_{3,1}\, e2 + R_{4,1}\, e1we2$$

$$R_{1,2}\, Id + R_{2,2}\, e1 + R_{3,2}\, e2 + R_{4,2}\, e1we2$$

$$R_{1,4}\, Id + R_{2,4}\, e1 + R_{3,4}\, e2 + R_{4,4}\, e1we2$$

Get the matrix form of linop(x,R). First we define an operator R from which we derive the matrix form, but we show in a second line that parameters may be passed to op2mat which allows us to use linop directly:

```
> R:=proc(x) linop(x,R) end:   # define a proper operator
  op2mat(R);                   #
  op2mat(linop,R);             #
```

$$\begin{bmatrix} R_{1,1} & R_{1,2} & R_{1,3} & R_{1,4} \\ R_{2,1} & R_{2,2} & R_{2,3} & R_{2,4} \\ R_{3,1} & R_{3,2} & R_{3,3} & R_{3,4} \\ R_{4,1} & R_{4,2} & R_{4,3} & R_{4,4} \end{bmatrix}$$

$$\begin{bmatrix} R_{1,1} & R_{1,2} & R_{1,3} & R_{1,4} \\ R_{2,1} & R_{2,2} & R_{2,3} & R_{2,4} \\ R_{3,1} & R_{3,2} & R_{3,3} & R_{3,4} \\ R_{4,1} & R_{4,2} & R_{4,3} & R_{4,4} \end{bmatrix}$$

Derive the matrix of the Grassmann antipode, in dim_V=2 we get a 4x4 matrix:

```
> bas:=cbasis(dim_V):
  matS:=op2mat(gantipode,1);
  map(linop,bas,matS);
  map(gantipode,bas,1);
```

$$matS := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[Id, -e1, -e2, e1we2]$$

$$[Id, -e1, -e2, e1we2]$$

We checked using linop that this is the same operator as defined abstractly. Hence our indexing is compatible.

A few further examples are:

```
> conv_unit:=proc(x) Id*scalarpart(x) end:
  `conv_unit  ---->   `,op2mat(conv_unit);
  gr_loop:=proc(x) drop_t(&map(&gco(x),1,wedge)) end:
  `Grassmann loop  ---->   `,op2mat(gr_loop);
  X:=add(x[i]*bas[i],i=1..2^dim_V):
  scalar_right_conv:=proc(x) wedge(x,X) end:
  `scalar right conv  ---->   `,op2mat(scalar_right_conv);
```

```
scalar_left_conv:=proc(x) wedge(X,x) end:
`scalar left conv  ---->   `,op2mat(scalar_left_conv);
scalar_coright_conv:=proc(x)
drop_t(contract(&t(&gco(x),X),2,EV)) end:
`scalar coright conv  ---->   `,op2mat(scalar_coright_conv);
scalar_coleft_conv:=proc(x)
drop_t(contract(&t(X,&gco(x)),1,EV)) end:
`scalar coleft conv  ---->   `,op2mat(scalar_coleft_conv);
```

$$\textit{conv\_unit} \text{ ----> } , \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\textit{Grassmann loop} \text{ ----> } , \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

$$\textit{scalar right conv} \text{ ----> } , \begin{bmatrix} x_1 & 0 & 0 & 0 \\ x_2 & x_1 & 0 & 0 \\ x_3 & 0 & x_1 & 0 \\ x_4 & x_3 & -x_2 & x_1 \end{bmatrix}$$

$$\textit{scalar left conv} \text{ ----> } , \begin{bmatrix} x_1 & 0 & 0 & 0 \\ x_2 & x_1 & 0 & 0 \\ x_3 & 0 & x_1 & 0 \\ x_4 & -x_3 & x_2 & x_1 \end{bmatrix}$$

$$\textit{scalar coright conv} \text{ ----> } , \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ 0 & x_1 & 0 & x_3 \\ 0 & 0 & x_1 & -x_2 \\ 0 & 0 & 0 & x_1 \end{bmatrix}$$

$$\textit{scalar coleft conv} \text{ ----> } , \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ 0 & x_1 & 0 & -x_3 \\ 0 & 0 & x_1 & x_2 \\ 0 & 0 & 0 & x_1 \end{bmatrix}$$

op2mat2 is the counterpart for operators acting on V^ &t V^, in our case the vector space dimension is 2 (dim_V=2) and dim V^ = 2^2 = 4 so the output is a 16 times 16 matrix:

We check the function by some examples and will see that it is compatible in indexing with

linop2:

```
> GSW:=op2mat2(gswitch,1);
  `V^2_bas`:=[seq(seq(&t(bas[i],bas[j]),i=1..2^dim_V),j=1..2^dim_
  V)];
  `V^2_GSW_bas`:=convert(evalm(GSW &* `V^2_bas`),list);   #
  compose using linalg
  `V^2_gs_bas1`:=map(gswitch,`V^2_bas`,1);                # act
  using the operator
  `V^2_gs_bas2`:=map(linop2,`V^2_bas`,GSW);               # act
  using linop2 and the matrix GSW
  printf("Are the two lists V^2_GSW_bas and V^2_gs_bas1 equal ?
  %a \n",

  op({seq(is(`V^2_GSW_bas`[i]=`V^2_gs_bas1`[i]),i=1..4^dim_V)}));

  printf("Are the two lists V^2_GSW_bas and V^2_gs_bas2 equal ?
  %a \n",

  op({seq(is(`V^2_GSW_bas`[i]=`V^2_gs_bas2`[i]),i=1..4^dim_V)}));
```

$$GSW := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$V^2\_bas := [Id \,\&t\, Id, e1 \,\&t\, Id, e2 \,\&t\, Id, e1we2 \,\&t\, Id, Id \,\&t\, e1, e1 \,\&t\, e1, e2 \,\&t\, e1, e1we2 \,\&t\, e1, Id \,\&t\, e2, e1 \,\&t\, e2, e2 \,\&t\, e2, e1we2 \,\&t\, e2, Id \,\&t\, e1we2, e1 \,\&t\, e1we2, e2 \,\&t\, e1we2, e1we2 \,\&t\, e1we2]$

$V^2\_GSW\_bas := [Id \,\&t\, Id, Id \,\&t\, e1, Id \,\&t\, e2, Id \,\&t\, e1we2, e1 \,\&t\, Id, -(e1 \,\&t\, e1), -(e1 \,\&t\, e2), e1 \,\&t\, e1we2, e2 \,\&t\, Id, -(e2 \,\&t\, e1), -(e2 \,\&t\, e2), e2 \,\&t\, e1we2, e1we2 \,\&t\, Id, e1we2 \,\&t\, e1, e1we2 \,\&t\, e2, e1we2 \,\&t\, e1we2]$

$$V^2\_gs\_bas1 := [Id \,\&t\, Id, Id \,\&t\, e1, Id \,\&t\, e2, Id \,\&t\, e1we2, e1 \,\&t\, Id, -(e1 \,\&t\, e1), -(e1 \,\&t\, e2), e1 \,\&t\, e1we2, e2 \,\&t\, Id, -(e2 \,\&t\, e1), -(e2 \,\&t\, e2), e2 \,\&t\, e1we2, e1we2 \,\&t\, Id, e1we2 \,\&t\, e1, e1we2 \,\&t\, e2, e1we2 \,\&t\, e1we2]$$

$$V^2\_gs\_bas2 := [Id \,\&t\, Id, Id \,\&t\, e1, Id \,\&t\, e2, Id \,\&t\, e1we2, e1 \,\&t\, Id, -(e1 \,\&t\, e1), -(e1 \,\&t\, e2), e1 \,\&t\, e1we2, e2 \,\&t\, Id, -(e2 \,\&t\, e1), -(e2 \,\&t\, e2), e2 \,\&t\, e1we2, e1we2 \,\&t\, Id, e1we2 \,\&t\, e1, e1we2 \,\&t\, e2, e1we2 \,\&t\, e1we2]$$

```
Are the two lists V^2_GSW_bas and V^2_gs_bas1 equal ?  true
Are the two lists V^2_GSW_bas and V^2_gs_bas2 equal ?  true
```

Let us give a further example:

```
> BI:=linalg[matrix](2^dim_V,2^dim_V,(i,j)->C[i,j]):
  make_BI_Id():
  BC:=proc(x)
  tcollect(&t(scalarpart(drop_t(&map(x,1,cmul)))*&cco(Id))) end:
  seq(seq(BC(&t(bas[i],bas[j])),i=1..2^dim_V),j=1..2^dim_V);
```

$$(Id \,\&t\, Id) + C_{1,1}(e1 \,\&t\, e1) + C_{2,1}(e2 \,\&t\, e1) + C_{1,2}(e1 \,\&t\, e2) + C_{2,2}(e2 \,\&t\, e2) + (C_{2,1}C_{1,2} - C_{2,2}C_{1,1})(e1we2 \,\&t\, e1we2), \&t(0), \&t(0), \&t(0), \&t(0),$$
$$B_{1,1}C_{1,1}(e1 \,\&t\, e1) + B_{1,1}C_{2,1}(e2 \,\&t\, e1) + B_{1,1}C_{2,2}(e2 \,\&t\, e2) + B_{1,1}C_{1,2}(e1 \,\&t\, e2) + B_{1,1}(Id \,\&t\, Id) + B_{1,1}(C_{2,1}C_{1,2} - C_{2,2}C_{1,1})(e1we2 \,\&t\, e1we2), B_{2,1}C_{1,1}(e1 \,\&t\, e1) + B_{2,1}C_{2,1}(e2 \,\&t\, e1) + B_{2,1}C_{2,2}(e2 \,\&t\, e2) + B_{2,1}C_{1,2}(e1 \,\&t\, e2) + B_{2,1}(Id \,\&t\, Id) + B_{2,1}(C_{2,1}C_{1,2} - C_{2,2}C_{1,1})(e1we2 \,\&t\, e1we2), \&t(0), \&t(0), B_{1,2}C_{1,1}(e1 \,\&t\, e1) + B_{1,2}C_{2,1}(e2 \,\&t\, e1) + B_{1,2}C_{2,2}(e2 \,\&t\, e2) + B_{1,2}C_{1,2}(e1 \,\&t\, e2) + B_{1,2}(Id \,\&t\, Id) + B_{1,2}(C_{2,1}C_{1,2} - C_{2,2}C_{1,1})(e1we2 \,\&t\, e1we2), B_{2,2}C_{1,1}(e1 \,\&t\, e1) + B_{2,2}C_{2,1}(e2 \,\&t\, e1) + B_{2,2}C_{2,2}(e2 \,\&t\, e2) + B_{2,2}C_{1,2}(e1 \,\&t\, e2) + B_{2,2}(Id \,\&t\, Id) + B_{2,2}(C_{2,1}C_{1,2} - C_{2,2}C_{1,1})(e1we2 \,\&t\, e1we2), \&t(0), \&t(0), \&t(0), \&t(0),$$
$$(B_{2,1}B_{1,2} - B_{2,2}B_{1,1})C_{1,1}(e1 \,\&t\, e1) + (B_{2,1}B_{1,2} - B_{2,2}B_{1,1})C_{2,1}(e2 \,\&t\, e1) + (B_{2,1}B_{1,2} - B_{2,2}B_{1,1})C_{2,2}(e2 \,\&t\, e2) + (B_{2,1}B_{1,2} - B_{2,2}B_{1,1})C_{1,2}(e1 \,\&t\, e2) + (B_{2,1}B_{1,2} - B_{2,2}B_{1,1})(Id \,\&t\, Id) + (B_{2,1}B_{1,2} - B_{2,2}B_{1,1})(C_{2,1}C_{1,2} - C_{2,2}C_{1,1})(e1we2 \,\&t\, e1we2)$$

```
> op2mat2(BC);
```

$$[1, 0, 0, 0, 0, C_{1,1}, C_{2,1}, 0, 0, C_{1,2}, C_{2,2}, 0, 0, 0, 0, C_{2,1}C_{1,2} - C_{2,2}C_{1,1}]$$
$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$
$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$
$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$
$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$
$$[B_{1,1}, 0, 0, 0, 0, B_{1,1}C_{1,1}, B_{1,1}C_{2,1}, 0, 0, B_{1,1}C_{1,2}, B_{1,1}C_{2,2}, 0, 0, 0, 0, B_{1,1}(C_{2,1}C_{1,2} - C_{2,2}C_{1,1})]$$

$$[B_{2,1}, 0, 0, 0, 0, B_{2,1} C_{1,1}, B_{2,1} C_{2,1}, 0, 0, B_{2,1} C_{1,2}, B_{2,1} C_{2,2}, 0, 0, 0, 0, B_{2,1} (C_{2,1} C_{1,2} - C_{2,2} C_{1,1})]$$

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

$$[B_{1,2}, 0, 0, 0, 0, B_{1,2} C_{1,1}, B_{1,2} C_{2,1}, 0, 0, B_{1,2} C_{1,2}, B_{1,2} C_{2,2}, 0, 0, 0, 0, B_{1,2} (C_{2,1} C_{1,2} - C_{2,2} C_{1,1})]$$

$$[B_{2,2}, 0, 0, 0, 0, B_{2,2} C_{1,1}, B_{2,2} C_{2,1}, 0, 0, B_{2,2} C_{1,2}, B_{2,2} C_{2,2}, 0, 0, 0, 0, B_{2,2} (C_{2,1} C_{1,2} - C_{2,2} C_{1,1})]$$

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

$$[B_{2,1} B_{1,2} - B_{2,2} B_{1,1}, 0, 0, 0, 0, (B_{2,1} B_{1,2} - B_{2,2} B_{1,1}) C_{1,1}, (B_{2,1} B_{1,2} - B_{2,2} B_{1,1}) C_{2,1}, 0, 0, (B_{2,1} B_{1,2} - B_{2,2} B_{1,1}) C_{1,2}, (B_{2,1} B_{1,2} - B_{2,2} B_{1,1}) C_{2,2}, 0, 0, 0, 0, (B_{2,1} B_{1,2} - B_{2,2} B_{1,1}) (C_{2,1} C_{1,2} - C_{2,2} C_{1,1})]$$

As a last example we give switch and the time the antipode acting on the switch:

```
> aasw:=proc(x) switch(tcollect(gantipode(gantipode(x,1),2)),1)
  end:
  op2mat2(gswitch,1);
  op2mat2(aasw);
  convert(evalm(%-%%),set);  ## these operators are not
  identical!
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\{-2, 0, 2\}$$

```
> printf("The worksheet took %f seconds to compute on Intel
  Pentium M 2.13 GHz 2GB RAM WinXP Prof",time()-bench);
```

```
The worksheet took 5.609000 seconds to compute on Intel Pentium M 2.13 GHz 2GB
RAM WinXP Prof
```

```
>
```

NOTE: If the entries of the tensor polynom are out of the bound of the matrix, this function may go into an **endless loop**! E.g. mapop2(&t(e5,e6),1,gs); in our example, since dim_V was 2.

See Also: Bigebra:-mapop, Bigebra:-mapop2, Bigebra:-EV, Bigebra:-pairing, Bigebra:-linop, Bigebra:-list2mat2, Bigebra:-help

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-op2mat - derive a matrix of an operator acting in V^ which is possibly unfaithful and not irreducible

Bigebra:-op2mat2 - derive a matrix of an operator acting in V^ &t V^ which is possibly unfaithful and not irreducible

Calling Sequence:

m1:= op2mat(fkt)

m1:= op2mat2(fkt)

Parameters:

- fkt : an linear operator acting on V^ (or V^ &t V^)

Output:

- m1 : a 2^dim_V x 2^dim_V matrix (a 4^dim_V x 4^dim_V matrix)

Global parameters:

- dim_V

Description:

- The op2mat command is useful to derive matrix forms of linear operators. It can be used together with linop (linop2) to move from an operator description to matrix form and back. The derived matrices are regarded as elements of End V^ (End V^ &t V^) . The basis used is assumed to be the standard Grassmann basis of Clifford or the following sum^{2^dim_V}_{i,j=1} &t(b[i],b[j]), where the b[i] are Grassmann bases of V^.
- The main purpose of these two functions is to get a matrix form of heavily used operators needed in long computation time, e.g. the antipode of a Clifford Hopf gebra or the switch (may be derived from op2mat2). The functionality of linalg is then available to those matrices.

Examples:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
  dim_V:=2:
```

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

Let the name of the operator be R, its matrix elements be R[i,j], we have:

```
> linop(Id,R);
  linop(e1,R);
  linop(e1we2,R);
```

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

$$R_{1,1}\, Id + R_{2,1}\, e1 + R_{3,1}\, e2 + R_{4,1}\, e1we2$$

$$R_{1,2}\, Id + R_{2,2}\, e1 + R_{3,2}\, e2 + R_{4,2}\, e1we2$$

$$R_{1,4}\, Id + R_{2,4}\, e1 + R_{3,4}\, e2 + R_{4,4}\, e1we2$$

Get the matrix form of linop(x,R). First we define an operator R from which we derive the matrix form, but we show in a second line that parameters may be passed to op2mat which allows us to use linop directly:

```
> R:=proc(x) linop(x,R) end:   # define a proper operator
  op2mat(R);                   #
  op2mat(linop,R);             #
```

$$\begin{bmatrix} R_{1,1} & R_{1,2} & R_{1,3} & R_{1,4} \\ R_{2,1} & R_{2,2} & R_{2,3} & R_{2,4} \\ R_{3,1} & R_{3,2} & R_{3,3} & R_{3,4} \\ R_{4,1} & R_{4,2} & R_{4,3} & R_{4,4} \end{bmatrix}$$

$$\begin{bmatrix} R_{1,1} & R_{1,2} & R_{1,3} & R_{1,4} \\ R_{2,1} & R_{2,2} & R_{2,3} & R_{2,4} \\ R_{3,1} & R_{3,2} & R_{3,3} & R_{3,4} \\ R_{4,1} & R_{4,2} & R_{4,3} & R_{4,4} \end{bmatrix}$$

Derive the matrix of the Grassmann antipode, in dim_V=2 we get a 4x4 matrix:

```
> bas:=cbasis(dim_V):
  matS:=op2mat(gantipode,1);
  map(linop,bas,matS);
  map(gantipode,bas,1);
```

$$matS := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[Id, -e1, -e2, e1we2]$$

$$[Id, -e1, -e2, e1we2]$$

We checked using linop that this is the same operator as defined abstractly. Hence our indexing is compatible.

A few further examples are:

```
> conv_unit:=proc(x) Id*scalarpart(x) end:
  `conv_unit  ---->   `,op2mat(conv_unit);
  gr_loop:=proc(x) drop_t(&map(&gco(x),1,wedge)) end:
  `Grassmann loop  ---->   `,op2mat(gr_loop);
  X:=add(x[i]*bas[i],i=1..2^dim_V):
  scalar_right_conv:=proc(x) wedge(x,X) end:
  `scalar right conv  ---->   `,op2mat(scalar_right_conv);
```

```
scalar_left_conv:=proc(x) wedge(X,x) end:
`scalar left conv  ---->   `,op2mat(scalar_left_conv);
scalar_coright_conv:=proc(x)
drop_t(contract(&t(&gco(x),X),2,EV)) end:
`scalar coright conv  ---->   `,op2mat(scalar_coright_conv);
scalar_coleft_conv:=proc(x)
drop_t(contract(&t(X,&gco(x)),1,EV)) end:
`scalar coleft conv  ---->   `,op2mat(scalar_coleft_conv);
```

$$\textit{conv\_unit} \text{ ----> } , \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\textit{Grassmann loop} \text{ ----> } , \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

$$\textit{scalar right conv} \text{ ----> } , \begin{bmatrix} x_1 & 0 & 0 & 0 \\ x_2 & x_1 & 0 & 0 \\ x_3 & 0 & x_1 & 0 \\ x_4 & x_3 & -x_2 & x_1 \end{bmatrix}$$

$$\textit{scalar left conv} \text{ ----> } , \begin{bmatrix} x_1 & 0 & 0 & 0 \\ x_2 & x_1 & 0 & 0 \\ x_3 & 0 & x_1 & 0 \\ x_4 & -x_3 & x_2 & x_1 \end{bmatrix}$$

$$\textit{scalar coright conv} \text{ ----> } , \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ 0 & x_1 & 0 & x_3 \\ 0 & 0 & x_1 & -x_2 \\ 0 & 0 & 0 & x_1 \end{bmatrix}$$

$$\textit{scalar coleft conv} \text{ ----> } , \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ 0 & x_1 & 0 & -x_3 \\ 0 & 0 & x_1 & x_2 \\ 0 & 0 & 0 & x_1 \end{bmatrix}$$

op2mat2 is the counterpart for operators acting on V^ &t V^, in our case the vector space dimension is 2 (dim_V=2) and dim V^ = 2^2 = 4 so the output is a 16 times 16 matrix:

We check the function by some examples and will see that it is compatible in indexing with

linop2:

```
> GSW:=op2mat2(gswitch,1);
  `V^2_bas`:=[seq(seq(&t(bas[i],bas[j]),i=1..2^dim_V),j=1..2^dim_
  V)];
  `V^2_GSW_bas`:=convert(evalm(GSW &* `V^2_bas`),list);   #
  compose using linalg
  `V^2_gs_bas1`:=map(gswitch,`V^2_bas`,1);                # act
  using the operator
  `V^2_gs_bas2`:=map(linop2,`V^2_bas`,GSW);               # act
  using linop2 and the matrix GSW
  printf("Are the two lists V^2_GSW_bas and V^2_gs_bas1 equal ?
  %a \n",

  op({seq(is(`V^2_GSW_bas`[i]=`V^2_gs_bas1`[i]),i=1..4^dim_V)}));

  printf("Are the two lists V^2_GSW_bas and V^2_gs_bas2 equal ?
  %a \n",

  op({seq(is(`V^2_GSW_bas`[i]=`V^2_gs_bas2`[i]),i=1..4^dim_V)}));
```

$$GSW := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$V^2\_bas := [Id \,\&t\, Id, e1 \,\&t\, Id, e2 \,\&t\, Id, e1we2 \,\&t\, Id, Id \,\&t\, e1, e1 \,\&t\, e1, e2 \,\&t\, e1, e1we2 \,\&t\, e1, Id \,\&t\, e2, e1 \,\&t\, e2, e2 \,\&t\, e2, e1we2 \,\&t\, e2, Id \,\&t\, e1we2, e1 \,\&t\, e1we2, e2 \,\&t\, e1we2, e1we2 \,\&t\, e1we2]$

$V^2\_GSW\_bas := [Id \,\&t\, Id, Id \,\&t\, e1, Id \,\&t\, e2, Id \,\&t\, e1we2, e1 \,\&t\, Id, -(e1 \,\&t\, e1), -(e1 \,\&t\, e2), e1 \,\&t\, e1we2, e2 \,\&t\, Id, -(e2 \,\&t\, e1), -(e2 \,\&t\, e2), e2 \,\&t\, e1we2, e1we2 \,\&t\, Id, e1we2 \,\&t\, e1, e1we2 \,\&t\, e2, e1we2 \,\&t\, e1we2]$

$$V^2\_gs\_bas1 := [Id \,\&t\, Id, Id \,\&t\, e1, Id \,\&t\, e2, Id \,\&t\, e1we2, e1 \,\&t\, Id, -(e1 \,\&t\, e1), -(e1 \,\&t\, e2), e1 \,\&t\, e1we2, e2 \,\&t\, Id, -(e2 \,\&t\, e1), -(e2 \,\&t\, e2), e2 \,\&t\, e1we2, e1we2 \,\&t\, Id, e1we2 \,\&t\, e1, e1we2 \,\&t\, e2, e1we2 \,\&t\, e1we2]$$

$$V^2\_gs\_bas2 := [Id \,\&t\, Id, Id \,\&t\, e1, Id \,\&t\, e2, Id \,\&t\, e1we2, e1 \,\&t\, Id, -(e1 \,\&t\, e1), -(e1 \,\&t\, e2), e1 \,\&t\, e1we2, e2 \,\&t\, Id, -(e2 \,\&t\, e1), -(e2 \,\&t\, e2), e2 \,\&t\, e1we2, e1we2 \,\&t\, Id, e1we2 \,\&t\, e1, e1we2 \,\&t\, e2, e1we2 \,\&t\, e1we2]$$

```
Are the two lists V^2_GSW_bas and V^2_gs_bas1 equal ?  true
Are the two lists V^2_GSW_bas and V^2_gs_bas2 equal ?  true
```

Let us give a further example:

```
> BI:=linalg[matrix](2^dim_V,2^dim_V,(i,j)->C[i,j]):
  make_BI_Id():
  BC:=proc(x)
  tcollect(&t(scalarpart(drop_t(&map(x,1,cmul)))*&cco(Id))) end:
  seq(seq(BC(&t(bas[i],bas[j])),i=1..2^dim_V),j=1..2^dim_V);
```

$$(Id \,\&t\, Id) + C_{1,1}(e1 \,\&t\, e1) + C_{2,1}(e2 \,\&t\, e1) + C_{1,2}(e1 \,\&t\, e2) + C_{2,2}(e2 \,\&t\, e2) + (C_{2,1}C_{1,2} - C_{2,2}C_{1,1})(e1we2 \,\&t\, e1we2), \&t(0), \&t(0), \&t(0), \&t(0),$$
$$B_{1,1}C_{2,1}(e2 \,\&t\, e1) + B_{1,1}(Id \,\&t\, Id) + B_{1,1}C_{1,1}(e1 \,\&t\, e1) + B_{1,1}C_{1,2}(e1 \,\&t\, e2) + B_{1,1}C_{2,2}(e2 \,\&t\, e2) + B_{1,1}(C_{2,1}C_{1,2} - C_{2,2}C_{1,1})(e1we2 \,\&t\, e1we2), B_{2,1}C_{2,1}(e2 \,\&t\, e1) + B_{2,1}(Id \,\&t\, Id) + B_{2,1}C_{1,1}(e1 \,\&t\, e1) + B_{2,1}C_{1,2}(e1 \,\&t\, e2) + B_{2,1}C_{2,2}(e2 \,\&t\, e2) + B_{2,1}(C_{2,1}C_{1,2} - C_{2,2}C_{1,1})(e1we2 \,\&t\, e1we2), \&t(0), \&t(0), B_{1,2}C_{2,1}(e2 \,\&t\, e1) + B_{1,2}(Id \,\&t\, Id) + B_{1,2}C_{1,1}(e1 \,\&t\, e1) + B_{1,2}C_{1,2}(e1 \,\&t\, e2) + B_{1,2}C_{2,2}(e2 \,\&t\, e2) + B_{1,2}(C_{2,1}C_{1,2} - C_{2,2}C_{1,1})(e1we2 \,\&t\, e1we2), B_{2,2}C_{2,1}(e2 \,\&t\, e1) + B_{2,2}(Id \,\&t\, Id) + B_{2,2}C_{1,1}(e1 \,\&t\, e1) + B_{2,2}C_{1,2}(e1 \,\&t\, e2) + B_{2,2}C_{2,2}(e2 \,\&t\, e2) + B_{2,2}(C_{2,1}C_{1,2} - C_{2,2}C_{1,1})(e1we2 \,\&t\, e1we2), \&t(0), \&t(0), \&t(0), \&t(0),$$
$$(B_{2,1}B_{1,2} - B_{2,2}B_{1,1})C_{2,1}(e2 \,\&t\, e1) + (B_{2,1}B_{1,2} - B_{2,2}B_{1,1})(Id \,\&t\, Id) + (B_{2,1}B_{1,2} - B_{2,2}B_{1,1})C_{1,1}(e1 \,\&t\, e1) + (B_{2,1}B_{1,2} - B_{2,2}B_{1,1})C_{1,2}(e1 \,\&t\, e2) + (B_{2,1}B_{1,2} - B_{2,2}B_{1,1})C_{2,2}(e2 \,\&t\, e2) + (B_{2,1}B_{1,2} - B_{2,2}B_{1,1})(C_{2,1}C_{1,2} - C_{2,2}C_{1,1})(e1we2 \,\&t\, e1we2)$$

```
> op2mat2(BC);
```

$$[1, 0, 0, 0, 0, C_{1,1}, C_{2,1}, 0, 0, C_{1,2}, C_{2,2}, 0, 0, 0, 0, C_{2,1}C_{1,2} - C_{2,2}C_{1,1}]$$
$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$
$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$
$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$
$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$
$$[B_{1,1}, 0, 0, 0, 0, B_{1,1}C_{1,1}, B_{1,1}C_{2,1}, 0, 0, B_{1,1}C_{1,2}, B_{1,1}C_{2,2}, 0, 0, 0, 0, B_{1,1}(C_{2,1}C_{1,2} - C_{2,2}C_{1,1})]$$

$$[B_{2,1}, 0, 0, 0, 0, B_{2,1} C_{1,1}, B_{2,1} C_{2,1}, 0, 0, B_{2,1} C_{1,2}, B_{2,1} C_{2,2}, 0, 0, 0, 0, B_{2,1} (C_{2,1} C_{1,2} - C_{2,2} C_{1,1})]$$

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

$$[B_{1,2}, 0, 0, 0, 0, B_{1,2} C_{1,1}, B_{1,2} C_{2,1}, 0, 0, B_{1,2} C_{1,2}, B_{1,2} C_{2,2}, 0, 0, 0, 0, B_{1,2} (C_{2,1} C_{1,2} - C_{2,2} C_{1,1})]$$

$$[B_{2,2}, 0, 0, 0, 0, B_{2,2} C_{1,1}, B_{2,2} C_{2,1}, 0, 0, B_{2,2} C_{1,2}, B_{2,2} C_{2,2}, 0, 0, 0, 0, B_{2,2} (C_{2,1} C_{1,2} - C_{2,2} C_{1,1})]$$

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

$$[B_{2,1} B_{1,2} - B_{2,2} B_{1,1}, 0, 0, 0, 0, (B_{2,1} B_{1,2} - B_{2,2} B_{1,1}) C_{1,1}, (B_{2,1} B_{1,2} - B_{2,2} B_{1,1}) C_{2,1}, 0, 0, (B_{2,1} B_{1,2} - B_{2,2} B_{1,1}) C_{1,2}, (B_{2,1} B_{1,2} - B_{2,2} B_{1,1}) C_{2,2}, 0, 0, 0, 0, (B_{2,1} B_{1,2} - B_{2,2} B_{1,1}) (C_{2,1} C_{1,2} - C_{2,2} C_{1,1})]$$

As a last example we give switch and the time the antipode acting on the switch:

```
> aasw:=proc(x) switch(tcollect(gantipode(gantipode(x,1),2)),1)
  end:
  op2mat2(gswitch,1);
  op2mat2(aasw);
  convert(evalm(%-%%),set);  ## these operators are not
  identical!
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\{-2, 0, 2\}$$

```
> printf("The worksheet took %f seconds to compute on Intel
  Pentium M 2.13 GHz 2GB RAM WinXP Prof",time()-bench);
```

```
The worksheet took 5.642000 seconds to compute on Intel Pentium M 2.13 GHz 2GB
RAM WinXP Prof
```

```
>
```

NOTE: If the entries of the tensor polynom are out of the bound of the matrix, this function may go into an **endless loop**! E.g. mapop2(&t(e5,e6),1,gs); in our example, since dim_V was 2.

See Also: Bigebra:-mapop, Bigebra:-mapop2, Bigebra:-EV, Bigebra:-pairing, Bigebra:-linop, Bigebra:-list2mat2, Bigebra:-help

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-pairing - computes pairing w.r.t. an exponentially generated bilinear form B^.

Calling Sequence:

- sc := pairing(c1,c2,name)

Parameters:

- c1,c2 : are Clifford polynoms.
- name : optional name to be used as kernel symbol for the pairing.

Output:

- sc : is a scalar.

Description:

- The pairing is most often used together with contract function on tensors. However, it acts generically on two Clifford polynomials.
- The pairing acts w.r.t. the bilinear form B^ which is a global variable used in CLIFFORD.
- Note that the relation between vectors and co-vectors is not fixed. One is therefore free to choose any nondegenerate (or even degenerate) bilinear form to establish this connection. If the usual canonical duality is used, this should be achieved by using EV rather than the pairing.
- A pairing is called *exponentially generated* if it can be written as exterior exponential of a pairing of the generating vector space: B^ = exp^(B).

Examples:

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

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

The pairing on homogeneous decomposable elements of the same grade, we use the optional parameter A and Z:

```
> pairing(Id,Id);
  pairing(e1,e2),pairing(e2,e1),pairing(ei,ej);
  pairing(e1we2,e1we2),pairing(e1we3,e1we2,A);
  pairing(e1we2we3,e1we2we3,Z);
```

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

$$1$$

$$B_{1,2}, B_{2,1}, B_{i,j}$$

$$B_{2,1}\,B_{1,2} - B_{2,2}\,B_{1,1}, A_{3,1}\,A_{1,2} - A_{3,2}\,A_{1,1}$$

$$Z_{3,1}\,Z_{2,2}\,Z_{1,3} - Z_{3,1}\,Z_{2,3}\,Z_{1,2} - Z_{3,2}\,Z_{2,1}\,Z_{1,3} + Z_{3,2}\,Z_{2,3}\,Z_{1,1} + Z_{3,3}\,Z_{2,1}\,Z_{1,2} - Z_{3,3}\,Z_{2,2}\,Z_{1,1}$$

The pairing on homogeneous decomposable elements of different grades:

```
> pairing(Id,ei);
  pairing(e1we2,e2),pairing(e2,e1we2);
```

$$0$$

$$0, 0$$

The pairing on inhomogeneous elements:

```
> pairing(a*Id-b*e1-e1we2+d*e2we3we4,Id+e2we3-4*sin(x)*e1we2);
```

$$a - B_{2,2}\, B_{1,3} + B_{2,3}\, B_{1,2} + 4\sin(x)\, B_{2,1}\, B_{1,2} - 4\sin(x)\, B_{2,2}\, B_{1,1}$$

Use contract to map the pairing onto adjacent tensor slots.

```
> contract(&t(e1,e2we3,e3we1,e2),2,pairing);
```

$$(B_{3,3}\, B_{2,1} - B_{3,1}\, B_{2,3})\,(e1\ \&\mathrm{t}\ e2)$$

```
> printf("The worksheet took %f seconds to compute on Intel
  Pentium M 2.13 GHz 2GB RAM WinXP Prof",time()-bench);
The worksheet took 0.263000 seconds to compute on Intel Pentium M 2.13 GHz 2GB
RAM WinXP Prof
>
```

See Also: Bigebra:-`&t`, Bigebra:-`type/tensorpolynom`, Bigebra:-contract, Bigebra:-EV, Bigebra help page

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-peek - pick elements from tensor slots.

Calling Sequence:

- [a,b | S] := peek(p1,i)

Parameters:

- p1 : a tensorpolynom which is of rank not less than i in each factor
- i : the slot number (first slot is from the left is 1) of the pair (i,i+1) on which the switch acts

Output:

- IF p1 was homogeneous:
- a : the entry of the i-th slot
- b : the remaining tensor
- IF p1 was an inhomogeneous element:
- S : a sequence of pairs S[i] = a,b of elements as in the homogeneous case.

Description:

- Given a tensor monom or tensor polynomial peek selects the element of the i-th slot of the tensor product. This function is for internal use, but can be used to form user supplied functions on tensors.
- The output of peek depends on the type of the tensor being processed. On homogeneous tensors, i.e. tensor monoms, peek simply returns a pair (expression sequence) composed of the element in the i-th tensor slot and a tensor composed from the i-1 slots. On tensorpolynoms peek acts on every term of the sum as described above, and returns a sequence of lists of type clipolynom, tensorpolynom.
- Scalar prefactors are returned with the extracted element a.

Examples:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

Peek on homogeneous tensors:

```
> peek(&t(e1,e2),1);
  peek(&t(e1,e2),2);
  peek(&t(e1,e2,e3,e4,e5,e6),4);
```

$$e1, \&t(e2)$$

$$e2, \&t(e1)$$

$$e4, \&t(e1, e2, e3, e5, e6)$$

Having scalar prefactors:

```
> peek(&t(a*e1we2,b*e3),1);
  peek(&t(a*e1we2,b*e3),2);
```

$$a\ b\ e1we2, \&t(e3)$$

$$a\ b\ e3, \&t(e1we2)$$

Peek on inhomogeneous elements:

```
> peek(&t(a*e1+b*e2we3,c*e2we3+d+e1we4),1);
  peek(&t(a*e1+b*e2we3,c*e2we3+d+e1we4),2);
```

$$[a\ c\ e1, \&t(e2we3)], [a\ d\ e1, \&t(1)], [a\ e1, \&t(e1we4)], [b\ c\ e2we3, \&t(e2we3)], [b\ d\ e2we3, \&t(1)], [b\ e2we3, \&t(e1we4)]$$

$$[a\ c\ e2we3, \&t(e1)], [a\ d, \&t(e1)], [a\ e1we4, \&t(e1)], [b\ c\ e2we3, \&t(e2we3)], [b\ d, \&t(e2we3)], [b\ e1we4, \&t(e2we3)]$$

```
> peek(&t(e1,e2)+&t(e3,e4),1);
```

$$[e1, \&t(e2)], [e3, \&t(e4)]$$

If the slot i is not available in a tensor, peek fails!

```
> peek(&t(e1,e2),3);
Error, (in Bigebra:-peek) improper op or subscript selector
```

```
> printf("The worksheet took %f seconds to compute on AMD Athlon 2700+ 1GB RAM WinXP Prof",time()-bench);
The worksheet took 0.124000 seconds to compute on AMD Athlon 2700+ 1GB RAM WinXP Prof
>
```

See Also: Bigebra:-`&t`, Bigebra:-`type/tensorpolynom`, Bigebra:-poke

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-poke - puts elements into tensor slots.

Calling Sequence:

- t1 := poke(p1,c1,i)

Parameters:

- p1 : a tensorpolynom of rank not less than i in each factor
- c1 : a Clifford polynomial
- i : the slot number (first slot is from the left is 1) where to insert the element, i.e. the tail of elements from the i-th one onwards in moved by 1 to the right and the new element is placed at the i-th slot.

Output:

- t1 : a tensor

Description:

- Given a tensor monom or tensor polynomial poke puts a Clifford monom or polynom into the i-th slot of the tensor product. This function is for internal use, but can be used to form user supplied functions on tensors.
  E.g.: poke(&t(p1,..,pi,..,pn),c,i) = &t(p1,..,p(i-1),c,pi,..pn)
  poke(&t(p1,..,pn),c,(n+1)) = &t(p1,..,pn,c) i.e. append c.
- Poke raises the rank of a tensor by one.

Examples:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

Poke into homogeneous tensors:

```
> poke(&t(e1,e2),e4,1);
  poke(&t(e1,e2),e4,2);
  poke(&t(e1,e2),e4,3);
```

$$\&t(e4, e1, e2)$$
$$\&t(e1, e4, e2)$$
$$\&t(e1, e2, e4)$$

Having scalar prefactors in p1 and c1:

```
> poke(&t(a*e1we2,a*e3),x*e4,1);
  poke(&t(a*e1we2,b*e3),x*e4,2);
  poke(&t(a*e1we2,b*e3),x*e4,3);  # i.e. behind the last slot !

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

$$a^2\, x\, \&t(e4, e1we2, e3)$$
$$a\, b\, x\, \&t(e1we2, e4, e3)$$
$$a\, b\, x\, \&t(e1we2, e3, e4)$$

Poke inhomogeneous Clifford elements into homogeneous tensors:

```
> poke(&t(a*e2we3,e1we4),x*e1+y*e2,1);
  poke(&t(a*e2we3,e1we4),x*e1+y*e2,2);
  poke(&t(a*e2we3,e1we4),x*e1+y*e2,3);
```

$$a\,(x\, \&t(e1, e2we3, e1we4) + y\, \&t(e2, e2we3, e1we4))$$
$$a\,(x\, \&t(e2we3, e1, e1we4) + y\, \&t(e2we3, e2, e1we4))$$
$$a\,(x\, \&t(e2we3, e1we4, e1) + y\, \&t(e2we3, e1we4, e2))$$

Poke inhomogeneous Clifford elements into inhomogeneous tensors:

```
> poke(&t(e1,e2)+&t(e3,e4),e5+e6,2);
```

$$\&t(e1, e5, e2) + \&t(e1, e6, e2) + \&t(e3, e5, e4) + \&t(e3, e6, e4)$$

If the number of slots is i, you can poke to i+1 (append a slot) but not to i+2 which causes an error!

```
> poke(&t(e2we3,e1we4),e5,3); # OK, appends
  poke(&t(e2we3,e1we4),e5,4); # error
```

$$\&t(e2we3, e1we4, e5)$$

```
Error, (in Bigebra:-poke) invalid subscript selector
```

```
> printf("The worksheet took %f seconds to compute on Intel
  Pentium M 2.13 GHz 2GB RAM WinXP Prof",time()-bench);
```

```
The worksheet took 0.077000 seconds to compute on Intel Pentium M 2.13 GHz 2GB
RAM WinXP Prof
```

```
>
```

See Also: Bigebra:-`&t`, Bigebra:-`type/tensorpolynom`, Bigebra:-peek

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-remove_eq - helper function to remove tautologies.

Calling Sequence:

- s2 := map(remove_eq,s2)

Parameters:

- s1 : set of equations.

Output:

- s2 : set of equations free of tautologies

Description:

- Remove_eq(uations) is used e.g. in tsolve1 and may be useful to the user in solving large sets of equations
- Remove_eq is usually mapped to a set of equations to remove tautologies. This is helpful to figure out those equalities which are conditions for some variables.

Examples:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

Reducing a set of equations:

```
> s1:={e1=e1,x[1]=x[2],x[2]=x[2],a=e1*b};
  s2:={seq(seq(op([x[i]=x[i+(j mod 2)]]),i=1..5),j=1..10)};
```

$$s1 := \{a = e1\ b, e1 = e1, x_1 = x_2, x_2 = x_2\}$$

$$s2 := \{x_1 = x_2, x_2 = x_2, x_2 = x_3, x_3 = x_4, x_4 = x_5, x_5 = x_6, x_1 = x_1, x_3 = x_3, x_4 = x_4, x_5 = x_5\}$$

```
> map(remove_eq,s1);
```

$$\{a = e1\ b, x_1 = x_2\}$$

```
> map(remove_eq,s2);
```

$$\{x_1 = x_2, x_2 = x_3, x_3 = x_4, x_4 = x_5, x_5 = x_6\}$$

```
> printf("The worksheet took %f seconds to compute on Intel
  Pentium M 2.13 GHz 2GB RAM WinXP Prof",time()-bench);
The worksheet took 0.016000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM WinXP Prof
>
```

See Also: Bigebra:-help, Bigebra:-`tsolve1`

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-switch - switch of tensor slots

Calling Sequence:

- p1 = switch(p2,i)

Parameters:

- p2 : a tensorpolynom of rank in each factor not less than i
- i : the slot number (first slot is from the left is 1) of the pair (i,i+1) on which the switch acts.

Output:

- p1 : a tensorpolynom

Description:

- Given a tensor polynomial the switch swaps two adjacent slots in a tensor product. No other action is performed.
- Note that switch generates not signs like gswitch.

Examples:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

Switch/swap tensor factors:

```
> &t(e1,e2);
  switch(%,1);
```

$$e1 \,\&t\, e2$$

$$e2 \,\&t\, e1$$

```
> &t(e1,a*e2+b*e2we3,e1we4-sin(x)*e5);
  switch(%,1);
  switch(%%,2);
```

$$a\,\&t(e1, e2, e1we4) - a\sin(x)\,\&t(e1, e2, e5) + b\,\&t(e1, e2we3, e1we4) - b\sin(x)\,\&t(e1, e2we3, e5)$$

$$a\,\&t(e2, e1, e1we4) - a\sin(x)\,\&t(e2, e1, e5) + b\,\&t(e2we3, e1, e1we4) - b\sin(x)\,\&t(e2we3, e1, e5)$$

$$a\,\&t(e1, e1we4, e2) - a\sin(x)\,\&t(e1, e5, e2) + b\,\&t(e1, e1we4, e2we3) - b\sin(x)\,\&t(e1, e5, e2we3)$$

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

```
> switch(&t(e1,e2),3);
Error, (in Bigebra:-switch) invalid subscript selector
```

```
> printf("The worksheet took %f seconds to compute on AMD Athlon
```

```
  2700+ 1GB RAM WinXP Prof",time()-bench);
The worksheet took 0.062000 seconds to compute on AMD Athlon 2700+ 1GB RAM WinX
P Prof
>
```

See Also: Bigebra:-`&t`, Bigebra:-gswitch

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-tcollect - collects coefficients of tensor polynomial

Calling Sequence:

- p1 := tcollect(p2)

Parameters:

- p2 : a tensor polynomial

Output:

- p1 : a tensor polynomial.

Description:

- The function tcollect is used to collect coefficients of tensor polynomials. ***This function is sometimes needed to feed output into other functions of Bigebra***. Moreover it allows for a better comparison between tensor polynomials.
- In later versions of BIGEBRA some functions will automatically tcollect their output for convenience. However in this version the user is called to do this for performance reasons.

Examples:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

Examples:

```
> tcollect(a*(&t(e1we2)+b*&t(e1we2)));
  tcollect(x*&t(e1we2we3,e4)+y*&t(e1we2we3,e4));
```

$$a\,(1+b)\,\&t(e1we2)$$

$$(x+y)\,(e1we2we3\ \&t\ e4)$$

Tcollect simply returns Clifford polynomials without clicollecting them!

```
> tcollect(e1+e2we3-4+sin(x)*e1we2+e1we2);
```

$$e1+e2we3-4+\sin(x)\,e1we2+e1we2$$

```
> printf("The worksheet took %f seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM WinXP Prof",time()-bench);
The worksheet took 0.016000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM WinXP Prof
>
```

See Also: Bigebra:-`&t`, Bigebra:-`type/tensorpolynom`, Bigebra:-help

Last modified: December 20, 2007 /BF/RA.

Function: `type/tensorbasmonom`
`type/tensormonom`
`type/tensorpolynom` -- new types for tensors

Calling Sequence:

- b1 = type(p, tensorbasmonom)
- b1 = type(p, tensormonom)
- b1 = type(p, tensorpolynom)

Parameters:

- p - an algebraic expression of type 'anything'.

Output:

- A Boolean value 'true' or 'false'

Description:

- Elements of the tensor algebra share this type, see tensor product.
- The procedure returns 'true' or 'false' depending whether its argument is or is not of one of the types 'tensorbasmonom', 'tensormonom' or 'tensorpolynom'.
- The types are inclusive, i.e. a 'tensorbasmonom' is also a 'tensormonom' which happens to be also a 'tensorpolynom'.
- Types are designed for mostly internal use.
- **Note:** During initialization of the Bigebra package these types are defined and have been placed in the top-level name space, no long form available/needed/useful.

Examples:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> Basmonom1:=&t(e1,e2,e3);
  Basmonom2:=&t(e1,e1we2);
  Monom1:=exp(I*phi)*&t(e1,e2we3,e4);
  Monom2:=-a*&t(e3we4,e1,e2);
  Polynom:=&t(Monom1+Monom2+Basmonom2);
```

$$Basmonom1 := \&t(e1, e2, e3)$$

$$Basmonom2 := e1 \ \&t \ e1we2$$

$$Monom1 := \mathbf{e}^{(\phi\, I)} \ \&t(e1, e2we3, e4)$$

$$Monom2 := -a \ \&t(e3we4, e1, e2)$$

$$Polynom := \mathbf{e}^{(\phi\, I)} \ \&t(e1, e2we3, e4) - a \ \&t(e3we4, e1, e2) + (e1 \ \&t \ e1we2)$$

```
> type(Basmonom1,tensorbasmonom),
  type(Basmonom1,tensormonom),
  type(Basmonom1,tensorpolynom);
```

$$true, true, true$$

```
> type(Basmonom2,tensorbasmonom),
  type(Basmonom2,tensormonom),
  type(Basmonom2,tensorpolynom);
```

$$true, true, true$$

```
> type(Monom2,tensorbasmonom),
  type(Monom2,tensormonom),
  type(Monom2,tensorpolynom);
```

$$false, true, true$$

```
> type(Polynom,tensorbasmonom),
  type(Polynom,tensormonom),
  type(Polynom,tensorpolynom);
```

$$false, false, true$$

However, be careful with the infix form of &(ampersand) operators, see define, tensor product.

```
> type(a*e2 &t b*e3,tensorbasmonom),
  type(a*e2 &t b*e3,tensormonom);  ## NOTE: infix is buggy, use
  parentheses!!
  eval(a*e2 &t b*e3);  ## second tensor slot is not properly
  treated, see e3!!
  eval((a*e1) &t (b*e3)), &t(a*e2,b*e3); ## works out correctly.
```

$$false, true$$

$$a\ b\ (e2\ \&t\ 1)\ e3$$

$$a\ b\ (e1\ \&t\ e3), a\ b\ (e2\ \&t\ e3)$$

```
> add(a[i]*&t(seq(e||j,j=1..i)),i=1..4);
  type(%,tensorpolynom);
```

$$a_1\ \&t(e1) + a_2\ (e1\ \&t\ e2) + a_3\ \&t(e1, e2, e3) + a_4\ \&t(e1, e2, e3, e4)$$

$$true$$

```
> remove(type,a*&t(e1,e2),tensorbasmonom);
```

$$a$$

```
> select(type,a*&t(e1,e2),tensorbasmonom);
```

$$e1\ \&t\ e2$$

But be careful about this:

```
> select(type,a*&t(e1,e2)+b*&t(e3,e4),tensorbasmonom);
```

$$0$$

```
> printf("The worksheet took %f seconds to compute on Intel
  Pentium M 2.13 GHz 2GB RAM WinXP Prof",time()-bench);
The worksheet took 0.046000 seconds to compute on Intel Pentium M 2.13 GHz 2GB
```

```
RAM WinXP Prof
>
```

See Also: Bigebra:-`&t`, Bigebra:-define, Bigebra:-help

Last modified: December 20, 2007 /BF/RA.

Function: `type/tensorbasmonom`
`type/tensormonom`
`type/tensorpolynom` -- new types for tensors

Calling Sequence:

- b1 = type(p, tensorbasmonom)
- b1 = type(p, tensormonom)
- b1 = type(p, tensorpolynom)

Parameters:

- p - an algebraic expression of type 'anything'.

Output:

- A Boolean value 'true' or 'false'

Description:

- Elements of the tensor algebra share this type, see tensor product.
- The procedure returns 'true' or 'false' depending whether its argument is or is not of one of the types 'tensorbasmonom', 'tensormonom' or 'tensorpolynom'.
- The types are inclusive, i.e. a 'tensorbasmonom' is also a 'tensormonom' which happens to be also a 'tensorpolynom'.
- Types are designed for mostly internal use.
- **Note:** During initialization of the Bigebra package these types are defined and have been placed in the top-level name space, no long form available/needed/useful.

Examples:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> Basmonom1:=&t(e1,e2,e3);
  Basmonom2:=&t(e1,e1we2);
  Monom1:=exp(I*phi)*&t(e1,e2we3,e4);
  Monom2:=-a*&t(e3we4,e1,e2);
  Polynom:=&t(Monom1+Monom2+Basmonom2);
```

$$Basmonom1 := \&t(e1, e2, e3)$$

$$Basmonom2 := e1 \ \&t \ e1we2$$

$$Monom1 := \mathbf{e}^{(\phi\, I)} \ \&t(e1, e2we3, e4)$$

$$Monom2 := -a \ \&t(e3we4, e1, e2)$$

$$Polynom := \mathbf{e}^{(\phi\, I)} \ \&t(e1, e2we3, e4) - a \ \&t(e3we4, e1, e2) + (e1 \ \&t \ e1we2)$$

```
> type(Basmonom1,tensorbasmonom),
  type(Basmonom1,tensormonom),
  type(Basmonom1,tensorpolynom);
```

$$true, true, true$$

```
> type(Basmonom2,tensorbasmonom),
  type(Basmonom2,tensormonom),
  type(Basmonom2,tensorpolynom);
```

$$true, true, true$$

```
> type(Monom2,tensorbasmonom),
  type(Monom2,tensormonom),
  type(Monom2,tensorpolynom);
```

$$false, true, true$$

```
> type(Polynom,tensorbasmonom),
  type(Polynom,tensormonom),
  type(Polynom,tensorpolynom);
```

$$false, false, true$$

However, be careful with the infix form of &(ampersand) operators, see define, tensor product.

```
> type(a*e2 &t b*e3,tensorbasmonom),
  type(a*e2 &t b*e3,tensormonom);  ## NOTE: infix is buggy, use
  parentheses!!
  eval(a*e2 &t b*e3);  ## second tensor slot is not properly
  treated, see e3!!
  eval((a*e1) &t (b*e3)), &t(a*e2,b*e3); ## works out correctly.
```

$$false, true$$

$$a\ b\ (e2\ \&t\ 1)\ e3$$

$$a\ b\ (e1\ \&t\ e3), a\ b\ (e2\ \&t\ e3)$$

```
> add(a[i]*&t(seq(e||j,j=1..i)),i=1..4);
  type(%,tensorpolynom);
```

$$a_1\ \&t(e1) + a_2\ (e1\ \&t\ e2) + a_3\ \&t(e1, e2, e3) + a_4\ \&t(e1, e2, e3, e4)$$

$$true$$

```
> remove(type,a*&t(e1,e2),tensorbasmonom);
```

$$a$$

```
> select(type,a*&t(e1,e2),tensorbasmonom);
```

$$e1\ \&t\ e2$$

But be careful about this:

```
> select(type,a*&t(e1,e2)+b*&t(e3,e4),tensorbasmonom);
```

$$0$$

```
> printf("The worksheet took %f seconds to compute on Intel
  Pentium M 2.13 GHz 2GB RAM WinXP Prof",time()-bench);
The worksheet took 0.032000 seconds to compute on Intel Pentium M 2.13 GHz 2GB
```

```
RAM WinXP Prof
>
```

See Also: Bigebra:-`&t`, Bigebra:-define, Bigebra:-help

Last modified: December 20, 2007 /BF/RA.

Function: `type/tensorbasmonom`
`type/tensormonom`
`type/tensorpolynom` -- new types for tensors

Calling Sequence:

- b1 = type(p, tensorbasmonom)
- b1 = type(p, tensormonom)
- b1 = type(p, tensorpolynom)

Parameters:

- p - an algebraic expression of type 'anything'.

Output:

- A Boolean value 'true' or 'false'

Description:

- Elements of the tensor algebra share this type, see tensor product.
- The procedure returns 'true' or 'false' depending whether its argument is or is not of one of the types 'tensorbasmonom', 'tensormonom' or 'tensorpolynom'.
- The types are inclusive, i.e. a 'tensorbasmonom' is also a 'tensormonom' which happens to be also a 'tensorpolynom'.
- Types are designed for mostly internal use.
- **Note:** During initialization of the Bigebra package these types are defined and have been placed in the top-level name space, no long form available/needed/useful.

Examples:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

```
> Basmonom1:=&t(e1,e2,e3);
  Basmonom2:=&t(e1,e1we2);
  Monom1:=exp(I*phi)*&t(e1,e2we3,e4);
  Monom2:=-a*&t(e3we4,e1,e2);
  Polynom:=&t(Monom1+Monom2+Basmonom2);
```

$$Basmonom1 := \&t(e1, e2, e3)$$

$$Basmonom2 := e1 \ \&t \ e1we2$$

$$Monom1 := \mathbf{e}^{(\phi\, I)} \ \&t(e1, e2we3, e4)$$

$$Monom2 := -a \ \&t(e3we4, e1, e2)$$

$$Polynom := \mathbf{e}^{(\phi\, I)} \ \&t(e1, e2we3, e4) - a \ \&t(e3we4, e1, e2) + (e1 \ \&t \ e1we2)$$

```
> type(Basmonom1,tensorbasmonom),
  type(Basmonom1,tensormonom),
  type(Basmonom1,tensorpolynom);
```

$$true, true, true$$

```
> type(Basmonom2,tensorbasmonom),
  type(Basmonom2,tensormonom),
  type(Basmonom2,tensorpolynom);
```

$$true, true, true$$

```
> type(Monom2,tensorbasmonom),
  type(Monom2,tensormonom),
  type(Monom2,tensorpolynom);
```

$$false, true, true$$

```
> type(Polynom,tensorbasmonom),
  type(Polynom,tensormonom),
  type(Polynom,tensorpolynom);
```

$$false, false, true$$

However, be careful with the infix form of &(ampersand) operators, see define, tensor product.

```
> type(a*e2 &t b*e3,tensorbasmonom),
  type(a*e2 &t b*e3,tensormonom);  ## NOTE: infix is buggy, use
  parentheses!!
  eval(a*e2 &t b*e3);  ## second tensor slot is not properly
  treated, see e3!!
  eval((a*e1) &t (b*e3)), &t(a*e2,b*e3); ## works out correctly.
```

$$false, true$$

$$a\ b\ (e2\ \&t\ 1)\ e3$$

$$a\ b\ (e1\ \&t\ e3), a\ b\ (e2\ \&t\ e3)$$

```
> add(a[i]*&t(seq(e||j,j=1..i)),i=1..4);
  type(%,tensorpolynom);
```

$$a_1\ \&t(e1) + a_2\ (e1\ \&t\ e2) + a_3\ \&t(e1, e2, e3) + a_4\ \&t(e1, e2, e3, e4)$$

$$true$$

```
> remove(type,a*&t(e1,e2),tensorbasmonom);
```

$$a$$

```
> select(type,a*&t(e1,e2),tensorbasmonom);
```

$$e1\ \&t\ e2$$

But be careful about this:

```
> select(type,a*&t(e1,e2)+b*&t(e3,e4),tensorbasmonom);
```

$$0$$

```
> printf("The worksheet took %f seconds to compute on Intel
  Pentium M 2.13 GHz 2GB RAM WinXP Prof",time()-bench);
The worksheet took 0.046000 seconds to compute on Intel Pentium M 2.13 GHz 2GB
```

```
RAM WinXP Prof
>
```

See Also: Bigebra:-`&t`, Bigebra:-define, Bigebra:-help

Last modified: December 20, 2007 /BF/RA.

Function: Bigebra:-tsolve1 - solves *n*->1 tangle equations for endomorphisms

Calling Sequence:

- lst := tsolve1(eq,vars,param)

Parameters:

- eq : an expression which is subjected to be zero
- vars : variables to be solved for
- param : parameters occurring in eq but not to be solved for

Output:

- lst : a list of solution sets (if any exists, otherwise an empty list)

Description:

- The n->1 tangle solver `tsolve1` is an extension of clisolve which solves Clifford polynomial equations to n->1 tangle equations. n->1 tangles have n inputs and one output. But we do not seek for solutions in the input/output space /\V but in End /\V, therefore the name 'tsolve1'.
- A detailed analysis of the problem to solve in n->1 mappings for endomorphisms on shows, that the parameters play the role of 'co-vectors' to get a sufficient amount of equations.
- The tsolve1 facility is most effectively used with mapop and linop to seek for endomorphisms fulfilling certain n->1 tangle relations.
- If one is interested in convolution algebras n(=1)->1 tangle equations are the generic case.
- Be careful to think properly about the variables and parameters!

Examples:

```
> restart:bench:=time():with(Clifford):with(Bigebra):
  dim_V:=2:
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
```

Example 1: We will show, how to find the convolutive unit of a Grassmann Hopf algebra using tsolve1. The tangle equation is

(*) wedge (F &t U) Δ (x) = F(x) = wedge (U &t F) Δ(x)

for arbitrary F. The problem is to find the operator U. We compute in dim_V= 2, dim /\V = 2^2=4, hence U, F can be represented by 4 times 4 matrices.

```
> matU:=linalg[matrix](2^dim_V,2^dim_V,(i,j)->U[i,j]);
  matF:=linalg[matrix](2^dim_V,2^dim_V,(i,j)->F[i,j]);
```

```
U:=proc(x) linop(x,U) end:
F:=proc(x) linop(x,F) end:
bas:=cbasis(dim_V):
X:=add(_x[i]*bas[i],i=1..2^dim_V);
```

$$matU := \begin{bmatrix} U_{1,1} & U_{1,2} & U_{1,3} & U_{1,4} \\ U_{2,1} & U_{2,2} & U_{2,3} & U_{2,4} \\ U_{3,1} & U_{3,2} & U_{3,3} & U_{3,4} \\ U_{4,1} & U_{4,2} & U_{4,3} & U_{4,4} \end{bmatrix}$$

$$matF := \begin{bmatrix} F_{1,1} & F_{1,2} & F_{1,3} & F_{1,4} \\ F_{2,1} & F_{2,2} & F_{2,3} & F_{2,4} \\ F_{3,1} & F_{3,2} & F_{3,3} & F_{3,4} \\ F_{4,1} & F_{4,2} & F_{4,3} & F_{4,4} \end{bmatrix}$$

$$X := \_x_1\, Id + \_x_2\, e1 + \_x_3\, e2 + \_x_4\, e1we2$$

X is an arbitrary element of /\V on which the endomorphisms act. First of all, we compute the LHS of eq (*) as expression1:

```
> exp1:=clicollect(drop_t(&map(mapop(tcollect(mapop(tcollect(&gco
  (X)),2,U)),1,F),1,wedge)));
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$\begin{aligned}
exp1 := {} & (_x_2\, U_{1,1} F_{1,2} - _x_4\, U_{1,2} F_{1,3} + _x_3\, U_{1,1} F_{1,3} + _x_1\, U_{1,1} F_{1,1} + _x_4\, U_{1,3} F_{1,2} \\
& + _x_4\, U_{1,1} F_{1,4} + _x_2\, U_{1,2} F_{1,1} + _x_4\, U_{1,4} F_{1,1} + _x_3\, U_{1,3} F_{1,1})\, Id + (_x_4\, U_{4,1} F_{1,4} \\
& + _x_3\, U_{4,3} F_{1,1} - _x_2\, U_{2,2} F_{3,1} + _x_4\, U_{1,4} F_{4,1} + _x_4\, U_{1,1} F_{4,4} - _x_4\, U_{1,2} F_{4,3} + _x_2\, U_{4,2} F_{1,1} \\
& - _x_4\, U_{4,2} F_{1,3} + _x_1\, U_{3,1} F_{2,1} + _x_4\, U_{3,3} F_{2,2} + _x_3\, U_{3,3} F_{2,1} + _x_4\, U_{3,4} F_{2,1} - _x_2\, U_{2,1} F_{3,2} \\
& + _x_2\, U_{3,2} F_{2,1} + _x_2\, U_{1,1} F_{4,2} + _x_2\, U_{1,2} F_{4,1} - _x_1\, U_{2,1} F_{3,1} - _x_4\, U_{2,1} F_{3,4} + _x_2\, U_{3,1} F_{2,2} \\
& - _x_4\, U_{2,4} F_{3,1} - _x_3\, U_{2,3} F_{3,1} + _x_3\, U_{4,1} F_{1,3} - _x_4\, U_{3,2} F_{2,3} + _x_3\, U_{3,1} F_{2,3} - _x_4\, U_{2,3} F_{3,2} \\
& + _x_3\, U_{1,3} F_{4,1} + _x_4\, U_{2,2} F_{3,3} + _x_4\, U_{1,3} F_{4,2} + _x_4\, U_{3,1} F_{2,4} + _x_1\, U_{1,1} F_{4,1} + _x_4\, U_{4,3} F_{1,2} \\
& - _x_3\, U_{2,1} F_{3,3} + _x_4\, U_{4,4} F_{1,1} + _x_2\, U_{4,1} F_{1,2} + _x_3\, U_{1,1} F_{4,3} + _x_1\, U_{4,1} F_{1,1})\, e1we2 + (\\
& _x_3\, U_{1,3} F_{2,1} + _x_3\, U_{2,3} F_{1,1} + _x_2\, U_{2,1} F_{1,2} - _x_4\, U_{2,2} F_{1,3} + _x_4\, U_{2,4} F_{1,1} + _x_3\, U_{1,1} F_{2,3} \\
& + _x_2\, U_{1,1} F_{2,2} + _x_1\, U_{1,1} F_{2,1} + _x_4\, U_{2,3} F_{1,2} + _x_3\, U_{2,1} F_{1,3} - _x_4\, U_{1,2} F_{2,3} + _x_4\, U_{1,4} F_{2,1} \\
& + _x_2\, U_{1,2} F_{2,1} + _x_4\, U_{2,1} F_{1,4} + _x_4\, U_{1,3} F_{2,2} + _x_4\, U_{1,1} F_{2,4} + _x_1\, U_{2,1} F_{1,1} + _x_2\, U_{2,2} F_{1,1})\, e1 + (-_x_4\, U_{3,2} F_{1,3} + _x_4\, U_{3,3} F_{1,2} + _x_4\, U_{3,1} F_{1,4} + _x_1\, U_{1,1} F_{3,1} + _x_3\, U_{1,1} F_{3,3} \\
& + _x_2\, U_{1,2} F_{3,1} + _x_4\, U_{3,4} F_{1,1} + _x_4\, U_{1,1} F_{3,4} + _x_2\, U_{1,1} F_{3,2} + _x_2\, U_{3,1} F_{1,2} - _x_4\, U_{1,2} F_{3,3} \\
& + _x_4\, U_{1,4} F_{3,1} + _x_3\, U_{1,3} F_{3,1} + _x_2\, U_{3,2} F_{1,1} + _x_1\, U_{3,1} F_{1,1} + _x_3\, U_{3,1} F_{1,3} + _x_3\, U_{3,3} F_{1,1} \\
& + _x_4\, U_{1,3} F_{3,2})\, e2
\end{aligned}$$

The middle term of equation (*) is expr2=F(x)

```
> exp2:=eval(eval(F(X)));
```

$$\exp2 := (\_x_2 F_{1,2} + \_x_3 F_{1,3} + \_x_1 F_{1,1} + \_x_4 F_{1,4})\, Id + (\_x_4 F_{4,4} + \_x_2 F_{4,2} + \_x_1 F_{4,1} + \_x_3 F_{4,3})\, e1we2 + (\_x_3 F_{2,3} + \_x_2 F_{2,2} + \_x_1 F_{2,1} + \_x_4 F_{2,4})\, e1 + (\_x_1 F_{3,1} + \_x_3 F_{3,3} + \_x_4 F_{3,4} + \_x_2 F_{3,2})\, e2$$

Now we can search for a solution to this equation varying the U[i,j] alone, i.e. having an arbitrary operator F and an arbitrary element X.

```
> sol1:=tsolve1(exp1-exp2,[seq(seq(U[i,j],i=1..2^dim_V),j=1..2^di
  m_V)],

  [seq(_x[i],i=1..2^dim_V),seq(seq(F[i,j],i=1..2^dim_V),j=1..2^di
  m_V)]);
```

$$sol1 := [\{U_{1,1} = 1, U_{1,2} = 0, U_{1,3} = 0, U_{1,4} = 0, U_{2,1} = 0, U_{2,2} = 0, U_{2,3} = 0, U_{2,4} = 0, U_{3,1} = 0, U_{3,2} = 0, U_{3,3} = 0, U_{3,4} = 0, U_{4,1} = 0, U_{4,2} = 0, U_{4,3} = 0, U_{4,4} = 0\}]$$

This yields the matrix representation of U as:

```
> matU:=subs(sol1[1],evalm(matU));
```

$$matU := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

However, equation (*) has a RHS and is thus overdetermined. We have to check that our solution is also a solution of the second equality in (*). This is done as follows:

```
> exp3:=clicollect(drop_t(&map(mapop(tcollect(subs(sol1[1],mapop(
  tcollect(&gco(X)),2,U))),1,F),1,wedge)));
  printf("The second equality is %s\n",evalb(exp2=exp3));
```

$$\exp3 := (\_x_2 F_{1,2} + \_x_3 F_{1,3} + \_x_1 F_{1,1} + \_x_4 F_{1,4})\, Id + (\_x_4 F_{4,4} + \_x_2 F_{4,2} + \_x_1 F_{4,1} + \_x_3 F_{4,3})\, e1we2 + (\_x_3 F_{2,3} + \_x_2 F_{2,2} + \_x_1 F_{2,1} + \_x_4 F_{2,4})\, e1 + (\_x_1 F_{3,1} + \_x_3 F_{3,3} + \_x_4 F_{3,4} + \_x_2 F_{3,2})\, e2$$

```
The second equality is true
```

This computation showed that we have a unique convolution unit in the Grassmann Hopf gebra over /\V, dim V=2. This allows to ask for the antipode of this algebra.

Example 2: We compute the antipode of a Grassmann Hopf gebra over /\V, dim V = 2 (continuing example 1). The antipode axioms read

(**) wedge (S &t Id) Δ (x) = U(x) = wedge (Id &t S) Δ(x)

where U is the convolutive unit as computed above and we have to solve for S.

First we define and compute the LHS and middle term, but suppress the output for brevity:

```
> matS:=linalg[matrix](4,4,(i,j)->S[i,j]):
  S:=proc(x) linop(x,S) end:
  exp4:=clicollect(drop_t(&map(tcollect(mapop(tcollect(&gco(X)),1
  ,S)),1,wedge))):
  exp5:=linop(X,matU);
  # Note that U = \eta \epsilon in Hopf algebraic terms,
  # i.e. display_id@scalarpart in terms of CLIFFORD
```

$$\text{exp5} := \_x_1\, Id$$

And solve for the antipode S in terms of its matrix representation matS:

```
> sol2:=tsolve1(exp4-exp5,[seq(seq(S[i,j],i=1..4),j=1..4)],[seq(_
  x[i],i=1..4)]);
  matS=subs(sol2[1],evalm(matS));
```

$$sol2 := [\{S_{1,1}=1, S_{1,2}=0, S_{1,3}=0, S_{1,4}=0, S_{2,1}=0, S_{2,3}=0, S_{3,1}=0, S_{3,2}=0, S_{4,1}=0, S_{2,2}=-1, S_{2,4}=0, S_{3,3}=-1, S_{3,4}=0, S_{4,2}=0, S_{4,3}=0, S_{4,4}=1\}]$$

$$matS = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note that the Grassmann Hopf antipode is exactly the grade involution on /\V (if dim V=2, but this can be proved algebraically for arbitrary dim V):

```
> map(gradeinv,bas);
  subs(sol2[1],map(S,bas));
```

$$[Id, -e1, -e2, e1we2]$$

$$[Id, -e1, -e2, e1we2]$$

Once more we should test that the second equality in (**) is fulfilled, which might be done by the reader!

Example 3:

We want to exemplify the tsolve1 facility to prove that in a Clifford Hopf gebra (over dim V=2) there exists no right(/left) integrals!

The definition of an integral h in a Hopf algebra is as follows:

(***) $\quad (\mathrm{Id} \,\&t\, h)\, \Delta(x) = \eta\, h(x)$

where η is the algebra unit. Note that Δ is the Clifford co-product here. Which we have to initialize using the co-scalarproduct BI.

```
> BI:=linalg[matrix](2,2,[u,z,t,v]);unprotect(`type/clipolynom`):

  dim_V;make_BI_Id():
```

$$BI := \begin{bmatrix} u & z \\ t & v \end{bmatrix}$$

2

First, we define h as a 'co-vector' (**NOTE**: we use the same symbols for the co-vector basis elements!). Then we compute now the LHS on (***) as follows:

```
> h:=add(_h[i]*bas[i],i=1..4);
  exp6:=clicollect(simplify(drop_t(contract(&t(&cco(X,1),h),2,EV)
  )));
```

$$h := \_h_1\, Id + \_h_2\, e1 + \_h_3\, e2 + \_h_4\, e1we2$$

$$\begin{aligned} \text{exp6} := {} & (\_x_2\,\_h_2 + \_x_3\,\_h_3 + \_x_4\,\_h_4 + \_x_1\,\_h_1)\, Id - (\_x_1\, v\, u\,\_h_4 - \_x_2\, v\,\_h_3 + \_x_3\, u\,\_h_2 \\ & - \_x_4\,\_h_1 + \_x_3\, z\,\_h_3 - \_x_2\, t\,\_h_2 + \_x_4\, z\,\_h_4 - \_x_1\, t\, z\,\_h_4 - \_x_4\, t\,\_h_4)\, e1we2 \\ & - (-\_x_4\,\_h_3 - \_x_1\, u\,\_h_2 - \_x_2\,\_h_1 - \_x_3\, u\,\_h_4 - \_x_1\, z\,\_h_3 + \_x_2\, z\,\_h_4)\, e1 \\ & - (-\_x_3\,\_h_1 - \_x_3\, t\,\_h_4 + \_x_2\, v\,\_h_4 - \_x_1\, t\,\_h_2 + \_x_4\,\_h_2 - \_x_1\, v\,\_h_3)\, e2 \end{aligned}$$

The RHS of (***) is computed as:

```
> exp7:=displayid(contract(&t(X,h),1,EV));
```

$$\text{exp7} := (\_x_2\,\_h_2 + \_x_3\,\_h_3 + \_x_4\,\_h_4 + \_x_1\,\_h_1)\, Id$$

And we can solve for h using tsolve1 (Note that we have to add the parameters of the co-scalarproduct BI which are present in expr6 and expr7.)

```
> sol3:=tsolve1(exp6-exp7,[seq(_h[i],i=1..4)],[seq(_x[i],i=1..4),
  u,z,t,v]);
```

$$sol3 := [\{\_h_3 = 0, \_h_1 = 0, \_h_2 = 0, \_h_4 = 0\}]$$

Hence this shows, that **there are no non zero integrals in a Clifford convolution for arbitrary co-scalarproduct.** However, we can ask, if there are co-scalarprodcts which allow an integral to exits. In fact we know that a Grassmann Hopf gebra has a non zero left and right integral. To answer this question, we have to put parameters of the co-scalarproduct into the variables and not the parameters of tsolve1:

```
> sol4:=tsolve1(exp6-exp7,[seq(_h[i],i=1..4),u,z,t,v],[seq(_x[i],
  i=1..4)]);
  select_sol:=()-> if
  1=nops(select(has,map(evalb,[op(sol4[1])]),true)) then
            sol4[1] else sol4[2] fi:
  new_sol:=select_sol();
```

$$\begin{aligned} sol4 := [& \{\_h_3 = 0, \_h_1 = 0, \_h_2 = 0, v = 0, u = 0, \_h_4 = \_h_4, t = 0, z = 0\}, \\ & \{\_h_3 = 0, \_h_1 = 0, \_h_2 = 0, \_h_4 = 0, v = v, z = z, u = u, t = t\}] \end{aligned}$$

$$new\_sol := \{\_h_3 = 0, \_h_1 = 0, \_h_2 = 0, v = 0, u = 0, \_h_4 = \_h_4, t = 0, z = 0\}$$

One of these solutions (since Maple arranges solutions at random every time the worksheet is executed we had to pick the right one) is that of sol3, but we found a second, called new_sol, which provides a non-trivial integral, however **for a non zero integral to exist the co-scalarproduct, assigned to BI, has to vanish identically!**

If we set h(x) = int_H[x], its action is given in the next expression, the co-scalarproduct vanishes.

```
> int_H[x]:=contract(&t(X,subs(new_sol,h)),1,EV);
  int_BI:=subs(new_sol,evalm(BI));
```

$$int\_H_x := \_x_4 \, \_h_4$$

$$int\_BI := \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Note that this is closely related to the bracket which we needed to define the meet (&v vee product). However the integral is a linear form (multi-co-vector) and obtains this result in a much clearer way. Hence the integral h(X) of X can be computed as follows:

```
> bracket(X)*bracket(h);
```

$$\_x_4 \, \_h_4$$

```
> printf("The worksheet took %f seconds to compute on Intel
  Pentium M 2.13 GHz 2GB RAM WinXP Prof",time()-bench);
The worksheet took 1.876000 seconds to compute on Intel Pentium M 2.13 GHz 2GB
RAM WinXP Prof
>
```

If the integral is normalized to 1, i.e. _h[4] =1, then the bracket is exactly the value of the integral. In a projective setting the normalization is not needed and only _h[4] <> 0 has to be asserted.

See Also: Bigebra:-linop, Bigebra:-tcollect, Bigebra:-contract, Bigebra:-`&map`, Bigebra:-EV, Bigebra:-`&gco`, Bigebra:-`&cco`, Bigebra help page, Clifford intro

Last modified: December 20, 2007 /BF/RA.

Help For:

++

CLIFFORD - A Maple 11 Package for Clifford Algebras with "Bigebra"

(Version 10 with environmental variables given by CLIFFORD_ENV())

Last revised: December 20, 2007 (Source file: clifford_M11_08.mws)

(*) Department of Mathematics, Box 5054

Tennessee Technological University, Cookeville, TN 38505

tel: USA (931) 372-3569, fax: USA (931) 372-6353

rablamowicz@tntech.edu

http://math.tntech.edu/rafal/Cliff8/

($) Universit"at Konstanz, Fachbereich Physik, Fach M678

78457 Konstanz, Germany

Bertfried.Fauser@uni-konstanz.de

http://kaluza.physik.uni-konstanz.de/~fauser/

If you are a Clifford algebra pro, assign 'true' to '_prolevel' and see

how much faster your computations will be! But watch your syntax!

Use 'useproduct' to change value of _default_Clifford_product in Cl(B) from

cmulRS when B is symbolic to cmulNUM when B is numeric. Type ?cmul for help.

Type CLIFFORD_ENV() to see current values of environmental variables.

++++++++++++This is CLIFFORD version 11++++++++++++

Calling Sequence:

function(args)

Clifford:-function](args)

Description:

- This package allows the user to perform a variety of algebraic computations in the Clifford algebra Cl(V,B) on a real vector space V endowed with a bilinear form B. The dimension of V is must be between 1 and 9, while the bilinear form B may be undefined, numeric, symbolic, symmetric, or antisymmetric. In fact, in most computations the form B doesn't even have to be assigned in which case coefficients of B appear in the Maple output. To use any function from the package, load the package using the command with(Clifford).
- Upon loading the package, default dimension of V is set to 9 (maximum) by assigning value of 9 to the global variable dim_V in the initialization procedure Clifford:-setup. Then, computations can be immediately performed in Cl(V,B) where B is unspecified and dim(V) = 9. This and other environmental variables can be displayed by the procedure Clifford:-CLIFFORD_ENV. User can change the value of dim_V by assigning a square matrix to B of size n by n, in which case dim_V is assigned value of n and 'CLIFFORD' performs computations in Cl(V,B) assuming that dim(V) = n. Another way to change value of dim_V is to simply assign a positive integer p to dim_V such that 1 <= p <= 9. The value of dim_V is used to determine whether an index larger than dim_V has been entered by the user with parsing done by a procedure Clifford:-cliparse. Also, when symbolic or mixed, symbolic and numeric, indices are used in Grassmann basis monomials, procedure wedge, which computes the wedge product, suppresses monomial terms in the output of grade higher than dim_V.
- In this version, user can select one of two procedures that compute Clifford product of any two multivectors p1, p2 in Cl(B): cmulRS and cmulNUM. Both procedures require that an additional parameter K of type 'name', 'symbol' that is passed on to them from cmul as in cmul(p1,p2,K). Then, the contraction is internally computed with respect to this parameter. Procedure 'cmulRS' uses Rota-Stein cliffordization and is faster for symbolic B. Procedure 'cmulNUM' uses Chevalley's recursive definition of the Clifford product in Cl(B) and is faster when B is numeric. In addition, user can supply his/her own procedure in place of these two under a generic name cmul_user_defined. The current choice is displayed by Clifford:-CLIFFORD_ENV and is stored in a global variable _default_Clifford_product. See also procedure Clifford:-useproduct which is used to select the internal Clifford product.
- In comparison with release 4, several procedures in this release have been revised to achieve greater speed. Some have obtained new optional features, and some have been revised to save memory. In addition, a new package 'Bigebra' has been added to supplement 'CLIFFORD'. Other supplementary packages that extend abilities of 'CLIFFORD' are 'Cliplus', 'GTP', and 'Octonion'.
- 'CLIFFORD' performs computations in Cl(V,B) using Grassmann basis monomials written as Id, e1, e1we2, e1we3we5, etc., with Id denoting the identity element in Cl(V,B). Thus, the fundamental types include Clifford:-`type/cliscalar`, Clifford:-`type/clibasmon`, Clifford:-`type/climon' and Clifford:-`type/clipolynom`. Computations in Clifford basis with Clifford monomials written as Id, e1, e2, e1 &C e2, &C(e1,e2,e3), etc. can be performed upon loading the supplementary package 'Cliplus'. Note that the indices of basis monomials range from 1 to 9; symbolic indices as strings of length 1 are also allowed, and can be mixed with numeric indices.

- Clifford product in Cl(V,B) is given by the procedure Clifford:-cmul, while the wedge product is given by the procedure Clifford:-wedge. For example, to multiply two (Grassmann) monomials or polynomials in Cl(V,B) enter cmul(p1,p2) or p1 &c p2, where p1, p2 are of one of these types: Clifford:-`type/clibasmon`, Clifford:-`type/climon'] or Clifford:-`type/clipolynom`. Likewise, the wedge product is entered as wedge(p1,p2) or p1 &w p2. Multiple inputs are also allowed, e.g., cmul(p1,p2,p3,p4), or, &c(p1,p2,p3,p4), or, wedge(p1,p2,p3,p4), or &w(p1,p2,p3,p4). It is also possible to compute with Clifford matrices of Clifford:-`type/climatrix` and apply various products to the matrix elements. See procedure Clifford:-rmulm for more help.
- 'CLIFFORD' uses Claude Chevalley's definition of the Clifford product or one based on Rota-Stein cliffordization technique. For more information, see page for Clifford:-cmul. All computations performed internally by 'CLIFFORD' are purely symbolic, that is, they do not involve any matrices. While matrices in the regular or spinor representations of Cl(V,B) could be used, 'CLIFFORD' performs operations on strings such as e1we2, e1we3we4, etc. that is, on Grassmann monomials whose type is Clifford:-`type/clibasmon`.
- For more information on a particular 'function' see its help page. In a Maple worksheet, type ?Clifford,function, or, ?function, at the Maple prompt. It is also possible to access all help pages for 'CLIFFORD' via the Maple browser by typing ?Clifford and following links in the browser.
- This version includes a global variable _prolevel which is set to its default value of 'false' by the 'CLIFFORD' initialization file Clifford:-setup. When _prolevel=false, all type checking is done and user's input is parsed for errors. This can take considerable time though. User can set _prolevel to 'true' in which case procedure 'cliparse' (see Clifford:-cliparse) always returns 'true' and no parsing of user input is done.
- There are additional packages available below which extend the 'CLIFFORD' package. They need to be loaded separately and each requires that 'CLIFFORD' be loaded first. Help pages for these packages are available also. For example:

  - 'Cliplus' is a collection of additional procedures; it allows, for example, to use a Clifford basis instead of a Grassmann basis,
  - 'Bigebra' is a collection of procedures to compute with coproducts,
  - 'GTP' - Graded Tensor Package allows for certain computations in graded tensor products of Clifford algebras,
  - 'Octonion' allows for computations with octonions considered as para-vectors in Cl(0,7),

This is how to load the main package and the supplementary packages:

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

[*&m*, *Bsignature*, *CLIFFORD_ENV*, *Kfield*, *LC*, *LCQ*, *RC*, *RCQ*, *RHnumber*, *adfmatrix*, *all_sigs*, *beta_minus*, *beta_plus*, *buildm*, *bygrade*, *c_conjug*, *cbasis*, *cdfmatrix*, *cexp*, *cexpQ*, *cinv*, *clibilinear*, *clicollect*, *clidata*, *clilinear*, *climinpoly*, *cliparse*, *cliremove*, *clisolve*, *clisort*, *cliterms*, *cmul*, *cmulNUM*, *cmulQ*, *cmulRS*, *cmulgen*, *cocycle*, *commutingelements*,

conjugation, *ddfmatrix*, *diagonalize*, *displayid*, *extract*, *factoridempotent*, *find1str*, *findbasis*, *gradeinv*, *init*, *isVahlenmatrix*, *isproduct*, *makealiases*, *makeclibasmon*, *matKrepr*, *maxgrade*, *maxindex*, *mdfmatrix*, *minimalideal*, *ord*, *permsign*, *pseudodet*, *q_conjug*, *qdisplay*, *qinv*, *qmul*, *qnorm*, *rd_clibasmon*, *rd_climon*, *rd_clipolynom*, *reorder*, *reversion*, *rmulm*, *rot3d*, *scalarpart* *sexp*, *specify_constants*, *spinorKbasis*, *spinorKrepr*, *squaremodf*, *subs_clipolynom*, *useproduct*, *vectorpart*, *version*, *wedge*, *wexp*]

```
> version();
```

++

CLIFFORD - A Maple 11 Package for Clifford Algebras with "Bigebra"

(Version 10 with environmental variables given by CLIFFORD_ENV())

Last revised: December 20, 2007 (Source file: clifford_M11_08.mws)

(*) Department of Mathematics, Box 5054

Tennessee Technological University, Cookeville, TN 38505

tel: USA (931) 372-3569, fax: USA (931) 372-6353

rablamowicz@tntech.edu

http://math.tntech.edu/rafal/Cliff8/

($) Universit"at Konstanz, Fachbereich Physik, Fach M678

78457 Konstanz, Germany

Bertfried.Fauser@uni-konstanz.de

http://kaluza.physik.uni-konstanz.de/~fauser/

If you are a Clifford algebra pro, assign 'true' to '_prolevel' and see

how much faster your computations will be! But watch your syntax!

Use 'useproduct' to change value of _default_Clifford_product in Cl(B) from

cmulRS when B is symbolic to cmulNUM when B is numeric. Type ?cmul for help.

Type CLIFFORD_ENV() to see current values of environmental variables.

++++++++++++This is CLIFFORD version 11++++++++++++

To load 'CLIFFORD' and the 'GTP' package type:

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

[*cmulB*, *gbasis*, *gcollect*, *gprod*, *grade*, *gradedprod*, *tensorrank*]

To load 'CLIFFORD' and the 'Octonion' package, type:

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

[Φ, *associator*, *commutator*, *def_omultable*, *o_conjug*, *oinv*, *omul*, *omultable*, *onorm*, *oversion*, *purevectorpart*, *realpart*]

To load 'CLIFFORD' and the 'Cliplus' package, type:

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

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

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

To load 'CLIFFORD' and the 'Bigebra' package, type:

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

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

[*&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*]

```
> CLIFFORD_ENV();
```

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

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

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

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

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

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

>

Listing of available procedures in 'CLIFFORD':

&c - infix form of Clifford product
&cm - multiplication of Clifford matrices with the Clifford product &c applied to matrix entries
&cQm - multiplication of Clifford matrices with the Clifford product &cQ applied to matrix entries
&om - multiplication of Clifford matrices with the octonionic product omul (ampersand form &om) applied to matrix entries
&q - quaternionic multiplication
&qm - multiplication of quaternionic matrices with the quaternion product &q applied to matrix entries
&rm - multiplication of Clifford matrices with a user-defined product &r applied to matrix entries
&r - a possibly non-commutative user-defined product applied to entries in Cl(B)
&w - infix form of the exterior product
&wm - multiplication of matrices with the wedge product &w applied to matrix entries
Bsignature - to display signature of the current form Q
CLIFFORD_ENV - to display current values of environmental variables in the worksheet
Kfield - to find a field K of the spinor representation of Cl(Q)
LC - left contraction in Cl(B)
LCQ - left contraction in CL(Q)
RC - right contraction in Cl(B)
RCQ - right contraction in Cl(Q)
RHnumber - Radon-Hurwitz function
adfmatrix - add matrices over double fields
all_sigs - display all signatures of Clifford algebras
beta_minus - scalar product in S = Cl(Q)f related to reversion
beta_plus - scalar product in S = Cl(Q)f related to conjugation
buildm - finding a left-regular representation

bygrade - sorting by grade
c_conjug - complex conjugation in complexified Cl(B)
cbasis - create a canonical (Grassmann) basis in Cl(B)
cdfmatrix - create a matrix over a double field from two matrices in a list
cexp - Clifford exponentiation in Cl(B)
cexpQ - Clifford exponentiation in Cl(Q)
cinv - inverse in Clifford algebra Cl(B)
clibilinear - making a user-defined procedure bilinear or multilinear
clicollect - collecting with respect to Clifford basis monomials
clidata - to find basis information about Cl(Q) for the given signature
clilinear - making a user-defined procedure linear
climinpoly - finding minimal polynomial of any Clifford polynomial
cliparse - checking user input for syntax errors
cliremove - deleting some parts of a string (internal use mostly)
clisolve - solves equations in Cl(V,B)
clisort - sorting with respect to Clifford basis monomials
cliterms - finding Clifford basis monomials in the given polynomial
cmul - Clifford multiplication in Cl(B) with ampersand form &c
cmulgen - generic (dummy) Clifford product
cmulNUM - Clifford multiplication of two multivectors using Chevalley's recursive definition
cmulQ - Clifford multiplication in Cl(Q) with ampersand form &cQ
cmulRS - Clifford multiplication of two multivectors using Rota-Stein cliffordization technique
cmul_user_defined - dummy (user defined) Clifford product
cocycle - finding element x in Cl(V,B) that connects two given polynomials
commutingelements - finding commuting basis monomials that square to 1
conjugation - conjugation in Cl(B)
convert/dfmatrix - converts a list of two matrices into a matrix with entries in a double field
convert/mlist - converting matrix entries to a list
convert/str_to_int - converting numerical strings to digits
ddfmatrix - decompose a matrix over a double field into a list of two matrices
diagonalize - procedure to diagonalize a matrix
displayid - display algebra unit as Id in the given polynomial
extract - extract indices from the given basis monomial (mostly internal use)
factoridempotent - factor an idempotent into a product of simple idempotents
find1str - finds locations of string of length one in another string (mostly internal use)
findbasis - finding a basis in a linear span
gradeinv - grade involution in Cl(B)
intro - this help page
isproduct - checking if the given polynomial is a product of 1-vectors
isVahlenmatrix - procedure checking if a 2 x 2 matrix is a Vahlen matrix
makealiases - alias basis monomials for faster entry

makeclibasmon - makes a Grassmann monomial with the given indices
matKrepr - matrix representation of Cl(Q) using pre-stored matrices
maxgrade - finding the largest grade in the given polynomial
maxindex - finding the largest index in the given polynomial
mdfmatrix - multiply two matrices over a double field
minimalideal - finding a real basis in a left or right minimal ideal
ord - finding location of indices in a monomial string (internal use)
permsign - computes sign of a permutation
pseudodet - computing a pseudodeterminant of a matrix
q_conjug - quaternion conjugation
qdisplay - displaying quaternion in the standard basis {1, qi, qj, qk}
qinv - quaternionic inverse
qmul - quaternionic multiplication with the infix form &q
qnorm - quaternionic norm
rd_clibasmon - random Grassmann basis monomial
rd_climon - random Grassmann monomial
rd_clipolynom - random Grassmann polynomial
reorder - reorder monomial indices in a polynomial or monomial
reversion - reversion in Cl(B)
rmulm - various ways to multiply matrices with entries in Cl(B)
rot3d- rotation in 3 dimensions with quaternions
scalarpart - finding the scalar part in a polynomial
setup - initialization procedure for the 'CLIFFORD' package
sexp - exponentiation in Cl(B) modulo minimal polynomial
specify_constants - to specify new constants by the user
spinorKbasis - finding a basis in a minimal (spinor) ideal over a field K
spinorKrepr - finding a matrix over K in spinor representation of Cl(Q)
squaremodf - computing the square of an element in an ideal Cl(Q)f or fCl(Q) modulo f
subs_clipolynom - substitution of a Clifford polynomial into a polynomial in one variable
useproduct - procedure that allows user to switch between these Clifford products: cmulRS, cmulNUM, cmulgen, or cmul_user_defined
vectorpart - finding a k-vector part in a polynomial
version - display current version of the package and copyrights
wedge - wedge/exterior multiplication in Cl(B)
wexp - wedge/exterior exponentiation in Cl(B)

- To see code of any of these procedures, follow the example below.

Types in 'CLIFFORD':

type/antisymmatrix - an antisymmetric matrix
type/clibasmon - a basis monomial in Cl(B)

type/climatrix - matrix of with entries in Cl(B)
type/climon - a monomial in Cl(B)
type/clipolynom - a polynomial in Cl(B)
type/cliprod - basis monomial in a Clifford basis, used in the 'Cliplus' package described below
type/cliscalar - a scalar in Cl(B)
type/dfmatrix - matrix over a double field
type/diagmatrix - a diagonal matrix
type/evenelement - even element in Cl(B)
type/fieldelement - a basis element in Cl(Q)f or fCl(Q) which is in some field K
type/gencomplex - a generalized complex number
type/genquatbasis - a generalized quaternion basis
type/genquaternion - a generalized quaternion
type/idempotent - an idempotent element in Cl(B)
type/nilpotent - a nilpotent element in Cl(B)
type/oddelement - an odd element in Cl(B)
type/primitiveidemp - a primitive idempotent in Cl(B)
type/purequatbasis - a basis for pure quaternions
type/quaternion - quaternion type
type/symmatrix - a symmetric matrix
type/tensorprod - place holders of tensor products

'GTP' - Graded Tensor Product of Clifford Algebras

Procedures in 'GTP':

cmulB - Clifford product in Cl(B) when B is specified as the third argument (may be also used in the package 'CLIFFORD' upon reading in the 'GTP' package)
gbasis - standard basis in a graded tensor product
gcollect - collection or terms in a graded tensor product
gprod - product of basis monomials in a graded tensor product
grade - grade of an element in a graded tensor product
gradedprod - product of polynomials in a graded tensor product
setup - initialization procedure for 'GTP' (no help page)
tensorrank - gives rank of a tensor
&t - tensor product

Types in 'GTP':

type/gradedeven - even element in GTP
type/gradedodd - odd element in GTP
type/gradedmonom - basis monomial in GTP
type/gradedpolynom - a polynomial in GTP

'Octonion' - Package for Computations with Octonions

Procedures in 'Octonion':

Phi - associative 3-form
&om - octonionic multiplication applied to matrix entries (procedure from 'CLIFFORD')
associator - associator of three octonions
commutator - commutator of two octonions
def_omultable - procedure to define an octonion multiplication table different than a default one
o_conjug - octonionic conjugation
oinv - octonionic inverse
omul - octonionic non-associative multiplication with infix form `&o`
omultable - procedure that displays current octonionic multiplication table
onorm - octonionic norm
oversion - displays current version of the package
purevectorpart - pure vector part of an octonion
realpart - real part of an octonion
setup - initialization procedure for 'Octonion'

Types in 'Octonion':

type/Fano_triples - type Fano triples [needed to define octonionic multiplication using the Fano plane concepts]
type/octonion - type 'octonion'

'Cliplus' - Package that extends certain functions from 'CLIFFORD'

Procedures in 'Cliplus':

&dw - dotted wedge
LCbig - procedure that extends left contraction LC from 'CLIFFORD'
RCbig - procedure that extends right contraction procedure RC from 'CLIFFORD'
clibasis - procedure that defines a Clifford basis consisting of Clifford monomials, e.g., e1 &C e2, &C(e1,e2,e3), etc.
clieval - procedure that converts polynomials in Cl(B) from the Clifford basis to the Grassmann basis
cliexpand - procedure that converts polynomials in Cl(B) from the Grassmann basis to the Clifford basis
climul - procedure that extends Clifford product cmul to polynomials expressed in terms of the Clifford basis
clirev - procedure that extends reversion to polynomials expressed in terms of the Clifford basis
convert/dwedge_to_wedge - procedure that converts from the undotted wedge basis to the dotted wedge basis

convert/wedge_to_dwedge - procedure that converts from the dotted wedge basis to the undotted wedge basis
dottedcbasis - procedure that returns a dotted wedge basis for Cl(B)
dwedge - dotted wedge procedure
makeclialiases - defines aliases for Clifford basis monomials, for example, e1 &C e2 will be aliased as e12, etc.
setup - initialization procedure for 'Cliplus'

'Bigebra' - Package for computations with products and co-prodocts:

Procedures in 'Bigebra':

&cco - Clifford co-product
&gco - Grassmann co-product
&gco_d - Grassmann *co-product w.r.t. the dotted wedge* product *computed in the undotted basis*!
&gco_pl -- Grassmann-Plücker co-product acting on hyperplanes in Plücker coordinatization
&map - maps a product of adjacent slots onto a tensor polynomial
&v - the vee (meet) product
EV - the evaluation map
VERSION - displays version of 'Bigebra'
bracket - the bracket of Peano space
contract - contraction of adjacent slots in tensor
define - partially patched version of Maple' original 'define' facility
drop_t - drops the tensor symbol from tensors of rank one
gantipode - the antipode map for Grassmann Hopf algebra
gco_unit - Grassmann co-unit
gswitch - graded switch of tensor slots
help - the main help page for 'Bigebra'
linop - defines a linear operator on /\ V
linop2 - defines a linear operator on /\V x /\V
lists2mat - computes a matrix representation from the action of an operator O : V^ ---> V^ given on two lists of source and target elements
lists2mat2 - same functionality as above, but for operators O2 : V^ &t V^ ---> V^ &t V^.

make_BI_Id - initialize Clifford co-product
mapop - maps a linear operator from End /\V onto tensor slots
mapop2 - maps a linear operator from End (/\ V &t /\ V) onto two tensor slots
meet - the meet (vee) product
op2mat - computes a matrix representation of an operator O : V^ ---> V^ given as function.
op2mat2 - same as op2mat but for operators O2 : V^ &t V^ ---> V^ &t V^.
pairing - computes pairing w.r.t. a bilinear form B

peek - picks elements from tensor slots
poke - puts elements into tensor slots
remove_eq - helper function that removes tautologies from equation sets
switch - switch tensor slots
tcollect - collects coefficients of a tensor polynomial
tsolve1 - solves n --> 1 equations for endomorphisms

Types in 'Bigebra':

type/tensorbasmonom - new type: tensor basis monomial
type/tensormonom - new type: tensor monomial
type/tensorpolynom - new type: tensor polynomial

Sample code:

```
> restart:with(Clifford):interface(verboseproc=2):
> print(`&c`);
proc()
local NP, ARGS, coB, nameB, lname, decindex, flagdec;
option `Copyright (c) 1995-2008 by Rafal Ablamowicz and Bertfried Fauser. All rights re\
served.`;
description `Last revised: December 20, 2007`;
    flagdec := true;
    if type(op(procname), procedure) then
        if type([args], listlist) then
            if type(op(args), array) then
                WARNING("enclose index in double quotes as in &c["B"] or &c["-B"\
                    '] when B has been assigned a matrix to avoid the following:");
                return 'procname(args)'
            end if
        else coB := 1; nameB := B; lname := B; ARGS := [args]; flagdec := false
        end if
    else
        lname := op(procname);
        ARGS := [args];
        if type(lname, `&*`(numeric, name)) then
            coB := op(select(type, {op(lname)}, numeric));
            nameB := op(select(type, {op(lname)}, name))
        else coB := 1; nameB := lname
        end if;
```

```
            flagdec := false
        end if;
        decindex := proc()
            local ARGS, coB, nameB;
            global B;
                if type([ args ], listlist) then
                    if type( op( args ), function ) then
                        ARGS := op( op( args ));
                        coB := 1;
                        nameB := eval( op( 0, op( args ) ));
                        if type( nameB, `&*`( numeric, name ) ) then
                            coB := op( select( type, { op( nameB ) }, numeric ));
                            nameB := op( select( type, { op( nameB ) }, name ))
                        end if
                    elif type( op( args ), `&*`( numeric, function ) ) then
                        nameB := { op( op( args ) ) };
                        coB := op( select( type, nameB, numeric ));
                        nameB := op( select( type, nameB, function ));
                        ARGS := op( nameB );
                        nameB := op( 0, nameB )
                    else error "unable to determine index or wrong index, use name in d\
                        ouble quotes as in &c["B"] or &c["-B"]"
                    end if
                elif type( [ args ], list ) then ARGS := args; coB := 1; nameB := B
                else error "cannot determine arguments and/or index from arguments"
                end if;
                return coB, nameB, [ ARGS ]
            end proc;
        if flagdec then coB, nameB, ARGS := decindex( args ); lname := coB*nameB end if;
        NP := nops( ARGS );
        if member( 0, ARGS ) then return 0 end if;
        if NP ≤ 1 then return op( ARGS ) end if;
        return cmul[ eval( lname ) ]( op( ARGS ) )
    end proc
> 
>
```

See Also: Clifford:-makealiases, Clifford:-setup, Clifford:-cbasis, Clifford:-version

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-wexp - exterior exponential in the Clifford algebra Cl(B)

Calling Sequence:

wexp(p,k);

Parameters:

p - an expression of type 'cliscalar' or 'clipolynom' or 'numeric'
k - a non-negative integer

Description:

- Procedure 'wexp' computes the Clifford exponential of a Clifford polynomial p in Cl(B) up to the order specified by the second argument k which is expected to be a nonnegative integer.
- If k = 0 then the procedure returns 1 or 'Id' depending whether p is of type 'cliscalar'. See `type/cliscalar` and `type/clipolynom` for more help on these basic types.
- Use 'cexp' to compute the Clifford exponential in Cl(B) and 'cexpQ' to compute the Clifford exponential in Cl(Q). See cexp, cexpQ and sexp for more help.
- Note that one may exponentiate not only polynomials but also polynomials times some parameter.
- It is not necessary that the form Q (or B) be defined.
- Use 'clicollect' to collect terms with respect to Clifford monomials. See clicollect for more help.

Examples:

```
> restart:with(Clifford):
> wexp(e1we2*t,3):clicollect(%);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$Id + t\ e1we2$$

```
> p:=e1we2+e1we3+e2we3: #define p
> wexp(p,0);
```

$$1$$

```
> wexp(p,2);
```

$$e1we3 + e2we3 + e1we2 + 1$$

```
> wexp(p*t,0);
```

$$1$$

```
> wexp((e1we2+e3we4)*s,3):clicollect(%);
```

$$Id + s\ e1we2 + s\ e3we4 + s^2\ e1we2we3we4$$

```
> wexp(2*alpha,0);
```

$$1$$

```
> wexp(2*alpha,7);
```

$$1 + 2\,\alpha + 2\,\alpha^2 + \frac{4}{3}\,\alpha^3 + \frac{2}{3}\,\alpha^4 + \frac{4}{15}\,\alpha^5 + \frac{4}{45}\,\alpha^6 + \frac{8}{315}\,\alpha^7$$

```
> cexp(2*alpha,7);
```

$$1 + 2\,\alpha + 2\,\alpha^2 + \frac{4}{3}\,\alpha^3 + \frac{2}{3}\,\alpha^4 + \frac{4}{15}\,\alpha^5 + \frac{4}{45}\,\alpha^6 + \frac{8}{315}\,\alpha^7$$

```
> cexpQ(2*alpha,7);
```

$$1 + 2\,\alpha + 2\,\alpha^2 + \frac{4}{3}\,\alpha^3 + \frac{2}{3}\,\alpha^4 + \frac{4}{15}\,\alpha^5 + \frac{4}{45}\,\alpha^6 + \frac{8}{315}\,\alpha^7$$

```
> wexp(.6,100);
```

$$1.822118801$$

```
>
```

See Also: Clifford:-sexp, Clifford:-cexpQ, Clifford:-cexp, Clifford:-clicollect, Clifford:-`type/clipolynom`

Last modified: December 20, 2007, RA/BF.

Function: `&c`, `&cQ`, `&w`, `&q`, `&cm`, `&cQm`, `&om`, `&wm`, `&qm`, `rm`, `&C`

Calling Sequence:

&c(p1,p2,...pn) or p1 &c p2 &c ... &c pn
&c[K](p1,p2,...pn)
&C(p1,p2,...pn) or p1 &C p2 &C ... &C pn
&C[K](p1,p2,...pn)
&cQ(p1,p2,...pn) or p1 &cQ p2 &cQ ... &cQ pn
&cQ[K](p1,p2,...pn)
&w(p1,p2,...pn) or p1 &w p2 &w ... &w pn
&q(q1,q2,...qn) or q1 &w q2 &w ... &w qn

&cm(M1,M2,...,Mn) or M1 &cm M2 &cm ... &cm Mn
&cm[K](M1,M2,...,Mn)
&cQm(M1,M2,...,Mn) or M1 &cQm M2 &cQm ... &cQm Mn
&cQm[K](M1,M2,...,Mn)
&om(o1,o2,...,on) (octonionic multiplication of matrices - requires package 'Octonion')
&wm(M1,M2,...,Mn) or M1 &wm M2 &wm ... &wm Mn
&qm(Q1,Q2,...,Qn) or Q1 &qm Q2 &qm ... &qm Qn
&rm(M1,M2,...,Mn) or M1 &rm M2 &rm ... &rm Mn
&rm[K](M1,M2,...,Mn)

Parameters:

p1, p2, ..., pn - expressions of the type 'cliscalar', 'clibasmon', 'climon', or 'clipolynom'
q1, q2, ..., qn - expressions of the type 'quaternion'
o1, o2, ..., on - expressions of type 'octonion' (see package 'Octonion' for more information)
M1, M2, ..., Mn - matrices of the type 'climatrix'
Q1, Q2, ..., Qn - matrices of the type 'climatrix' with quaternionic entries, that is, entries of type 'quaternion'
K - (optional) bilinear form of type `name`, `symbol`, `matrix`, `array`, or `&*`(numeric,{name,symbol,array,matrix})

Description:

- CLIFFORD defines the following infix forms upon loading:

- &c is the infix form for Clifford multiplication 'cmul' in Cl(B) (see cmul for more help),

- &C is the inert form for Clifford multiplication 'cmul' in Cl(B) (see `type/cliprod` for more help),

- &cQ is the infix form for Clifford multiplication 'cmulQ' in Cl(Q) (see cmulQ for more help),

- &w is the infix form for wedge (exterior) multiplication 'wedge' in Cl(B) (see wedge for more help),

- &q is the infix form for quaternionic multiplication 'qmul' in Cl([1,1,1]) (see qmul for more help),

- &cm is the infix form for multiplication of Clifford matrices, that is, matrices with entries in Cl(B) when Clifford multiplication 'cmul' is applied to matrix entries (see `type/climatrix` and rmulm for more help),
- &cQm is the infix form for multiplication of Clifford matrices, that is, matrices with entries in Cl(Q) when Clifford multiplication 'cmulQ' is applied to matrix entries (see `type/climatrix` and rmulm for more help),
- &wm is the infix form for multiplication of Clifford matrices, that is, matrices with entries in Cl(B) when wedge (exterior) multiplication 'wedge' is applied to matrix entries (see `type/climatrix` and rmulm for more help),
- &qm is the infix form for multiplication of Clifford matrices, that is, matrices with entries in Cl(B) of type 'quaternion' when quaternionic multiplication 'qmul' is applied to matrix entries (see quaternion, qmul, `type/climatrix` and rmulm for more help),
- &om is the infix form for multiplication of Clifford matrices, that is, matrices with entries in Cl(B) of type 'octonion' when octonionic multiplication 'omul' is applied to matrix entries (see octonion, omul, `type/climatrix` and rmulm for more help),
- &rm is the infix form for multiplication of Clifford matrices, that is, matrices with entries in Cl(B) (matrices of type 'climatrix') when when some generic, yet-to-be defined, multiplication 'r' is applied to matrix entries (see `type/climatrix` and rmulm for more help),

- The infix forms `&c`, `&cQ`, `&cm`, `&rm`,and `&C` can accept an optional index of type `name`, `symbol`, `matrix`, `array`, or `&*`(numeric,{name,symbol,array,matrix}) as in, for example, &c[K](p1,p2,...,pn), &c[-K](p1,p2,...,pn).
- NOTE: When index K has been assigned a matrix, double quotes must be used around K to stop premature evaluation of the matrix by Maple as in &c["K"](p1,p2,...,pn), &c["-K"](p1,p2,...,pn), &c["2*K"](p1,p2,...,pn), &c["-2*K"](p1,p2,...,pn). See examples below.

Examples:

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

Example 1: There are two different ways to use the infix form:

A. Infix form of 'cmul':

```
> cmul(1+e1we2+2*e3,e2+3*e4);      #long form in Cl(B)
  cmul[B](1+e1we2+2*e3,e2+3*e4);   #long form in Cl(B)
  cmul[K](1+e1we2+2*e3,e2+3*e4);   #long form in Cl(K)
  cmul[-K](1+e1we2+2*e3,e2+3*e4);  #long form in Cl(-K)
```

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

$$2\,(B_{3,2}+3\,B_{3,4})\,Id-2\,e2we3-(-1+B_{1,2}+3\,B_{1,4})\,e2+3\,e4+(B_{2,2}+3\,B_{2,4})\,e1 + 3\,e1we2we4+6\,e3we4$$

$$2\,(B_{3,2}+3\,B_{3,4})\,Id-2\,e2we3-(-1+B_{1,2}+3\,B_{1,4})\,e2+3\,e4+(B_{2,2}+3\,B_{2,4})\,e1 + 3\,e1we2we4+6\,e3we4$$

$$2\,(K_{3,2}+3\,K_{3,4})\,Id-2\,e2we3-(-1+K_{1,2}+3\,K_{1,4})\,e2+3\,e4+(K_{2,2}+3\,K_{2,4})\,e1 + 3\,e1we2we4+6\,e3we4$$

$$-2\,(K_{3,2}+3\,K_{3,4})\,Id-2\,e2we3+(1+K_{1,2}+3\,K_{1,4})\,e2+3\,e4-(K_{2,2}+3\,K_{2,4})\,e1 + 3\,e1we2we4+6\,e3we4$$

```
> &c(1+e1we2+2*e3,e2+3*e4);     #infix form for cmul in Cl(B)
  (1+e1we2+2*e3) &c (e2+3*e4); #infix form for cmul in Cl(B)
  &c[K](1+e1we2+2*e3,e2+3*e4); #infix form for cmul in Cl(K)
  &c[-K](1+e1we2+2*e3,e2+3*e4);#infix form for cmul in Cl(-K)
```

$$2\,(B_{3,2}+3\,B_{3,4})\,Id-2\,e2we3-(-1+B_{1,2}+3\,B_{1,4})\,e2+3\,e4+(B_{2,2}+3\,B_{2,4})\,e1 + 3\,e1we2we4+6\,e3we4$$

$$2\,(B_{3,2}+3\,B_{3,4})\,Id-2\,e2we3-(-1+B_{1,2}+3\,B_{1,4})\,e2+3\,e4+(B_{2,2}+3\,B_{2,4})\,e1 + 3\,e1we2we4+6\,e3we4$$

$$2\,(K_{3,2}+3\,K_{3,4})\,Id-2\,e2we3-(-1+K_{1,2}+3\,K_{1,4})\,e2+3\,e4+(K_{2,2}+3\,K_{2,4})\,e1 + 3\,e1we2we4+6\,e3we4$$

$$-2\,(K_{3,2}+3\,K_{3,4})\,Id-2\,e2we3+(1+K_{1,2}+3\,K_{1,4})\,e2+3\,e4-(K_{2,2}+3\,K_{2,4})\,e1 + 3\,e1we2we4+6\,e3we4$$

K can be a matrix::

```
> K:=linalg[diag](1$4):
> &c[''K''](1+e1we2+2*e3,e2+3*e4); #infix form for cmul in Cl(K)
```

$$-2\,e2we3+e2+3\,e4+e1+3\,e1we2we4+6\,e3we4$$

```
> &c['K'](1+e1we2+2*e3,e2+3*e4); #<<<- Intended error, infix form
  for cmul in Cl(K) / single quotes are not enough
Error, (in Clifford:-cliparse) check spelling of sparse or define it as a
constant or an alias
> &c[K](1+e1we2+2*e3,e2+3*e4); #<<<- Intended error,infix form
  for cmul in Cl(K)
Error, (in Clifford:-cliparse) check spelling of sparse or define it as a
constant or an alias
> &c[''-K''](1+e1we2+2*e3,e2+3*e4); #infix form for cmul in Cl(K)
```

$$-2\,e2we3+e2+3\,e4-e1+3\,e1we2we4+6\,e3we4$$

```
> &c[''-2*K''](1+e1we2+2*e3,e2+3*e4); #infix form for cmul in
  Cl(K)
```

$$-2\,e2we3+e2+3\,e4-2\,e1+3\,e1we2we4+6\,e3we4$$

```
> K:='K': #unassigning K for next examples
> cmul(e1,e2-e5,e3-2*e1);cmul[K](e1,e2-e5,e3-2*e1);cmul[-K](e1,e2
  -e5,e3-2*e1);
```

$$(B_{1,2}-B_{1,5})\,e3-(-2\,B_{1,1}+B_{1,3})\,e2-(B_{5,3}-2\,B_{1,5}+2\,B_{2,1}-B_{2,3}+2\,B_{1,2}-2\,B_{5,1})\,e1 + (-2\,B_{1,1}+B_{1,3})\,e5+e1we2we3+e1we3we5$$

$$(K_{1,2}-K_{1,5})\,e3-(-2\,K_{1,1}+K_{1,3})\,e2-(K_{5,3}-2\,K_{1,5}+2\,K_{2,1}-K_{2,3}+2\,K_{1,2}-2\,K_{5,1})\,e1 + (-2\,K_{1,1}+K_{1,3})\,e5+e1we2we3+e1we3we5$$

$$-(K_{1,2}-K_{1,5})\,e3+(-2\,K_{1,1}+K_{1,3})\,e2+(K_{5,3}-2\,K_{1,5}+2\,K_{2,1}-K_{2,3}+2\,K_{1,2}-2\,K_{5,1})\,e1 - (-2\,K_{1,1}+K_{1,3})\,e5+e1we2we3+e1we3we5$$

```
> &c(e1,e2,e3);
```

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

```
> cmul[B](e1,e2,e3);
```

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

```
> &c[''B''](e1+3*e3,e2-e3,e4);
```

$$(3\,B_{2,4}+B_{1,4})\,e3-(3\,B_{3,4}+B_{1,4})\,e2+(B_{1,2}-3\,B_{3,3}+3\,B_{3,2}-B_{1,3})\,e4+(B_{2,4}-B_{3,4})\,e1 + e1we2we4-e1we3we4-3\,e2we3we4$$

```
> &c[''B''](e1,e2);
```

$$B_{1,2}\,Id+e1we2$$

```
> &c[''B''](e1+3*e3,e2-e3,e4);
```

$$(3\,B_{2,4}+B_{1,4})\,e3-(3\,B_{3,4}+B_{1,4})\,e2+(B_{1,2}-3\,B_{3,3}+3\,B_{3,2}-B_{1,3})\,e4+(B_{2,4}-B_{3,4})\,e1 + e1we2we4-e1we3we4-3\,e2we3we4$$

```
> e1 &c e2;
```

$$B_{1,2}\,Id+e1we2$$

```
> (e1+e2) &c (e3-2*e4);&c[K]((e1+e2),(e3-2*e4));
```

$$-(-B_{1,3}-B_{2,3}+2\,B_{1,4}+2\,B_{2,4})\,Id+e1we3+e2we3-2\,e2we4-2\,e1we4$$

$$-(-K_{1,3}-K_{2,3}+2\,K_{1,4}+2\,K_{2,4})\,Id+e1we3+e2we3-2\,e2we4-2\,e1we4$$

```
> (e1+e2) &c (e3-2*e4) &c (2*e9);&c[K]((e1+e2),(e3-2*e4),2*e9);
```

$$-2\,(B_{2,9}+B_{1,9})\,e3-2\,(-B_{3,9}+2\,B_{4,9})\,e2+4\,(B_{2,9}+B_{1,9})\,e4-2\,(-B_{3,9}+2\,B_{4,9})\,e1 + 2\,e1we3we9+2\,e2we3we9-4\,e2we4we9-4\,e1we4we9 - 2\,(-B_{1,3}-B_{2,3}+2\,B_{1,4}+2\,B_{2,4})\,e9$$

$$-2\,(K_{2,9}+K_{1,9})\,e3+2\,(K_{3,9}-2\,K_{4,9})\,e2+4\,(K_{2,9}+K_{1,9})\,e4+2\,(K_{3,9}-2\,K_{4,9})\,e1 + 2\,e1we3we9+2\,e2we3we9-4\,e2we4we9-4\,e1we4we9 - 2\,(-K_{1,3}-K_{2,3}+2\,K_{1,4}+2\,K_{2,4})\,e9$$

B. Infix form of 'cmulQ':

```
> B:=linalg[diag](1$9):
> cmulQ(e1+e2,e1-2*e4); #long form
  cmulQ[B](e1+e2,e1-2*e4); #long form
  cmulQ[K](e1+e2,e1-2*e4); #optional parameter in cmulQ
```

```
cmulQ[-K](e1+e2,e1-2*e4); #optional parameter in cmulQ
```

$$Id - e1we2 - 2\,e2we4 - 2\,e1we4$$

$$Id - e1we2 - 2\,e2we4 - 2\,e1we4$$

$$-(-K_{1,1} - K_{2,1} + 2\,K_{1,4} + 2\,K_{2,4})\,Id - e1we2 - 2\,e2we4 - 2\,e1we4$$

$$(-K_{1,1} - K_{2,1} + 2\,K_{1,4} + 2\,K_{2,4})\,Id - e1we2 - 2\,e2we4 - 2\,e1we4$$

```
> &cQ(e1+e2,e1-2*e4); #infix form for cmulQ
  &cQ[''B''](e1+e2,e1-2*e4); #infix form for cmulQ
  (e1+e2) &cQ (e1-2*e4); #infix form for cmulQ
  &cQ[K](e1+e2,e1-2*e4); #infix form for cmulQ
  &cQ[-K](e1+e2,e1-2*e4); #infix form for cmulQ
```

$$Id - e1we2 - 2\,e2we4 - 2\,e1we4$$

$$Id - e1we2 - 2\,e2we4 - 2\,e1we4$$

$$Id - e1we2 - 2\,e2we4 - 2\,e1we4$$

$$-(-K_{1,1} - K_{2,1} + 2\,K_{1,4} + 2\,K_{2,4})\,Id - e1we2 - 2\,e2we4 - 2\,e1we4$$

$$(-K_{1,1} - K_{2,1} + 2\,K_{1,4} + 2\,K_{2,4})\,Id - e1we2 - 2\,e2we4 - 2\,e1we4$$

Also in `&cQ` double quotes must be used around a name used as index if that name has been assigned a matrix:

```
> K:=linalg[diag](1$4):
> &cQ[K](e1+e2,e1-2*e4); #<<<Intended error, infix form for cmulQ
Error, (in Clifford:-cliparse) check spelling of sparse or define it as a
constant or an alias
```

```
> &cQ['K'](e1+e2,e1-2*e4); #<<< Intended error, infix form for
  cmulQ / single quotes are not enough
Error, (in Clifford:-cliparse) check spelling of sparse or define it as a
constant or an alias
```

```
> &cQ[''K''](e1+e2,e1-2*e4); #infix form for cmulQ / double
  quotes are needed
```

$$Id - e1we2 - 2\,e2we4 - 2\,e1we4$$

```
> e1 &cQ e1;
```

$$Id$$

C. Infix form of 'wedge':

```
> wedge(e1+e2,e3-2*e4); #long form
```

$$e1we3 + e2we3 - 2\,e1we4 - 2\,e2we4$$

```
> &w(e1+e2,e3-2*e4); #short form for wedge
```

$$e1we3 + e2we3 - 2\,e1we4 - 2\,e2we4$$

```
> (e1+e2) &w (e3-2*e4);
```

$$e1we3 + e2we3 - 2\,e1we4 - 2\,e2we4$$

There is a dwedge procedure in 'Cliplus' package for computation of dotted wedge.

D. Infix form of 'qmul'. Defining B to be 3 x 3 diagonal matrix so that we could use quaternions:

```
> B:=linalg[diag](1,1,1):
> q1:=2+2*e1we2+e1we3+e2we3;q2:=-2*e2we3+3+e1we2;
```

$$q1 := 2 + 2\ e1we2 + e1we3 + e2we3$$

$$q2 := -2\ e2we3 + 3 + e1we2$$

```
> type(q1,quaternion),type(q2,quaternion);
```

$$true, true$$

```
> qmul(q1,q2); # long form of the quaternionic product
  &q(q1,q2); # short form of the quaternionic product
  q1 &q q2; # short form of the quaternionic product
```

$$6 - 10\ qk - 2\ qj$$

$$6 - 10\ qk - 2\ qj$$

$$6 - 10\ qk - 2\ qj$$

This is how Maple will display quaternions if you use 'qdisplay':

```
> _quatbasis;
```

$$[[Id, e3we2, e1we3, e2we1], \{Maple\ has\ assigned\ qi:=-e2we3,\ qj:=e1we3,\ qk:=-e1we2\}]$$

```
> q1:=qdisplay(q1);q2:=qdisplay(q2);
```

$$q1 := 2 - 2\ qk + qj - qi$$

$$q2 := 3 - qk + 2\ qi$$

```
> type(q1,quaternion);type(q2,quaternion);
```

$$true$$

$$true$$

Quaternion multiplication is done with 'qmul' or '&q':

```
> qmul(q1,q2);
  &q(q1,q2);
```

$$6 - 10\ qk - 2\ qj$$

$$6 - 10\ qk - 2\ qj$$

Let's find the multiplication table for H:

```
> Q:=[1,'qi','qj','qk'];
```

$$Q := [1, qi, qj, qk]$$

```
> A := array(1..4,1..4):
  for i from 1 to 4 do for j from 1 to 4 do A[i,j]:=Q[i] &q Q[j]
  od od;
> print(A);
```

$$\begin{bmatrix} 1 & qi & qj & qk \\ qi & -1 & qk & -qj \\ qj & -qk & -1 & qi \\ qk & qj & -qi & -1 \end{bmatrix}$$

E. Infix form `&rm` for user-defined product `&r` when applied to matrices:

```
> M1:=matrix(2,2,[e1+e2,e3-3,e2+4,Id]);
```

$$M1 := \begin{bmatrix} e1+e2 & e3-3 \\ e2+4 & Id \end{bmatrix}$$

```
> type(M1,climatrix);
```

$$true$$

```
> &rm(M1,M1); #implictly Clifford product is computed in Cl(B)
```

$$\begin{bmatrix} \&r_B(e1+e2, e1+e2) + \&r_B(e3-3, e2+4) & \&r_B(e1+e2, e3-3) + \&r_B(e3-3, 1) \\ \&r_B(e2+4, e1+e2) + \&r_B(1, e2+4) & \&r_B(e2+4, e3-3) + \&r_B(1, 1) \end{bmatrix}$$

```
> evalm(B);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Since B has been assigned a matrix, we must use double quotes as follows:

```
> &rm[''-B''](M1,M1); #explicitly Clifford product is computed in
  Cl(B)
```

$$\begin{bmatrix} \&r_{-B}(e1+e2, e1+e2) + \&r_{-B}(e3-3, e2+4) & \&r_{-B}(e1+e2, e3-3) + \&r_{-B}(e3-3, 1) \\ \&r_{-B}(e2+4, e1+e2) + \&r_{-B}(1, e2+4) & \&r_{-B}(e2+4, e3-3) + \&r_{-B}(1, 1) \end{bmatrix}$$

If double quotes are not used, an error message will result:

```
> &rm['B'](M1,M1); #<<< Intended error: explicitly Clifford
  product is computed in Cl(B)
Warning, enclose index in double quotes as in &rm[''B''] or &rm[''-B''] when B
has been assigned a matrix to avoid the following:
```

$$\&\mathrm{rm}\left(\left[\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\right]\right)$$

```
> &rm['-B'](M1,M1); #<<< Intended error: single quotes are not
  enough
Error, (in decindex) unable to determine index or wrong index type for &rm, try
enclosing name of the index in double quotes as in &rm[''B''] or &rm[''-B'']
```

```
> &rm[''-B''](M1,M1); #explicitly Clifford product is computed in
  Cl(B)
```

$$\begin{bmatrix} \&r_{-B}(e1+e2, e1+e2) + \&r_{-B}(e3-3, e2+4) & \&r_{-B}(e1+e2, e3-3) + \&r_{-B}(e3-3, 1) \\ \&r_{-B}(e2+4, e1+e2) + \&r_{-B}(1, e2+4) & \&r_{-B}(e2+4, e3-3) + \&r_{-B}(1, 1) \end{bmatrix}$$

Example 2: Multiplication of matrices with entries in Cl(B) is done as follows. As an example, we define B to have both the symmetric and antisymmetric parts.

E. Infix form of the Clifford product applied to matrices with entries in a Clifford algebra.

```
> B:=matrix(3,3,[ 1, a, b,
                 -a, 1, c,
                 -b,-c, 1]);
```

$$B := \begin{bmatrix} 1 & a & b \\ -a & 1 & c \\ -b & -c & 1 \end{bmatrix}$$

```
> eval(makealiases(3)):
> M1:=matrix(2,2,[1+2*e3,e3+e2,e1we2we3,e2+2*e3]);
```

$$M1 := \begin{bmatrix} 1 + 2\,e3 & e3 + e2 \\ e123 & e2 + 2\,e3 \end{bmatrix}$$

```
> M2:=matrix(2,2,[2*e1we2,-e1+e2,2*e2we3,e2-2*e3]);
```

$$M2 := \begin{bmatrix} 2\,e12 & -e1 + e2 \\ 2\,e23 & e2 - 2\,e3 \end{bmatrix}$$

```
> type(M1,climatrix);type(M2,climatrix);
```

$$true$$

$$true$$

```
> M1 &cm M2; #Clifford product 'cmul' is applied to matrix
  entries
```

$$[2\,e12 + 4\,e123 - 4\,b\,e2 + 4\,c\,e1 - 2\,e3\,c + 2\,e3 - 2\,e2 - 2\,c\,e2\,,$$
$$-5\,c + 2\,b + 2\,e13 - 5\,e23 + e2 - e1 - 1]$$
$$[-2\,e3\,a^2 + 2\,e2\,a\,b - 4\,c\,e2 - 2\,e1\,b - 2\,e1\,c\,a - 4\,a\,e123 - 4\,e3\,c - 4\,e2\,,$$
$$-a\,e13 - e13 - 5\,e23 + a\,e23 - c\,e12 + b\,e12 - 3 - 4\,c]$$

```
> map(clicollect,%);
```

$$[2\,e12 - 2\,(c - 1)\,e3 - 2\,(2\,b + c + 1)\,e2 + 4\,c\,e1 + 4\,e123\,,$$
$$(-5\,c + 2\,b - 1)\,Id + 2\,e13 - 5\,e23 + e2 - e1]$$
$$[-2\,(a^2 + 2\,c)\,e3 + 2\,(a\,b - 2 - 2\,c)\,e2 - 2\,(b + c\,a)\,e1 - 4\,a\,e123\,,$$
$$-(3 + 4\,c)\,Id - (a + 1)\,e13 + (-5 + a)\,e23 + (-c + b)\,e12]$$

To use infix form, make sure to enclose the name of the index in double quotes if it has been assigned a matrix. For example,

```
> &cm[X](M1,M2); # No need for ''X'' since X is a name, not a
  matrix
```

$$[2\,e12 + 4\,e123 + 4\,X_{3,1}\,e2 - 4\,X_{3,2}\,e1 + 2\,X_{3,2}\,e3 + 2\,X_{2,2}\,e3 - 2\,X_{3,3}\,e2 - 2\,X_{2,3}\,e2\,,$$
$$3\,X_{3,2} - 2\,X_{3,1} + 2\,e13 - 5\,e23 + e2 - e1 - 2\,X_{2,3} + X_{2,2} - 2\,X_{3,3}]$$
$$[2\,X_{2,1}\,X_{1,2}\,e3 - 2\,X_{2,2}\,X_{1,1}\,e3 - 2\,X_{3,1}\,X_{1,2}\,e2 + 2\,X_{3,2}\,X_{1,1}\,e2 + 2\,X_{3,1}\,X_{2,2}\,e1$$
$$- 2\,X_{3,2}\,X_{2,1}\,e1 + 2\,X_{2,1}\,e123 - 2\,X_{1,2}\,e123 + 2\,X_{2,2}\,e3 + 4\,X_{3,2}\,e3 - 2\,X_{2,3}\,e2 - 4\,X_{3,3}\,e2\,,$$
$$X_{2,1}\,e13 - X_{2,2}\,e13 - X_{1,1}\,e23 + X_{1,2}\,e23 + X_{3,2}\,e12 - X_{3,1}\,e12 + X_{2,2} + 2\,X_{3,2} - 2\,X_{2,3}$$

$-4\,X_{3,3} - 4\,e23]$

```
> &cm[''B''](M1,M2); # Need for ''B'' since B is a matrix
```

$[2\,e12 + 4\,e123 - 4\,b\,e2 + 4\,c\,e1 - 2\,e3\,c + 2\,e3 - 2\,e2 - 2\,c\,e2,$
$-5\,c + 2\,b + 2\,e13 - 5\,e23 + e2 - e1 - 1]$
$[-2\,e3\,a^2 + 2\,e2\,a\,b - 4\,c\,e2 - 2\,e1\,b - 2\,e1\,c\,a - 4\,a\,e123 - 4\,e3\,c - 4\,e2,$
$-a\,e13 - e13 - 5\,e23 + a\,e23 - c\,e12 + b\,e12 - 3 - 4\,c]$

```
> &cm[''-B''](M1,M2); # Need for ''B'' since B is a matrix
```

$[2\,e12 + 4\,e123 + 4\,b\,e2 - 4\,c\,e1 + 2\,e3\,c - 2\,e3 + 2\,e2 + 2\,c\,e2,$
$5\,c - 2\,b + 2\,e13 - 5\,e23 + e2 - e1 + 1]$
$[-2\,e3\,a^2 - 4\,e3 + 2\,e2\,a\,b - 2\,e1\,b - 2\,e1\,c\,a + 4\,a\,e123 + 4\,e3\,c + 4\,e2,$
$a\,e13 + e13 - 3\,e23 - a\,e23 + c\,e12 - b\,e12 + 3 + 4\,c]$

The same applies to `&cQm`:

```
> M1 &cQm M2; #Clifford product 'cmulQ' is applied to matrix
  entries
```

$$\begin{bmatrix} 2\,e12 + 4\,e123 - 2\,e2 + 2\,e3 & -e1 + 2\,e13 + e2 - 5\,e23 - 1 \\ -4\,e2 & -5\,e23 - e13 - 3 \end{bmatrix}$$

```
> &cQm(M1,M2); #default matrix B is used implicitly
```

$$\begin{bmatrix} 2\,e12 + 4\,e123 - 2\,e2 + 2\,e3 & -e1 + 2\,e13 + e2 - 5\,e23 - 1 \\ -4\,e2 & -5\,e23 - e13 - 3 \end{bmatrix}$$

```
> &cQm[''B''](M1,M2); #correct result when ''B'' is used
```

$$\begin{bmatrix} 2\,e12 + 4\,e123 - 2\,e2 + 2\,e3 & -e1 + 2\,e13 + e2 - 5\,e23 - 1 \\ -4\,e2 & -5\,e23 - e13 - 3 \end{bmatrix}$$

```
> &cQm[''-B''](M1,M2); #correct result
```

$$\begin{bmatrix} 2\,e12 + 4\,e123 + 2\,e2 - 2\,e3 & -e1 + 2\,e13 + e2 - 5\,e23 + 1 \\ -4\,e3 + 4\,e2 & -3\,e23 + e13 + 3 \end{bmatrix}$$

```
>
```

Procedure `&wm` applies the wedge product to the matrix entries and as such it does not need an index.

```
> M1 &wm M2; #wedge (exterior) 'wedge' is applied to matrix
  entries
Warning, since B has been (re-)assigned, value of dim_V has been reduced by 'we
dge' to 3
```

$$\begin{bmatrix} 2\,e12 + 4\,e123 & -e1 + 2\,e13 + e2 - 5\,e23 \\ 0 & -4\,e23 \end{bmatrix}$$

```
> &wm(M1,M2); #wedge (exterior) 'wedge' is applied to matrix
  entries
```

$$\begin{bmatrix} 2\,e12 + 4\,e123 & -e1 + 2\,e13 + e2 - 5\,e23 \\ 0 & -4\,e23 \end{bmatrix}$$

Note: In this last example, an arbitrary, yet-to-be-defined by the user product 'r' has been applied to matrix entries.

```
> B:=linalg[diag](1,1,1):
> q1:=2+2*qi+qj+3*qk;q2:=-2*qi+3+qk;q3:=-2-2*qk+qj;q4:=-2*qj+3*qi
  +qk;
```

$$q1 := 2 - 2\,e23 + e13 - 3\,e12$$

$$q2 := 2\,e23 + 3 - e12$$

$$q3 := -2 + 2\,e12 + e13$$

$$q4 := -2\,e13 - 3\,e23 - e12$$

```
> type(q1,quaternion),type(q2,quaternion),
  type(q3,quaternion),type(q4,quaternion);
```

$$true, true, true, true$$

```
> Q1:=matrix(2,2,[q1,q2,q3,q4]);Q2:=matrix(2,2,[q2,-q1,q4,-q3]);
```

$$Q1 := \begin{bmatrix} 2 - 2\,e23 + e13 - 3\,e12 & 2\,e23 + 3 - e12 \\ -2 + 2\,e12 + e13 & -2\,e13 - 3\,e23 - e12 \end{bmatrix}$$

$$Q2 := \begin{bmatrix} 2\,e23 + 3 - e12 & -2 + 2\,e23 - e13 + 3\,e12 \\ -2\,e13 - 3\,e23 - e12 & 2 - 2\,e12 - e13 \end{bmatrix}$$

```
> type(Q1,climatrix),type(Q2,climatrix);
```

$$true, true$$

```
> Q1 &qm Q2;
```

$$\begin{bmatrix} 12 + 20\,qk - 6\,qj + 14\,qi & 14 - 2\,qk - 3\,qj - 11\,qi \\ -18 - 6\,qk + 7\,qj + 5\,qi & -5 + 11\,qk - 6\,qj + 2\,qi \end{bmatrix}$$

```
> printf("Worksheet took %f seconds to compute on Pentium M 2.13
  GHz 2GB machine with Win XP Professional\n",time()-bench);
Worksheet took 2.585000 seconds to compute on Pentium M 2.13 GHz 2GB machine wi
th Win XP Professional
>
```

See Also: GTP:-`&t`, Clifford:-rmulm, Clifford:-`type/clipolynom`, Cliplus:-dwedge, Clifford:-useproduct

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-adfmatrix, Clifford:-mdfmatrix - add or multiply matrices over double field

Calling Sequence:

adfmatrix(m1,m2) - add two matrices m1 and m2 of `type/dfmatrix` and of the same dimension n x n,
mdfmatrix(m1,m2) - multiply two matrices m1 and m2 of `type/dfmatrix` and of the same dimension n x n,

Parameters:

m1,m2 - matrices of `type/dfmatrix` of the same size

Description:

- A matrix M is of type 'ddfmatrix' if it is a square matrix whose entries are two element lists. These matrices arise when faithful matrix representations of semi-simple Clifford algebras are calculated. Use all_sigs to display signatures of semisimple Clifford algebras in dimensions 1 through 9.
- Faithful spinor representations of semisimple Clifford algebras have been precomputed. They are stored in a library file and can be retrieved with the procedure clidata.

Examples:

```
> restart:bench:=time():
  with(Clifford):with(linalg):
  _default_Clifford_product;
  #useproduct(cmulNUM);
  _default_Clifford_product;
```

$$Clifford{:}\text{-}cmulRS$$

$$Clifford{:}\text{-}cmulRS$$

Let's display signatures of semi-simple Clifford algebras in dimensions 1 through 9 which are isomorphic to rings of matrices over double real field R+R:

```
> all_sigs(1..9,'real','semisimple');
```

$$[[0, 7], [1, 0], [1, 8], [2, 1], [3, 2], [4, 3], [5, 4], [9, 0]]$$

Let's display signatures of semi-simple Clifford algebras in dimensions 1 through 9 which are isomorphic to rings of matrices over double quaternionic field H+H:

```
> all_sigs(1..9,'quat','semisimple');
```

$$[[0, 3], [1, 4], [2, 5], [3, 6], [5, 0], [6, 1], [7, 2]]$$

There are no semisimple Clifford algebras that would be isomorphic to rings of matrices of double complex field C+C:

```
> all_sigs(1..9,'complex','semisimple');
```

$$[\ ]$$

More information about Clifford algebra Cl(Q) of the quadratic form of signature (2,1) can be

found using procedure clidata as follows:

```
> clidata([2,1]);
```

$$\left[real, 2, semisimple, 'cmulQ'\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e2we3}{2}\right), [Id, e2], [Id], [Id, e2]\right]$$

Example 1: Let's view matrices m[i], i=1..3, representing 1-vectors {e1,e2,e3} in Cl(2,1) = Mat(2,2,R+R). These matrices are of type `type/dfmatrix` and have been precomputed. They can be displayed with the procedure matKrepr:

```
> pq:=[2,1];
  B:=diag(1$pq[1],-1$pq[2]);
```

$$pq := [2, 1]$$

$$B := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

```
> L:=matKrepr();
```

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

$$L := \left[e1 = \begin{bmatrix} [1, -1] & [0, 0] \\ [0, 0] & [-1, 1] \end{bmatrix}, e2 = \begin{bmatrix} [0, 0] & [1, 1] \\ [1, 1] & [0, 0] \end{bmatrix}, e3 = \begin{bmatrix} [0, 0] & [-1, -1] \\ [1, 1] & [0, 0] \end{bmatrix}\right]$$

Let's assign these matrices to m[i], i=1..3:

```
> for i from 1 to nops(L) do m[i]:=rhs(L[i]) od;
```

$$m_1 := \begin{bmatrix} [1, -1] & [0, 0] \\ [0, 0] & [-1, 1] \end{bmatrix}$$

$$m_2 := \begin{bmatrix} [0, 0] & [1, 1] \\ [1, 1] & [0, 0] \end{bmatrix}$$

$$m_3 := \begin{bmatrix} [0, 0] & [-1, -1] \\ [1, 1] & [0, 0] \end{bmatrix}$$

Observe that matrices m[1], m[2], and m[3] satisfy appropriate relations:

(A) Squares of m[1], m[2], and m[3] are:

```
> mdfmatrix(m[1],m[1]),mdfmatrix(m[2],m[2]),mdfmatrix(m[3],m[3]);
```

$$\begin{bmatrix} [1, 1] & [0, 0] \\ [0, 0] & [1, 1] \end{bmatrix}, \begin{bmatrix} [1, 1] & [0, 0] \\ [0, 0] & [1, 1] \end{bmatrix}, \begin{bmatrix} [-1, -1] & [0, 0] \\ [0, 0] & [-1, -1] \end{bmatrix}$$

(B) Pair-wise, matrices m[1], m[2], and m[3] anticommute:

```
> adfmatrix(mdfmatrix(m[1],m[2]),mdfmatrix(m[2],m[1]));
```

$$\begin{bmatrix} [0, 0] & [0, 0] \\ [0, 0] & [0, 0] \end{bmatrix}$$

```
> adfmatrix(mdfmatrix(m[1],m[3]),mdfmatrix(m[3],m[1]));
```

$$\begin{bmatrix} [0, 0] & [0, 0] \\ [0, 0] & [0, 0] \end{bmatrix}$$

```
> adfmatrix(mdfmatrix(m[2],m[3]),mdfmatrix(m[3],m[2]));
```

$$\begin{bmatrix} [0,0] & [0,0] \\ [0,0] & [0,0] \end{bmatrix}$$

Thus, we can find a unique matrix representing each basis Grassmann monomial in Cl(Q) as follows. Let's define a homomorphism f from Cl(Q) to Mat(2,2,R+R) as a Maple function:

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

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

```
> f:=proc() end:
  f(Id):=mdfmatrix(m[1],m[1]):
  f(e1):=evalm(m[1]):
  f(e2):=evalm(m[2]):
  f(e3):=evalm(m[3]):
  f(e1we2):=mdfmatrix(m[1],m[2]):
  f(e1we3):=mdfmatrix(m[1],m[3]):
  f(e2we3):=mdfmatrix(m[2],m[3]):
  f(e1we2we3):=mdfmatrix(mdfmatrix(m[1],m[2]),m[3]):
```

Then, we can apply f to the entries of clibas:

```
> for x in clibas do
  x, " is mapped by f to ", f(x);
  od;
```

$$Id, \text{" is mapped by f to "}, \begin{bmatrix} [1,1] & [0,0] \\ [0,0] & [1,1] \end{bmatrix}$$

$$e1, \text{" is mapped by f to "}, \begin{bmatrix} [1,-1] & [0,0] \\ [0,0] & [-1,1] \end{bmatrix}$$

$$e2, \text{" is mapped by f to "}, \begin{bmatrix} [0,0] & [1,1] \\ [1,1] & [0,0] \end{bmatrix}$$

$$e3, \text{" is mapped by f to "}, \begin{bmatrix} [0,0] & [-1,-1] \\ [1,1] & [0,0] \end{bmatrix}$$

$$e1we2, \text{" is mapped by f to "}, \begin{bmatrix} [0,0] & [1,-1] \\ [-1,1] & [0,0] \end{bmatrix}$$

$$e1we3, \text{" is mapped by f to "}, \begin{bmatrix} [0,0] & [-1,1] \\ [-1,1] & [0,0] \end{bmatrix}$$

$$e2we3, \text{" is mapped by f to "}, \begin{bmatrix} [1,1] & [0,0] \\ [0,0] & [-1,-1] \end{bmatrix}$$

$$e1we2we3, \text{" is mapped by f to "}, \begin{bmatrix} [1,-1] & [0,0] \\ [0,0] & [1,-1] \end{bmatrix}$$

```
>
```

Example 2: Let's view matrices m[i], i=1..3, representing 1-vectors {e1,e2,e3} in Cl(0,3) = Mat(1,1,H+H). These matrices are of type `type/dfmatrix` and have been precomputed. They can be displayed with the procedure matKrepr:

```
> pq:=[0,3];
  B:=diag(1$pq[1],-1$pq[2]);
```

$$pq := [0, 3]$$

$$B := \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

```
> L:=matKrepr();
```

$$L := [e1 = [[e1, e1]], e2 = [[e2, e2]], e3 = [[-e1we2, e1we2]]]$$

Let's assign these matrices to m[i], i=1..3:

```
> for i from 1 to nops(L) do m[i]:=rhs(L[i]) od;
```

$$m_1 := [[e1, e1]]$$

$$m_2 := [[e2, e2]]$$

$$m_3 := [[-e1we2, e1we2]]$$

```
>
```

Observe that matrices m[1], m[2], and m[3] satisfy appropriate relations:

(A) Squares of m[1], m[2], and m[3] are:

```
> mdfmatrix(m[1],m[1]),mdfmatrix(m[2],m[2]),mdfmatrix(m[3],m[3]);
```

$$[[-1, -1]], [[-1, -1]], [[-1, -1]]$$

(B) Pair-wise, matrices m[1], m[2], and m[3] anticommute:

```
> adfmatrix(mdfmatrix(m[1],m[2]),mdfmatrix(m[2],m[1]));
```

$$[[0, 0]]$$

```
> adfmatrix(mdfmatrix(m[1],m[3]),mdfmatrix(m[3],m[1]));
```

$$[[0, 0]]$$

```
> adfmatrix(mdfmatrix(m[2],m[3]),mdfmatrix(m[3],m[2]));
```

$$[[0, 0]]$$

Thus, we can find a unique matrix representing each basis Grassmann monomial in Cl(Q) as follows. Let's define a homomorphism h from Cl(Q) to Mat(1,1,H+H) as a Maple function:

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

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

```
> h:=proc() end:
  h(Id):=mdfmatrix(m[1],m[1]):
  h(e1):=evalm(m[1]):
  h(e2):=evalm(m[2]):
  h(e3):=evalm(m[3]):
  h(e1we2):=mdfmatrix(m[1],m[2]):
  h(e1we3):=mdfmatrix(m[1],m[3]):
  h(e2we3):=mdfmatrix(m[2],m[3]):
  h(e1we2we3):=mdfmatrix(mdfmatrix(m[1],m[2]),m[3]):
```

Then, we can apply f to the entries of clibas:

```
> for x in clibas do
  x, " is mapped by h to ", h(x);
```

```
od;
```

$$Id, \text{" is mapped by h to "}, [[-1, -1]]$$
$$e1, \text{" is mapped by h to "}, [[e1, e1]]$$
$$e2, \text{" is mapped by h to "}, [[e2, e2]]$$
$$e3, \text{" is mapped by h to "}, [[-e1we2, e1we2]]$$
$$e1we2, \text{" is mapped by h to "}, [[e1we2, e1we2]]$$
$$e1we3, \text{" is mapped by h to "}, [[e2, -e2]]$$
$$e2we3, \text{" is mapped by h to "}, [[-e1, e1]]$$
$$e1we2we3, \text{" is mapped by h to "}, [[1, -1]]$$

It is easy to see from the above display, that in order for f to be an isomorphism, there is a need to represent each element in the Grassmann basis by a pair of matrices, or, equivalently, by a matrix in a double fields. For example, let's split into a pair of matrices from Mat(1,1,H) double-field matrices representing e3 and e1we2 elements:

```
> ddfmatrix(h(e3));ddfmatrix(h(e1we2));
```

$$[[-e1we2], [e1we2]]$$
$$[[e1we2], [e1we2]]$$

Notice, that the second matrix [e1we2] is identical in both pairs, or, equivalently, element e1we2 is present in both matrices

```
> h(e3),h(e1we2);
```

$$[[-e1we2, e1we2]], [[e1we2, e1we2]]$$

Thus, the assignment e3 --> e1we2, e1we2 --> e1we2 would not be an isomorphism. There is a need for another element to distinguish these two images by adding a second element. That is, e3 --> [-e1we2, e1we2] and e1we2 --> [e1we2,e1we2].

It may be worth recalling that these pairs of matrices are simply representations of the given element u from Cl(Q) in two left (or right) minimal (spinor) ideals S1 and S2 generated by the primitive idempotents f and gradeinv(f) respectively where 'gradeinv' is the grade involution in Cl(Q). We can find f and generators for S1:=Cl(Q)f from 'clidata':

```
> cdata:=clidata();
```

$$cdata := \left[quaternionic, 1, semisimple, \frac{Id}{2} + \frac{e1we2we3}{2}, [Id, e1, e2, e3], [Id, e1, e2, e1we2], [Id]\right]$$

```
> f:=cdata[4];field:=cdata[6];gens:=cdata[7];
```

$$f := \frac{Id}{2} + \frac{e1we2we3}{2}$$
$$field := [Id, e1, e2, e1we2]$$
$$gens := [Id]$$

Notice that f is an idempotent:

```
> cmul(f,f)-f;
```

$$0$$

and that f is a primitive idempotent:

```
> type(f,primitiveidemp);
```

$$true$$

Notice also that elements in the list 'field' generate a subalgebra in Cl(Q) isomorphic with the ring of quaternions. Let's display the multiplication table of these elements:

```
> M:=matrix(4,4,(i,j)->cmul(field[i],field[j]));
```

$$M := \begin{bmatrix} Id & e1 & e2 & e1we2 \\ e1 & -Id & e1we2 & -e2 \\ e2 & -e1we2 & -Id & e1 \\ e1we2 & e2 & -e1 & -Id \end{bmatrix}$$

Thus, the basis in S1=Cl(Q)f over the K = gen(field) = H is just f, that is, S1=Cl(Q)f=<f>. Recall also that S1 = Cl(Q)f is a *right* K module. Thus, we have

```
> cmul(e3,f)=cmul(f,-e1we2); #representation in S1=Cl(Q)f
```

$$-\frac{e1we2}{2}+\frac{e3}{2}=-\frac{e1we2}{2}+\frac{e3}{2}$$

which means that element '-e1we2' from K represents e3 in S1. Likewise, in S2=Cl(Q)gradeinv(f). Recall also that S1 = Cl(Q)gradeinv(f) is also a *right* K module.

```
> fg:=gradeinv(f);
```

$$fg := \frac{Id}{2}-\frac{e1we2we3}{2}$$

```
> cmul(e3,fg)=cmul(fg,e1we2); #representation in S2=Cl(Q)fg
  f(e3);
```

$$\frac{e1we2}{2}+\frac{e3}{2}=\frac{e1we2}{2}+\frac{e3}{2}$$

$$\frac{1}{2}\,\mathrm{Id}(e3)+\frac{1}{2}\,\mathrm{e1we2we3}(e3)$$

Thus, the pair (-e1we2,e1we2) represents e3 in S1+S2 (here `+` denotes the direct sum), which is the same as matrix m[3] above with an entry in H+H. Let's compute the pair representing e1we2:

```
> cmul(e1we2,f)=cmul(f,e1we2); #representation in S1=Cl(Q)f
```

$$\frac{e1we2}{2}-\frac{e3}{2}=\frac{e1we2}{2}-\frac{e3}{2}$$

```
> cmul(e1we2,fg)=cmul(fg,e1we2); #representation in S2=Cl(Q)fg
```

$$\frac{e1we2}{2}+\frac{e3}{2}=\frac{e1we2}{2}+\frac{e3}{2}$$

Thus, the pair (e1we2,e1we2) represents e1we2 in S1+S2. Let's verify in a similar manner that matrices representing e1, e2, e1we3, e2we3, and e1we2we3 shown above are correct:

```
> cmul(e1,f)=cmul(f,e1);cmul(e1,fg)=cmul(fg,e1);h(e1);
```

$$-\frac{e2we3}{2}+\frac{e1}{2}=-\frac{e2we3}{2}+\frac{e1}{2}$$

$$\frac{e2we3}{2}+\frac{e1}{2}=\frac{e2we3}{2}+\frac{e1}{2}$$

$$[[e1, e1]]$$

Thus, the pair (e1,e1) represents e1 in S1+S2, which is the same as matrix m[1] above with an entry in H+H.

```
> cmul(e2,f)=cmul(f,e2);cmul(e2,fg)=cmul(fg,e2);h(e2);
```

$$\frac{e1we3}{2}+\frac{e2}{2}=\frac{e1we3}{2}+\frac{e2}{2}$$

$$-\frac{e1we3}{2}+\frac{e2}{2}=-\frac{e1we3}{2}+\frac{e2}{2}$$

$$[[e2, e2]]$$

Thus, the pair (e2,e2) represents e2 in S1+S2, which is the same as matrix m[2] above with an entry in H+H.

```
> cmul(e1we3,f)=cmul(f,e2);cmul(e1we3,fg)=cmul(fg,-e2);h(e1we3);
```

$$\frac{e1we3}{2}+\frac{e2}{2}=\frac{e1we3}{2}+\frac{e2}{2}$$

$$\frac{e1we3}{2}-\frac{e2}{2}=\frac{e1we3}{2}-\frac{e2}{2}$$

$$[[e2, -e2]]$$

Thus, the pair (e2,-e2) represents e1we3 in S1+S2.

```
> cmul(e2we3,f)=cmul(f,-e1);cmul(e2we3,fg)=cmul(fg,e1);h(e2we3);
```

$$\frac{e2we3}{2}-\frac{e1}{2}=\frac{e2we3}{2}-\frac{e1}{2}$$

$$\frac{e2we3}{2}+\frac{e1}{2}=\frac{e2we3}{2}+\frac{e1}{2}$$

$$[[-e1, e1]]$$

Thus, the pair (-e1,e1) represents e2we3 in S1+S2.

```
> cmul(e1we2we3,f)=cmul(f,1);cmul(e1we2we3,fg)=cmul(fg,-1);h(e1we
  2we3);
```

$$\frac{Id}{2}+\frac{e1we2we3}{2}=\frac{Id}{2}+\frac{e1we2we3}{2}$$

$$-\frac{Id}{2}+\frac{e1we2we3}{2}=-\frac{Id}{2}+\frac{e1we2we3}{2}$$

$$[[1, -1]]$$

Thus, the pair (1,1) represents e1we2we3 in S1+S2.

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 6.686000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
```

```
>
```

For more information how to find spinor representations in higher dimensions, see help pages for Kfield, gradeinv, marKrepr, minimalideal, spinorKbasis, spinorKrepr. See also cbasis, clidata, `type/primitiveidemp`, `type/idempotent`.

See Also: Clifford:-`type/dfmatrix`, Clifford:-ddfmatrix, Clifford:-cdfmatrix

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-all_sigs - display signatures of Clifford algebras in dimensions 1 through 9

Calling Sequence:

all_sigs(range);
all_sigs(range,s1);
all_sigs(range,s1,s2);

Parameters:

range - expression of the type 'range', for example, 1..9, 2..5, 5..5,
s1 - one of the strings 'real', 'complex', or 'quat',
s2 - one of the strings 'simple', or 'semisimple'.

Description:

- Procedure 'all_sigs' displays signatures [p,q] of the quadratic form Q in dimensions specified by the input 'range' such that the Clifford algebra Cl(Q) is isomorphic to a matrix ring over R (the real numbers), C (the complex numbers), H (the quaternions) , R + R (the double real field), or H + H (the double quaternion ring).
- Signatures are displayed in a list.

Examples:

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

We can display all signatures from the dimension 1 to the dimension 9 as follows:

```
> all_sigs(1..9);
```

[[0, 1], [0, 2], [0, 3], [0, 4], [0, 5], [0, 6], [0, 7], [0, 8], [0, 9], [1, 0], [1, 1], [1, 2], [1, 3], [1, 4], [1, 5], [1, 6], [1, 7], [1, 8], [2, 0], [2, 1], [2, 2], [2, 3], [2, 4], [2, 5], [2, 6], [2, 7], [3, 0], [3, 1], [3, 2], [3, 3], [3, 4], [3, 5], [3, 6], [4, 0], [4, 1], [4, 2], [4, 3], [4, 4], [4, 5], [5, 0], [5, 1], [5, 2], [5, 3], [5, 4], [6, 0], [6, 1], [6, 2], [6, 3], [7, 0], [7, 1], [7, 2], [8, 0], [8, 1], [9, 0]]

Signatures [p,q] of Q in dimensions 1 through 9 such the Cl(p,q) is isomorphic to a matrix ring over the reals or double reals:

```
> all_sigs(1..9,'real');
```

[[0, 6], [0, 7], [0, 8], [1, 0], [1, 1], [1, 7], [1, 8], [2, 0], [2, 1], [2, 2], [3, 1], [3, 2], [3, 3], [4, 2], [4, 3], [4, 4], [5, 3], [5, 4], [8, 0], [9, 0]]

Signatures [p,q] of Q in dimensions 1 through 9 such the Cl(p,q) is isomorphic to a simple matrix ring over the reals:

```
> all_sigs(1..9,'real','simple');
```

[[0, 6], [0, 8], [1, 1], [1, 7], [2, 0], [2, 2], [3, 1], [3, 3], [4, 2], [4, 4], [5, 3], [8, 0]]

Signatures [p,q] of Q in dimensions 1 through 9 such the Cl(p,q) is isomorphic to a semi-simple matrix ring over the double reals:

```
> all_sigs(1..9,'real','semisimple');
```

[[0, 7], [1, 0], [1, 8], [2, 1], [3, 2], [4, 3], [5, 4], [9, 0]]

In the complex case, all Clifford algebras are simple:

```
> all_sigs(1..9,'complex');
```

$$[[0, 1], [0, 5], [0, 9], [1, 2], [1, 6], [2, 3], [2, 7], [3, 0], [3, 4], [4, 1], [4, 5], [5, 2], [6, 3], [7, 0], [8, 1]]$$

```
> all_sigs(1..9,'complex','simple');
```

$$[[0, 1], [0, 5], [0, 9], [1, 2], [1, 6], [2, 3], [2, 7], [3, 0], [3, 4], [4, 1], [4, 5], [5, 2], [6, 3], [7, 0], [8, 1]]$$

```
> all_sigs(1..9,'complex','semisimple');
```

$$[\ ]$$

Here is the quaternionic case:

```
> all_sigs(1..9,'quat');
```

$$[[0, 2], [0, 3], [0, 4], [1, 3], [1, 4], [1, 5], [2, 4], [2, 5], [2, 6], [3, 5], [3, 6], [4, 0], [5, 0], [5, 1], [6, 0], [6, 1], [6, 2], [7, 1], [7, 2]]$$

```
> all_sigs(1..9,'quat','simple');
```

$$[[0, 2], [0, 4], [1, 3], [1, 5], [2, 4], [2, 6], [3, 5], [4, 0], [5, 1], [6, 0], [6, 2], [7, 1]]$$

```
> all_sigs(1..9,'quat','semisimple');
```

$$[[0, 3], [1, 4], [2, 5], [3, 6], [5, 0], [6, 1], [7, 2]]$$

```
>
```

Of course, just to find Clifford algebras in one single dimension 5, try this:
(a) Clifford algebras isomorphic to matrix rings over the reals or double reals in dimension 5:

```
> all_sigs(5..5,'real');
```

$$[[3, 2]]$$

(b) Clifford algebras isomorphic to simple matrix rings over the reals in dimension 5:

```
> all_sigs(5..5,'real','simple');
```

$$[\ ]$$

(c) Clifford algebras isomorphic to semi-simple matrix rings over double reals in dimension 5:

```
> all_sigs(5..5,'real','semisimple');
```

$$[[3, 2]]$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 0.062000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM with Win XP Professional
>
```

See Also: Clifford:-scalarpart, Clifford:-`type/clipolynom`, Clifford:-`type/cliscalar`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-`type/antisymmatrix` - define a new type: an anti-symmetric matrix

Calling Sequence:

type(m, antisymmatrix);

Parameters:

m - a name type 'matrix'

Description:

- The procedure checks if the matrix m is antisymmetric or not.
- The procedure returns 'true' or 'false' depending whether its argument is or is not of the type 'antisymmatrix'.
- See also related types 'symmatrix' (`type/symmatrix`), 'diagmatrix' (`type/diagmatrix`), and 'climatrix' (`type/climatrix`).

Examples:

```
> restart:bench:=time():with(Clifford):
> B:=linalg[diag](1,1,-1);
```

$$B := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

```
> type(B,diagmatrix);
```

$$true$$

```
> type(B,symmatrix);
```

$$true$$

```
> type(B,antisymmatrix);
```

$$false$$

```
> type(B,climatrix);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$false$$

```
> B:=linalg[matrix](2,2,[a,b,-b,c]);
```

$$B := \begin{bmatrix} a & b \\ -b & c \end{bmatrix}$$

```
> type(B,diagmatrix);
```

$$false$$

```
> type(B,symmatrix);
```

$$false$$

```
> type(B,antisymmatrix);
```

$$false$$

```
> type(B,climatrix);
```

$$false$$

```
> BA:=linalg[matrix](2,2,[0,b,-b,0]);
```

$$BA := \begin{bmatrix} 0 & b \\ -b & 0 \end{bmatrix}$$

```
> type(BA,symmatrix);
```

$$false$$

```
> type(BA,diagmatrix);
```

$$false$$

```
> type(BA,antisymmatrix);
```

$$true$$

```
> type(BA,climatrix);
```

$$false$$

```
> cliB:=linalg[matrix](2,2,[1+e1,2*e1we2-e2we3,e4,e5]);
```

$$cliB := \begin{bmatrix} 1 + e1 & 2\, e1we2 - e2we3 \\ e4 & e5 \end{bmatrix}$$

```
> type(cliB,diagmatrix);
```

$$false$$

```
> type(cliB,symmatrix);
```

$$false$$

```
> type(cliB,antisymmatrix);
```

$$false$$

```
> type(cliB,climatrix);
```

$$true$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM machine with Win XP
  Professional\n",time()-bench);
Worksheet took 0.064000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
machine with Win XP Professional
>
```

See Also: Clifford:-`type/climatrix`, Clifford:-`type/diagmatrix`, Clifford:-`type/symmatrix`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-beta_plus, Clifford:-beta_minus - computes scalar product in spinor ideals

Calling Sequence:

beta_plus(psi, phi, f, 's') - computes a scalar product of two spinors 'psi' and 'phi' in S=Cl(Q)f when one of them is subject to reversion anti-automorphism where S is a spinor ideal in Cl(Q) generated by a primitive idempotent 'f'; argument 's' is optional

beta_minus(psi, phi, f, 's') - computes a scalar product of two spinors 'psi' and 'phi' in S=Cl(Q)f when one of them is subject to conjugation anti-automorphism where S is a spinor ideal in Cl(Q) generated by a primitive idempotent 'f'; argument 's' is optional

Parameters:

psi, phi - elements in S=Cl(Q)f which are of one of the following types: `type/cliscalar`, `type/clibasmon`, `type/climon`, or `type/clipolynom`
f - element of `type/primitiveidemp`, that is, a primitive idempotent in Cl(Q)
's' (optional) - optional argument of `type/name` (it must be unevaluated name)

Description:

- These two procedures give beta_{+} and beta_{-} scalar products on S=Cl(Q)f considered as right K-modules that have been described in [1] and [4].
- The first two arguments are spinors 'psi' and 'phi' which are expected to be of type one of the types `type/cliscalar`, `type/clibasmon`, `type/climon`, or `type/clipolynom` with the most general one being the last one. Procedures reversion and conjugation realize operations of reversion and conjugation in Cl(Q) (more generally in Cl(V,B)), where Q is a quadratic form defined in V, that is, Cl(Q)=Cl(V,Q). The third argument is a primitive idempotent f in Cl(Q), that is, an element of `type/primitiveidemp'. This idempotent, as it is explained below, can be arbitrary in case when K = R, while there are some restrictions what it can be when K = C and K = H.
- The fourth optional argument 's' is of `type/name': it will be a placeholder for an invertible element s in Cl(Q) such that cmul(f, s) = cmul(s, auto(f)) for a simple Clifford algebra Cl(Q) (when the signature of Q is (p,q) and p - q <> 1 mod 4), or cmul(g, s) = cmul(g, auto(g)) for a semisimple Clifford algebra Cl(Q) (when the signature of Q is (p,q) and p - q = 1 mod 4). Here, cmul denotes Clifford product in Cl(Q), g = f + conjugation(f), and 'auto' is either 'reversion' or 'conjugation'. Invertible element s is needed so that cmul(s, auto(psi), phi) belongs to K, K^ or K + K^, depending on the chosen (anti-) automorphism and whether Cl(Q) is simple or semisimple. Here K is a subalgebra of Cl(Q) isomorphic with the reals, complex numbers, or quaternions depending on the signature of Q (for more information see [1]) and the hat in K^ denotes grade involution in Cl(Q). The grade involution is computed with the procedure gradeinv. Upon exiting, 'beta' stores the value of s under the name 's' supplied by the user. These special elements s in Cl(Q) are called 'purespinors'. It must be entered as 's', that is, as an unevaluated name. Otherwise, an error message will appear if s has already been assigned some other basis monomial as a purespinor since basis monomials are protected in 'CLIFFORD'.

- Procedure Kfield computes a basis for the ring K, while procedures spinorKbasis and spinorKrepr compute a K-basis in S = Cl(Q)f and a spinor representation of Cl(Q) in S, respectively.
- For more information on symbolic computations in Clifford algebras, see [2] and [3].
- Faithful matrix representations of simple and semi-simple Clifford algebras are stored in a library file and can be retrieved with the procedure clidata. Use procedure all_sigs to display signatures and basic structure of Clifford algebras in dimensions 1 through 9.
- **References:**

[1] R. Ablamowicz, "Spinor representations of Clifford algebras: A symbolic approach", Computer Physics Communications Thematic Issue ‘‘Computer Algebra in Physics Research’’, Vol.**115** (1998), Numbers 2--3, pages 510--535.
[2] R. Ablamowicz, P. Lounesto, and J.M. Parra, eds., "Clifford Algebras with Numeric and Symbolic Computations", Birkhauser, Boston, 1996.
[3] R. Ablamowicz and P. Lounesto, "On Clifford algebras of a bilinear form with an antisymmetric part", in: R. Ablamowicz, P. Lounesto, and J.M. Parra, eds., "On Clifford Algebras with Numeric and Symbolic Computations", Birkhauser, Boston, 1996, page 167--188.
[4] P. Lounesto, "Scalar products of spinors and an extension of Brauer-Wall groups", Foundations of Physics, Vol. **11**, Nos. 9/10 (1981) pages 721--740.

Examples:

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

Let's display signatures of simple and semi-simple Clifford algebras in dimensions 1 through 9 which are isomorphic to rings of matrices over R and over the double real field R+R, respectively:

```
> all_sigs(1..9,'real','simple');
```

$$[[0, 6], [0, 8], [1, 1], [1, 7], [2, 0], [2, 2], [3, 1], [3, 3], [4, 2], [4, 4], [5, 3], [8, 0]]$$

```
> all_sigs(1..9,'real','semisimple');
```

$$[[0, 7], [1, 0], [1, 8], [2, 1], [3, 2], [4, 3], [5, 4], [9, 0]]$$

Let's display signatures of simple Clifford algebras in dimensions 1 through 9 which are isomorphic to rings of matrices over the complex field C. There are no semi-simple algebras in this case:

```
> all_sigs(1..9,'complex','simple');
```

$$[[0, 1], [0, 5], [0, 9], [1, 2], [1, 6], [2, 3], [2, 7], [3, 0], [3, 4], [4, 1], [4, 5], [5, 2], [6, 3], [7, 0], [8, 1]]$$

```
> all_sigs(1..9,'complex','semisimple');
```

$$[\ ]$$

Let's display signatures of simple and semi-simple Clifford algebras in dimensions 1 through 9 which are isomorphic to rings of matrices over quaternions H and over the double quaternionic field H+H, respectively:

```
> all_sigs(1..9,'quat','simple');
```

$$[[0, 2], [0, 4], [1, 3], [1, 5], [2, 4], [2, 6], [3, 5], [4, 0], [5, 1], [6, 0], [6, 2], [7, 1]]$$

```
> all_sigs(1..9,'quat','semisimple');
```

$$[[0,3],[1,4],[2,5],[3,6],[5,0],[6,1],[7,2]]$$

More information about any Clifford algebra Cl(Q) of the quadratic form of signature (p,q), 1<= p+q <= 9, can be displayed using procedure clidata as follows:

```
> clidata([2,0]); #Clifford algebra of the Ecuclidean plane R^2
```

$$\left[real, 2, simple, \frac{Id}{2}+\frac{e1}{2}, [Id, e2], [Id], [Id, e2]\right]$$

```
> clidata([3,0]); #Clifford algebra of the Ecuclidean space R^3
```

$$\left[complex, 2, simple, \frac{Id}{2}+\frac{e1}{2}, [Id, e2, e3, e2we3], [Id, e2we3], [Id, e2]\right]$$

```
> clidata([1,3]); #Clifford algebra of the Minkowski space
```

$$\left[quaternionic, 2, simple, \frac{Id}{2}+\frac{e1we4}{2}, [Id, e1, e2, e3, e1we2, e1we3, e2we3, e1we2we3], [Id, e2, e3, e2we3], [Id, e1]\right]$$

```
> clidata([3,1]); #Clifford algebra of the Minkowski space
```

$$\left[real, 4, simple, 'cmulQ'\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e3we4}{2}\right), [Id, e2, e3, e2we3], [Id], [Id, e2, e3, e2we3]\right]$$

```
>
```

In order to view matrices representing 1-vectors {e1,e2,e3,..., en} in Cl(V,Q), dim V = n, use procedure matKrepr. For example, let's see matrices for the above-listed algebras and for the semi-simple algebra Cl(Cl(2,1) = Mat(2,2,R+R). These last matrices, with entries in the double field R+R are of `type/dfmatrix` and have been precomputed.

```
> matKrepr([2,0]);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$\left[e1=\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}, e2=\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\right]$$

```
> matKrepr([3,0]); #Pauli matrices
```

$$\left[e1=\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}, e2=\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}, e3=\begin{bmatrix}0 & -e2we3\\ e2we3 & 0\end{bmatrix}\right]$$

```
> matKrepr([1,3]);
```

$$\left[e1=\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}, e2=\begin{bmatrix}e2 & 0\\ 0 & -e2\end{bmatrix}, e3=\begin{bmatrix}e3 & 0\\ 0 & -e3\end{bmatrix}, e4=\begin{bmatrix}0 & -1\\ 1 & 0\end{bmatrix}\right]$$

```
> matKrepr([3,1]); #Dirac matrices
```

$$\left[e1=\begin{bmatrix}1&0&0&0\\0&-1&0&0\\0&0&-1&0\\0&0&0&1\end{bmatrix}, e2=\begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}, e3=\begin{bmatrix}0&0&1&0\\0&0&0&-1\\1&0&0&0\\0&-1&0&0\end{bmatrix}, e4=\begin{bmatrix}0&0&-1&0\\0&0&0&1\\1&0&0&0\\0&-1&0&0\end{bmatrix}\right]$$

```
> matKrepr([2,1]);
```

$$\left[e1=\begin{bmatrix}[1,-1]&[0,0]\\[0,0]&[-1,1]\end{bmatrix}, e2=\begin{bmatrix}[0,0]&[1,1]\\[1,1]&[0,0]\end{bmatrix}, e3=\begin{bmatrix}[0,0]&[-1,-1]\\[1,1]&[0,0]\end{bmatrix}\right]$$

For more information how to find spinor representations in higher dimensions, see help pages for Kfield, gradeinv, marKrepr, minimalideal, spinorKbasis, spinorKrepr. See also cbasis, clidata, `type/primitiveidemp`, `type/idempotent`.

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 0.171000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

Example 1: Let's compute the two bilinear forms beta_{+} and beta_{-} on S = Cl(Q)f where Cl(Q)=Cl(3,0) is the Clifford algebra of the Euclidean space R^3. To shorten output, procedure makealiases is used.

```
> restart:bench:=time():with(Clifford):with(linalg):
  B:=diag(1,1,1); #define the diagonal form B for Cl(3,0)
```

$$B:=\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$$

```
> dim:=coldim(B):eval(makealiases(dim)):
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data:=\left[complex, 2, simple, \frac{Id}{2}+\frac{e1}{2}, [Id, e2, e3, e23], [Id, e23], [Id, e2]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own
  here
> for i from 1 to nops(data[7]) do f||i:=data[7][i] &c f od;
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$f1:=\frac{Id}{2}+\frac{e1}{2}$$

$$f2:=\frac{e2}{2}-\frac{e12}{2}$$

```
> Kbasis:=data[6]; #here K = C
```

$$Kbasis:=[Id, e23]$$

Let's define arbitrary (complex) spinor coefficients psi1, psi2, phi1, and phi2 for two spinors 'psi' and 'phi' in S = Cl(Q)f = C^2. Notice, that psi1, psi2, phi1, and phi2 belong to a subalgebra K of

Cl(Q) spanned by {Id, e2we3} that is isomorphic to C since (e2we3)^2 = -Id. Recall also that the left minimal ideal S = Cl(D)f is a RIGHT K-module. That's why the 'complex' coefficients must be written on the RIGHT of the spinor basis elements f1 and f2 in S:

```
> psi1:=psi11 * Id + psi12 * e23;psi2:=psi21 * Id + psi22 * e23;
  #entries in K
```

$$\psi 1 := \psi 11\, Id + \psi 12\, e23$$

$$\psi 2 := \psi 21\, Id + \psi 22\, e23$$

```
> phi1:=phi11 * Id + phi12 * e23;phi2:=phi21 * Id + phi22 * e23;
  #entries in K
```

$$\phi 1 := \phi 11\, Id + \phi 12\, e23$$

$$\phi 2 := \phi 21\, Id + \phi 22\, e23$$

```
> psi:='f1 &c psi1' + 'f2 &c psi2'; #here psi1 ... are 'complex'
  components of psi
  phi:='f1 &c phi1' + 'f2 &c phi2'; #here phi1 ... are 'complex'
  components of phi
```

$$\psi := \mathrm{climul}(f1, \psi 1) + \mathrm{climul}(f2, \psi 2)$$

$$\phi := \mathrm{climul}(f1, \phi 1) + \mathrm{climul}(f2, \phi 2)$$

```
>
```

Now, we compute beta_{+} on 'psi' and 'phi', while we store the purespinor under the name 'purespinor1'. Notice, that beta_{+} is invariant under the unitary group U(2) (for more information see [1], [4]).

```
> x:=beta_plus(psi,phi,f,'purespinor1');purespinor1; #beta_plus
  is invariant under U(2)
```

$$x := (\psi 12\, \phi 12 + \psi 11\, \phi 11 + \psi 22\, \phi 22 + \psi 21\, \phi 21)\, Id + (\psi 11\, \phi 12 + \psi 21\, \phi 22 - \psi 22\, \phi 21 - \psi 12\, \phi 11)\, e23$$

$$Id$$

```
> psi1c:=reversion(psi1);psi2c:=reversion(psi2);
```

$$psi1c := \psi 11\, Id - \psi 12\, e23$$

$$psi2c := \psi 21\, Id - \psi 22\, e23$$

```
> clicollect(psi1c &c phi1 + psi2c &c phi2); #this equals x and
  is invariant under U(2)
```

$$(\psi 12\, \phi 12 + \psi 11\, \phi 11 + \psi 22\, \phi 22 + \psi 21\, \phi 21)\, Id + (\psi 11\, \phi 12 + \psi 21\, \phi 22 - \psi 22\, \phi 21 - \psi 12\, \phi 11)\, e23$$

```
>
```

Now, we compute beta_{-} on 'psi' and 'phi', while we store the purespinor under the name 'purespinor2'. Notice, that beta_{-} is invariant under the complex symplectic group Sp(2,C) (for more information see [1], [4]).

```
> y:=beta_minus(psi,phi,f,'purespinor2');purespinor2; #beta_minus
  is invariant under Sp(2,C)
```

$$y := (-\psi 12\, \phi 22 + \psi 11\, \phi 21 + \psi 22\, \phi 12 - \psi 21\, \phi 11)\, Id$$

$$+ (-\psi 21\ \phi 12 - \psi 22\ \phi 11 + \psi 11\ \phi 22 + \psi 12\ \phi 21)\ e23$$

$$e2$$

```
> clicollect(psi1 &c phi2 - psi2 &c phi1); #this equals y and is invariant under Sp(2,C)
```

$$(-\psi 12\ \phi 22 + \psi 11\ \phi 21 + \psi 22\ \phi 12 - \psi 21\ \phi 11)\ Id$$
$$+ (-\psi 21\ \phi 12 - \psi 22\ \phi 11 + \psi 11\ \phi 22 + \psi 12\ \phi 21)\ e23$$

```
>
```

Observe that 'purespinor1' and 'purespinor2' have the desired commuting properties with the idempotent f:

```
> u:=purespinor1; f &c u - u &c reversion(f);
```

$$u := Id$$

$$0$$

```
> u:=purespinor2; f &c u - u &c conjugation(f);
```

$$u := e2$$

$$0$$

Testing error message since K = span{Id, e23} <> R:

```
> f1:=1/2*Id+1/2*e2;
```

$$f1 := \frac{Id}{2} + \frac{e2}{2}$$

```
> type(f1,primitiveidemp);
```

$$true$$

```
> beta_plus(psi,phi,f1);
Error, (in Clifford:-beta_plus) when K = C or K = H, primitive idempotent f = plus/minus clidata(B)[4] or its grade involution
```

```
> beta_minus(psi,phi,f1);
Error, (in Clifford:-beta_minus) when K = C or K = H, primitive idempotent f = plus/minus clidata(B)[4] or its grade involution
```

The error message is returned due to the fact that in the case when K = C or K = H, the only flexibility in these two procedures in accepting a different primitive idempotent 'f ' than the one stored under clidata(B)[4] is that 'f' must equal plus or minus the stored idempotent, or its grade involution. In the real case when K = R, this doesn't matter and the user can enter any primitive idempotent.

```
> f:=clidata(B)[4];
```

$$f := \frac{Id}{2} + \frac{e1}{2}$$

```
> evalb(f1=f),evalb(f1=-f),evalb(f1=gradeinv(f)),evalb(f1=-gradeinv(f));
```

$$false, false, false, false$$

This requirement has to do with the fact that both procedures beta_{+} and beta_{-} use pre-computed basis for K which contains two or four elements depending whether K = C or K =

H. Certainly, this basis depends on the choice of the idempotent! In general, such basis for K can be computed for any primitive idempotent f with the procedure Kbasis.

```
> 'K'=clidata()[6];
```

$$K = [Id, e23]$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 2.839000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM with Win XP Professional
>
```

Example 2: Scalar products of spinors of the Clifford algebra Cl(1,3) that is isomorphic to H(2).

```
> restart:bench:=time():with(Clifford):with(linalg):
  B:=linalg[diag](1,-1,-1,-1): #define the bilinear form B for Cl(1,3)
> dim:=coldim(B):eval(makealiases(dim)):
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data := \left[quaternionic, 2, simple, \frac{Id}{2} + \frac{e14}{2}, [Id, e1, e2, e3, e12, e13, e23, e123], [Id, e2, e3, e23], [Id, e1]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own here
> for i from 1 to nops(data[7]) do f||i:=data[7][i] &c f od;
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in clude &C and &C[K]. Type ?cliprod for help.
```

$$f1 := \frac{Id}{2} + \frac{e14}{2}$$

$$f2 := \frac{e1}{2} + \frac{e4}{2}$$

Elements f1 and f2 computed above give a K-basis in S = Cl(1,3)f where K = span{Id,e2,e3,e2we3} is a subalgebra of Cl(1,3) isomorphic with the quaternionic ring H. The elements spanning K are stored in data[6] displayed above. One can really check that these elements form a quaternionic basis and form a list of `type/genquatbasis`:

```
> Kbasis:=data[6]; #here K = H
```

$$Kbasis := [Id, e2, e3, e23]$$

```
> type(Kbasis,genquatbasis);
```

$$true$$

It is also possible to check it directly by computing all possible products and then displaying them in a matrix:

```
> M:=matrix(4,4,(i,j)->cmul(Kbasis[i],Kbasis[j]));
```

$$M := \begin{bmatrix} Id & e2 & e3 & e23 \\ e2 & -Id & e23 & -e3 \\ e3 & -e23 & -Id & e2 \\ e23 & e3 & -e2 & -Id \end{bmatrix}$$

Like in the example above, we first define arbitrary quaternionic coefficients for two spinors 'psi' and 'phi' in S = Cl(1,3)f:

```
> psi1:=psi11 * Id + psi12 * e2 + psi13 * e3 + psi14 * e23;#first
  coefficient of psi in K
  psi2:=psi21 * Id + psi22 * e2 + psi23 * e3 + psi24 *
  e23;#second coefficient of psi in K
```

$$\psi 1 := \psi 11\, Id + \psi 12\, e2 + \psi 13\, e3 + \psi 14\, e23$$

$$\psi 2 := \psi 21\, Id + \psi 22\, e2 + \psi 23\, e3 + \psi 24\, e23$$

```
> phi1:=phi11 * Id + phi12 * e2 + phi13 * e3 + phi14 * e23;#first
  coefficient of phi in K
  phi2:=phi21 * Id + phi22 * e2 + phi23 * e3 + phi24 *
  e23;#second coefficient of phi in K
```

$$\phi 1 := \phi 11\, Id + \phi 12\, e2 + \phi 13\, e3 + \phi 14\, e23$$

$$\phi 2 := \phi 21\, Id + \phi 22\, e2 + \phi 23\, e3 + \phi 24\, e23$$

Now, we define spinors 'psi' and 'phi':

```
> psi:='f1 &c psi1' + 'f2 &c psi2'; #here psi1 ... are
  quaternionic components of psi
  phi:='f1 &c phi1' + 'f2 &c phi2'; #here phi1 ... are
  quaternionic components of phi
```

$$\psi := Cliplus{:}\text{-}climul(f1, \psi 1) + Cliplus{:}\text{-}climul(f2, \psi 2)$$

$$\phi := Cliplus{:}\text{-}climul(f1, \phi 1) + Cliplus{:}\text{-}climul(f2, \phi 2)$$

The beta_{+} and the beta_{-} forms on S x S are invariant under the symplectic group Sp(2,2) as can be checked directly from the output below:

```
> x:=beta_plus(psi,phi,f,'purespinor1');purespinor1; #beta_plus
  is invariant under Sp(2,2)
```

$$x := (\psi 12\, \phi 22 + \psi 13\, \phi 23 + \psi 21\, \phi 11 + \psi 11\, \phi 21 + \psi 22\, \phi 12 + \psi 24\, \phi 14 + \psi 23\, \phi 13 + \psi 14\, \phi 24)\, Id + (-\psi 23\, \phi 14 - \psi 12\, \phi 21 - \psi 13\, \phi 24 + \psi 21\, \phi 12 + \psi 24\, \phi 13 + \psi 14\, \phi 23 + \psi 11\, \phi 22 - \psi 22\, \phi 11)\, e2 + (-\psi 24\, \phi 12 + \psi 11\, \phi 23 + \psi 12\, \phi 24 - \psi 14\, \phi 22 - \psi 23\, \phi 11 + \psi 22\, \phi 14 + \psi 21\, \phi 13 - \psi 13\, \phi 21)\, e3 + (-\psi 22\, \phi 13 + \psi 11\, \phi 24 + \psi 23\, \phi 12 + \psi 13\, \phi 22 - \psi 14\, \phi 21 - \psi 24\, \phi 11 + \psi 21\, \phi 14 - \psi 12\, \phi 23)\, e23$$

$$e1$$

```
> y:=beta_minus(psi,phi,f,'purespinor2');purespinor2; #beta_minus
  is invariant under Sp(2,2)
```

$$y := (-\psi 12\,\phi 22 - \psi 13\,\phi 23 + \psi 23\,\phi 13 + \psi 14\,\phi 24 - \psi 24\,\phi 14 + \psi 11\,\phi 21 + \psi 22\,\phi 12 - \psi 21\,\phi 11)\,Id + (\psi 12\,\phi 21 + \psi 13\,\phi 24 + \psi 14\,\phi 23 - \psi 22\,\phi 11 - \psi 24\,\phi 13 - \psi 23\,\phi 14 - \psi 21\,\phi 12 + \psi 11\,\phi 22)\,e2 + (\psi 22\,\phi 14 - \psi 23\,\phi 11 - \psi 12\,\phi 24 - \psi 14\,\phi 22 + \psi 13\,\phi 21 - \psi 21\,\phi 13 + \psi 24\,\phi 12 + \psi 11\,\phi 23)\,e3 + (-\psi 14\,\phi 21 + \psi 24\,\phi 11 - \psi 22\,\phi 13 + \psi 11\,\phi 24 - \psi 21\,\phi 14 - \psi 13\,\phi 22 + \psi 12\,\phi 23 + \psi 23\,\phi 12)\,e23$$

$$e1$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 2.784000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

Example 3: Let's compute the two bilinear forms beta_{+} and beta_{-} on S = Cl(Q)f where Cl(Q)=Cl(2,3) (complexification of Cl(1,3)) is the Clifford algebra of the de Sitter space of signature (2,3). Cl(2,3) is isomorphic to C(4) and is a complexification of Cl(1,3).

```
> restart:bench:=time():with(Clifford):dim:=5:eval(makealiases(di
  m)):
> B:=linalg[diag](1,1,-1,-1,-1):#define form B for Cl(2,3)
> data:=clidata(B); #retrieve and display data about Cl(2,3)
```

$$data := \left[complex, 4, simple, 'cmulQ'\left(\frac{Id}{2} + \frac{e14}{2}, \frac{Id}{2} + \frac{e25}{2}\right), [Id, e1, e2, e3, e12, e13, e23, e123], [Id, e3], [Id, e1, e2, e12]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own
  here
```

Here are the matrices representing basis 1-monomials of Cl(2,3):

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

$$\left[e1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, e2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}, e3 = \begin{bmatrix} e3 & 0 & 0 & 0 \\ 0 & -e3 & 0 & 0 \\ 0 & 0 & -e3 & 0 \\ 0 & 0 & 0 & e3 \end{bmatrix}, e4 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, e5 = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}\right]$$

```
for i from 1 to 4 do f||i:=data[7][i] &c f od;
```

$$f1 := \frac{Id}{4} + \frac{e14}{4} + \frac{e25}{4} - \frac{e1245}{4}$$

$$f2 := \frac{e1}{4} + \frac{e125}{4} - \frac{e245}{4} + \frac{e4}{4}$$

$$f3 := \frac{e2}{4} - \frac{e124}{4} + \frac{e145}{4} + \frac{e5}{4}$$

$$f4 := \frac{e12}{4} + \frac{e15}{4} - \frac{e24}{4} + \frac{e45}{4}$$

Thus, a spinor psi from S=Cl(2,3)f is a four component vector with real entries. It can be written in terms of the basis [f1,f2,f3,f4] and its complex entries in K = <Id>_R +<e3>_R as follows:

```
> psi1:=psi11+psi12*e3:psi2:=psi21+psi22*e3:psi3:=psi31+psi32*e3:
  psi4:=psi41+psi42*e3:
> phi1:=phi11+phi12*e3:phi2:=phi21+phi22*e3:phi3:=phi31+phi32*e3:
  phi4:=phi41+phi42*e3:
```

Let's define now arbitrary spinors 'psi' and 'phi' in S considered as the RIGHT K-module:

```
> psi:='f1 &c psi1' + 'f2 &c psi2' + 'f3 &c psi3' + 'f4 &c psi4';

  phi:='f1 &c phi1' + 'f2 &c phi2' + 'f3 &c phi3' + 'f4 &c phi4';
```

$$\psi := Cliplus\text{:-}climul(f1, \psi 1) + Cliplus\text{:-}climul(f2, \psi 2) + Cliplus\text{:-}climul(f3, \psi 3) + Cliplus\text{:-}climul(f4, \psi 4)$$

$$\phi := Cliplus\text{:-}climul(f1, \phi 1) + Cliplus\text{:-}climul(f2, \phi 2) + Cliplus\text{:-}climul(f3, \phi 3) + Cliplus\text{:-}climul(f4, \phi 4)$$

The two scalar forms beta_{+} and beta_{-} are:

```
> x:=beta_plus(psi,phi,f,'s1');s1; #beta_plus is invariant under
  Sp(4,C)
```

$$x := (-\psi 11\, \phi 41 - \psi 42\, \phi 12 - \psi 21\, \phi 31 + \psi 12\, \phi 42 - \psi 32\, \phi 22 + \psi 31\, \phi 21 + \psi 41\, \phi 11 + \psi 22\, \phi 32)\, Id + (\psi 41\, \phi 12 - \psi 21\, \phi 32 + \psi 32\, \phi 21 + \psi 42\, \phi 11 - \psi 22\, \phi 31 + \psi 31\, \phi 22 - \psi 12\, \phi 41 - \psi 11\, \phi 42)\, e3\, e12$$

```
> y:=beta_minus(psi,phi,f,'s2');s2; #beta_minus is invariant
  under U(2,2)
```

$$y := (\psi 21\, \phi 31 + \psi 42\, \phi 12 - \psi 31\, \phi 21 + \psi 22\, \phi 32 - \psi 12\, \phi 42 + \psi 41\, \phi 11 - \psi 32\, \phi 22 - \psi 11\, \phi 41)\, Id + (-\psi 31\, \phi 22 + \psi 41\, \phi 12 - \psi 22\, \phi 31 + \psi 32\, \phi 21 - \psi 42\, \phi 11 + \psi 12\, \phi 41 - \psi 11\, \phi 42 + \psi 21\, \phi 32)\, e3\, e12$$

We can verify that the automorphism group of beta_{-} is U(2,2) as follows:

```
> psi1c:=gradeinv(psi1):psi2c:=gradeinv(psi2):psi3c:=gradeinv(psi
```

```
3):psi4c:=gradeinv(psi4):
> y1:=clicollect(cmul(psi4c,phi1)+cmul(psi2c,phi3)-cmul(psi3c,phi
2)-cmul(psi1c,phi4));
```

$$y1 := (\psi 21\ \phi 31 + \psi 42\ \phi 12 - \psi 31\ \phi 21 + \psi 22\ \phi 32 - \psi 12\ \phi 42 + \psi 41\ \phi 11 - \psi 32\ \phi 22 - \psi 11\ \phi 41)\ Id - (\psi 31\ \phi 22 - \psi 41\ \phi 12 + \psi 22\ \phi 31 - \psi 32\ \phi 21 + \psi 42\ \phi 11 - \psi 12\ \phi 41 + \psi 11\ \phi 42 - \psi 21\ \phi 32)\ e3$$

```
> simplify(y1-y);
```

$$0$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 7.173000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

Example 4: Scalar products in the spinor spaces of the Clifford algebra Cl(3,1), that is isomorphic to R(4), are both invariant under the symplectic group Sp(4).

```
> restart:bench:=time():with(Clifford):dim:=4:eval(makealiases(di
m)):
> B:=linalg[diag](1,1,1,-1):#define form B for Cl(3,1)
> data:=clidata(B); #retrieve and display data about Cl(3,1)
```

$$data := \left[real, 4, simple, 'cmulQ'\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e34}{2}\right), [Id, e2, e3, e23], [Id], [Id, e2, e3, e23]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own
here
```

Here are the matrices representing some basis monomials of Cl(B). Those that represent the 1-vectors are usually referred to as *Dirac gamma matrices* (since 'gamma' is a protected Maple name, we will use here 'g' instead):

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

$$\left[e1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, e2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, e3 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}, e4 = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}\right]$$

Let's find matrices representing the four basis vectors in S:

```
> for i from 1 to 4 do f||i:=data[7][i] &c f od;
```

$$f1 := \frac{Id}{4} + \frac{e134}{4} + \frac{e34}{4} + \frac{e1}{4}$$

$$f2 := -\frac{e12}{4} - \frac{e1234}{4} + \frac{e234}{4} + \frac{e2}{4}$$

$$f3 := -\frac{e13}{4} + \frac{e3}{4} - \frac{e14}{4} + \frac{e4}{4}$$

$$f4 := \frac{e23}{4} + \frac{e123}{4} + \frac{e124}{4} + \frac{e24}{4}$$

Thus, a spinor psi from S=Cl(3,1)f is a four component vector with real entries. It can be written in terms of the basis [f1,f2,f3,f4] and its real entries in K = <Id>_R as follows:

```
> psi:= psi1 * 'f1' + psi2 * 'f2' + psi3 * 'f3' + psi4 * 'f4';
  phi:= phi1 * 'f1' + phi2 * 'f2' + phi3 * 'f3' + phi4 * 'f4';
```

$$\psi := \psi 1\, f1 + \psi 2\, f2 + \psi 3\, f3 + \psi 4\, f4$$

$$\phi := \phi 1\, f1 + \phi 2\, f2 + \phi 3\, f3 + \phi 4\, f4$$

```
> x:=beta_plus(psi,phi,f,'s1');s1; #automorphism group of
  beta_plus is Sp(4,R)
```

$$x := (\psi 4\, \phi 1 + \psi 3\, \phi 2 - \psi 2\, \phi 3 - \psi 1\, \phi 4)\, Id$$

$$e23$$

```
> y:=beta_minus(psi,phi,f,'s2');s2; #automorphism group of
  beta_minus is Sp(4,R)
```

$$y := (\psi 1\, \phi 3 - \psi 3\, \phi 1 + \psi 4\, \phi 2 - \psi 2\, \phi 4)\, Id$$

$$e3$$

Since the coefficients of 'psi' and 'phi' are here assumed to be real, it doesn't matter that they were written on the left side of the basis elements [f1,f2,f3,f4].

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 1.800000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

Example 5: Let's consider finally the semi-simple case of Cl(2,1) = Mat(2,R) + Mat(2,R). Here, we will have to deal with pairs of spinor spinor spaces S and S^ where S^ is a spinor space of spinors of the form gradeinv(psi) where gradeinv is the grade involution in Cl(2,1) and 'psi' is a spinor from S = Cl(1,2)f.

```
> restart:bench:=time():with(Clifford):with(linalg):_shortcut_in_
  spinorKrepr:=false:
> B:=diag(1$2,-1$1):dim:=coldim(B):eval(makealiases(dim)):clibasi
  s:=cbasis(dim):
```

Display information stored:

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

$$\left[e1 = \begin{bmatrix} [1, -1] & [0, 0] \\ [0, 0] & [-1, 1] \end{bmatrix}, e2 = \begin{bmatrix} [0, 0] & [1, 1] \\ [1, 1] & [0, 0] \end{bmatrix}, e3 = \begin{bmatrix} [0, 0] & [-1, -1] \\ [1, 1] & [0, 0] \end{bmatrix}\right]$$

```
> data:=clidata();f:=data[4];
```

$$data := \left[real, 2, semisimple, 'Clifford{:}\text{-}cmulQ'\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e23}{2}\right), [Id, e2], [Id], [Id, e2]\right]$$

$$f := Clifford{:}\text{-}cmulQ\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e23}{2}\right)$$

We will need now one spinor basis for S and for S^:

```
> K_basis1:=[seq(data[7][i] &c f,i=1..nops(data[7]))];#spinor
  basis in S
  K_basis2:=map(gradeinv,K_basis1);#spinor basis in gradeinv(S)
  or S hat
```

$$K\_basis1 := \left[\frac{Id}{4}+\frac{e23}{4}+\frac{e123}{4}+\frac{e1}{4}, -\frac{e13}{4}-\frac{e12}{4}+\frac{e2}{4}+\frac{e3}{4}\right]$$

$$K\_basis2 := \left[\frac{Id}{4}+\frac{e23}{4}-\frac{e123}{4}-\frac{e1}{4}, -\frac{e13}{4}-\frac{e12}{4}-\frac{e2}{4}-\frac{e3}{4}\right]$$

```
> FBgens1:=data[6];FBgens2:=map(gradeinv,FBgens1);#generators for
  K and K hat
```

$$FBgens1 := [Id]$$

$$FBgens2 := [Id]$$

Although the basis elements e1, e2, e3 already have their matrices with entries in R+R pre-computed and displayed above, we show how they were computed. First, we need to find pairs of matrices representing e1, e2, and e3 respectively

```
> for i from 1 to dim do
  M[i],N[i]:=spinorKrepr(e||i,K_basis1,FBgens1,'left'),spinorKrep
  r(e||i,K_basis2,FBgens2,'left');
  od;
```

$$M_1, N_1 := \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$M_2, N_2 := \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$$

$$M_3, N_3 := \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

By combining them pairwise, we get matrices representing faithfully e1, e2, and e3 in Mat(2,2,R+R). This operation can be done with cdfmatrix:

```
> E[1]:=cdfmatrix(M[1],N[1]); #matrix for e1
```

$$E_1 := \begin{bmatrix} [1, -1] & [0, 0] \\ [0, 0] & [-1, 1] \end{bmatrix}$$

```
> E[2]:=cdfmatrix(M[2],N[2]); #matrix for e2
```

$$E_2 := \begin{bmatrix} [0, 0] & [1, -1] \\ [1, -1] & [0, 0] \end{bmatrix}$$

```
> E[3]:=cdfmatrix(M[3],N[3]); #matrix for e3
```

$$E_3 := \begin{bmatrix} [0,0] & [-1,1] \\ [1,-1] & [0,0] \end{bmatrix}$$

Notice that these matrices observe appropriate commutation relations:

```
> mdfmatrix(E[1],E[1]); #square of E[1] or e1
```

$$\begin{bmatrix} [1,1] & [0,0] \\ [0,0] & [1,1] \end{bmatrix}$$

```
> mdfmatrix(E[2],E[2]); #square of E[2] or e2
```

$$\begin{bmatrix} [1,1] & [0,0] \\ [0,0] & [1,1] \end{bmatrix}$$

```
> mdfmatrix(E[3],E[3]); #square of E[3] or e3
```

$$\begin{bmatrix} [-1,-1] & [0,0] \\ [0,0] & [-1,-1] \end{bmatrix}$$

```
> adfmatrix(mdfmatrix(E[1],E[2]),mdfmatrix(E[2],E[1])); #product
  cmul(e1,e2)+cmul(e2,e1)
```

$$\begin{bmatrix} [0,0] & [0,0] \\ [0,0] & [0,0] \end{bmatrix}$$

```
> adfmatrix(mdfmatrix(E[1],E[3]),mdfmatrix(E[3],E[1])); #product
  cmul(e1,e3)+cmul(e3,e1)
```

$$\begin{bmatrix} [0,0] & [0,0] \\ [0,0] & [0,0] \end{bmatrix}$$

```
> adfmatrix(mdfmatrix(E[2],E[3]),mdfmatrix(E[3],E[2])); #product
  cmul(e2,e3)+cmul(e3,e2)
```

$$\begin{bmatrix} [0,0] & [0,0] \\ [0,0] & [0,0] \end{bmatrix}$$

```
>
```

Let's make assignment of the basis elements for S and S^:

```
> f1:=K_basis1[1];f2:=K_basis1[2]; #basis in S
```

$$f1 := \frac{Id}{4} + \frac{e23}{4} + \frac{e123}{4} + \frac{e1}{4}$$

$$f2 := -\frac{e13}{4} - \frac{e12}{4} + \frac{e2}{4} + \frac{e3}{4}$$

```
> g1:=K_basis2[1];g2:=-K_basis2[2]; #basis in S hat
```

$$g1 := \frac{Id}{4} + \frac{e23}{4} - \frac{e123}{4} - \frac{e1}{4}$$

$$g2 := \frac{e13}{4} + \frac{e12}{4} + \frac{e2}{4} + \frac{e3}{4}$$

That is, S = span{f1,f2} and S^ = span{g1,g2}. Thus, spinors in S = Cl(Q)f = R^2 can be displayed as follows:

```
> psi:='f1' &c (psi1 &c Id) + 'f2' &c (psi2 &c Id);
  phi:='f1' &c (phi1 &c Id) + 'f2' &c (phi2 &c Id);
```

$$\psi := \frac{\psi 1\, Id}{4} + \frac{\psi 1\, e23}{4} + \frac{\psi 1\, e123}{4} + \frac{\psi 1\, e1}{4} - \frac{\psi 2\, e13}{4} - \frac{\psi 2\, e12}{4} + \frac{\psi 2\, e2}{4} + \frac{\psi 2\, e3}{4}$$

$$\phi := \frac{\phi 1\, Id}{4} + \frac{\phi 1\, e23}{4} + \frac{\phi 1\, e123}{4} + \frac{\phi 1\, e1}{4} - \frac{\phi 2\, e13}{4} - \frac{\phi 2\, e12}{4} + \frac{\phi 2\, e2}{4} + \frac{\phi 2\, e3}{4}$$

The matrix representation of 'psi' and 'phi' will then be:

```
> matKrepr(psi),matKrepr(phi);
```

$$\begin{bmatrix} [\psi 1, 0] & [0, 0] \\ [\psi 2, 0] & [0, 0] \end{bmatrix}, \begin{bmatrix} [\phi 1, 0] & [0, 0] \\ [\phi 2, 0] & [0, 0] \end{bmatrix}$$

The two scalar forms in S are then:

```
> beta_plus(psi,phi,f1,'s1');s1; #beta_plus is invariant under
  GL(2,R)
```

$$0$$

$$s1$$

```
> beta_minus(psi,phi,f1,'s2');s2; #beta_minus is invariant under
  Sp(2,R)
```

$$(-\psi 2\, \phi 1 + \psi 1\, \phi 2)\, Id$$

$$e2$$

Verification:

```
> out1:=clicollect(s2 &c conjugation(psi) &c phi &c f1);
```

$$out1 := -\frac{(\psi 2\, \phi 1 - \psi 1\, \phi 2)\, Id}{4} - \frac{(\psi 2\, \phi 1 - \psi 1\, \phi 2)\, e23}{4} - \frac{(\psi 2\, \phi 1 - \psi 1\, \phi 2)\, e123}{4} - \frac{(\psi 2\, \phi 1 - \psi 1\, \phi 2)\, e1}{4}$$

```
> psiphi:=psi1*phi2-psi2*phi1;
```

$$psiphi := -\psi 2\, \phi 1 + \psi 1\, \phi 2$$

```
> out2:=clicollect(expand(psiphi*f1));
```

$$out2 := -\frac{(\psi 2\, \phi 1 - \psi 1\, \phi 2)\, Id}{4} - \frac{(\psi 2\, \phi 1 - \psi 1\, \phi 2)\, e23}{4} - \frac{(\psi 2\, \phi 1 - \psi 1\, \phi 2)\, e123}{4} - \frac{(\psi 2\, \phi 1 - \psi 1\, \phi 2)\, e1}{4}$$

```
> out1-out2;
```

$$0$$

Spinors in S^ = (Cl(Q)f)^ = R^2 can be displayed as follows::

```
> psi:='g1' &c (psi1 &c Id) + 'g2' &c (psi2 &c Id);
  phi:='g1' &c (phi1 &c Id) + 'g2' &c (phi2 &c Id);
```

$$\psi := \frac{\psi 1\, Id}{4} + \frac{\psi 1\, e23}{4} - \frac{\psi 1\, e123}{4} - \frac{\psi 1\, e1}{4} + \frac{\psi 2\, e13}{4} + \frac{\psi 2\, e12}{4} + \frac{\psi 2\, e2}{4} + \frac{\psi 2\, e3}{4}$$

$$\phi := \frac{\phi 1\, Id}{4} + \frac{\phi 1\, e23}{4} - \frac{\phi 1\, e123}{4} - \frac{\phi 1\, e1}{4} + \frac{\phi 2\, e13}{4} + \frac{\phi 2\, e12}{4} + \frac{\phi 2\, e2}{4} + \frac{\phi 2\, e3}{4}$$

```
> matKrepr(psi),matKrepr(phi);
```

$$\begin{bmatrix} [0, \psi 1] & [0, 0] \\ [0, \psi 2] & [0, 0] \end{bmatrix}, \begin{bmatrix} [0, \phi 1] & [0, 0] \\ [0, \phi 2] & [0, 0] \end{bmatrix}$$

The scalar forms then are:

```
> beta_plus(psi,phi,g1); #beta_plus is invariant under GL(2,R)
```

$$0$$

```
> beta_minus(psi,phi,g1,'s'); #invariant under Sp(2,R)
```

$$(-\psi 2\ \phi 1 + \psi 1\ \phi 2)\ Id$$

Verification:

```
> s; #purespinor
```

$$e2$$

```
> out1:=clicollect(s &c conjugation(psi) &c phi &c g1); #all in S
  hat
```

$$out1 := -\frac{(\psi 2\ \phi 1 - \psi 1\ \phi 2)\ Id}{4} - \frac{(\psi 2\ \phi 1 - \psi 1\ \phi 2)\ e23}{4} + \frac{(\psi 2\ \phi 1 - \psi 1\ \phi 2)\ e123}{4} + \frac{(\psi 2\ \phi 1 - \psi 1\ \phi 2)\ e1}{4}$$

```
> out2:=clicollect(expand(psiphi*g1));
```

$$out2 := -\frac{(\psi 2\ \phi 1 - \psi 1\ \phi 2)\ Id}{4} - \frac{(\psi 2\ \phi 1 - \psi 1\ \phi 2)\ e23}{4} + \frac{(\psi 2\ \phi 1 - \psi 1\ \phi 2)\ e123}{4} + \frac{(\psi 2\ \phi 1 - \psi 1\ \phi 2)\ e1}{4}$$

```
> out1-out2;
```

$$0$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 3.550000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

See Also: Clifford:-`type/dfmatrix`, Clifford:-ddfmatrix, Clifford:-cdfmatrix

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-beta_plus, Clifford:-beta_minus - computes scalar product in spinor ideals

Calling Sequence:

beta_plus(psi, phi, f, 's') - computes a scalar product of two spinors 'psi' and 'phi' in S=Cl(Q)f when one of them is subject to reversion anti-automorphism where S is a spinor ideal in Cl(Q) generated by a primitive idempotent 'f'; argument 's' is optional

beta_minus(psi, phi, f, 's') - computes a scalar product of two spinors 'psi' and 'phi' in S=Cl(Q)f when one of them is subject to conjugation anti-automorphism where S is a spinor ideal in Cl(Q) generated by a primitive idempotent 'f'; argument 's' is optional

Parameters:

psi, phi - elements in S=Cl(Q)f which are of one of the following types: `type/cliscalar`, `type/clibasmon`, `type/climon`, or `type/clipolynom`
f - element of `type/primitiveidemp`, that is, a primitive idempotent in Cl(Q)
's' (optional) - optional argument of `type/name` (it must be unevaluated name)

Description:

- These two procedures give beta_{+} and beta_{-} scalar products on S=Cl(Q)f considered as right K-modules that have been described in [1] and [4].
- The first two arguments are spinors 'psi' and 'phi' which are expected to be of type one of the types `type/cliscalar`, `type/clibasmon`, `type/climon`, or `type/clipolynom` with the most general one being the last one. Procedures reversion and conjugation realize operations of reversion and conjugation in Cl(Q) (more generally in Cl(V,B)), where Q is a quadratic form defined in V, that is, Cl(Q)=Cl(V,Q). The third argument is a primitive idempotent f in Cl(Q), that is, an element of `type/primitiveidemp'. This idempotent, as it is explained below, can be arbitrary in case when K = R, while there are some restrictions what it can be when K = C and K = H.
- The fourth optional argument 's' is of `type/name': it will be a placeholder for an invertible element s in Cl(Q) such that cmul(f, s) = cmul(s, auto(f)) for a simple Clifford algebra Cl(Q) (when the signature of Q is (p,q) and p - q <> 1 mod 4), or cmul(g, s) = cmul(g, auto(g)) for a semisimple Clifford algebra Cl(Q) (when the signature of Q is (p,q) and p - q = 1 mod 4). Here, cmul denotes Clifford product in Cl(Q), g = f + conjugation(f), and 'auto' is either 'reversion' or 'conjugation'. Invertible element s is needed so that cmul(s, auto(psi), phi) belongs to K, K^ or K + K^, depending on the chosen (anti-) automorphism and whether Cl(Q) is simple or semisimple. Here K is a subalgebra of Cl(Q) isomorphic with the reals, complex numbers, or quaternions depending on the signature of Q (for more information see [1]) and the hat in K^ denotes grade involution in Cl(Q). The grade involution is computed with the procedure gradeinv. Upon exiting, 'beta' stores the value of s under the name 's' supplied by the user. These special elements s in Cl(Q) are called 'purespinors'. It must be entered as 's', that is, as an unevaluated name. Otherwise, an error message will appear if s has already been assigned some other basis monomial as a purespinor since basis monomials are protected in 'CLIFFORD'.

- Procedure Kfield computes a basis for the ring K, while procedures spinorKbasis and spinorKrepr compute a K-basis in S = Cl(Q)f and a spinor representation of Cl(Q) in S, respectively.
- For more information on symbolic computations in Clifford algebras, see [2] and [3].
- Faithful matrix representations of simple and semi-simple Clifford algebras are stored in a library file and can be retrieved with the procedure clidata. Use procedure all_sigs to display signatures and basic structure of Clifford algebras in dimensions 1 through 9.
- **References:**

[1] R. Ablamowicz, "Spinor representations of Clifford algebras: A symbolic approach", Computer Physics Communications Thematic Issue ‘‘Computer Algebra in Physics Research’’, Vol.**115** (1998), Numbers 2--3, pages 510--535.
[2] R. Ablamowicz, P. Lounesto, and J.M. Parra, eds., "Clifford Algebras with Numeric and Symbolic Computations", Birkhauser, Boston, 1996.
[3] R. Ablamowicz and P. Lounesto, "On Clifford algebras of a bilinear form with an antisymmetric part", in: R. Ablamowicz, P. Lounesto, and J.M. Parra, eds., "On Clifford Algebras with Numeric and Symbolic Computations", Birkhauser, Boston, 1996, page 167--188.
[4] P. Lounesto, "Scalar products of spinors and an extension of Brauer-Wall groups", Foundations of Physics, Vol. **11**, Nos. 9/10 (1981) pages 721--740.

Examples:

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

Let's display signatures of simple and semi-simple Clifford algebras in dimensions 1 through 9 which are isomorphic to rings of matrices over R and over the double real field R+R, respectively:

```
> all_sigs(1..9,'real','simple');
```

$$[[0, 6], [0, 8], [1, 1], [1, 7], [2, 0], [2, 2], [3, 1], [3, 3], [4, 2], [4, 4], [5, 3], [8, 0]]$$

```
> all_sigs(1..9,'real','semisimple');
```

$$[[0, 7], [1, 0], [1, 8], [2, 1], [3, 2], [4, 3], [5, 4], [9, 0]]$$

Let's display signatures of simple Clifford algebras in dimensions 1 through 9 which are isomorphic to rings of matrices over the complex field C. There are no semi-simple algebras in this case:

```
> all_sigs(1..9,'complex','simple');
```

$$[[0, 1], [0, 5], [0, 9], [1, 2], [1, 6], [2, 3], [2, 7], [3, 0], [3, 4], [4, 1], [4, 5], [5, 2], [6, 3], [7, 0], [8, 1]]$$

```
> all_sigs(1..9,'complex','semisimple');
```

$$[\ ]$$

Let's display signatures of simple and semi-simple Clifford algebras in dimensions 1 through 9 which are isomorphic to rings of matrices over quaternions H and over the double quaternionic field H+H, respectively:

```
> all_sigs(1..9,'quat','simple');
```

$$[[0, 2], [0, 4], [1, 3], [1, 5], [2, 4], [2, 6], [3, 5], [4, 0], [5, 1], [6, 0], [6, 2], [7, 1]]$$

```
> all_sigs(1..9,'quat','semisimple');
```

$$[[0,3],[1,4],[2,5],[3,6],[5,0],[6,1],[7,2]]$$

More information about any Clifford algebra Cl(Q) of the quadratic form of signature (p,q), 1<= p+q <= 9, can be displayed using procedure clidata as follows:

```
> clidata([2,0]); #Clifford algebra of the Ecuclidean plane R^2
```

$$\left[real, 2, simple, \frac{Id}{2}+\frac{e1}{2}, [Id, e2], [Id], [Id, e2]\right]$$

```
> clidata([3,0]); #Clifford algebra of the Ecuclidean space R^3
```

$$\left[complex, 2, simple, \frac{Id}{2}+\frac{e1}{2}, [Id, e2, e3, e2we3], [Id, e2we3], [Id, e2]\right]$$

```
> clidata([1,3]); #Clifford algebra of the Minkowski space
```

$$\left[quaternionic, 2, simple, \frac{Id}{2}+\frac{e1we4}{2}, [Id, e1, e2, e3, e1we2, e1we3, e2we3, e1we2we3], [Id, e2, e3, e2we3], [Id, e1]\right]$$

```
> clidata([3,1]); #Clifford algebra of the Minkowski space
```

$$\left[real, 4, simple, 'cmulQ'\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e3we4}{2}\right), [Id, e2, e3, e2we3], [Id], [Id, e2, e3, e2we3]\right]$$

```
>
```

In order to view matrices representing 1-vectors {e1,e2,e3,..., en} in Cl(V,Q), dim V = n, use procedure matKrepr. For example, let's see matrices for the above-listed algebras and for the semi-simple algebra Cl(Cl(2,1) = Mat(2,2,R+R). These last matrices, with entries in the double field R+R are of `type/dfmatrix` and have been precomputed.

```
> matKrepr([2,0]);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$\left[e1=\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}, e2=\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\right]$$

```
> matKrepr([3,0]); #Pauli matrices
```

$$\left[e1=\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}, e2=\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}, e3=\begin{bmatrix}0 & -e2we3\\ e2we3 & 0\end{bmatrix}\right]$$

```
> matKrepr([1,3]);
```

$$\left[e1=\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}, e2=\begin{bmatrix}e2 & 0\\ 0 & -e2\end{bmatrix}, e3=\begin{bmatrix}e3 & 0\\ 0 & -e3\end{bmatrix}, e4=\begin{bmatrix}0 & -1\\ 1 & 0\end{bmatrix}\right]$$

```
> matKrepr([3,1]); #Dirac matrices
```

$$\left[e1=\begin{bmatrix}1&0&0&0\\0&-1&0&0\\0&0&-1&0\\0&0&0&1\end{bmatrix}, e2=\begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}, e3=\begin{bmatrix}0&0&1&0\\0&0&0&-1\\1&0&0&0\\0&-1&0&0\end{bmatrix}, e4=\begin{bmatrix}0&0&-1&0\\0&0&0&1\\1&0&0&0\\0&-1&0&0\end{bmatrix}\right]$$

```
> matKrepr([2,1]);
```

$$\left[e1=\begin{bmatrix}[1,-1]&[0,0]\\[0,0]&[-1,1]\end{bmatrix}, e2=\begin{bmatrix}[0,0]&[1,1]\\[1,1]&[0,0]\end{bmatrix}, e3=\begin{bmatrix}[0,0]&[-1,-1]\\[1,1]&[0,0]\end{bmatrix}\right]$$

For more information how to find spinor representations in higher dimensions, see help pages for Kfield, gradeinv, marKrepr, minimalideal, spinorKbasis, spinorKrepr. See also cbasis, clidata, `type/primitiveidemp`, `type/idempotent`.

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 0.079000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

Example 1: Let's compute the two bilinear forms beta_{+} and beta_{-} on S = Cl(Q)f where Cl(Q)=Cl(3,0) is the Clifford algebra of the Euclidean space R^3. To shorten output, procedure makealiases is used.

```
> restart:bench:=time():with(Clifford):with(linalg):
  B:=diag(1,1,1); #define the diagonal form B for Cl(3,0)
```

$$B := \begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$$

```
> dim:=coldim(B):eval(makealiases(dim)):
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data := \left[complex, 2, simple, \frac{Id}{2}+\frac{e1}{2}, [Id, e2, e3, e23], [Id, e23], [Id, e2]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own
  here
> for i from 1 to nops(data[7]) do f||i:=data[7][i] &c f od;
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$f1 := \frac{Id}{2}+\frac{e1}{2}$$

$$f2 := \frac{e2}{2}-\frac{e12}{2}$$

```
> Kbasis:=data[6]; #here K = C
```

$$Kbasis := [Id, e23]$$

Let's define arbitrary (complex) spinor coefficients psi1, psi2, phi1, and phi2 for two spinors 'psi' and 'phi' in S = Cl(Q)f = C^2. Notice, that psi1, psi2, phi1, and phi2 belong to a subalgebra K of

Cl(Q) spanned by {Id, e2we3} that is isomorphic to C since (e2we3)^2 = -Id. Recall also that the left minimal ideal S = Cl(D)f is a RIGHT K-module. That's why the 'complex' coefficients must be written on the RIGHT of the spinor basis elements f1 and f2 in S:

```
> psi1:=psi11 * Id + psi12 * e23;psi2:=psi21 * Id + psi22 * e23;
  #entries in K
```

$$\psi 1 := \psi 11\, Id + \psi 12\, e23$$

$$\psi 2 := \psi 21\, Id + \psi 22\, e23$$

```
> phi1:=phi11 * Id + phi12 * e23;phi2:=phi21 * Id + phi22 * e23;
  #entries in K
```

$$\phi 1 := \phi 11\, Id + \phi 12\, e23$$

$$\phi 2 := \phi 21\, Id + \phi 22\, e23$$

```
> psi:='f1 &c psi1' + 'f2 &c psi2'; #here psi1 ... are 'complex'
  components of psi
  phi:='f1 &c phi1' + 'f2 &c phi2'; #here phi1 ... are 'complex'
  components of phi
```

$$\psi := \mathrm{climul}(f1, \psi 1) + \mathrm{climul}(f2, \psi 2)$$

$$\phi := \mathrm{climul}(f1, \phi 1) + \mathrm{climul}(f2, \phi 2)$$

```
>
```

Now, we compute beta_{+} on 'psi' and 'phi', while we store the purespinor under the name 'purespinor1'. Notice, that beta_{+} is invariant under the unitary group U(2) (for more information see [1], [4]).

```
> x:=beta_plus(psi,phi,f,'purespinor1');purespinor1; #beta_plus
  is invariant under U(2)
```

$$x := (\psi 12\, \phi 12 + \psi 11\, \phi 11 + \psi 22\, \phi 22 + \psi 21\, \phi 21)\, Id + (\psi 11\, \phi 12 + \psi 21\, \phi 22 - \psi 22\, \phi 21 - \psi 12\, \phi 11)\, e23$$

$$Id$$

```
> psi1c:=reversion(psi1);psi2c:=reversion(psi2);
```

$$psi1c := \psi 11\, Id - \psi 12\, e23$$

$$psi2c := \psi 21\, Id - \psi 22\, e23$$

```
> clicollect(psi1c &c phi1 + psi2c &c phi2); #this equals x and
  is invariant under U(2)
```

$$(\psi 12\, \phi 12 + \psi 11\, \phi 11 + \psi 22\, \phi 22 + \psi 21\, \phi 21)\, Id + (\psi 11\, \phi 12 + \psi 21\, \phi 22 - \psi 22\, \phi 21 - \psi 12\, \phi 11)\, e23$$

```
>
```

Now, we compute beta_{-} on 'psi' and 'phi', while we store the purespinor under the name 'purespinor2'. Notice, that beta_{-} is invariant under the complex symplectic group Sp(2,C) (for more information see [1], [4]).

```
> y:=beta_minus(psi,phi,f,'purespinor2');purespinor2; #beta_minus
  is invariant under Sp(2,C)
```

$$y := (-\psi 12\, \phi 22 + \psi 11\, \phi 21 + \psi 22\, \phi 12 - \psi 21\, \phi 11)\, Id$$

$$+ (-\psi 21\ \phi 12 - \psi 22\ \phi 11 + \psi 11\ \phi 22 + \psi 12\ \phi 21)\ e23$$

$$e2$$

```
> clicollect(psi1 &c phi2 - psi2 &c phi1); #this equals y and is invariant under Sp(2,C)
```

$$(-\psi 12\ \phi 22 + \psi 11\ \phi 21 + \psi 22\ \phi 12 - \psi 21\ \phi 11)\ Id$$
$$+ (-\psi 21\ \phi 12 - \psi 22\ \phi 11 + \psi 11\ \phi 22 + \psi 12\ \phi 21)\ e23$$

```
>
```

Observe that 'purespinor1' and 'purespinor2' have the desired commuting properties with the idempotent f:

```
> u:=purespinor1; f &c u - u &c reversion(f);
```

$$u := Id$$

$$0$$

```
> u:=purespinor2; f &c u - u &c conjugation(f);
```

$$u := e2$$

$$0$$

Testing error message since K = span{Id, e23} <> R:

```
> f1:=1/2*Id+1/2*e2;
```

$$f1 := \frac{Id}{2} + \frac{e2}{2}$$

```
> type(f1,primitiveidemp);
```

$$true$$

```
> beta_plus(psi,phi,f1);
Error, (in Clifford:-beta_plus) when K = C or K = H, primitive idempotent f = plus/minus clidata(B)[4] or its grade involution
```

```
> beta_minus(psi,phi,f1);
Error, (in Clifford:-beta_minus) when K = C or K = H, primitive idempotent f = plus/minus clidata(B)[4] or its grade involution
```

The error message is returned due to the fact that in the case when K = C or K = H, the only flexibility in these two procedures in accepting a different primitive idempotent 'f ' than the one stored under clidata(B)[4] is that 'f' must equal plus or minus the stored idempotent, or its grade involution. In the real case when K = R, this doesn't matter and the user can enter any primitive idempotent.

```
> f:=clidata(B)[4];
```

$$f := \frac{Id}{2} + \frac{e1}{2}$$

```
> evalb(f1=f),evalb(f1=-f),evalb(f1=gradeinv(f)),evalb(f1=-gradeinv(f));
```

$$false, false, false, false$$

This requirement has to do with the fact that both procedures beta_{+} and beta_{-} use pre-computed basis for K which contains two or four elements depending whether K = C or K =

H. Certainly, this basis depends on the choice of the idempotent! In general, such basis for K can be computed for any primitive idempotent f with the procedure Kbasis.

```
> 'K'=clidata()[6];
```

$$K = [Id, e23]$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 1.252000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM with Win XP Professional
>
```

Example 2: Scalar products of spinors of the Clifford algebra Cl(1,3) that is isomorphic to H(2).

```
> restart:bench:=time():with(Clifford):with(linalg):
  B:=linalg[diag](1,-1,-1,-1): #define the bilinear form B for Cl(1,3)
> dim:=coldim(B):eval(makealiases(dim)):
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data := \left[quaternionic, 2, simple, \frac{Id}{2} + \frac{e14}{2}, [Id, e1, e2, e3, e12, e13, e23, e123], [Id, e2, e3, e23], [Id, e1]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own here
> for i from 1 to nops(data[7]) do f||i:=data[7][i] &c f od;
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in clude &C and &C[K]. Type ?cliprod for help.
```

$$f1 := \frac{Id}{2} + \frac{e14}{2}$$

$$f2 := \frac{e1}{2} + \frac{e4}{2}$$

Elements f1 and f2 computed above give a K-basis in S = Cl(1,3)f where K = span{Id,e2,e3,e2we3} is a subalgebra of Cl(1,3) isomorphic with the quaternionic ring H. The elements spanning K are stored in data[6] displayed above. One can really check that these elements form a quaternionic basis and form a list of `type/genquatbasis`:

```
> Kbasis:=data[6]; #here K = H
```

$$Kbasis := [Id, e2, e3, e23]$$

```
> type(Kbasis,genquatbasis);
```

$$true$$

It is also possible to check it directly by computing all possible products and then displaying them in a matrix:

```
> M:=matrix(4,4,(i,j)->cmul(Kbasis[i],Kbasis[j]));
```

$$M := \begin{bmatrix} Id & e2 & e3 & e23 \\ e2 & -Id & e23 & -e3 \\ e3 & -e23 & -Id & e2 \\ e23 & e3 & -e2 & -Id \end{bmatrix}$$

Like in the example above, we first define arbitrary quaternionic coefficients for two spinors 'psi' and 'phi' in S = Cl(1,3)f:

```
> psi1:=psi11 * Id + psi12 * e2 + psi13 * e3 + psi14 * e23;#first coefficient of psi in K
  psi2:=psi21 * Id + psi22 * e2 + psi23 * e3 + psi24 * e23;#second coefficient of psi in K
```

$$\psi 1 := \psi 11\, Id + \psi 12\, e2 + \psi 13\, e3 + \psi 14\, e23$$

$$\psi 2 := \psi 21\, Id + \psi 22\, e2 + \psi 23\, e3 + \psi 24\, e23$$

```
> phi1:=phi11 * Id + phi12 * e2 + phi13 * e3 + phi14 * e23;#first coefficient of phi in K
  phi2:=phi21 * Id + phi22 * e2 + phi23 * e3 + phi24 * e23;#second coefficient of phi in K
```

$$\phi 1 := \phi 11\, Id + \phi 12\, e2 + \phi 13\, e3 + \phi 14\, e23$$

$$\phi 2 := \phi 21\, Id + \phi 22\, e2 + \phi 23\, e3 + \phi 24\, e23$$

Now, we define spinors 'psi' and 'phi':

```
> psi:='f1 &c psi1' + 'f2 &c psi2'; #here psi1 ... are quaternionic components of psi
  phi:='f1 &c phi1' + 'f2 &c phi2'; #here phi1 ... are quaternionic components of phi
```

$$\psi := Cliplus{:}\text{-}climul(f1, \psi 1) + Cliplus{:}\text{-}climul(f2, \psi 2)$$

$$\phi := Cliplus{:}\text{-}climul(f1, \phi 1) + Cliplus{:}\text{-}climul(f2, \phi 2)$$

The beta_{+} and the beta_{-} forms on S x S are invariant under the symplectic group Sp(2,2) as can be checked directly from the output below:

```
> x:=beta_plus(psi,phi,f,'purespinor1');purespinor1; #beta_plus is invariant under Sp(2,2)
```

$$x := (\psi 11\, \phi 21 + \psi 23\, \phi 13 + \psi 22\, \phi 12 + \psi 21\, \phi 11 + \psi 12\, \phi 22 + \psi 13\, \phi 23 + \psi 24\, \phi 14 + \psi 14\, \phi 24)\, Id + (-\psi 22\, \phi 11 - \psi 12\, \phi 21 + \psi 21\, \phi 12 - \psi 23\, \phi 14 + \psi 11\, \phi 22 + \psi 24\, \phi 13 + \psi 14\, \phi 23 - \psi 13\, \phi 24)\, e2 + (\psi 11\, \phi 23 + \psi 21\, \phi 13 - \psi 23\, \phi 11 - \psi 24\, \phi 12 + \psi 12\, \phi 24 + \psi 22\, \phi 14 - \psi 13\, \phi 21 - \psi 14\, \phi 22)\, e3 + (-\psi 24\, \phi 11 + \psi 23\, \phi 12 - \psi 22\, \phi 13 + \psi 21\, \phi 14 - \psi 12\, \phi 23 + \psi 11\, \phi 24 - \psi 14\, \phi 21 + \psi 13\, \phi 22)\, e23$$

$$e1$$

```
> y:=beta_minus(psi,phi,f,'purespinor2');purespinor2; #beta_minus is invariant under Sp(2,2)
```

$$y := (-\psi 12\,\phi 22 - \psi 13\,\phi 23 - \psi 24\,\phi 14 + \psi 23\,\phi 13 + \psi 22\,\phi 12 - \psi 21\,\phi 11 + \psi 11\,\phi 21 + \psi 14\,\phi 24)\,Id + (-\psi 21\,\phi 12 - \psi 24\,\phi 13 + \psi 12\,\phi 21 + \psi 11\,\phi 22 - \psi 22\,\phi 11 - \psi 23\,\phi 14 + \psi 13\,\phi 24 + \psi 14\,\phi 23)\,e2 + (-\psi 21\,\phi 13 + \psi 22\,\phi 14 + \psi 11\,\phi 23 + \psi 13\,\phi 21 + \psi 24\,\phi 12 - \psi 23\,\phi 11 - \psi 12\,\phi 24 - \psi 14\,\phi 22)\,e3 + (\psi 23\,\phi 12 - \psi 13\,\phi 22 - \psi 22\,\phi 13 + \psi 24\,\phi 11 + \psi 11\,\phi 24 - \psi 14\,\phi 21 - \psi 21\,\phi 14 + \psi 12\,\phi 23)\,e23$$

$$e1$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 2.172000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

Example 3: Let's compute the two bilinear forms beta_{+} and beta_{-} on S = Cl(Q)f where Cl(Q)=Cl(2,3) (complexification of Cl(1,3)) is the Clifford algebra of the de Sitter space of signature (2,3). Cl(2,3) is isomorphic to C(4) and is a complexification of Cl(1,3).

```
> restart:bench:=time():with(Clifford):dim:=5:eval(makealiases(di
  m)):
> B:=linalg[diag](1,1,-1,-1,-1):#define form B for Cl(2,3)
> data:=clidata(B); #retrieve and display data about Cl(2,3)
```

$$data := \left[complex, 4, simple, 'cmulQ'\left(\frac{Id}{2} + \frac{e14}{2}, \frac{Id}{2} + \frac{e25}{2}\right), [Id, e1, e2, e3, e12, e13, e23, e123], [Id, e3], [Id, e1, e2, e12]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own
  here
```

Here are the matrices representing basis 1-monomials of Cl(2,3):

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

$$\left[e1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, e2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}, e3 = \begin{bmatrix} e3 & 0 & 0 & 0 \\ 0 & -e3 & 0 & 0 \\ 0 & 0 & -e3 & 0 \\ 0 & 0 & 0 & e3 \end{bmatrix}, e4 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, e5 = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}\right]$$

```
for i from 1 to 4 do f||i:=data[7][i] &c f od;
```

$$f1 := \frac{Id}{4} + \frac{e14}{4} + \frac{e25}{4} - \frac{e1245}{4}$$

$$f2 := \frac{e4}{4} + \frac{e1}{4} + \frac{e125}{4} - \frac{e245}{4}$$

$$f3 := -\frac{e124}{4} + \frac{e145}{4} + \frac{e5}{4} + \frac{e2}{4}$$

$$f4 := \frac{e12}{4} + \frac{e15}{4} - \frac{e24}{4} + \frac{e45}{4}$$

Thus, a spinor psi from S=Cl(2,3)f is a four component vector with real entries. It can be written in terms of the basis [f1,f2,f3,f4] and its complex entries in K = <Id>_R +<e3>_R as follows:

```
> psi1:=psi11+psi12*e3:psi2:=psi21+psi22*e3:psi3:=psi31+psi32*e3:
  psi4:=psi41+psi42*e3:
> phi1:=phi11+phi12*e3:phi2:=phi21+phi22*e3:phi3:=phi31+phi32*e3:
  phi4:=phi41+phi42*e3:
```

Let's define now arbitrary spinors 'psi' and 'phi' in S considered as the RIGHT K-module:

```
> psi:='f1 &c psi1' + 'f2 &c psi2' + 'f3 &c psi3' + 'f4 &c psi4';

  phi:='f1 &c phi1' + 'f2 &c phi2' + 'f3 &c phi3' + 'f4 &c phi4';
```

$$\psi := Cliplus\text{:-}climul(f1, \psi 1) + Cliplus\text{:-}climul(f2, \psi 2) + Cliplus\text{:-}climul(f3, \psi 3) + Cliplus\text{:-}climul(f4, \psi 4)$$

$$\phi := Cliplus\text{:-}climul(f1, \phi 1) + Cliplus\text{:-}climul(f2, \phi 2) + Cliplus\text{:-}climul(f3, \phi 3) + Cliplus\text{:-}climul(f4, \phi 4)$$

The two scalar forms beta_{+} and beta_{-} are:

```
> x:=beta_plus(psi,phi,f,'s1');s1; #beta_plus is invariant under
  Sp(4,C)
```

$$x := (\psi 22\, \phi 32 + \psi 12\, \phi 42 - \psi 11\, \phi 41 - \psi 32\, \phi 22 + \psi 41\, \phi 11 - \psi 21\, \phi 31 - \psi 42\, \phi 12 + \psi 31\, \phi 21)\, Id + (\psi 31\, \phi 22 + \psi 42\, \phi 11 + \psi 32\, \phi 21 - \psi 22\, \phi 31 - \psi 21\, \phi 32 - \psi 12\, \phi 41 - \psi 11\, \phi 42 + \psi 41\, \phi 12)\, e3\, e12$$

```
> y:=beta_minus(psi,phi,f,'s2');s2; #beta_minus is invariant
  under U(2,2)
```

$$y := (\psi 22\, \phi 32 + \psi 41\, \phi 11 - \psi 11\, \phi 41 + \psi 21\, \phi 31 - \psi 12\, \phi 42 - \psi 31\, \phi 21 - \psi 32\, \phi 22 + \psi 42\, \phi 12)\, Id + (\psi 21\, \phi 32 + \psi 12\, \phi 41 + \psi 32\, \phi 21 - \psi 31\, \phi 22 + \psi 41\, \phi 12 - \psi 22\, \phi 31 - \psi 42\, \phi 11 - \psi 11\, \phi 42)\, e3\, e12$$

We can verify that the automorphism group of beta_{-} is U(2,2) as follows:

```
> psi1c:=gradeinv(psi1):psi2c:=gradeinv(psi2):psi3c:=gradeinv(psi
```

```
3):psi4c:=gradeinv(psi4):
> y1:=clicollect(cmul(psi4c,phi1)+cmul(psi2c,phi3)-cmul(psi3c,phi
2)-cmul(psi1c,phi4));
```

$$y1 := -(-\psi22\,\phi32 - \psi41\,\phi11 + \psi11\,\phi41 - \psi21\,\phi31 + \psi12\,\phi42 + \psi31\,\phi21 + \psi32\,\phi22 - \psi42\,\phi12)\,Id + (\psi21\,\phi32 + \psi12\,\phi41 + \psi32\,\phi21 - \psi31\,\phi22 + \psi41\,\phi12 - \psi22\,\phi31 - \psi42\,\phi11 - \psi11\,\phi42)\,e3$$

```
> simplify(y1-y);
```

$$0$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 7.127000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

Example 4: Scalar products in the spinor spaces of the Clifford algebra Cl(3,1), that is isomorphic to R(4), are both invariant under the symplectic group Sp(4).

```
> restart:bench:=time():with(Clifford):dim:=4:eval(makealiases(di
m)):
> B:=linalg[diag](1,1,1,-1):#define form B for Cl(3,1)
> data:=clidata(B); #retrieve and display data about Cl(3,1)
```

$$data := \left[real, 4, simple, 'cmulQ'\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e34}{2}\right), [Id, e2, e3, e23], [Id], [Id, e2, e3, e23]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own
here
```

Here are the matrices representing some basis monomials of Cl(B). Those that represent the 1-vectors are usually referred to as *Dirac gamma matrices* (since 'gamma' is a protected Maple name, we will use here 'g' instead):

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

$$\left[e1 = \begin{bmatrix}1&0&0&0\\0&-1&0&0\\0&0&-1&0\\0&0&0&1\end{bmatrix}, e2 = \begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}, e3 = \begin{bmatrix}0&0&1&0\\0&0&0&-1\\1&0&0&0\\0&-1&0&0\end{bmatrix}, e4 = \begin{bmatrix}0&0&-1&0\\0&0&0&1\\1&0&0&0\\0&-1&0&0\end{bmatrix}\right]$$

Let's find matrices representing the four basis vectors in S:

```
> for i from 1 to 4 do f||i:=data[7][i] &c f od;
```

$$f1 := \frac{Id}{4}+\frac{e134}{4}+\frac{e1}{4}+\frac{e34}{4}$$

$$f2 := -\frac{e12}{4}+\frac{e2}{4}+\frac{e234}{4}-\frac{e1234}{4}$$

$$f3 := -\frac{e13}{4} + \frac{e3}{4} - \frac{e14}{4} + \frac{e4}{4}$$

$$f4 := \frac{e23}{4} + \frac{e123}{4} + \frac{e124}{4} + \frac{e24}{4}$$

Thus, a spinor psi from S=Cl(3,1)f is a four component vector with real entries. It can be written in terms of the basis [f1,f2,f3,f4] and its real entries in K = <Id>_R as follows:

```
> psi:= psi1 * 'f1' + psi2 * 'f2' + psi3 * 'f3' + psi4 * 'f4';
  phi:= phi1 * 'f1' + phi2 * 'f2' + phi3 * 'f3' + phi4 * 'f4';
```

$$\psi := \psi 1\, f1 + \psi 2\, f2 + \psi 3\, f3 + \psi 4\, f4$$

$$\phi := \phi 1\, f1 + \phi 2\, f2 + \phi 3\, f3 + \phi 4\, f4$$

```
> x:=beta_plus(psi,phi,f,'s1');s1; #automorphism group of
  beta_plus is Sp(4,R)
```

$$x := (\psi 3\, \phi 2 - \psi 1\, \phi 4 - \psi 2\, \phi 3 + \psi 4\, \phi 1)\, Id$$

$$e23$$

```
> y:=beta_minus(psi,phi,f,'s2');s2; #automorphism group of
  beta_minus is Sp(4,R)
```

$$y := (\psi 1\, \phi 3 + \psi 4\, \phi 2 - \psi 2\, \phi 4 - \psi 3\, \phi 1)\, Id$$

$$e3$$

Since the coefficients of 'psi' and 'phi' are here assumed to be real, it doesn't matter that they were written on the left side of the basis elements [f1,f2,f3,f4].

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 1.876000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

Example 5: Let's consider finally the semi-simple case of Cl(2,1) = Mat(2,R) + Mat(2,R). Here, we will have to deal with pairs of spinor spinor spaces S and S^ where S^ is a spinor space of spinors of the form gradeinv(psi) where gradeinv is the grade involution in Cl(2,1) and 'psi' is a spinor from S = Cl(1,2)f.

```
> restart:bench:=time():with(Clifford):with(linalg):_shortcut_in_
  spinorKrepr:=false:
> B:=diag(1$2,-1$1):dim:=coldim(B):eval(makealiases(dim)):clibasi
  s:=cbasis(dim):
```

Display information stored:

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

$$\left[e1 = \begin{bmatrix} [1, -1] & [0, 0] \\ [0, 0] & [-1, 1] \end{bmatrix}, e2 = \begin{bmatrix} [0, 0] & [1, 1] \\ [1, 1] & [0, 0] \end{bmatrix}, e3 = \begin{bmatrix} [0, 0] & [-1, -1] \\ [1, 1] & [0, 0] \end{bmatrix}\right]$$

```
> data:=clidata();f:=data[4];
```

$$data := \left[real, 2, semisimple, 'Clifford{:}\text{-}cmulQ'\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e23}{2}\right), [Id, e2], [Id], [Id, e2]\right]$$

$$f := Clifford{:}\text{-}cmulQ\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e23}{2}\right)$$

We will need now one spinor basis for S and for S^:

```
> K_basis1:=[seq(data[7][i] &c f,i=1..nops(data[7]))];#spinor
  basis in S
  K_basis2:=map(gradeinv,K_basis1);#spinor basis in gradeinv(S)
  or S hat
```

$$K\_basis1 := \left[\frac{e23}{4}+\frac{Id}{4}+\frac{e1}{4}+\frac{e123}{4}, -\frac{e13}{4}-\frac{e12}{4}+\frac{e2}{4}+\frac{e3}{4}\right]$$

$$K\_basis2 := \left[\frac{e23}{4}+\frac{Id}{4}-\frac{e1}{4}-\frac{e123}{4}, -\frac{e13}{4}-\frac{e12}{4}-\frac{e2}{4}-\frac{e3}{4}\right]$$

```
> FBgens1:=data[6];FBgens2:=map(gradeinv,FBgens1);#generators for
  K and K hat
```

$$FBgens1 := [Id]$$

$$FBgens2 := [Id]$$

Although the basis elements e1, e2, e3 already have their matrices with entries in R+R pre-computed and displayed above, we show how they were computed. First, we need to find pairs of matrices representing e1, e2, and e3 respectively

```
> for i from 1 to dim do
  M[i],N[i]:=spinorKrepr(e||i,K_basis1,FBgens1,'left'),spinorKrep
  r(e||i,K_basis2,FBgens2,'left');
  od;
```

$$M_1, N_1 := \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$M_2, N_2 := \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$$

$$M_3, N_3 := \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

By combining them pairwise, we get matrices representing faithfully e1, e2, and e3 in Mat(2,2,R+R). This operation can be done with cdfmatrix:

```
> E[1]:=cdfmatrix(M[1],N[1]); #matrix for e1
```

$$E_1 := \begin{bmatrix} [1, -1] & [0, 0] \\ [0, 0] & [-1, 1] \end{bmatrix}$$

```
> E[2]:=cdfmatrix(M[2],N[2]); #matrix for e2
```

$$E_2 := \begin{bmatrix} [0, 0] & [1, -1] \\ [1, -1] & [0, 0] \end{bmatrix}$$

```
> E[3]:=cdfmatrix(M[3],N[3]); #matrix for e3
```

$$E_3 := \begin{bmatrix} [0,0] & [-1,1] \\ [1,-1] & [0,0] \end{bmatrix}$$

Notice that these matrices observe appropriate commutation relations:

```
> mdfmatrix(E[1],E[1]); #square of E[1] or e1
```

$$\begin{bmatrix} [1,1] & [0,0] \\ [0,0] & [1,1] \end{bmatrix}$$

```
> mdfmatrix(E[2],E[2]); #square of E[2] or e2
```

$$\begin{bmatrix} [1,1] & [0,0] \\ [0,0] & [1,1] \end{bmatrix}$$

```
> mdfmatrix(E[3],E[3]); #square of E[3] or e3
```

$$\begin{bmatrix} [-1,-1] & [0,0] \\ [0,0] & [-1,-1] \end{bmatrix}$$

```
> adfmatrix(mdfmatrix(E[1],E[2]),mdfmatrix(E[2],E[1])); #product
  cmul(e1,e2)+cmul(e2,e1)
```

$$\begin{bmatrix} [0,0] & [0,0] \\ [0,0] & [0,0] \end{bmatrix}$$

```
> adfmatrix(mdfmatrix(E[1],E[3]),mdfmatrix(E[3],E[1])); #product
  cmul(e1,e3)+cmul(e3,e1)
```

$$\begin{bmatrix} [0,0] & [0,0] \\ [0,0] & [0,0] \end{bmatrix}$$

```
> adfmatrix(mdfmatrix(E[2],E[3]),mdfmatrix(E[3],E[2])); #product
  cmul(e2,e3)+cmul(e3,e2)
```

$$\begin{bmatrix} [0,0] & [0,0] \\ [0,0] & [0,0] \end{bmatrix}$$

```
>
```

Let's make assignment of the basis elements for S and S^:

```
> f1:=K_basis1[1];f2:=K_basis1[2]; #basis in S
```

$$f1 := \frac{e23}{4} + \frac{Id}{4} + \frac{e1}{4} + \frac{e123}{4}$$

$$f2 := -\frac{e13}{4} - \frac{e12}{4} + \frac{e2}{4} + \frac{e3}{4}$$

```
> g1:=K_basis2[1];g2:=-K_basis2[2]; #basis in S hat
```

$$g1 := \frac{e23}{4} + \frac{Id}{4} - \frac{e1}{4} - \frac{e123}{4}$$

$$g2 := \frac{e13}{4} + \frac{e12}{4} + \frac{e2}{4} + \frac{e3}{4}$$

That is, S = span{f1,f2} and S^ = span{g1,g2}. Thus, spinors in S = Cl(Q)f = R^2 can be displayed as follows:

```
> psi:='f1' &c (psi1 &c Id) + 'f2' &c (psi2 &c Id);
  phi:='f1' &c (phi1 &c Id) + 'f2' &c (phi2 &c Id);
```

$$\psi := \frac{\psi 1\, e23}{4} + \frac{\psi 1\, Id}{4} + \frac{\psi 1\, e1}{4} + \frac{\psi 1\, e123}{4} - \frac{\psi 2\, e13}{4} - \frac{\psi 2\, e12}{4} + \frac{\psi 2\, e2}{4} + \frac{\psi 2\, e3}{4}$$

$$\phi := \frac{\phi 1\, e23}{4} + \frac{\phi 1\, Id}{4} + \frac{\phi 1\, e1}{4} + \frac{\phi 1\, e123}{4} - \frac{\phi 2\, e13}{4} - \frac{\phi 2\, e12}{4} + \frac{\phi 2\, e2}{4} + \frac{\phi 2\, e3}{4}$$

The matrix representation of 'psi' and 'phi' will then be:

```
> matKrepr(psi),matKrepr(phi);
```

$$\begin{bmatrix} [\psi 1, 0] & [0, 0] \\ [\psi 2, 0] & [0, 0] \end{bmatrix}, \begin{bmatrix} [\phi 1, 0] & [0, 0] \\ [\phi 2, 0] & [0, 0] \end{bmatrix}$$

The two scalar forms in S are then:

```
> beta_plus(psi,phi,f1,'s1');s1; #beta_plus is invariant under
  GL(2,R)
```

$$0$$

$$s1$$

```
> beta_minus(psi,phi,f1,'s2');s2; #beta_minus is invariant under
  Sp(2,R)
```

$$(-\psi 2\, \phi 1 + \psi 1\, \phi 2)\, Id$$

$$e2$$

Verification:

```
> out1:=clicollect(s2 &c conjugation(psi) &c phi &c f1);
```

$$out1 := -\frac{(\psi 2\, \phi 1 - \psi 1\, \phi 2)\, e23}{4} - \frac{(\psi 2\, \phi 1 - \psi 1\, \phi 2)\, Id}{4} - \frac{(\psi 2\, \phi 1 - \psi 1\, \phi 2)\, e1}{4} - \frac{(\psi 2\, \phi 1 - \psi 1\, \phi 2)\, e123}{4}$$

```
> psiphi:=psi1*phi2-psi2*phi1;
```

$$psiphi := -\psi 2\, \phi 1 + \psi 1\, \phi 2$$

```
> out2:=clicollect(expand(psiphi*f1));
```

$$out2 := -\frac{(\psi 2\, \phi 1 - \psi 1\, \phi 2)\, e23}{4} - \frac{(\psi 2\, \phi 1 - \psi 1\, \phi 2)\, Id}{4} - \frac{(\psi 2\, \phi 1 - \psi 1\, \phi 2)\, e1}{4} - \frac{(\psi 2\, \phi 1 - \psi 1\, \phi 2)\, e123}{4}$$

```
> out1-out2;
```

$$0$$

Spinors in S^ = (Cl(Q)f)^ = R^2 can be displayed as follows::

```
> psi:='g1' &c (psi1 &c Id) + 'g2' &c (psi2 &c Id);
  phi:='g1' &c (phi1 &c Id) + 'g2' &c (phi2 &c Id);
```

$$\psi := \frac{\psi 1\, e23}{4} + \frac{\psi 1\, Id}{4} - \frac{\psi 1\, e1}{4} - \frac{\psi 1\, e123}{4} + \frac{\psi 2\, e13}{4} + \frac{\psi 2\, e12}{4} + \frac{\psi 2\, e2}{4} + \frac{\psi 2\, e3}{4}$$

$$\phi := \frac{\phi 1\, e23}{4} + \frac{\phi 1\, Id}{4} - \frac{\phi 1\, e1}{4} - \frac{\phi 1\, e123}{4} + \frac{\phi 2\, e13}{4} + \frac{\phi 2\, e12}{4} + \frac{\phi 2\, e2}{4} + \frac{\phi 2\, e3}{4}$$

```
> matKrepr(psi),matKrepr(phi);
```

$$\begin{bmatrix} [0, \psi 1] & [0, 0] \\ [0, \psi 2] & [0, 0] \end{bmatrix}, \begin{bmatrix} [0, \phi 1] & [0, 0] \\ [0, \phi 2] & [0, 0] \end{bmatrix}$$

The scalar forms then are:

```
> beta_plus(psi,phi,g1); #beta_plus is invariant under GL(2,R)
```

$$0$$

```
> beta_minus(psi,phi,g1,'s'); #invariant under Sp(2,R)
```

$$(-\psi 2\ \phi 1 + \psi 1\ \phi 2)\ Id$$

Verification:

```
> s; #purespinor
```

$$e2$$

```
> out1:=clicollect(s &c conjugation(psi) &c phi &c g1); #all in S
  hat
```

$$out1 := -\frac{(\psi 2\ \phi 1 - \psi 1\ \phi 2)\ e23}{4} - \frac{(\psi 2\ \phi 1 - \psi 1\ \phi 2)\ Id}{4} + \frac{(\psi 2\ \phi 1 - \psi 1\ \phi 2)\ e1}{4} + \frac{(\psi 2\ \phi 1 - \psi 1\ \phi 2)\ e123}{4}$$

```
> out2:=clicollect(expand(psiphi*g1));
```

$$out2 := -\frac{(\psi 2\ \phi 1 - \psi 1\ \phi 2)\ e23}{4} - \frac{(\psi 2\ \phi 1 - \psi 1\ \phi 2)\ Id}{4} + \frac{(\psi 2\ \phi 1 - \psi 1\ \phi 2)\ e1}{4} + \frac{(\psi 2\ \phi 1 - \psi 1\ \phi 2)\ e123}{4}$$

```
> out1-out2;
```

$$0$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 3.732000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

See Also: Clifford:-`type/dfmatrix`, Clifford:-ddfmatrix, Clifford:-cdfmatrix

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-Bsignature - display the signature of the quadratic form Q related to the symmetric part of a bilinear form B

Calling Sequence:

Bsignature();
Bsignature(B);
Bsignature(M);

Parameters:

B - bilinear form in a vector space V
M - any square matrix

Description:

- Procedure 'Bsignature' finds the signature of the bilinear form B, assuming that B has been defined.
- Once B has been defined, 'Bsignature' attempts first to diagonalize the symmetric part of the matrix of B using procedure diagonalize. Then, it tries to find the signum of each diagonal element.
- If B contains symbolic parameters, most likely an error message will be returned as the signum of the such parameters or expressions involving such parameters cannot be determined. However, as examples below show, once the user makes certain assumptions about these parameters, computation of the signum will be proceed and answer will be returned.
- The procedure returns a list with two or three integers depending whether the quadratic form Q defined by the symmetric part of B is non-degenerate or degenerate. If Q is non-degenerate, the list [p,q] gives the number of positive and the number of negative entries in the diagonal form of the symmetric part of B, that is, the signature (p,q) of Q. If Q is degenerate, the returned list [p,q,d] contains three integers where d = dim(rad V).
- Note that B=diag(1,-1) and B=diag(-1,1) will have the same signature [1,1] returned by this procedure.
- Package 'Cliplus' is an additional package with extends functionality of 'CLIFFORD'. To find out more, go to Clifford:-intro.

Examples:

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

Example 1:

```
> Bsignature(); #testing an error message
Error, (in Clifford:-Bsignature) square matric should be assigned to B first
```

```
> B:=diag(1,1,-1,-1);
```

$$B := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

```
> Bsignature();
```

$$[2, 2]$$

```
> B:=diag(1,1,0);
```

$$B := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

```
> Bsignature();
```

$$[2, 0, 1]$$

```
> e1 &c e1,e2 &c e2,e3 &c e3;
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$Id, Id, 0$$

```
> B:=diag(1,-1);e1 &c e1,e2 &c e2;Bsignature();
```

$$B := \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$Id, -Id$$

$$[1, 1]$$

```
> B:=diag(-1,1);e1 &c e1,e2 &c e2;Bsignature();
```

$$B := \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$-Id, Id$$

$$[1, 1]$$

Example 2: In this example, we show what happens if the given matrix contains symbolic parameters.

```
> B:=matrix(3,3,[1,a,b,a,-2,c,-b,-c,3]);
```

$$B := \begin{bmatrix} 1 & a & b \\ a & -2 & c \\ -b & -c & 3 \end{bmatrix}$$

```
> type(B,symmatrix);
```

$$false$$

Procedure diagonalize can be used to diagonalize a symmetric matrix. However, procedure 'Bsignature' will need that the symbolic parameters be defined using command 'assume'.

```
> diagonalize(B);  ###<<<-Intended error message
Error, invalid input: Clifford:-diagonalize expects its 1st argument, m, to be
of type symmatrix, but received B
```

```
> Bsignature(B); ###<<<- Intended error message
Error, (in Clifford:-Bsignature) unable to determine signs of expressions [1,
```

```
signum(-1/2+1/2*(9+4*a^2)^(1/2)), -signum(1/2+1/2*(9+4*a^2)^(1/2))]
```

```
> assume(a>0);
> about(a);
Originally a, renamed a~:
  is assumed to be: RealRange(Open(0),infinity)
> Bsignature(B);
Error, (in Clifford:-diagonalize) since matrix entered does not have a complete
set of linearly independent eigenvectors, it is not diagonalizable
```

If we erase the assumption about a, 'Bsignature' again yields an error message.

```
> a:='a';
```

$$a := a$$

```
> Bsignature(); #testing an error message
Error, (in Clifford:-Bsignature) unable to determine signs of expressions
[signum(-1/2+1/2*(9+4*a^2)^(1/2)), -signum(1/2+1/2*(9+4*a^2)^(1/2)), 1]
>
```

Example 3: Let's see some more examples.

```
> B:=diag(1,1,-1,-1):
> Bsignature();
```

$$[2, 2]$$

```
> B:=diag(1,1,0):
> Bsignature();
```

$$[2, 0, 1]$$

```
> e1 &c e1,e2 &c e2,e3 &c e3;
```

$$Id, Id, 0$$

```
> B:=diag(1,-1);e1 &c e1,e2 &c e2;Bsignature();
```

$$B := \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$Id, -Id$$

$$[1, 1]$$

```
> B:=diag(-1,1);e1 &c e1,e2 &c e2;Bsignature();
```

$$B := \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$-Id, Id$$

$$[1, 1]$$

```
> B:=matrix(2,2,[1,b,b,-1]);
```

$$B := \begin{bmatrix} 1 & b \\ b & -1 \end{bmatrix}$$

```
> diagonalize(B);
```

$$\begin{bmatrix} \sqrt{1+b^2} & 0 \\ 0 & -\sqrt{1+b^2} \end{bmatrix}$$

Thus, as we see in the above, 'diagonalize' can diagonalize a symmetric, symbolic matrix, but since nothing has been assumed about the parameter b, 'Bsignature' gives an error:

```
> Bsignature(B); ###<<<-Intended error message
Error, (in Clifford:-Bsignature) unable to determine signs of expressions
[signum(1+b^2)^(1/2), -signum(1+b^2)^(1/2)]
```

However, it is enough to assume that b is real:

```
> assume(b,real);
> Bsignature(B);
Error, (in Clifford:-diagonalize) since matrix entered does not have a complete
set of linearly independent eigenvectors, it is not diagonalizable
```

Another symbolic example:

```
> B:=matrix(2,2,[1,c+b,c-b,-1]);
```

$$B := \begin{bmatrix} 1 & c+b\sim \\ c-b\sim & -1 \end{bmatrix}$$

```
> diagonalize(B);  ###<<<-Intended error message
Error, invalid input: Clifford:-diagonalize expects its 1st argument, m, to be
of type symmatrix, but received B
> Bsym:=evalm((B+transpose(B))/2);
```

$$Bsym := \begin{bmatrix} 1 & c \\ c & -1 \end{bmatrix}$$

```
> diagonalize(Bsym);
```

$$\begin{bmatrix} \sqrt{1+c^2} & 0 \\ 0 & -\sqrt{1+c^2} \end{bmatrix}$$

```
> Bsignature(Bsym); ###<<<-Intended error message
Error, (in Clifford:-Bsignature) unable to determine signs of expressions
[signum(1+c^2)^(1/2), -signum(1+c^2)^(1/2)]
> assume(c>0);
> Bsignature(Bsym);
Error, (in Clifford:-diagonalize) since matrix entered does not have a complete
set of linearly independent eigenvectors, it is not diagonalizable
>
```

Example 4: Some more examples:

```
> a:='a':b:='b':c:='c':
> B:=matrix(3,3,[1,a,b,-a,-2,c,-b,-c,3]);
  Bsignature(B);
```

$$B := \begin{bmatrix} 1 & a & b \\ -a & -2 & c \\ -b & -c & 3 \end{bmatrix}$$

$$[2, 1]$$

```
> Bsym:=evalm((B+transpose(B))/2);
```

$$Bsym := \begin{bmatrix} 1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

```
> M:=diagonalize(Bsym);
```

$$M := \begin{bmatrix} 1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

```
> Bsignature(Bsym);
```

$$[2, 1]$$

```
> B:=matrix(2,2,[0,1,0,0]);
```

$$B := \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

```
> diagonalize(B); ###<<<- Intended error message
Error, invalid input: Clifford:-diagonalize expects its 1st argument, m, to be
of type symmatrix, but received B
```

```
> B:=matrix(2,2,[0,1/2,1/2,1]);
```

$$B := \begin{bmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 1 \end{bmatrix}$$

```
> diagonalize(B);
```

$$\begin{bmatrix} \frac{1}{2} + \frac{\sqrt{2}}{2} & 0 \\ 0 & \frac{1}{2} - \frac{\sqrt{2}}{2} \end{bmatrix}$$

```
> Bsignature(B);
```

$$[1, 1]$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 2.625000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

See Also: Clifford:-version, Clifford:-diagonalize, Clifford:-`type/symmatrix`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-buildm - find a matrix representation of a Clifford polynomial in some basis

Calling Sequence:

buildm(p,L);
buildm(p,L,s);
buildm(p,L,s1,s2);

Parameters:

p - expression of the type 'cliscalar', 'clibasmon, 'climon', or 'clipolynom'
L - a list of basis elements
s - (optional) element of type symbol: 'false' or 'true', or 'left', 'right', 'Lie' or 'auto'
s1 - (optional) element of type symbol: 'left', 'right', 'Lie' or 'auto'
s2 - (optional) element of type symbol: 'true' or 'false'

Description:

- Procedure 'buildm' builds a matrix for the given element p of the Clifford algebra Cl(B) using one of the actions 'left' (default), 'right', 'Lie', or 'auto' with respect to an ordered basis specified as the list L.
- The element p is entered as the first argument and the basis in the form of a list is specified as the second argument. The elements in the list L are checked for linear independence by default. If they are linearly dependent, an error message is returned. Use procedure findbasis to reduce L to a linearly independent set.
- When only two arguments p and L are used, the 'left' regular action is assumed by default and the matrix of p in the left regular representation is found provided elements in the list L are linearly independent.
- When used with three arguments p, L, and s, with the last (optional) argument s being either 'false' or 'true', user may explicitly demand that the list L not be checked or be checked for linear independence, respectively. However, if the elements in L happen to be dependent, unpredictable output will result.
- If the third optional parameter s is 'left', 'right', 'Lie' or 'auto', the call to buildm(p,L,s) returns matrix of p in:
  (a) the left regular representation when s = 'left': p --> g(p), g(p): u --> pu, where u is any element in the span of L;
  (b) the right regular representation when s = 'right': p --> g(p), g(p): u --> up, where u is any element in the span of L;
  (c) the Lie action when s = 'Lie': p --> g(p), g(p): u --> [p,u], where u is any element in the span of L and [. , .] is the Lie bracket;
  (d) the automorphism action when s = 'auto': p --> g(p), g(p): u --> pup^(-1), where u is any element in the span of L and p^(-1) denotes the inverse of p.

- For example, one can find the left-regular representation of the algebra on itself or, when Cl(B) is simple and isomorphic to a ring of real matrices, one can find matrices representing Clifford polynomials in a real basis of a minimal ideal.
- There are new procedures specifically designed for finding spinor representations of orthogonal Clifford algebras in terms of real, complex, and quaternionic matrices such as 'minimalideal', 'Kfield', 'spinorKrepr', 'matKrepr', 'spinorKbasis'. See spinorKbasis, matKrepr, minimalideal, Kfield, spinorKrepr.
- To multiply matrices with entries in a Clifford algebra use procedure rmulm.
- Clifford product is given by the procedure cmul.
- The warning messages about the value of environmental variable dim_V being reduced are created by the procedure wedge. For more information about CLIFFORD's environmental variables, see CLIFFORD_ENV.

Examples:

```
> restart:bench:=time():
  with(Clifford):with(linalg):
  _default_Clifford_product;
  useproduct(cmulNUM);
  _default_Clifford_product;
```

Clifford:-cmulRS

```
Warning, cmul will use cmulNUM; for help see pages ?cmul, ?Clifford:-intro, or
?cmulNUM
```

Clifford:-cmulNUM

Example 1: Let's see various ways if using this procedure.

```
> M1:=buildm(e1,[Id,e1,e2,e1we2]); #left regular representation
  of e1 (default)
  M2:=buildm(e2,[Id,e1,e2,e1we2]); #left regular representation
  of e2 (default)
  clicollect(cmul(e1,e2)+cmul(e2,e1));
  evalm(M1 &* M2 + M2 &* M1);
```

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

$$M1 := \begin{bmatrix} 0 & B_{1,1} & B_{1,2} & 0 \\ 1 & 0 & 0 & -B_{1,2} \\ 0 & 0 & 0 & B_{1,1} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$M2 := \begin{bmatrix} 0 & B_{2,1} & B_{2,2} & 0 \\ 0 & 0 & 0 & -B_{2,2} \\ 1 & 0 & 0 & B_{2,1} \\ 0 & -1 & 0 & 0 \end{bmatrix}$$

$$(B_{1,2} + B_{2,1})\, Id$$

$$\begin{bmatrix} B_{1,2} + B_{2,1} & 0 & 0 & 0 \\ 0 & B_{1,2} + B_{2,1} & 0 & 0 \\ 0 & 0 & B_{1,2} + B_{2,1} & 0 \\ 0 & 0 & 0 & B_{1,2} + B_{2,1} \end{bmatrix}$$

```
> M1:=buildm(e1,[Id,e1,e2,e1we2],'left'); #left regular
representation of e1 (explicit call)
M2:=buildm(e2,[Id,e1,e2,e1we2],'left'); #left regular
representation of e2 (explicit call)
```

$$M1 := \begin{bmatrix} 0 & B_{1,1} & B_{1,2} & 0 \\ 1 & 0 & 0 & -B_{1,2} \\ 0 & 0 & 0 & B_{1,1} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$M2 := \begin{bmatrix} 0 & B_{2,1} & B_{2,2} & 0 \\ 0 & 0 & 0 & -B_{2,2} \\ 1 & 0 & 0 & B_{2,1} \\ 0 & -1 & 0 & 0 \end{bmatrix}$$

```
> M1:=buildm(e1,[Id,e1,e2,e1we2],'right'); #right regular
representation of e1 (default)
M2:=buildm(e2,[Id,e1,e2,e1we2],'right'); #right regular
representation of e2 (default)
clicollect(cmul(e1,e2)+cmul(e2,e1));
evalm(M1 &* M2 + M2 &* M1);
```

$$M1 := \begin{bmatrix} 0 & B_{1,1} & B_{2,1} & 0 \\ 1 & 0 & 0 & B_{2,1} \\ 0 & 0 & 0 & -B_{1,1} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$M2 := \begin{bmatrix} 0 & B_{1,2} & B_{2,2} & 0 \\ 0 & 0 & 0 & B_{2,2} \\ 1 & 0 & 0 & -B_{1,2} \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$(B_{1,2} + B_{2,1})\, Id$$

$$\begin{bmatrix} B_{1,2}+B_{2,1} & 0 & 0 & 0 \\ 0 & B_{1,2}+B_{2,1} & 0 & 0 \\ 0 & 0 & B_{1,2}+B_{2,1} & 0 \\ 0 & 0 & 0 & B_{1,2}+B_{2,1} \end{bmatrix}$$

```
> M12:=buildm(e1we2,[Id,e1,e2,e1we2],'true'); #left regular representation of e1we2 (default)
```

$$M12 := \begin{bmatrix} 0 & 0 & 0 & B_{2,1}B_{1,2}-B_{1,1}B_{2,2} \\ 0 & B_{2,1} & B_{2,2} & 0 \\ 0 & -B_{1,1} & -B_{1,2} & 0 \\ 1 & 0 & 0 & B_{2,1}-B_{1,2} \end{bmatrix}$$

Testing an error message:

```
> buildm(e1we2,[Id,e1,e2,e1+e2,e1we2],'true'); ###<<<- Intended error message
Error, (in Clifford:-buildm) elements of the list [Id, e1, e2, e1+e2, e1we2] are linearly dependent. Apply 'findbasis' to this list first.
```

```
> L:=findbasis([Id,e1,e2,e1+e2,e1we2]);
```

$$L := [Id, e1, e2, e1we2]$$

When option 'false' is used, no testing for linear independence is done. When elements in the list entered as a second argument are linearly dependent and option 'false' is used', result is unexpected. It shows internal entries for which Maple was unable to find values.

```
> buildm(e1we2,[Id,e1,e2,e1+e2,e1we2],'false'); ###<<<- Bad output due to linear dependence of [...] elements
```

$$[0, 0, 0, 0, B_{2,1}B_{1,2}-B_{1,1}B_{2,2}]$$

$$[xm_{2,1}, xm_{2,2}, xm_{2,3}, xm_{2,4}, xm_{2,5}]$$

$$[xm_{2,1}, -B_{1,1}-B_{2,1}+xm_{2,2}, -B_{1,2}-B_{2,2}+xm_{2,3}, -B_{1,1}+xm_{2,4}-B_{1,2}-B_{2,1}-B_{2,2}, xm_{2,5}]$$

$$[-xm_{2,1}, B_{2,1}-xm_{2,2}, B_{2,2}-xm_{2,3}, B_{2,1}+B_{2,2}-xm_{2,4}, -xm_{2,5}]$$

$$[1, 0, 0, 0, B_{2,1}-B_{1,2}]$$

```
> L:=[Id,e1,e2,e1we2];
```

$$L := [Id, e1, e2, e1we2]$$

```
> buildm(Id,L,'left'),buildm(Id,L,'right'),buildm(Id,L,'Lie'),buildm(Id,L,'auto');
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
> buildm(e1,L,'left'),buildm(e1,L,'right'),buildm(e1,L,'Lie'),buildm(e1,L,'auto');
```

$$\begin{bmatrix} 0 & B_{1,1} & B_{1,2} & 0 \\ 1 & 0 & 0 & -B_{1,2} \\ 0 & 0 & 0 & B_{1,1} \\ 0 & 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & B_{1,1} & B_{2,1} & 0 \\ 1 & 0 & 0 & B_{2,1} \\ 0 & 0 & 0 & -B_{1,1} \\ 0 & 0 & -1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & B_{2,1} - B_{1,2} & 0 \\ 0 & 0 & 0 & B_{2,1} + B_{1,2} \\ 0 & 0 & 0 & -2\,B_{1,1} \\ 0 & 0 & -2 & 0 \end{bmatrix},$$

$$\begin{bmatrix} 1 & 0 & 0 & B_{2,1} - B_{1,2} \\ 0 & 1 & \frac{B_{2,1} + B_{1,2}}{B_{1,1}} & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

```
> buildm(e2,L,'left'),buildm(e2,L,'right'),buildm(e2,L,'Lie'),bui
  ldm(e2,L,'auto');
```

$$\begin{bmatrix} 0 & B_{2,1} & B_{2,2} & 0 \\ 0 & 0 & 0 & -B_{2,2} \\ 1 & 0 & 0 & B_{2,1} \\ 0 & -1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & B_{1,2} & B_{2,2} & 0 \\ 0 & 0 & 0 & B_{2,2} \\ 1 & 0 & 0 & -B_{1,2} \\ 0 & 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & B_{1,2} - B_{2,1} & 0 & 0 \\ 0 & 0 & 0 & 2\,B_{2,2} \\ 0 & 0 & 0 & -B_{1,2} - B_{2,1} \\ 0 & 2 & 0 & 0 \end{bmatrix},$$

$$\begin{bmatrix} 1 & 0 & 0 & B_{2,1} - B_{1,2} \\ 0 & -1 & 0 & 0 \\ 0 & \frac{B_{2,1} + B_{1,2}}{B_{2,2}} & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

```
> buildm(e1we2,L,'left');
  buildm(e1we2,L,'right');
  buildm(e1we2,L,'Lie');
```

$$\begin{bmatrix} 0 & 0 & 0 & B_{2,1}\,B_{1,2} - B_{1,1}\,B_{2,2} \\ 0 & B_{2,1} & B_{2,2} & 0 \\ 0 & -B_{1,1} & -B_{1,2} & 0 \\ 1 & 0 & 0 & B_{2,1} - B_{1,2} \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & B_{2,1}\,B_{1,2} - B_{1,1}\,B_{2,2} \\ 0 & -B_{1,2} & -B_{2,2} & 0 \\ 0 & B_{1,1} & B_{2,1} & 0 \\ 1 & 0 & 0 & B_{2,1} - B_{1,2} \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -B_{1,2} - B_{2,1} & -2\,B_{2,2} & 0 \\ 0 & 2\,B_{1,1} & B_{2,1} + B_{1,2} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

```
> printlevel:=1:buildm(e1we2,L,'auto');
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -\frac{B_{1,1}B_{2,2}-B_{2,1}^{2}}{-B_{2,1}B_{1,2}+B_{1,1}B_{2,2}} & \frac{B_{2,2}(B_{2,1}-B_{1,2})}{-B_{2,1}B_{1,2}+B_{1,1}B_{2,2}} & 0 \\ 0 & -\frac{B_{1,1}(B_{2,1}-B_{1,2})}{-B_{2,1}B_{1,2}+B_{1,1}B_{2,2}} & -\frac{-B_{1,2}^{2}+B_{1,1}B_{2,2}}{-B_{2,1}B_{1,2}+B_{1,1}B_{2,2}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

If the elements in the list are linearly dependent, an error message is returned:

```
> buildm(e1,[Id,e1,e2,e1+e2]); #testing an error message
Error, (in Clifford:-buildm) elements of the list [Id, e1, e2, e1+e2] are
linearly dependent. Apply 'findbasis' to this list first.
```

```
> L:=findbasis([Id,e1,e2,e1+e2]);
```

$$L := [Id, e1, e2]$$

```
> buildm(e1,L,false);
```

$$\begin{bmatrix} 0 & B_{1,1} & B_{1,2} \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

```
> M12:=buildm(e1we2,[Id,e1,e2,e1we2]);
```

$$M12 := \begin{bmatrix} 0 & 0 & 0 & B_{2,1}B_{1,2}-B_{1,1}B_{2,2} \\ 0 & B_{2,1} & B_{2,2} & 0 \\ 0 & -B_{1,1} & -B_{1,2} & 0 \\ 1 & 0 & 0 & B_{2,1}-B_{1,2} \end{bmatrix}$$

```
>
```

Example 2: Finding a matrix representation of an element p in Cl(3) under all four actions on Cl(3) spanned by the Grassmann basis:

```
> B:=diag(1,1,1);
```

$$B := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
> p:=-2+e1+2*e2-4*e1we3+e1we2we3;
```

$$p := -2 + e1 + 2\,e2 - 4\,e1we3 + e1we2we3$$

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

$$Grassmann\_basis := [Id, e1, e2, e3, e1we2, e1we3, e2we3, e1we2we3]$$

```
> buildm(p,Grassmann_basis,'left');
```

$$\begin{bmatrix} -2 & 1 & 2 & 0 & 0 & 4 & 0 & -1 \\ 1 & -2 & 0 & -4 & -2 & 0 & -1 & 0 \\ 2 & 0 & -2 & 0 & 1 & 1 & 0 & -4 \\ 0 & 4 & 0 & -2 & -1 & 1 & 2 & 0 \\ 0 & -2 & 1 & 1 & -2 & 0 & 4 & 0 \\ -4 & 0 & -1 & 1 & 0 & -2 & 0 & -2 \\ 0 & 1 & 0 & 2 & -4 & 0 & -2 & 1 \\ 1 & 0 & 4 & 0 & 0 & -2 & 1 & -2 \end{bmatrix}$$

```
> buildm(p,Grassmann_basis,'right');
```

$$\begin{bmatrix} -2 & 1 & 2 & 0 & 0 & 4 & 0 & -1 \\ 1 & -2 & 0 & 4 & 2 & 0 & -1 & 0 \\ 2 & 0 & -2 & 0 & -1 & 1 & 0 & -4 \\ 0 & -4 & 0 & -2 & -1 & -1 & -2 & 0 \\ 0 & 2 & -1 & 1 & -2 & 0 & -4 & 0 \\ -4 & 0 & -1 & -1 & 0 & -2 & 0 & -2 \\ 0 & 1 & 0 & -2 & 4 & 0 & -2 & 1 \\ 1 & 0 & 4 & 0 & 0 & -2 & 1 & -2 \end{bmatrix}$$

```
> buildm(p,Grassmann_basis,'Lie');
```

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 \\ 0 & -8 & 0 & 0 & 0 & -2 & -4 & 0 \\ 0 & 4 & -2 & 0 & 0 & 0 & -8 & 0 \\ 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -4 & 8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

```
> buildm(p,Grassmann_basis,'auto');
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{-95}{149} & \frac{26}{149} & \frac{122}{149} & \frac{-4}{149} & \frac{-48}{149} & \frac{8}{149} & 0 \\ 0 & \frac{26}{149} & \frac{156}{149} & \frac{-13}{149} & \frac{-24}{149} & \frac{10}{149} & \frac{48}{149} & 0 \\ 0 & \frac{-122}{149} & \frac{13}{149} & \frac{-88}{149} & \frac{-2}{149} & \frac{-24}{149} & \frac{4}{149} & 0 \\ 0 & \frac{-4}{149} & \frac{-24}{149} & \frac{2}{149} & \frac{-88}{149} & \frac{-13}{149} & \frac{-122}{149} & 0 \\ 0 & \frac{48}{149} & \frac{-10}{149} & \frac{-24}{149} & \frac{13}{149} & \frac{156}{149} & \frac{-26}{149} & 0 \\ 0 & \frac{-8}{149} & \frac{-48}{149} & \frac{4}{149} & \frac{122}{149} & \frac{-26}{149} & \frac{-95}{149} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Example 3: Finding spinor representation on Cl(2) in the left and right minimal ideals generated by a primitive idempotent f. For more information, see all_sigs and clidata, and `type/primitiveidemp`.

```
> all_sigs(2..2,'real','simple');
```

$$[[1, 1], [2, 0]]$$

```
> B:=diag(1,1);
```

$$B := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

```
> cdata:=clidata(B);
```

$$cdata := \left[real, 2, simple, \frac{Id}{2} + \frac{e1}{2}, [Id, e2], [Id], [Id, e2] \right]$$

```
> f:=cdata[4];
```

$$f := \frac{Id}{2} + \frac{e1}{2}$$

```
> cmul(f,f) - f;#notice that f is an idempotent
```

$$0$$

```
> L:=[cmul(Id,f),cmul(e2,f)]; #basis for the left minimal ideal
  Cl(2)f
  R:=[cmul(f,Id),cmul(f,e2)]; #basis for the right minimal ideal
  fCl(2)
```

$$L := \left[\frac{Id}{2} + \frac{e1}{2}, -\frac{e1we2}{2} + \frac{e2}{2} \right]$$

$$R := \left[\frac{Id}{2} + \frac{e1}{2}, \frac{e1we2}{2} + \frac{e2}{2} \right]$$

```
> ML1,ML2:=buildm(e1,L,'left'),buildm(e2,L,'left'); #left regular
  spinor representation
  MR1,MR2:=buildm(e1,R,'right'),buildm(e2,R,'right'); #right
  regular spinor representation
```

$$ML1, ML2 := \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$MR1, MR2 := \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

```
> cmul(e1,e1),cmul(e1,e2)+cmul(e2,e1),cmul(e2,e2);
```

$$Id, 0, Id$$

```
> evalm(ML1 &* ML1),evalm(ML1 &* ML2 + ML2 &* ML1),evalm(ML2 &*
  ML2);
  evalm(MR1 &* MR1),evalm(MR1 &* MR2 + MR2 &* MR1),evalm(MR2 &*
  MR2);
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

```
>
```

Example 4: Finding the left regular reducible representation of the generators e1, e2, and e3 on the Clifford algebra Cl(3).

```
> B:=diag(1,1,1):
> Grassmann_basis:=cbasis(3);
```

$$Grassmann\_basis := [Id, e1, e2, e3, e1we2, e1we3, e2we3, e1we2we3]$$

```
> M1,M2:=buildm(e1,Grassmann_basis,'left'),buildm(e2,Grassmann_basis,'left');
  M3:=buildm(e3,Grassmann_basis,'left');
```

$$M1, M2 := \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

$$M3 := \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

```
> cmul(e1,e1),cmul(e2,e2),cmul(e3,e3);
  evalm(M1 &* M1),evalm(M2 &* M2),evalm(M3 &* M3);
```

$$Id, Id, Id$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

```
> cmul(e1,e2)+cmul(e2,e1),cmul(e1,e3)+cmul(e3,e1),cmul(e2,e3)+cmul(e3,e2);
  iszero(M1 &* M2 + M2 &* M1),iszero(M1 &* M3 + M3 &*
  M1),iszero(M2 &* M3 + M3 &* M2);
```

$$0, 0, 0$$

$$\mathit{true}, \mathit{true}, \mathit{true}$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
```

```
Worksheet took 4.559000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
```

```
>
```

See Also: Clifford:-rmulm, Clifford:-spinorKrepr, Clifford:-spinorKbasis, Clifford:-minimalideal, Clifford:-matKrepr, Clifford:-findbasis, Clifford:-Kfield, Clifford:-`type/clipolynom`, Clifford:-`type/climon`, Clifford:-`type/clibasmon`, Clifford:-`type/cliscalar`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-bygrade - sort a list of basis monomials by grade

Calling Sequence:

bygrade(a1,a2);

sort(L, bygrade);

Parameters:

a1, a2 - expressions of type 'clibasmon', 'climon', or 'clipolynom'
L - a list of basis monomials, Clifford monomials, or Clifford polynomials, e.g., of type(list{'clibasmon', 'climon', 'clipolynom'})

Description:

- Function 'bygrade' may be used to sort two basis monomials, Clifford monomials, or Clifford polynomials, or a list of such elements.
- Basis monomials and Clifford monomials are sorted by grade; in case of a tie 'bygrade' sorts by lexicographic order based on the basis monomials. However, Clifford basis monomials are put before Clifford monomials.
- If any of the elements a1 or a2 is a Clifford polynomial, then grade ties are resolved by sorting by the weight of each Clifford polynomial defined as the sum of the grades of each terms, and then by the number of Clifford basis monomials in each expression.
- Procedure returns true or false in each case, and it can be used to sort a list of basis monomials, Clifford monomials, and Clifford polynomials.
- As a sorting function, it returns 'true' if a1 should precede a2 and 'false' if not.

Examples:

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

Example 1: Sorting basis monomials or Clifford monomials:

```
> bygrade(Id,e1we2);
```

$$true$$

```
> bygrade(e2we3,e4);
```

$$false$$

```
> bygrade(2*e1we2we3,e2we3);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$false$$

```
> bygrade(2*e1we2we3,-5*a*e2we3we4);
```

$$true$$

```
> bygrade(e1we2,2*e1we2);
```

$$true$$

```
> bygrade(3*e1we2we3,e1we2we3);
```

$$false$$

```
> bygrade(3*e1we2we3,e1we3we4);
```

$$true$$

Example 2: Sorting basis monomials or Clifford monomials with Clifford polynomials:

```
> bygrade(2*e1we3,e1-2*e1we2);
```

$$true$$

```
> bygrade(Id+e1,e1);
```

$$false$$

```
> bygrade(1+e1we2+4*e1we2we3,e1we2we3we4);
```

$$true$$

Example 3: Sorting two Clifford polynomials:

```
> bygrade(1+e1we2+4*e1we2we3,e1we2we3+e1we3);
```

$$false$$

```
> bygrade(1+e1we2+4*e1we2we3,e1we2we3+e1we3+e1);
```

$$true$$

```
> bygrade(e1-2*e2we3+e1we2we3,e3we4we5+2*e1-e2+e3);
```

$$true$$

Example 4: Sorting a list:

```
> L:=[e4,e3we2,Id,e1we3+2*e1we2,e1,e2we3we4,e2];
```

$$L := [e4, e3we2, Id, e1we3 + 2\, e1we2, e1, e2we3we4, e2]$$

```
> sort(L,bygrade);
```

$$[Id, e1, e2, e4, e3we2, e1we3 + 2\, e1we2, e2we3we4]$$

```
> sort([e4,e3,e2,e1],bygrade);
```

$$[e1, e2, e3, e4]$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 0.091000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

See Also: Clifford:-`type/clipolynom`, Clifford:-`type/cliscalar`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-c_conjug - complex conjugation in a Clifford algebra Cl(B)

Calling Sequence:

c_conjug(p);

Parameters:

p - expression of the type 'cliscalar' or 'clipolynom' or 'matrix'

Description:

- Procedure 'c_conjug' calculates complex conjugation in a complexified Clifford algebra

  c_conjug(u) = c_conjug(a + I*b) = a - I*b

- where a and b are in the real Clifford algebra and I is the imaginary unit, i.e., I = sqrt(-1). This procedure is linear in its argument.
- Note that quaternionic conjugation is computed using q_conjug.
- This procedure can also be applied to matrices with entries in a complexified Clifford algebra in which case complex conjugation is applied to each entry of the matrix.

Examples:

```
> restart:with(Clifford):
> c_conjug((1 + 2*I)*e1 - 3*I*e1we2);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$3\,I\,e1we2 + (1 - 2\,I)\,e1$$

```
> c_conjug((1+I)*e1we2);
```

$$(1 - I)\,e1we2$$

```
> m:=linalg[matrix](2,2,
     [(1+2*I)*e1-3*I*e1we2,(1+2*I)*e1-3*I*e1we2,
      (1- 2*I)*e1we3-3*I*e1we2,(1+2*I)*e1+e1we2]);
```

$$m := \begin{bmatrix} (1 + 2\,I)\,e1 - 3\,I\,e1we2 & (1 + 2\,I)\,e1 - 3\,I\,e1we2 \\ (1 - 2\,I)\,e1we3 - 3\,I\,e1we2 & (1 + 2\,I)\,e1 + e1we2 \end{bmatrix}$$

```
> c_conjug(m);
```

$$\begin{bmatrix} 3\,I\,e1we2 + (1 - 2\,I)\,e1 & 3\,I\,e1we2 + (1 - 2\,I)\,e1 \\ (1 + 2\,I)\,e1we3 + 3\,I\,e1we2 & e1we2 + (1 - 2\,I)\,e1 \end{bmatrix}$$

```
>
```

See Also: Clifford:-q_conjug, Clifford:-conjugation, Clifford:-reversion, Clifford:-gradeinv, Clifford:-`type/clipolynom`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-cbasis - define a standard Clifford basis in the Clifford algebra Cl(V)

Calling Sequence:

cbasis(n);
cbasis(n, k);
cbasis(n, 'even');

Parameters:

n - dimension of the real vector space V where 0 <= n <= 9
k - the k-th grade, 0<= k <= n

Description:

- Procedure 'cbasis' writes a canonical un-dotted standard Grassmann wedge basis for a Clifford algebra Cl(V,B) over a real vector space V endowed with a bilinear form B. The dimension of V is specified as the first argument. There is a similar procedure Cliplus:-dottedcbasis which returns a dotted wedge basis for Cl(V,B). These two bases are different if the antisymmetric part of B is non-zero.
- 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).
- Below, 'Id' stands for the algebra unit element and 'w' is just a placeholder for the wedge/exterior product in the Clifford algebra. See wedge for more help.
- When n=0 then [Id] is returned.
- An option 'even' allows one to create a basis in the even subalgebra of the Clifford algebra Cl(V).
- Procedure makeclibasmon creates a basis Grassmann monomial, that is, an element of `type/clibasmon` with indices entered as a list.
- Procedure reorder can be used to reorder indices of a basis Grassmann monomial. It also computes the sign of the permutation that reorders the indices and puts it in front of the result.
- Procedure extract is a reverse procedure to makeclibasmon in that it extracts indices from a basis Grassmann monomial.
- The basis elements can be aliased, e.g., 'e1we2' can be aliased as 'e12' with the procedure makealiases.

Examples:

```
> restart:with(Clifford):
> cbasis(3);
```

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

```
> cbasis(3,2);
```

$$[e1we2, e1we3, e2we3]$$

```
> cbasis(3,'even');
```

$$[Id, e1we2, e1we3, e2we3]$$

```
> cbasis(0);
```

$$[Id]$$

```
> makeclibasmon([]);
```

$$Id$$

```
> makeclibasmon([1,4,2,j,i,3]);
```

$$e1we4we2wejweiwe3$$

```
> reorder(%);
```

$$-e1we2we3we4weiwej$$

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

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

```
> map(extract,L);
```

$$[[\ ], [1], [2], [3], [1, 2], [1, 3], [2, 3], [1, 2, 3]]$$

```
> map(makeclibasmon,%);
```

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

```
> with(Cliplus):
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
> F:=array(1..9,1..9,antisymmetric):
> dottedcbasis[F](4);
```

$$[Id, e1, e2, e3, e4, e1we2 + F_{1,2}\, Id, e1we3 + F_{1,3}\, Id, e1we4 + F_{1,4}\, Id, e2we3 + F_{2,3}\, Id, e2we4 + F_{2,4}\, Id, e3we4 + F_{3,4}\, Id, e1we2we3 + F_{2,3}\, e1 - F_{1,3}\, e2 + F_{1,2}\, e3, e1we2we4 + F_{2,4}\, e1 - F_{1,4}\, e2 + F_{1,2}\, e4, e1we3we4 + F_{3,4}\, e1 - F_{1,4}\, e3 + F_{1,3}\, e4, e2we3we4 + F_{3,4}\, e2 - F_{2,4}\, e3 + F_{2,3}\, e4, e1we2we3we4 + F_{2,3}\, F_{1,4}\, Id - F_{1,3}\, F_{2,4}\, Id + F_{1,2}\, F_{3,4}\, Id - F_{2,4}\, e1we3 + F_{1,4}\, e2we3 + F_{3,4}\, e1we2 + F_{2,3}\, e1we4 - F_{1,3}\, e2we4 + F_{1,2}\, e3we4]$$

```
>
```

See Also: Clifford:-extract, Clifford:-`type/clibasmon`, Clifford:-reorder, Clifford:-wedge, Clifford:-makeclibasmon, Cliplus:-dottedcbasis, Cliplus:-dwedge

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-ddfmatrix, Clifford:-cdfmatrix - decompose a double field matrix and create a double field matrix

Calling Sequence:

ddfmatrix(M) - decompose a matrix of type 'dfmatrix' into a list L of two square matrices,
cdfmatrix(L) - create a matrix M of type 'dfmatrix' from a list L containing two n x n matrices,

Parameters:

M - a matrix of `type/dfmatrix`
L - list containing two square matrices of the same size

Description:

- A matrix M is of `type/dfmatrix' if it is a square matrix whose entries are two element lists in a double field. These matrices arise when faithful matrix representations of semi-simple Clifford algebras are calculated. Use all_sigs to display signatures of semisimple Clifford algebras in dimensions 1 through 9.
- Faithful spinor representations of semisimple Clifford algebras have been precomputed. They are stored in a library file and can be retrieved with the procedure clidata.
- 'cdmatrix' is equivalent to procedure `convert/dfmatrix` except it is used differently.

Examples:

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

Let's display signatures of semi-simple Clifford algebras in dimensions 1 through 9 which are isomorphic to rings of matrices over double real field R+R:

```
> all_sigs(1..9,'real','semisimple');
```

$$[[0, 7], [1, 0], [1, 8], [2, 1], [3, 2], [4, 3], [5, 4], [9, 0]]$$

Let's display signatures of semi-simple Clifford algebras in dimensions 1 through 9 which are isomorphic to rings of matrices over double quaternionic field H+H:

```
> all_sigs(1..9,'quat','semisimple');
```

$$[[0, 3], [1, 4], [2, 5], [3, 6], [5, 0], [6, 1], [7, 2]]$$

There are no semisimple Clifford algebras that would be isomorphic to rings of matrices of double complex field C+C:

```
> all_sigs(1..9,'complex','semisimple');
```

$$[\ ]$$

More information about Clifford algebra Cl(Q) of the quadratic form of signature (2,1) can be found using procedure clidata as follows:

```
> clidata([2,1]);
```

$$\left[real, 2, semisimple, 'cmulQ'\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e2we3}{2}\right), [Id, e2], [Id], [Id, e2]\right]$$

Example 1: Let's view matrices m[i], i=1..3, representing 1-vectors {e1,e2,e3} in Cl(2,1) = Mat(2,2,R+R). These matrices are of type `type/dfmatrix` and have been precomputed. They can be displayed with the procedure matKrepr:

```
> pq:=[2,1];
  B:=diag(1$pq[1],-1$pq[2]);
```

$$pq := [2, 1]$$

$$B := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

```
> L:=matKrepr();
```

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

$$L := \left[e1 = \begin{bmatrix} [1, -1] & [0, 0] \\ [0, 0] & [-1, 1] \end{bmatrix}, e2 = \begin{bmatrix} [0, 0] & [1, 1] \\ [1, 1] & [0, 0] \end{bmatrix}, e3 = \begin{bmatrix} [0, 0] & [-1, -1] \\ [1, 1] & [0, 0] \end{bmatrix} \right]$$

Let's assign these matrices to m[i], i=1..3:

```
> for i from 1 to nops(L) do m[i]:=rhs(L[i]) od;
```

$$m_1 := \begin{bmatrix} [1, -1] & [0, 0] \\ [0, 0] & [-1, 1] \end{bmatrix}$$

$$m_2 := \begin{bmatrix} [0, 0] & [1, 1] \\ [1, 1] & [0, 0] \end{bmatrix}$$

$$m_3 := \begin{bmatrix} [0, 0] & [-1, -1] \\ [1, 1] & [0, 0] \end{bmatrix}$$

All matrices m[i], i=1..3, are of type 'dfmatrix':

```
> for i from 1 to 3 do type(m[i],dfmatrix) od;
```

$$true$$

$$true$$

$$true$$

We can now decompose each matrix into an ordered pair of two ordinary real matrices with the procedure 'ddfmatrix':

```
> L1:=ddfmatrix(m[1]);L2:=ddfmatrix(m[2]);L3:=ddfmatrix(m[3]);
```

$$L1 := \left[\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \right]$$

$$L2 := \left[\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \right]$$

$$L3 := \left[\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \right]$$

Then, we can create back matrices m[1], m[2], m[3] over R+R with the procedure 'cdfmatrix]:

```
> m1:=cdfmatrix(L1);m2:=cdfmatrix(L2);m3:=cdfmatrix(L3);
```

$$m1 := \begin{bmatrix} [1, -1] & [0, 0] \\ [0, 0] & [-1, 1] \end{bmatrix}$$

$$m2 := \begin{bmatrix} [0, 0] & [1, 1] \\ [1, 1] & [0, 0] \end{bmatrix}$$

$$m3 := \begin{bmatrix} [0, 0] & [-1, -1] \\ [1, 1] & [0, 0] \end{bmatrix}$$

or with the procedure `convert/dfmatrix`:

```
> convert(L1,dfmatrix);convert(L2,dfmatrix);convert(L3,dfmatrix);
```

$$\begin{bmatrix} [1, -1] & [0, 0] \\ [0, 0] & [-1, 1] \end{bmatrix}$$

$$\begin{bmatrix} [0, 0] & [1, 1] \\ [1, 1] & [0, 0] \end{bmatrix}$$

$$\begin{bmatrix} [0, 0] & [-1, -1] \\ [1, 1] & [0, 0] \end{bmatrix}$$

Example 2: Let's view matrices m[i], i=1..3, representing 1-vectors {e1,e2,e3} in Cl(0,3) = Mat(1,1,H+H). These matrices are of type `type/dfmatrix` and have been precomputed. They can be displayed with the procedure matKrepr:

```
> pq:=[0,3];
  B:=diag(1$pq[1],-1$pq[2]);
```

$$pq := [0, 3]$$

$$B := \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

```
> L:=matKrepr();
```

$$L := [e1 = [[e1, e1]], e2 = [[e2, e2]], e3 = [[-e1we2, e1we2]]]$$

Let's assign these matrices to m[i], i=1..3:

```
> for i from 1 to nops(L) do m[i]:=rhs(L[i]) od;
```

$$m_1 := [[e1, e1]]$$

$$m_2 := [[e2, e2]]$$

$$m_3 := [[-e1we2, e1we2]]$$

All matrices m[i], i=1..3, are of type 'dfmatrix':

```
> for i from 1 to nops(L) do type(m[i],dfmatrix) od;
```

$$true$$

$$true$$

$$true$$

```
> L1:=ddfmatrix(m[1]);L2:=ddfmatrix(m[2]);L3:=ddfmatrix(m[3]);
```

$$L1 := [[e1], [e1]]$$

$$L2 := [[e2], [e2]]$$

$$L3 := [[-e1we2], [e1we2]]$$

```
> m1:=cdfmatrix(L1);m2:=cdfmatrix(L2);m3:=cdfmatrix(L3);
```

$$m1 := [[e1, e1]]$$

$$m2 := [[e2, e2]]$$

$$m3 := [[-e1we2, e1we2]]$$

>

Algebraic operations on these matrices such as addition and multiplication can be performed with procedures Clifford:-adfmatrix and Clifford:-mdfmatrix.

See Also: Clifford:-`type/dfmatrix`, Clifford:-adfmatrix, Clifford:-mdfmatrix, Clifford:-spinorKrepr, Clifford:-matKrepr

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-cexp - Clifford exponential in the Clifford algebra Cl(B)

Calling Sequence:

cexp(p,n);
cexp(p,n,K);

Parameters:

p - an expression of type 'numeric', 'cliscalar' or 'clipolynom'
n - a non-negative integer
K - (optional) argument of type name, symbol, matrix, array, or
`&*`(numeric,{name,symbol,array,matrix})

Description:

- Procedure 'cexp' computes Clifford exponential of a Clifford polynomial p in Cl(B) up to the order specified by the second argument n which is expected to be a nonnegative integer. It uses cmulNUM for speedy computations.
- It can use optional argument as in cexp(p,n,K) in which case computations are performed in Cl(K).
- If n = 0 then the procedure returns 1 when p is of type 'numeric' or 'cliscalar' (see `type/cliscalar`) and 'Id' when p is of type 'clipolynom' (see `type/clipolynom`).
- When the bilinear form B is diagonal, use 'cexpQ' which is a special form of 'cexp' to be used in Cl(Q) (although 'cexp' still may be used but it will be slower than 'cexpQ'). See cexpQ for more help.
- Note that one may Clifford-exponentiate not only a polynomial but also a polynomial times some parameter.
- It is not necessary that the form Q (or B) be defined.
- Instead of 'cexp' or 'cexpQ' use procedure sexp when n is larger than the degree of the minimal polynomial of p since then the procedure 'sexp' is faster than either 'cexp' or 'cexpQ'. Recall that the minimal polynomial of any Clifford polynomial can be computed using procedure climinpoly .
- Use 'wexp' to compute exterior exponential in Cl(B). See wexp for more help.

Examples:

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

Example 1: Exponentiation in Cl(B) and in Cl(-B):.

```
> p:=e1we2*t;
```

$$p := e1we2\ t$$

```
> climinpoly[B](p);
```

```
climinpoly[-B](p);
```

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

$$-t^2 B_{2,1} B_{1,2} + t^2 B_{2,2} B_{1,1} + x^2 - (t B_{2,1} - t B_{1,2}) x$$

$$-t^2 B_{2,1} B_{1,2} + t^2 B_{2,2} B_{1,1} + x^2 - (-t B_{2,1} + t B_{1,2}) x$$

Since the degree of the minimal polynomial of p is 2, we will use procedure 'cexp' only when n <= 2; otherwise we will use the faster procedure 'sexp':

```
> s:=time():
  p1:=sexp(e1we2*t,5,B); #note the use of off-diagonal B entries
  time()-s;
```

$$p1 := \frac{1}{120} (120 + 20 t^3 B_{2,1}{}^2 B_{1,2} - 20 t^3 B_{2,1} B_{2,2} B_{1,1} - 20 t^3 B_{2,1} B_{1,2}{}^2 + 20 t^3 B_{1,2} B_{2,2} B_{1,1}$$
$$+ 60 t^2 B_{2,1} B_{1,2} - 60 t^2 B_{2,2} B_{1,1} - t^5 B_{2,1}{}^3 B_{1,2}{}^2 - t^5 B_{2,1}{}^2 B_{1,2} B_{2,2} B_{1,1} + t^5 B_{2,1}{}^2 B_{1,2}{}^3$$
$$+ t^5 B_{2,1} B_{1,2}{}^2 B_{2,2} B_{1,1} + 2 t^5 B_{2,1} B_{2,2}{}^2 B_{1,1}{}^2 - 2 t^5 B_{1,2} B_{2,2}{}^2 B_{1,1}{}^2 + t^5 B_{2,1}{}^4 B_{1,2}$$
$$- t^5 B_{2,1}{}^3 B_{2,2} B_{1,1} - t^5 B_{1,2}{}^4 B_{2,1} + t^5 B_{1,2}{}^3 B_{2,2} B_{1,1} - 5 t^4 B_{2,1}{}^2 B_{1,2}{}^2 + 5 t^4 B_{2,2}{}^2 B_{1,1}{}^2$$
$$+ 5 t^4 B_{2,1}{}^3 B_{1,2} - 5 t^4 B_{2,1}{}^2 B_{2,2} B_{1,1} + 5 t^4 B_{1,2}{}^3 B_{2,1} - 5 t^4 B_{1,2}{}^2 B_{2,2} B_{1,1}) Id + \frac{1}{120} t ($$
$$60 t B_{2,1} - 60 t B_{1,2} - 20 t^2 B_{2,1} B_{1,2} - 20 t^2 B_{2,2} B_{1,1} + 20 t^2 B_{2,1}{}^2 + 20 t^2 B_{1,2}{}^2$$
$$- 5 t^3 B_{2,1}{}^2 B_{1,2} + 5 t^3 B_{2,1} B_{1,2}{}^2 - 10 t^3 B_{2,1} B_{2,2} B_{1,1} + 10 t^3 B_{1,2} B_{2,2} B_{1,1} + 5 t^3 B_{2,1}{}^3$$
$$- 5 t^3 B_{1,2}{}^3 + t^4 B_{2,1}{}^2 B_{1,2}{}^2 + t^4 B_{2,2}{}^2 B_{1,1}{}^2 - t^4 B_{2,1}{}^3 B_{1,2} - 3 t^4 B_{2,1}{}^2 B_{2,2} B_{1,1} - t^4 B_{1,2}{}^3 B_{2,1}$$
$$- 3 t^4 B_{1,2}{}^2 B_{2,2} B_{1,1} + 4 t^4 B_{2,1} B_{2,2} B_{1,1} B_{1,2} + t^4 B_{2,1}{}^4 + t^4 B_{1,2}{}^4 + 120) e1we2$$

0.375

```
> s:=time():
  p2:=cexp(e1we2*t,5,B);
  time()-s;
```

$$p2 := Id + t\, e1we2 + \frac{1}{2} t^2 (B_{2,1} B_{1,2} - B_{2,2} B_{1,1}) Id + \frac{1}{2} t^2 (B_{2,1} - B_{1,2}) e1we2$$
$$+ \frac{1}{6} t^3 (B_{2,1}{}^2 B_{1,2} - B_{2,1} B_{2,2} B_{1,1} - B_{2,1} B_{1,2}{}^2 + B_{1,2} B_{2,2} B_{1,1}) Id$$
$$+ \frac{1}{6} t^3 (-B_{2,1} B_{1,2} + B_{2,1}{}^2 - B_{2,2} B_{1,1} + B_{1,2}{}^2) e1we2 +$$
$$\frac{1}{24} t^4 (-B_{2,1}{}^2 B_{1,2}{}^2 + B_{2,2}{}^2 B_{1,1}{}^2 + B_{2,1}{}^3 B_{1,2} - B_{2,1}{}^2 B_{2,2} B_{1,1} + B_{1,2}{}^3 B_{2,1} - B_{1,2}{}^2 B_{2,2} B_{1,1}) Id$$
$$+ \frac{1}{24} t^4 (2 B_{1,2} B_{2,2} B_{1,1} + B_{2,1}{}^3 - B_{2,1}{}^2 B_{1,2} - 2 B_{2,1} B_{2,2} B_{1,1} + B_{2,1} B_{1,2}{}^2 - B_{1,2}{}^3) e1we2 +$$

$$\frac{1}{120} t^5 (-B_{2,1}{}^3 B_{1,2}{}^2 - B_{2,1}{}^2 B_{1,2} B_{2,2} B_{1,1} + B_{2,1}{}^2 B_{1,2}{}^3 + B_{2,1} B_{1,2}{}^2 B_{2,2} B_{1,1}$$
$$+ 2 B_{2,1} B_{2,2}{}^2 B_{1,1}{}^2 - 2 B_{1,2} B_{2,2}{}^2 B_{1,1}{}^2 + B_{2,1}{}^4 B_{1,2} - B_{2,1}{}^3 B_{2,2} B_{1,1} - B_{1,2}{}^4 B_{2,1}$$
$$+ B_{1,2}{}^3 B_{2,2} B_{1,1}) Id + \frac{1}{120} t^5 (-3 B_{1,2}{}^2 B_{2,2} B_{1,1} + B_{2,1}{}^4 + 4 B_{1,2} B_{2,2} B_{1,1} B_{2,1} + B_{2,1}{}^2 B_{1,2}{}^2$$
$$+ B_{2,2}{}^2 B_{1,1}{}^2 - B_{2,1}{}^3 B_{1,2} - 3 B_{2,1}{}^2 B_{2,2} B_{1,1} - B_{1,2}{}^3 B_{2,1} + B_{1,2}{}^4) e1we2$$

$$0.266$$

Thus, the procedure 'sexp' was faster, but the results are the same:

```
> simplify(p1-p2);
```

$$0$$

When n <= 2 (in this case), procedure 'cexp' is faster than 'sexp':

```
> s:=time():
  p2:=cexp(e1we2*t,2,-K);
  time()-s;
```

$$p2 := Id + t\, e1we2 + \frac{1}{2} t^2 (K_{2,1} K_{1,2} - K_{2,2} K_{1,1}) Id - \frac{1}{2} t^2 (K_{2,1} - K_{1,2}) e1we2$$

$$0.015$$

```
> s:=time():
  p1:=sexp(e1we2*t,2,-K);
  time()-s;
```

$$p1 := \frac{1}{2} (2 + t^2 K_{2,1} K_{1,2} - t^2 K_{2,2} K_{1,1}) Id - \frac{1}{2} t (-2 + t K_{2,1} - t K_{1,2}) e1we2$$

$$0.016$$

Example 2: Exponentiation in Cl(K) and Cl(-K).

```
> p:=e1we2*t;
```

$$p := t\, e1we2$$

```
> cexp(p,1,K);
  cexp(p,2,K);
  cexp(p,3,K);
  cexp(p,1,-K);
  cexp(p,2,-K);
  cexp(p,3,-K);
```

$$Id + t\, e1we2$$

$$Id + t\, e1we2 + \frac{1}{2} t^2 (K_{2,1} K_{1,2} - K_{2,2} K_{1,1}) Id + \frac{1}{2} t^2 (K_{2,1} - K_{1,2}) e1we2$$

$$Id + t\, e1we2 + \frac{1}{2} t^2 (K_{2,1} K_{1,2} - K_{2,2} K_{1,1}) Id + \frac{1}{2} t^2 (K_{2,1} - K_{1,2}) e1we2$$

$$+\frac{1}{6}t^3\left(-K_{2,1}K_{1,2}^{\ 2}+K_{2,1}^{\ 2}K_{1,2}-K_{2,1}K_{2,2}K_{1,1}+K_{1,2}K_{2,2}K_{1,1}\right)Id$$

$$+\frac{1}{6}t^3\left(-K_{2,1}K_{1,2}+K_{2,1}^{\ 2}-K_{2,2}K_{1,1}+K_{1,2}^{\ 2}\right)e1we2$$

$$Id+t\,e1we2$$

$$Id+t\,e1we2+\frac{1}{2}t^2\left(K_{2,1}K_{1,2}-K_{2,2}K_{1,1}\right)Id-\frac{1}{2}t^2\left(K_{2,1}-K_{1,2}\right)e1we2$$

$$Id+t\,e1we2+\frac{1}{2}t^2\left(K_{2,1}K_{1,2}-K_{2,2}K_{1,1}\right)Id-\frac{1}{2}t^2\left(K_{2,1}-K_{1,2}\right)e1we2$$

$$-\frac{1}{6}t^3\left(-K_{2,1}K_{1,2}^{\ 2}+K_{2,1}^{\ 2}K_{1,2}-K_{2,1}K_{2,2}K_{1,1}+K_{1,2}K_{2,2}K_{1,1}\right)Id$$

$$+\frac{1}{6}t^3\left(-K_{2,1}K_{1,2}+K_{2,1}^{\ 2}-K_{2,2}K_{1,1}+K_{1,2}^{\ 2}\right)e1we2$$

```
>
```

Example 3: Let's see some additional examples in Cl(2,2). Since we will be using a numeric matrix B, we will use cmulNUM for the Clifford product instead of the default cmulRS. To change to cmulNUM, we will use procedure useproduct.

```
> useproduct(cmulNUM);
Warning, cmul will use cmulNUM; for help see pages ?cmul, ?Clifford:-intro, or
?cmulNUM
```

```
> B:=linalg[diag](1,1,-1,-1):
  p:=e1we2+e1we3+e4: #define B and p
```

```
> climinpoly(p); #finding the minimal polynomial of p
  climinpoly[B](p); #finding the minimal polynomial of p
```

$$x^4+2x^2+1$$

$$x^4+2x^2+1$$

```
> cexp(p,0);
```

$$Id$$

```
> cexp(p,2);
```

$$\frac{Id}{2}+e1we3+e1we2+e4+e1we3we4+e1we2we4$$

```
> cexp(p,3);
```

$$\frac{Id}{2}+\frac{e1we3}{2}+\frac{e1we2}{2}+\frac{5\,e4}{6}+e1we3we4+e1we2we4$$

```
> cexp(p,4);
```

$$\frac{13\,Id}{24}+\frac{e1we3}{2}+\frac{e1we2}{2}+\frac{5\,e4}{6}+\frac{5\,e1we3we4}{6}+\frac{5\,e1we2we4}{6}$$

```
> sexp(p,5);
```

$$\frac{13\,Id}{24}+\frac{13\,e1we3}{24}+\frac{13\,e1we2}{24}+\frac{101\,e4}{120}+\frac{5\,e1we3we4}{6}+\frac{5\,e1we2we4}{6}$$

```
> sexp(p,6);
```

$$\frac{389\,Id}{720}+\frac{13\,e1we3}{24}+\frac{13\,e1we2}{24}+\frac{101\,e4}{120}+\frac{101\,e1we3we4}{120}+\frac{101\,e1we2we4}{120}$$

Let's redefine p:

```
> p:=e1we2*t;
```

$$p := t\,e1we2$$

```
> climinpoly(p);
```

$$x^2+t^2$$

```
> cexp(e1we2*t,1);
```

$$Id+t\,e1we2$$

```
> cexp(e1we2*t,2);
```

$$Id+t\,e1we2-\frac{1}{2}t^2\,Id$$

```
> sexp(e1we2*t,3);
```

$$-\frac{(-2+t^2)\,Id}{2}-\frac{t\,(-6+t^2)\,e1we2}{6}$$

```
> sexp(e1we2*t,4);
```

$$\frac{(24-12\,t^2+t^4)\,Id}{24}-\frac{t\,(-6+t^2)\,e1we2}{6}$$

Additional examples:

```
> cexp((e1+e2we3)*t,5);
```

$$Id+t\,e2we3+t\,e1+t^2\,Id+t^2\,e1we2we3+\frac{2\,t^3\,e2we3}{3}+\frac{2\,t^3\,e1}{3}+\frac{t^4\,Id}{3}+\frac{t^4\,e1we2we3}{3}+\frac{2\,t^5\,e2we3}{15}+\frac{2\,t^5\,e1}{15}$$

```
> sexp((e1+e2we3)*t,15);
```

$$(42567525+14189175\,t^4+6006\,t^{10}+42567525\,t^2+182\,t^{12}+1891890\,t^6+4\,t^{14}+135135\,t^8)\,Id/42567525+t\,(638512875+8\,t^{14}+425675250\,t^2+85135050\,t^4+8108100\,t^6+450450\,t^8+16380\,t^{10}+420\,t^{12})\,e2we3/638512875+t\,(638512875+8\,t^{14}+425675250\,t^2+85135050\,t^4+8108100\,t^6+450450\,t^8+16380\,t^{10}+420\,t^{12})\,e1/638512875+\frac{t^2\,(1891890\,t^4+6006\,t^8+182\,t^{10}+42567525+4\,t^{12}+135135\,t^6+14189175\,t^2)\,e1we2we3}{42567525}$$

```
> cexp(2*alpha,0);
  sexp(2*alpha,0);
```

$$1$$

$$1$$

```
> cexp(2*alpha,7);
  sexp(2*alpha,7);
```

$$1+2\,\alpha+2\,\alpha^2+\frac{4}{3}\,\alpha^3+\frac{2}{3}\,\alpha^4+\frac{4}{15}\,\alpha^5+\frac{4}{45}\,\alpha^6+\frac{8}{315}\,\alpha^7$$

$$1+2\,\alpha+2\,\alpha^2+\frac{4}{3}\,\alpha^3+\frac{2}{3}\,\alpha^4+\frac{4}{15}\,\alpha^5+\frac{4}{45}\,\alpha^6+\frac{8}{315}\,\alpha^7$$

```
> cexp(2*alpha*Id,10);
  sexp(2*alpha*Id,10);
```

$$\left(1+2\,\alpha+2\,\alpha^2+\frac{4}{3}\,\alpha^3+\frac{2}{3}\,\alpha^4+\frac{4}{15}\,\alpha^5+\frac{4}{45}\,\alpha^6+\frac{8}{315}\,\alpha^7+\frac{2}{315}\,\alpha^8+\frac{4}{2835}\,\alpha^9+\frac{4}{14175}\,\alpha^{10}\right)Id$$

$$\left(1+2\,\alpha+2\,\alpha^2+\frac{4}{3}\,\alpha^3+\frac{2}{3}\,\alpha^4+\frac{4}{15}\,\alpha^5+\frac{4}{45}\,\alpha^6+\frac{8}{315}\,\alpha^7+\frac{2}{315}\,\alpha^8+\frac{4}{2835}\,\alpha^9+\frac{4}{14175}\,\alpha^{10}\right)Id$$

```
> cexp(.6,100);
  sexp(.6,100);
```

$$1.822118801$$

$$1.822118801$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 2.217000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

See Also: Clifford:-climinpoly, Clifford:-sexp, Clifford:-wexp, Clifford:-cexpQ, Clifford:-`type/clipolynom`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-cexpQ - Clifford exponential in the Clifford algebra Cl(Q)

Calling Sequence:

cexpQ(p,n);
cexpQ(p,n,K);

Parameters:

p - an expression of type 'numeric', 'cliscalar' or 'clipolynom'
n - a non-negative integer
K - (optional) argument of type name, symbol, matrix, array or
`&*`(numeric,{name,symbol,array,matrix})

Description:

- Procedure 'cexpQ' computes Clifford exponential of a Clifford polynomial p in Cl(Q) up to the order specified by the second argument n which is expected to be a nonnegative integer. This is just a special version of the procedure cexp.
- Optional parameter can be passed on as argument of type name, symbol, array, matrix, or `&*`(numeric,{name,symbol,array,matrix}). Then, computations are performed in Cl(K) with respect to the diagonal entries of K. Procedure doesn't check is diagonal or symmetric.
- If n = 0 then the procedure returns 1 when p is of type 'numeric' or 'cliscalar' (see `type/cliscalar`) and 'Id' when p is of type 'clipolynom' (see `type/clipolynom`).
- Note that one may Clifford-exponentiate not only a polynomial but also a polynomial times some parameter.
- Instead of 'cexpQ' or 'cexp' use procedure sexp when n is larger than the degree of the minimal polynomial of p since then the procedure 'sexp' is faster than either 'cexpQ' or 'cexp'. Recall that the minimal polynomial of any Clifford polynomial can be computed using procedure climinpoly .
- Use 'wexp' to compute exterior exponential in Cl(B). See wexp for more help.
- Be careful: when the bilinear form B is unspecified, 'cexpQ' uses only the diagonal entries of B.

Examples:

```
> restart:with(Clifford):
> p:=e1we2*t;
```

$$p := e1we2\, t$$

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

$$-t^2 B_{2,1} B_{1,2} + t^2 B_{2,2} B_{1,1} + x^2 - (t B_{2,1} - t B_{1,2})\, x$$

Since the degree of the minimal polynomial of p is 2, we will use procedure 'cexp' only when n <= 2; otherwise we will use the faster procedure 'sexp':

```
> s:=time():
  p1:=sexp(e1we2*t,5); #note the use of off-diagonal B entries
  time()-s;
```

$$p1 := \frac{1}{120}(120 + 20\,t^3\,B_{2,1}{}^2\,B_{1,2} - 20\,t^3\,B_{2,1}\,B_{2,2}\,B_{1,1} - 20\,t^3\,B_{2,1}\,B_{1,2}{}^2 + 20\,t^3\,B_{1,2}\,B_{2,2}\,B_{1,1}$$
$$+ 60\,t^2\,B_{2,1}\,B_{1,2} - 60\,t^2\,B_{2,2}\,B_{1,1} - t^5\,B_{2,1}{}^3\,B_{1,2}{}^2 - t^5\,B_{2,1}{}^2\,B_{1,2}\,B_{2,2}\,B_{1,1} + t^5\,B_{2,1}{}^2\,B_{1,2}{}^3$$
$$+ t^5\,B_{2,1}\,B_{1,2}{}^2\,B_{2,2}\,B_{1,1} + 2\,t^5\,B_{2,1}\,B_{2,2}{}^2\,B_{1,1}{}^2 - 2\,t^5\,B_{1,2}\,B_{2,2}{}^2\,B_{1,1}{}^2 + t^5\,B_{2,1}{}^4\,B_{1,2}$$
$$- t^5\,B_{2,1}{}^3\,B_{2,2}\,B_{1,1} - t^5\,B_{1,2}{}^4\,B_{2,1} + t^5\,B_{1,2}{}^3\,B_{2,2}\,B_{1,1} - 5\,t^4\,B_{2,1}{}^2\,B_{1,2}{}^2 + 5\,t^4\,B_{2,2}{}^2\,B_{1,1}{}^2$$
$$+ 5\,t^4\,B_{2,1}{}^3\,B_{1,2} - 5\,t^4\,B_{2,1}{}^2\,B_{2,2}\,B_{1,1} + 5\,t^4\,B_{1,2}{}^3\,B_{2,1} - 5\,t^4\,B_{1,2}{}^2\,B_{2,2}\,B_{1,1})\,Id + \frac{1}{120}\,t\,($$
$$60\,t\,B_{2,1} - 60\,t\,B_{1,2} - 20\,t^2\,B_{2,1}\,B_{1,2} - 20\,t^2\,B_{2,2}\,B_{1,1} + 20\,t^2\,B_{2,1}{}^2 + 20\,t^2\,B_{1,2}{}^2$$
$$- 5\,t^3\,B_{2,1}{}^2\,B_{1,2} + 5\,t^3\,B_{2,1}\,B_{1,2}{}^2 - 10\,t^3\,B_{2,1}\,B_{2,2}\,B_{1,1} + 10\,t^3\,B_{1,2}\,B_{2,2}\,B_{1,1} + 5\,t^3\,B_{2,1}{}^3$$
$$- 5\,t^3\,B_{1,2}{}^3 + t^4\,B_{2,1}{}^2\,B_{1,2}{}^2 + t^4\,B_{2,2}{}^2\,B_{1,1}{}^2 - t^4\,B_{2,1}{}^3\,B_{1,2} - 3\,t^4\,B_{2,1}{}^2\,B_{2,2}\,B_{1,1} - t^4\,B_{1,2}{}^3\,B_{2,1}$$
$$- 3\,t^4\,B_{1,2}{}^2\,B_{2,2}\,B_{1,1} + 4\,t^4\,B_{2,1}\,B_{2,2}\,B_{1,1}\,B_{1,2} + t^4\,B_{2,1}{}^4 + t^4\,B_{1,2}{}^4 + 120)\,e1we2$$

$$0.203$$

```
> s:=time():
  p2:=cexpQ(e1we2*t,5,B);
  time()-s;
```

$$p2 := Id + t\,e1we2 - \frac{1}{2}\,t^2\,Id\,B_{2,2}\,B_{1,1} - \frac{1}{6}\,e1we2\,t^3\,B_{2,2}\,B_{1,1} + \frac{1}{24}\,Id\,t^4\,B_{2,2}{}^2\,B_{1,1}{}^2$$
$$+ \frac{1}{120}\,t^5\,B_{2,2}{}^2\,B_{1,1}{}^2\,e1we2$$

$$0.063$$

Thus, the procedure 'sexp' was faster, but the results are not obviously the same.

Let's see some additional examples in Cl(2,2). Since we will be using a numeric matrix, we select cmulNUM instead of cmulRS using procedure useproduct for speedier computation.

```
> useproduct(cmulNUM);
Warning, cmul will use cmulNUM; for help see pages ?cmul, ?Clifford:-intro, or
?cmulNUM
```

```
> B:=linalg[diag](1,1,-1,-1):
  p:=e1we2+2*e3we4: #define B and p
> climinpoly(p); #finding the minimal polynomial of p
```

$$x^4 + 10\,x^2 + 9$$

```
> cexpQ(p,0);
```

$$Id$$

```
> cexpQ(p,2);
```

$$-\frac{3\,Id}{2}+e1we2+2\,e3we4+2\,e1we2we3we4$$

```
> cexpQ(p,3);
```

$$-\frac{3\,Id}{2}-\frac{7\,e1we2}{6}-\frac{e3we4}{3}+2\,e1we2we3we4$$

```
> cexpQ(p,4);
```

$$\frac{5\,Id}{24}-\frac{7\,e1we2}{6}-\frac{e3we4}{3}+\frac{e1we2we3we4}{3}$$

```
> sexp(p,4);
```

$$\frac{5\,Id}{24}-\frac{7\,e1we2}{6}-\frac{e3we4}{3}+\frac{e1we2we3we4}{3}$$

```
> sexp(p,5);
```

$$\frac{5\,Id}{24}-\frac{19\,e1we2}{120}+\frac{41\,e3we4}{60}+\frac{e1we2we3we4}{3}$$

Let's redefine p:

```
> p:=(e1+2*e2-e3we4)*t;
```

$$p := (e1+2\,e2-e3we4)\,t$$

```
> climinpoly(p);
```

$$x^4-8\,x^2\,t^2+36\,t^4$$

```
> cexpQ(p,1);
```

$$Id+t\,e1+2\,t\,e2-t\,e3we4$$

```
> cexpQ(p,2);
```

$$Id+t\,e1+2\,t\,e2-t\,e3we4+2\,t^2\,Id-t^2\,e1we3we4-2\,t^2\,e2we3we4$$

```
> cexpQ(p,3);
```

$$Id+t\,e1+2\,t\,e2-t\,e3we4+2\,t^2\,Id-t^2\,e1we3we4-2\,t^2\,e2we3we4+\frac{t^3\,e1}{3}-\frac{7\,t^3\,e3we4}{3}+\frac{2\,t^3\,e2}{3}$$

```
> sexp(p,4);
```

$$-\frac{(-6+t^4-12\,t^2)\,Id}{6}+\frac{t\,(3+t^2)\,e1}{3}+\frac{2\,t\,(3+t^2)\,e2}{3}-\frac{t\,(7\,t^2+3)\,e3we4}{3}-\frac{2\,t^2\,(2\,t^2+3)\,e2we3we4}{3}-\frac{t^2\,(2\,t^2+3)\,e1we3we4}{3}$$

Additional examples:

```
> evalm(B);
  cexpQ(e1,2,B);
  cexpQ(e1,2,-B);
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

$$\frac{3\,Id}{2} + e1$$

$$\frac{Id}{2} + e1$$

```
> sexp((e1-3*e2we3)*t,10,B);
  sexp((e1-3*e2we3)*t,10,-B);
```

$$\frac{(16065\,t^4 + 2835 + 14175\,t^2 + 2313\,t^8 + 410\,t^{10} + 8190\,t^6)\,Id}{2835} - \frac{t\,(315 + 462\,t^4 + 38\,t^8 + 630\,t^2 + 172\,t^6)\,e2we3}{105} + \frac{t\,(2835 + 13230\,t^2 + 1022\,t^8 + 11718\,t^4 + 4572\,t^6)\,e1}{2835} - \frac{t^2\,(682\,t^8 + 23625\,t^2 + 13230\,t^4 + 14175 + 3825\,t^6)\,e1we2we3}{4725}$$

$$-\frac{(-113400 - 453600\,t^2 + 55440\,t^6 + 23715\,t^8 - 132300\,t^4 + 3116\,t^{10})\,Id}{113400} + \frac{t\,(84\,t^4 - 2520 + 332\,t^6 - 2520\,t^2 + 71\,t^8)\,e2we3}{840} + \frac{t\,(481\,t^8 + 22680 + 11052\,t^6 + 59724\,t^4 + 98280\,t^2)\,e1}{22680} + \frac{t^2\,(-5040\,t^6 - 151200\,t^2 - 49140\,t^4 - 113400 + 79\,t^8)\,e1we2we3}{37800}$$

```
> cexpQ(2*alpha,0);
  sexp(2*alpha,0);
```

$$1$$

$$1$$

```
> cexpQ(2*alpha,7);
  sexp(2*alpha,7);
```

$$1 + 2\,\alpha + 2\,\alpha^2 + \frac{4}{3}\,\alpha^3 + \frac{2}{3}\,\alpha^4 + \frac{4}{15}\,\alpha^5 + \frac{4}{45}\,\alpha^6 + \frac{8}{315}\,\alpha^7$$

$$1 + 2\,\alpha + 2\,\alpha^2 + \frac{4}{3}\,\alpha^3 + \frac{2}{3}\,\alpha^4 + \frac{4}{15}\,\alpha^5 + \frac{4}{45}\,\alpha^6 + \frac{8}{315}\,\alpha^7$$

```
> cexpQ(2*alpha*Id,10);
```

```
sexp(2*alpha*Id,10);
```

$$\left(1+2\,\alpha+2\,\alpha^2+\frac{4}{3}\,\alpha^3+\frac{2}{3}\,\alpha^4+\frac{4}{15}\,\alpha^5+\frac{4}{45}\,\alpha^6+\frac{8}{315}\,\alpha^7+\frac{2}{315}\,\alpha^8+\frac{4}{2835}\,\alpha^9+\frac{4}{14175}\,\alpha^{10}\right)Id$$

$$\left(1+2\,\alpha+2\,\alpha^2+\frac{4}{3}\,\alpha^3+\frac{2}{3}\,\alpha^4+\frac{4}{15}\,\alpha^5+\frac{4}{45}\,\alpha^6+\frac{8}{315}\,\alpha^7+\frac{2}{315}\,\alpha^8+\frac{4}{2835}\,\alpha^9+\frac{4}{14175}\,\alpha^{10}\right)Id$$

```
> cexpQ(.6,100);
  sexp(.6,100);
```

$$1.822118801$$

$$1.822118801$$

```
>
```

See Also: Clifford:-climinpoly, Clifford:-sexp, Clifford:-wexp, Clifford:-cexp, Clifford:-`type/clipolynom`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-cinv - symbolic inverse in Cl(B) or Cl(K)

Calling Sequence:

cinv(p);
cinv(p,K);

Parameters:

p - expression of the type 'cliscalar', 'clibasmon, 'climon', or 'clipolynom'
K - (optional) argument of type name, symbol, matrix, or array, or
`&*`(numeric,{name,symbol,array,matrix})

Description:

- Procedure 'cinv' calculates a symbolic inverse, if such exists, of any Clifford polynomial p of type 'clipolynom' (see `type/clipolynom` for more help) or any Clifford scalar s of type 'cliscalar' (see `type/cliscalar` for more help) in the Clifford algebra Cl(B), when no optional argument is specified, or in Cl(K), where K is the name of some other bilinear form. For example, K = g, where g is the symmetric part of B, is often used.
- If the bilinear form B is assigned a matrix, computations are done in the Clifford algebra of that form. If B is not assigned, then purely symbolic inverse is returned and it contains then unassigned entries of B.
- If the inverse is not found and B has been assigned a matrix, the procedure attempts to find a reason why the inverse does not exist. For example, if p is one of the following types, the inverse does not exist in the current Clifford algebra Cl(B):

  - p is of the type 'nilpotent' (see `type/nilpotent` for more help) ,
  - p is a non-trivial 'idempotent' (see `type/idempotent` for more help),

- If B is not assigned or when none of the above cases applies, the procedure determines whether p is an even element in Cl(B), i.e., whether it is of the type 'evenelement' (see `type/evenelement` for more help). If so, then p belongs to some even Clifford subalgebra A of Cl(B) and if it has the inverse, the inverse also belongs to A. Thus, in the 'even' case, the procedure finds the maximum index in p and looks for the inverse in a suitable even Clifford subalgebra A of Cl(B) over a vector subspace of (V,B) of dimension maxindex(p) (see maxindex for more help).
- If p is not even, then the inverse is looked for in a Clifford subalgebra of Cl(B) over a vector subspace of (V,B) of dimension maxindex(p).
- If p has no inverse, an appropriate message is returned. If p does have the inverse, to verify its correctness sometimes additional simplifications must be done. See examples below.
- If no inverse is found and if the procedure is unable to determine the reason, it returns NULL.
- Note that any of the following is an illegal entry: 1/e1, e1^(-1), etc.

- Quaternionic inverse may be computed with the procedure qinv.
- Recall that Clifford product can be given in the long form as cmul and in short infix form as `&c`.

Examples:

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

Example 1: Some special cases when B is not yet assigned:

```
> p:=2*a; # p is a 'cliscalar'
```

$$p := 2\,a$$

```
> cinv(p,K);
```

$$\frac{1}{2\,a}$$

```
> p:=2*(a+b)*c*Id; # p is of grade 0
```

$$p := 2\,(a+b)\,c\,Id$$

```
> cinv(p,K);
```

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

$$\frac{Id}{2\,c\,a + 2\,c\,b}$$

```
> p:=2+e1+2*e1we2; # p is some polynomial in Cl(B);
```

$$p := 2 + e1 + 2\,e1we2$$

```
> pinv:=cinv(p,B);
  pinv2:=cinv(p,-B);
```

$$pinv := \frac{2\,(B_{2,1} - B_{1,2} + 1)\,Id}{-4\,B_{1,2} + 4\,B_{2,2}\,B_{1,1} - B_{1,1} + 4\,B_{2,1} + 4 - 4\,B_{2,1}\,B_{1,2}} - \frac{2\,e1we2}{-4\,B_{1,2} + 4\,B_{2,2}\,B_{1,1} - B_{1,1} + 4\,B_{2,1} + 4 - 4\,B_{2,1}\,B_{1,2}} - \frac{e1}{-4\,B_{1,2} + 4\,B_{2,2}\,B_{1,1} - B_{1,1} + 4\,B_{2,1} + 4 - 4\,B_{2,1}\,B_{1,2}}$$

$$pinv2 := -\frac{2\,(-B_{1,2} + B_{2,1} - 1)\,Id}{B_{1,1} + 4\,B_{2,2}\,B_{1,1} - 4\,B_{2,1} - 4\,B_{2,1}\,B_{1,2} + 4\,B_{1,2} + 4} - \frac{2\,e1we2}{B_{1,1} + 4\,B_{2,2}\,B_{1,1} - 4\,B_{2,1} - 4\,B_{2,1}\,B_{1,2} + 4\,B_{1,2} + 4} - \frac{e1}{B_{1,1} + 4\,B_{2,2}\,B_{1,1} - 4\,B_{2,1} - 4\,B_{2,1}\,B_{1,2} + 4\,B_{1,2} + 4}$$

```
> simplify(cmul[B](pinv,p)),simplify(cmul[B](p,pinv));
  simplify(cmul[-B](pinv2,p)),simplify(cmul[-B](p,pinv2));
```

$$Id, Id$$

$$Id, Id$$

```
> simplify(&c[B](pinv,p)),simplify(&c[B](p,pinv));
  simplify(&c[-B](pinv2,p)),simplify(&c[-B](p,pinv2));
```

$$Id, Id$$

$$Id, Id$$

```
> p:=2+e1we2+2*e1we3; #p is even in some Cl(B)
```

$$p := 2 + e1we2 + 2\, e1we3$$

```
> pinv:=cinv(p);
```

$$pinv := -(B_{2,1} - B_{1,2} + 2 + 2\, B_{3,1} - 2\, B_{1,3})\, Id \Big/ (-2\, B_{2,3}\, B_{1,1} - 4\, B_{3,3}\, B_{1,1} - B_{2,2}\, B_{1,1} - 2\, B_{3,2}\, B_{1,1} + B_{2,1}\, B_{1,2} + 2\, B_{1,2} - 4 - 4\, B_{3,1} + 4\, B_{1,3}) + 2\, e1we3 / (-2\, B_{2,3}\, B_{1,1} - 4\, B_{3,3}\, B_{1,1} - B_{2,2}\, B_{1,1} - 2\, B_{3,2}\, B_{1,1} + B_{2,1}\, B_{1,2} + 2\, B_{3,1}\, B_{1,2} + 4\, B_{3,1}\, B_{1,3} + 2\, B_{2,1}\, B_{1,3} - 2\, B_{2,1} + 2\, B_{1,2} - 4 - 4\, B_{3,1} + 4\, B_{1,3}) + e1we2 / (-2\, B_{2,3}\, B_{1,1} - 4\, B_{3,3}\, B_{1,1} - B_{2,2}\, B_{1,1} - 2\, B_{3,2}\, B_{1,1} + B_{2,1}\, B_{1,2} + 2\, B_{3,1}\, B_{1,2} + 4\, B_{3,1}\, B_{1,3} + 2\, B_{2,1}\, B_{1,3} - 2\, B_{2,1} + 2\, B_{1,2} - 4 - 4\, B_{3,1} + 4\, B_{1,3})$$

```
> simplify(cmul[B](p,pinv)),simplify(cmul[B](pinv,p));
```

$$Id, Id$$

Example 2: Same computations as in Example 1 but using a different form. For example, one could try to compute the inverse in Cl(g) where g is the symmetric part of the bilinear form B. Then, we need to use the Clifford product procedure cmul[g] or `&c`[g] with an appropriate index that gives Clifford product in Cl(g).

```
> p:=2*a; # p is a 'cliscalar'
```

$$p := 2\, a$$

```
> cinv(p,g);
```

$$\frac{1}{2\, a}$$

```
> p:=2*(a+b)*c*Id; # p is of grade 0
```

$$p := 2\,(a + b)\, c\, Id$$

```
> cinv(p,g);
```

$$\frac{Id}{2\, c\, a + 2\, c\, b}$$

```
> p:=2+e1+2*e1we2; # p is some polynomial in Cl(B);
```

$$p := 2 + e1 + 2\, e1we2$$

```
> pinv:=cinv(p,g);
```

$$pinv := \frac{2\,(g_{1,2} - g_{2,1} - 1)\, Id}{4\, g_{2,1}\, g_{1,2} + 4\, g_{1,2} - 4\, g_{2,2}\, g_{1,1} + g_{1,1} - 4\, g_{2,1} - 4} + \frac{2\, e1we2}{4\, g_{2,1}\, g_{1,2} + 4\, g_{1,2} - 4\, g_{2,2}\, g_{1,1} + g_{1,1} - 4\, g_{2,1} - 4}$$

$$+\frac{e1}{4\,g_{2,1}\,g_{1,2}+4\,g_{1,2}-4\,g_{2,2}\,g_{1,1}+g_{1,1}-4\,g_{2,1}-4}$$

```
> simplify(cmul[g](pinv,p)),simplify(cmul[g](p,pinv));
```

$$Id, Id$$

```
> simplify(&c[g](pinv,p)),simplify(&c[g](p,pinv));
```

$$Id, Id$$

```
> p:=2+e1we2+2*e1we3;
```

$$p := 2 + e1we2 + 2\,e1we3$$

```
> pinv:=cinv(p,g);
```

$$pinv := (g_{1,2}-2\,g_{3,1}-2-g_{2,1}+2\,g_{1,3})\,Id \Big/ (-4\,g_{3,3}\,g_{1,1}-g_{2,2}\,g_{1,1}-2\,g_{2,3}\,g_{1,1}-2\,g_{3,2}\,g_{1,1}+g_{2,1}\,g_{1,2}+2\,g_{2,1}\,g_{1,3}+4\,g_{3,1}\,g_{1,3}+2\,g_{3,1}\,g_{1,2}+2\,g_{1,2}-4\,g_{3,1}-4-2\,g_{2,1}+4\,g_{1,3})+2\,e1we3\,/\,(-4\,g_{3,3}\,g_{1,1}-g_{2,2}\,g_{1,1}-2\,g_{2,3}\,g_{1,1}-2\,g_{3,2}\,g_{1,1}+g_{2,1}\,g_{1,2}+2\,g_{2,1}\,g_{1,3}+4\,g_{3,1}\,g_{1,3}+2\,g_{3,1}\,g_{1,2}+2\,g_{1,2}-4\,g_{3,1}-4-2\,g_{2,1}+4\,g_{1,3})+e1we2\,/\,(-4\,g_{3,3}\,g_{1,1}-g_{2,2}\,g_{1,1}-2\,g_{2,3}\,g_{1,1}-2\,g_{3,2}\,g_{1,1}+g_{2,1}\,g_{1,2}+2\,g_{2,1}\,g_{1,3}+4\,g_{3,1}\,g_{1,3}+2\,g_{3,1}\,g_{1,2}+2\,g_{1,2}-4\,g_{3,1}-4-2\,g_{2,1}+4\,g_{1,3})$$

```
> simplify(cmul[g](p,pinv)),simplify(cmul[g](pinv,p));
```

$$Id, Id$$

```
>
```

Example 3: Let's see special cases when B is assigned. When the given element p has no inverse, the procedure 'cinv' tries to find the reason why. For example, p could be an idempotent or a nilpotent:

```
> restart:with(Clifford):B:=linalg[diag](1,-1,-1,-1):
> p:=a*(e1+e2); #since p is an isotropic 1-vector in Cl(1,3), p
  is nilpotent
```

$$p := a\,(e1+e2)$$

```
> cmul(p,p);type(p,nilpotent);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$0$$

$$true$$

```
> cinv(p); #testing an error message
Warning, testing why entered argument has no inverse...

Error, (in Clifford:-cinv) element a*(e1+e2) is nilpotent in signature [1, 3]
and as such it has no inverse
```

```
> f:=(1/2)*(1+e1); # f is an idempotent in Cl(1,3)
```

$$f := \frac{1}{2}+\frac{e1}{2}$$

```
> type(f,idempotent);cinv(f); #testing an error message
```

$$true$$

```
Warning, testing why entered argument has no inverse...

Error, (in Clifford:-cinv) element 1/2+1/2*e1 is an idempotent in signature [1, 3] and as such it has no inverse
```

Element f defined below is almost an idempotent, that is, f^2 = k*f where k is of type 'cliscalar'.

```
> B:=linalg[diag](1,-1,-1,-1):
> f:=2*a+2*a*e1; # f is almost an idempotent in Cl(1,3)
```

$$f := 2\,a + 2\,a\,e1$$

```
> cinv(f); #testing an error message
Warning, testing why entered argument has no inverse...

Error, (in Clifford:-cinv) element 'p'=2*a+2*a*e1 is almost an idempotent since Clifford:-cmul(p,p) = 4*a*p and as such it has no inverse in signature [1, 3]
> p:=2*alpha*Id + 2*(e1+e2); # p has the form a*Id+v, cmul(v,v)=0
```

$$p := 2\,\alpha\,Id + 2\,e1 + 2\,e2$$

```
> pinv:=cinv(p);cmul(pinv,p);
```

$$pinv := \frac{Id}{2\,\alpha} - \frac{e1}{2\,\alpha^2} - \frac{e2}{2\,\alpha^2}$$

$$Id$$

```
> p:=2+3*e2we4; # p is of the type 'gencomplex'
```

$$p := 2 + 3\,e2we4$$

```
> type(p,gencomplex);
```

$$true$$

```
> pinv:=cinv(p);
```

$$pinv := \frac{2\,Id}{13} - \frac{3\,e2we4}{13}$$

```
> cmul(pinv,p);
```

$$Id$$

```
> p:=2-e2+e3+a*e2we3; # p is of the type 'genquaternion'
```

$$p := 2 - e2 + e3 + a\,e2we3$$

```
> type(p,genquaternion);
```

$$true$$

```
> pinv:=cinv(p);simplify(cmul(pinv,p));
```

$$pinv := \frac{2\,Id}{6 + a^2} + \frac{e2}{6 + a^2} - \frac{e3}{6 + a^2} - \frac{a\,e2we3}{6 + a^2}$$

$$Id$$

```
> p:=2+e1we2+e2we3+e3we1; # p is even in Cl(1,3)
```

$$p := 2 + e1we2 + e2we3 + e3we1$$

```
> type(p,evenelement);
```

$$true$$

```
> pinv:=cinv(p);
```

$$pinv := \frac{2\,Id}{3} - \frac{e1we2}{3} + \frac{e1we3}{3} - \frac{e2we3}{3}$$

```
> reorder(cmul(pinv,p)),reorder(cmul(p,pinv));
```

$$Id, Id$$

Example 4: Some more examples:

```
> B:=linalg[diag](1,1,-1):
> f:=(1/2)*(1+e1):
> type(f,idempotent);
```

$$true$$

```
> cinv(f,B); #testing an error message
Warning, testing why entered argument has no inverse...

Error, (in Clifford:-cinv) element 1/2+1/2*e1 is an idempotent in signature [2,
1] and as such it has no inverse
```

```
> p:=1+e1-e3+e2we3;
```

$$p := 1 + e1 - e3 + e2we3$$

```
> pinv:=cinv(p,B);
```

$$pinv := \frac{e1}{2} + \frac{e1we2}{2} + \frac{e2we3}{2} - \frac{e1we2we3}{2}$$

```
> cmul[B](pinv,p);
```

$$Id$$

```
>
```

See Also: Clifford:-qinv, Clifford:-`type/nilpotent`, Clifford:-`type/idempotent`, Clifford:-`type/genquaternion`, Clifford:-`type/gencomplex`, Clifford:-maxindex, Clifford:-`type/oddelement`, Clifford:-`type/evenelement`, Clifford:-cmul, Clifford:-`type/clipolynom`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-`type/clibasmon` - define a new type: a basis Clifford monomial

Calling Sequence:

type(p, clibasmon);

Parameters:

p - an algebraic expression of type 'anything'

Description:

- Basis elements in the standard Clifford basis returned by the procedure 'cbasis' (see cbasis) are of this new type.
- The procedure returns 'true' or 'false' depending whether its argument is or is not of the type 'clibasmon'.
- Permutations of the indices also yield elements if this type.

Examples:

```
> restart:with(Clifford):
> clibasis:=cbasis(3);
```

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

```
> map(type,clibasis,clibasmon);
```

$$[true, true, true, true, true, true, true, true]$$

```
> type(e1we2,clibasmon);
```

$$true$$

```
> type(e2we1,clibasmon);
```

$$true$$

```
> type(2*e1we2,clibasmon);
```

$$false$$

```
> type(2+e1we2,clibasmon);
```

$$false$$

```
> type(eiwejwe1we2,clibasmon);
```

$$true$$

```
>
```

See Also: Clifford:-`type/cliscalar`, Clifford:-`type/climon`, Clifford:-`type/clipolynom`, Clifford:-`type/cliprod`, Clifford:-cbasis

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-clibilinear - make a procedure bilinear with respect to Clifford scalars

Calling Sequence:

clibilinear(a1,a2,K);
clibilinear(a1,a2,K,Ps);

Parameters:

a1, a2 - expressions of the type 'cliscalar', 'clibasmon', 'climon', 'clipolynom', or 'cliprod'
K - expression of the type 'name' or 'procedure'
Ps - (optional) sequence of arguments of type name, symbol, matrix, array that can be passed on to K

Description:

- Procedure 'clibilinear' makes any previously defined procedure K bilinear with respect to Clifford scalars of the type 'cliscalar' in the first two arguments which are expected to be of the type 'cliscalar', 'clibasmon', 'climon', 'clipolynom', or 'cliprod'.
- The name of the procedure to be made bilinear is entered as the third argument.
- This procedure can be used to make new procedures bilinear; it has been used to define, for example the Clifford multiplication procedures 'cmul' and 'cmulQ' as bilinear (and, in fact, as multilinear) operations. See cmul and cmulQ for more help.
- Recursive use of this procedure allows one to define multilinear procedures, e.g., see the code of 'cmul'.
- Any additional parameters are passed on to the procedure K. See below.

Examples:

```
> restart:with(Clifford):
  _warnings_flag;          #warnings are printed by default
  #_warnings_flag:=false:  #printing warnings can be stopped
```

$$true$$

Example 1: Using just three parameters:

```
> p1:=3*Pi+2*e2we3;p2:=2*alpha*e1we2we3+(2*e1-e3)/sqrt(a^2+b^2);
  p3:=-2*`&C`(e1,e2)+2*e3-3*`&C`(e2,e4,e5);
```

$$p1 := 3\,\pi + 2\,e2we3$$

$$p2 := 2\,\alpha\,e1we2we3 + \frac{2\,e1 - e3}{\sqrt{a^2 + b^2}}$$

$$p3 := -2\,(e1\ \&C\ e2) + 2\,e3 - 3\ \&C(e2, e4, e5)$$

```
> clibilinear(p1,p2,K);
  clibilinear(p1,p3,X);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
```

```
clude &C and &C[K]. Type ?cliprod for help.
```

$$6\,\pi\,\alpha\,\mathrm{K}(Id, e1we2we3) + 4\,\alpha\,\mathrm{K}(e2we3, e1we2we3) + \frac{6\,\pi\,\mathrm{K}(Id, e1)}{\sqrt{a^2+b^2}} + \frac{4\,\mathrm{K}(e2we3, e1)}{\sqrt{a^2+b^2}} - \frac{3\,\pi\,\mathrm{K}(Id, e3)}{\sqrt{a^2+b^2}} - \frac{2\,\mathrm{K}(e2we3, e3)}{\sqrt{a^2+b^2}}$$

$$-6\,\pi\,\mathrm{X}(Id, e1\ \&\mathrm{C}\ e2) - 4\,\mathrm{X}(e2we3, e1\ \&\mathrm{C}\ e2) + 6\,\pi\,\mathrm{X}(Id, e3) + 4\,\mathrm{X}(e2we3, e3) - 9\,\pi\,\mathrm{X}(Id, \&\mathrm{C}(e2, e4, e5)) - 6\,\mathrm{X}(e2we3, \&\mathrm{C}(e2, e4, e5))$$

```
> clibilinear(1+e1+e1we2,p3,H);
```

$$-2\,\mathrm{H}(Id, e1\ \&\mathrm{C}\ e2) - 2\,\mathrm{H}(e1, e1\ \&\mathrm{C}\ e2) - 2\,\mathrm{H}(e1we2, e1\ \&\mathrm{C}\ e2) + 2\,\mathrm{H}(Id, e3) + 2\,\mathrm{H}(e1, e3) + 2\,\mathrm{H}(e1we2, e3) - 3\,\mathrm{H}(Id, \&\mathrm{C}(e2, e4, e5)) - 3\,\mathrm{H}(e1, \&\mathrm{C}(e2, e4, e5)) - 3\,\mathrm{H}(e1we2, \&\mathrm{C}(e2, e4, e5))$$

```
> clibilinear(Pi*e1-e2we3,(2*e2+e3)/sqrt(q^2+1)+1,K);
```

$$\frac{2\,\pi\,\mathrm{K}(e1, e2)}{\sqrt{q^2+1}} - \frac{2\,\mathrm{K}(e2we3, e2)}{\sqrt{q^2+1}} + \frac{\pi\,\mathrm{K}(e1, e3)}{\sqrt{q^2+1}} - \frac{\mathrm{K}(e2we3, e3)}{\sqrt{q^2+1}} + \pi\,\mathrm{K}(e1, Id) - \mathrm{K}(e2we3, Id)$$

```
>
```

Example 2: Additional parameters beyond the third parameter are passed on to the procedure K:

```
> clibilinear(e1,e2,K,P);
```

$$\mathrm{K}(e1, e2, P)$$

```
> clibilinear(2+3*e1-e4,e4-5*e6,K,P1,P2);
```

$$2\,\mathrm{K}(Id, e4, P1, P2) + 3\,\mathrm{K}(e1, e4, P1, P2) - \mathrm{K}(e4, e4, P1, P2) - 10\,\mathrm{K}(Id, e6, P1, P2) - 15\,\mathrm{K}(e1, e6, P1, P2) + 5\,\mathrm{K}(e4, e6, P1, P2)$$

```
> assigned(Cliplus);
```

$$true$$

```
>
```

See Also: Clifford:-cmulQ, Clifford:-cmul, Clifford:-`type/clipolynom`, Clifford:-`type/cliscalar`, Clifford:-`type/cliprod`, Clifford:-cbasis

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-clicollect - collect a Clifford polynomial with respect to its indeterminates

Calling Sequence:

clicollect(p);

Parameters:

p - expression of the type 'cliscalar', 'clipolynom', or 'climatrix'

Description:

- Procedure 'clicollect' collects monomial terms in a Clifford polynomial.
- It does not reorder monomial terms. If reordering is needed, use 'reorder' procedure. See reorder for more help.
- This procedure may be applied to matrices with entries in a Clifford algebra, that is, matrices of the type `type/climatrix`.
- Recall that when package 'Cliplus' is loaded in, the following two types in CLIFFORD: `type/climon` and `type/clipolynom` get redefined and include now polynomial and monomial terms with expressions of `type/cliprod`. When such expressions are present, 'clicollect' collects with respect to these terms as well.

Examples:

```
> restart:with(Clifford):
> p1:=e1+alpha*e1-e2we1+3*e1we2-a*e1we2we3;
```

$$p1 := e1 + \alpha\, e1 - e2we1 + 3\, e1we2 - a\, e1we2we3$$

```
> clicollect(p1);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$-e2we1 + 3\, e1we2 + (1 + \alpha)\, e1 - a\, e1we2we3$$

```
> reorder(%);
```

$$4\, e1we2 + (1 + \alpha)\, e1 - a\, e1we2we3$$

```
> clicollect(6+7*Id);
```

$$13\, Id$$

```
> p2:=cmulQ(p1,p1);
```

$$p2 := (B_{1,1}\,\alpha^2 + 2\,B_{1,1}\,\alpha + B_{1,1} - a^2\,B_{3,2}\,B_{1,3}\,B_{2,1} + a^2\,B_{3,2}\,B_{1,1}\,B_{2,3} + a^2\,B_{3,1}\,B_{1,3}\,B_{2,2} - a^2\,B_{3,3}\,B_{1,1}\,B_{2,2} - a^2\,B_{3,1}\,B_{1,2}\,B_{2,3} + a^2\,B_{3,3}\,B_{1,2}\,B_{2,1} - 16\,B_{1,1}\,B_{2,2} + 16\,B_{1,2}\,B_{2,1})\,Id + a\,(-a\,B_{1,3}\,B_{2,2} + B_{2,1} + a\,B_{1,2}\,B_{2,3} + \alpha\,B_{2,1} + B_{1,2} + \alpha\,B_{1,2} + a\,B_{3,1}\,B_{2,2} - a\,B_{3,2}\,B_{2,1})\,e1we3 - a\,(a\,B_{3,1}\,B_{1,2} - a\,B_{3,2}\,B_{1,1} + 2\,B_{1,1}\,\alpha + a\,B_{1,1}\,B_{2,3} + 2\,B_{1,1} - a\,B_{1,3}\,B_{2,1})\,e2we3 - (a\,\alpha\,B_{1,3} - a^2\,B_{3,3}\,B_{2,1} + a\,\alpha\,B_{3,1} + a^2\,B_{3,3}\,B_{1,2} + a\,B_{3,1} - a^2\,B_{3,2}\,B_{1,3} + 16\,B_{1,2} + a\,B_{1,3} - 16\,B_{2,1} + a^2\,B_{3,1}\,B_{2,3})\,e1we2$$

$$+4\,(a\,B_{1,2}\,B_{2,3}-\alpha\,B_{1,2}+\alpha\,B_{2,1}-a\,B_{1,3}\,B_{2,2}+B_{2,1}+a\,B_{3,2}\,B_{2,1}-B_{1,2}-a\,B_{3,1}\,B_{2,2})\,e1$$
$$+8\,a\,(-B_{2,1}+B_{1,2})\,e1we2we3+4\,a\,(-B_{1,1}\,B_{2,3}-B_{3,2}\,B_{1,1}+B_{1,3}\,B_{2,1}+B_{3,1}\,B_{1,2})\,e2$$
$$-8\,a\,(B_{1,2}\,B_{2,1}-B_{1,1}\,B_{2,2})\,e3$$

```
> clicollect(%);
```

$$(B_{1,1}\,\alpha^2+2\,B_{1,1}\,\alpha+B_{1,1}-a^2\,B_{3,2}\,B_{1,3}\,B_{2,1}+a^2\,B_{3,2}\,B_{1,1}\,B_{2,3}+a^2\,B_{3,1}\,B_{1,3}\,B_{2,2}$$
$$-a^2\,B_{3,3}\,B_{1,1}\,B_{2,2}-a^2\,B_{3,1}\,B_{1,2}\,B_{2,3}+a^2\,B_{3,3}\,B_{1,2}\,B_{2,1}-16\,B_{1,1}\,B_{2,2}+16\,B_{1,2}\,B_{2,1})\,Id+$$
$$a\,(-a\,B_{1,3}\,B_{2,2}+B_{2,1}+a\,B_{1,2}\,B_{2,3}+\alpha\,B_{2,1}+B_{1,2}+\alpha\,B_{1,2}+a\,B_{3,1}\,B_{2,2}-a\,B_{3,2}\,B_{2,1})\,e1we3$$
$$-a\,(a\,B_{3,1}\,B_{1,2}-a\,B_{3,2}\,B_{1,1}+2\,B_{1,1}\,\alpha+a\,B_{1,1}\,B_{2,3}+2\,B_{1,1}-a\,B_{1,3}\,B_{2,1})\,e2we3-($$
$$a\,\alpha\,B_{1,3}-a^2\,B_{3,3}\,B_{2,1}+a\,\alpha\,B_{3,1}+a^2\,B_{3,3}\,B_{1,2}+a\,B_{3,1}-a^2\,B_{3,2}\,B_{1,3}+16\,B_{1,2}+a\,B_{1,3}$$
$$-16\,B_{2,1}+a^2\,B_{3,1}\,B_{2,3})\,e1we2$$
$$+4\,(a\,B_{1,2}\,B_{2,3}-\alpha\,B_{1,2}+\alpha\,B_{2,1}-a\,B_{1,3}\,B_{2,2}+B_{2,1}+a\,B_{3,2}\,B_{2,1}-B_{1,2}-a\,B_{3,1}\,B_{2,2})\,e1$$
$$+8\,a\,(-B_{2,1}+B_{1,2})\,e1we2we3+4\,a\,(-B_{1,1}\,B_{2,3}-B_{3,2}\,B_{1,1}+B_{1,3}\,B_{2,1}+B_{3,1}\,B_{1,2})\,e2$$
$$-8\,a\,(B_{1,2}\,B_{2,1}-B_{1,1}\,B_{2,2})\,e3$$

```
> M:=linalg[matrix](2,2,[2+3*Id,2*e1-a*e1+e3,e4,e3-e4]);
```

$$M:=\begin{bmatrix}2+3\,Id & 2\,e1-a\,e1+e3\\ e4 & e3-e4\end{bmatrix}$$

```
> clicollect(M);
```

$$\begin{bmatrix}5\,Id & -(-2+a)\,e1+e3\\ e4 & e3-e4\end{bmatrix}$$

```
> with(Cliplus):
  p3:=cliexpand(p1);p4:=cliexpand(p2);
```

$p3:=$

$$e1+\alpha\,e1+4\,(e1\ \&C\ e2)-4\,B_{1,2}\,Id-a\ \&C(e1,e2,e3)+a\,B_{2,3}\,e1-a\,B_{1,3}\,e2+a\,B_{1,2}\,e3$$

$p4:=$

$$Id\,B_{1,1}\,\alpha^2+8\,a\,B_{1,2}\ \&C(e1,e2,e3)+2\,Id\,B_{1,1}\,\alpha-a\,B_{1,3}\,(e1\ \&C\ e2)$$
$$+a\,B_{1,2}\,(e1\ \&C\ e3)+a^2\,B_{3,3}\,{B_{1,2}}^2\,Id+a\,B_{3,1}\,B_{1,2}\,Id-a\,\alpha\,B_{1,3}\,(e1\ \&C\ e2)-4\,e1\,\alpha\,B_{1,2}$$
$$+4\,e1\,\alpha\,B_{2,1}+a\,\alpha\,B_{3,1}\,B_{1,2}\,Id-a^2\,B_{3,2}\,B_{1,3}\,B_{1,2}\,Id-a^2\,B_{1,3}\,B_{2,2}\,(e1\ \&C\ e3)$$
$$+a^2\,{B_{1,3}}^2\,B_{2,2}\,Id+a^2\,B_{1,2}\,B_{2,3}\,(e1\ \&C\ e3)+a^2\,B_{3,1}\,B_{2,2}\,(e1\ \&C\ e3)$$
$$-a^2\,B_{3,2}\,B_{2,1}\,(e1\ \&C\ e3)-a\,B_{2,1}\,B_{1,3}\,Id+a\,\alpha\,B_{2,1}\,(e1\ \&C\ e3)+a\,\alpha\,B_{1,2}\,(e1\ \&C\ e3)$$
$$+a\,B_{2,1}\,(e1\ \&C\ e3)-4\,e1\,a\,B_{1,2}\,B_{2,3}-4\,e1\,a\,B_{1,3}\,B_{2,2}+4\,e1\,a\,B_{3,2}\,B_{2,1}-4\,e1\,a\,B_{3,1}\,B_{2,2}$$
$$-4\,a\,e2\,B_{1,1}\,B_{2,3}-4\,a\,e2\,B_{3,2}\,B_{1,1}-4\,a\,e2\,B_{1,3}\,B_{2,1}+4\,a\,e2\,B_{3,1}\,B_{1,2}+8\,a\,e3\,B_{1,1}\,B_{2,2}$$
$$-2\,a\,B_{1,1}\,(e2\ \&C\ e3)+a^2\,B_{3,3}\,B_{2,1}\,(e1\ \&C\ e2)-a\,\alpha\,B_{3,1}\,(e1\ \&C\ e2)$$
$$-a^2\,B_{3,3}\,B_{1,2}\,(e1\ \&C\ e2)+a^2\,B_{3,2}\,B_{1,3}\,(e1\ \&C\ e2)-a^2\,B_{3,1}\,B_{2,3}\,(e1\ \&C\ e2)$$

$$- a B_{3,1} (e1 \&C e2) + 8 a B_{2,1} B_{2,3} e1 + 8 a B_{1,2} B_{1,3} e2 - 16 B_{1,2} (e1 \&C e2) + 16 B_{1,2}^{2} Id$$
$$+ 16 B_{2,1} (e1 \&C e2) - 16 Id B_{1,1} B_{2,2} - a^2 B_{1,2} B_{2,3} B_{1,3} Id - a \alpha B_{2,1} B_{1,3} Id + Id B_{1,1}$$
$$- Id a^2 B_{3,3} B_{1,1} B_{2,2} + Id a^2 B_{3,1} B_{1,2} B_{2,3} + 4 e1 B_{2,1} - 4 e1 B_{1,2} - a^2 B_{3,1} B_{1,2} (e2 \&C e3)$$
$$+ a^2 B_{3,2} B_{1,1} (e2 \&C e3) - a^2 B_{1,1} B_{2,3} (e2 \&C e3) + a^2 B_{1,1} B_{2,3}^{2} Id$$
$$+ a^2 B_{1,3} B_{2,1} (e2 \&C e3) - 2 a B_{1,1} \alpha (e2 \&C e3) + 2 a B_{1,1} B_{2,3} Id$$
$$- 8 a B_{2,1} \&C(e1, e2, e3) - 8 a B_{1,2}^{2} e3 - a^2 B_{1,3} B_{2,1} B_{2,3} Id + 2 a B_{1,1} \alpha B_{2,3} Id$$

```
> p3:=clicollect(p3);
```

$$p3 := -4 B_{1,2} Id + (1 + \alpha + a B_{2,3}) e1 - a B_{1,3} e2 + a B_{1,2} e3 + 4 (e1 \&C e2) - a \&C(e1, e2, e3)$$

```
> p4:=clicollect(p4);
```

$$p4 := (B_{1,1} \alpha^2 - a^2 B_{3,2} B_{1,3} B_{1,2} + a^2 B_{1,3}^{2} B_{2,2} - a B_{1,3} B_{2,1} + 2 a B_{1,1} B_{2,3} + 16 B_{1,2}^{2}$$
$$- 16 B_{1,1} B_{2,2} - a^2 B_{1,2} B_{2,3} B_{1,3} + 2 B_{1,1} \alpha + a^2 B_{3,3} B_{1,2}^{2} + a B_{3,1} B_{1,2} + B_{1,1}$$
$$- a^2 B_{3,3} B_{1,1} B_{2,2} + a \alpha B_{3,1} B_{1,2} + a^2 B_{3,1} B_{1,2} B_{2,3} - a^2 B_{1,3} B_{2,1} B_{2,3} + 2 a B_{1,1} \alpha B_{2,3}$$
$$+ a^2 B_{1,1} B_{2,3}^{2} - a \alpha B_{2,1} B_{1,3}) Id - 4 (-B_{2,1} - 2 a B_{2,1} B_{2,3} + a B_{3,1} B_{2,2} + a B_{1,3} B_{2,2}$$
$$+ a B_{1,2} B_{2,3} + \alpha B_{1,2} - \alpha B_{2,1} + B_{1,2} - a B_{3,2} B_{2,1}) e1$$
$$+ 4 a (B_{3,1} B_{1,2} - B_{1,3} B_{2,1} + 2 B_{1,2} B_{1,3} - B_{3,2} B_{1,1} - B_{1,1} B_{2,3}) e2$$
$$- 8 a (B_{1,2}^{2} - B_{1,1} B_{2,2}) e3$$
$$- a (a B_{3,1} B_{1,2} - a B_{3,2} B_{1,1} + 2 B_{1,1} \alpha + a B_{1,1} B_{2,3} + 2 B_{1,1} - a B_{1,3} B_{2,1}) (e2 \&C e3)$$
$$+ 8 a (-B_{2,1} + B_{1,2}) \&C(e1, e2, e3) - (a \alpha B_{1,3} - a^2 B_{3,3} B_{2,1} + a \alpha B_{3,1} + a^2 B_{3,3} B_{1,2}$$
$$+ a B_{3,1} - a^2 B_{3,2} B_{1,3} + 16 B_{1,2} + a B_{1,3} - 16 B_{2,1} + a^2 B_{3,1} B_{2,3}) (e1 \&C e2) + a$$
$$(-a B_{1,3} B_{2,2} + B_{2,1} + a B_{1,2} B_{2,3} + \alpha B_{2,1} + B_{1,2} + \alpha B_{1,2} + a B_{3,1} B_{2,2} - a B_{3,2} B_{2,1})$$
$$(e1 \&C e3)$$

```
> reorder(clieval(p3)-p1);
```

$$0$$

```
> simplify(clieval(p4)-p2);
```

$$0$$

```
>
```

For more information, see cliexpand and clieval.

See Also: Clifford:-cmulQ, Clifford:-clisort, Clifford:-reorder, Clifford:-`type/clipolynom`, Clifford:-`type/cliscalar`, Clifford:-`type/cliscalar`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-clidata - display information about any Clifford algebra Cl(V,Q) in dimension up to 9

Calling Sequence:

clidata();
clidata(Q);
clidata([p,q]);

Parameters:

Q - of type 'matrix': a quadratic form in V of signature [p,q]
[p,q] - of type 'list': signature of Q

Description:

- Procedure 'clidata' returns a list containing basic information about the orthogonal Clifford algebra Cl(B) of the given bilinear form B which is assumed to have been diagonalized or defined as a diagonal matrix of type 'diagmatrix'. See `type/diagmatrix` for more help.
- One can get a list of all signatures for which data is available by using procedure all_sigs.
- The procedure may be called:

- without an argument as in clidata() in which case it returns information about the Clifford algebra of the currently defined bilinear form B; if the form is not defined then an error message is returned;
- with one argument of type 'matrix' as in clidata(B) where B is a bilinear form;
- with one argument of type 'list' as in clidata([p,q]) where the list [p,q] is the signature of Q where Q is the quadratic form defined by the symmetric part of B.

- It returns a list with the following information:

- the first entry is the string 'real', 'complex', or 'quaternionic' depending whether the spinor representation of Cl(Q) is over the field K of the reals, complexes, or quaternions;
- the second entry is the dimension of the spinor representation over the field K;
- the third entry is the string 'simple' or 'semisimple' depending on the structure of the algebra;
- the fourth entry is a primitive idempotent f which may be used to generate a left or right minimal ideal in the algebra;

  NOTE: the idempotents are stored here in an unevaluated form so that they could be easily recognized as Clifford products of simpler projection operators. The number of factors in these products is given by the the value of q - RHnumber(q-p) where [p,q] is the signature of Q (see Bsignature for more help) and 'RHnumber' is a procedure which gives the Radon-Hurwitz function (see RHnumber for more help).

- the fifth entry is a list of basis monomials ordered by grade which generate S=Cl(Q)f and S=fCl(Q);

- the sixth entry is a list of basis monomials ordered by grade which gives a basis for K: in terms of these monomials matrices representing Clifford polynomials will be written by the procedure 'spinorKrepr' (see spinorKrepr for more help).
- the seventh entry is a list of basis monomials ordered by grade which generate S over K.

Examples:

```
> restart:with(Clifford):
> all_sigs(1..9);
```

$$[[0,1],[0,2],[0,3],[0,4],[0,5],[0,6],[0,7],[0,8],[0,9],[1,0],[1,1],[1,2],[1,3],[1,4],[1,5],[1,6],[1,7],[1,8],[2,0],[2,1],[2,2],[2,3],[2,4],[2,5],[2,6],[2,7],[3,0],[3,1],[3,2],[3,3],[3,4],[3,5],[3,6],[4,0],[4,1],[4,2],[4,3],[4,4],[4,5],[5,0],[5,1],[5,2],[5,3],[5,4],[6,0],[6,1],[6,2],[6,3],[7,0],[7,1],[7,2],[8,0],[8,1],[9,0]]$$

```
> B:=linalg[diag](1,1,1):eval(makealiases(7,'ordered')):
> clidata(); #information about Cl(3)
```

$$\left[complex, 2, simple, \frac{Id}{2}+\frac{e1}{2}, [Id, e2, e3, e23], [Id, e23], [Id, e2]\right]$$

```
> clidata([3,4]); #information about Cl(3,4)
```

$$\left[complex, 8, simple, 'cmulQ'\left(\frac{Id}{2}+\frac{e15}{2}, \frac{Id}{2}+\frac{e26}{2}, \frac{Id}{2}+\frac{e37}{2}\right), [Id, e1, e2, e3, e4, e12, e13, e14, e23, e24, e34, e123, e124, e134, e234, e1234], [Id, e4], [Id, e1, e2, e3, e12, e13, e23, e123]\right]$$

```
> clidata([1,3]); #information about Cl(1,3)
```

$$\left[quaternionic, 2, simple, \frac{Id}{2}+\frac{e14}{2}, [Id, e1, e2, e3, e12, e13, e23, e123], [Id, e2, e3, e23], [Id, e1]\right]$$

```
> clidata([3,1]); #information about Cl(3,1)
```

$$\left[real, 4, simple, 'cmulQ'\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e34}{2}\right), [Id, e2, e3, e23], [Id], [Id, e2, e3, e23]\right]$$

```
> clidata([5,1]); #information about Cl(5,1)
```

$$\left[quaternionic, 4, simple, 'cmulQ'\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e56}{2}\right), [Id, e2, e3, e4, e5, e23, e24, e25, e34, e35, e45, e234, e235, e245, e345, e2345], [Id, e23, e24, e34], [Id, e2, e5, e25]\right]$$

```
> clidata([4,3]); #information about Cl(4,3)
```

$$\left[real, 8, semisimple, 'cmulQ'\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e25}{2}, \frac{Id}{2}+\frac{e36}{2}, \frac{Id}{2}+\frac{e47}{2}\right), [Id, e2, e3, e4, e23, e24, e34, e234], [Id], [Id, e2, e3, e4, e23, e24, e34, e234]\right]$$

```
> clidata([2,2]); #information about Cl(2,2)
```

$$\left[real, 4, simple, 'cmulQ'\left(\frac{Id}{2}+\frac{e13}{2}, \frac{Id}{2}+\frac{e24}{2}\right), [Id, e1, e2, e12], [Id], [Id, e1, e2, e12]\right]$$

```
> clidata([4,1]); #information about Cl(4,1)
```

$$\left[complex, 4, simple, 'cmulQ'\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e45}{2}\right), [Id, e2, e3, e4, e23, e24, e34, e234], [Id, e23], [Id, e2, e4, e24]\right]$$

```
>
```

See Also: Clifford:-spinorKrepr, Clifford:-spinorKbasis, Clifford:-Kfield, Clifford:-Bsignature

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-CLIFFORD_ENV **-** displays current values of environmental variables used in 'CLIFFORD' and other supplementary packages

Calling Sequence:

CLIFFORD_ENV();

Parameters:

none

Description:

- Procedure 'CLIFFORD_ENV' displays current values of all environmental variables used in 'CLIFFORD', 'Bigebra', 'GTP', 'Cliplus', and 'Octonion' packages. The default values of these variables are set when each package is loaded, but they can be changed by the user. If any of the current values is not in the range, a warning message is displayed.
- Environmental variables used in 'CLIFFORD' and their meanings are as follows:

  - dim_V - default value is 9, while possible values are positive integers between 1 and 9, inclusive: It sets the dimension of the vector space V in which a bilinear form B or a quadratic form Q may be defined or left undefined. Then, 'CLIFFORD' assumes that the dimension of the Clifford algebra Cl(V,B), or Cl(V,Q), or the Grassmann algebra /\ V is 2^(dim_V). The value of dim_V limits the maximum grade in the output from the procedure wedge when the indices of basis monomials are symbolic. The value of dim_V can be set by the user, but it is overwritten by wedge at its **first** invocation whenever bilinear from 'B' is defined or re-defined. When the value of dim_V is reduced by wedge, a warning message is printed (no such warning message is printed when the value of dim_V remains the same or is increased). These warning messages can be suppressed if 'warnings_flag', whose default value is 'true', is set to 'false' by the user. See **Example 1** below.

  - _default_Clifford_product - default value is 'cmulRS' (see cmulRS), while other possible values are: 'cmulNUM' (see cmulNUM), 'cmulgen' (see cmulgen) or 'cmul_user_defined' (see useproduct). Procedures 'cmulRS' and 'cmulNUM' essentially compute Clifford product between any two elements in the Clifford algebra (B) which are of type clibasmon, climon, clipolynom, or cliscalar using either Rota-Stein cliffordization technique or the recursive definition of the Clifford product due to Chevalley (see cmul for more information). Other possible values for this global variable are: 'cmulgen', a generic name for Clifford product that is also a name of a procedure cmulgen, and 'cmul_user_defined', a user defined Clifford product that can take three arguments x, y, and a name. The value of _default_Clifford_product is set by Clifford:-setup to 'cmulRS' upon loading of CLIFFORD. It can be changed by the user either by a direct assignment (see below) or, by using procedure useproduct which checks correctness of the user input and prints an informational message. See **Example 2** below.

- _prolevel - default value is 'false', while possible values are 'true' or 'false': If _prolevel is set to 'false', cliparse check user's input for syntax errors which is time consuming. If it is set to 'false', no syntax checking is done, that is, cliparse always returns 'true', and computations are done faster. However, results could be unexpected. Setting _prolevel to 'true' is recommended only for experienced users. See **Example 3** below.

- _shortcut_in_minimalideal - default value is 'true', while possible values are 'true' or 'false': When set to 'true', procedure minimalideal checks user's input if it matches input previously used for which results were computed and stored under procedure clidata. If yes, results are retrieved from a library file and returned to the user. If not, procedure continues to compute new results. If _shortcut_in_minimalideal is set to 'false', procedure doesn't check user's input for a match, but it moves directly to find a new output.

- _shortcut_in_Kfield - default value is 'true', while possible values are 'true' or 'false': When set to 'true', procedure Kfield checks user's input if it matches input previously used for which results were computed and stored under procedure clidata. If yes, results are retrieved from a library file and returned to the user. If not, procedure continues to compute new results. If _shortcut_in_Kfield ideal is set to 'false', procedure doesn't check user's input for a match, but it moves directly to find a new output.

- _shortcut_in_spinorKbasis - default value is 'true', while possible values are 'true' or 'false': When set to 'true', procedure spinorKbasis checks user's input if it matches input previously used for which results were computed and stored under procedure clidata. If yes, results are retrieved from a library file and returned to the user. If not, procedure continues to compute new results. If _shortcut_in_spinorKbasis ideal is set to 'false', procedure doesn't check user's input for a match, but it moves directly to find a new output.

- _shortcut_in_spinorKrepr - default value is 'true', while possible values are 'true' or 'false': When set to 'true', procedure spinorKrepr checks user's input if it matches input previously used for which results were computed and stored under procedure clidata. If yes, it calls procedure matKrepr which in turn uses previously computed and stored results to compute a new result. If not, procedure continues to compute new results. If _shortcut_in_spinorKrepr ideal is set to 'false', procedure doesn't check user's input for a match, but it moves directly to find a new output.

- _warnings_flag - default value is 'true', while possible values are 'true' or 'false': When set to 'true', certain procedures in 'CLIFFORD' print warning messages, for example, procedure wedge. If set to 'false', no warning messages are printed.

- _scalartypes - contains Maple types which are considered to be 'scalars' by procedures `type/cliscalar`, clibilinear, and clilinear. By adding new types to this set, user can make procedures in 'CLIFFORD' linear or bilinear, as appropriate, with respect to the new types added to _scalartypes. It is also conceivable that user may want procedures in 'CLIFFORD' to recognize additional types and treat them as 'scalars'. Thus it will be enough to add them to this set.

_quatbasis - contains a list of elements in Cl(3,0) which are used as default by procedure qdisplay to display quaternions in the quaternion basis contained in the list. That set can also be changed provided its elements give another ring in the given Clifford algebra that is isomorphic with the quaternionic ring H.

- There are no environmental variables in 'Cliplus'. Instead, there are macros (see macro) that are being defined in Cliplus:-setup:

  - macro(cmul = climul) - redefines procedure cmul internally as climul so that 'cmul' can handle now `type/cliprod` in polynomials
  - macro(cmulQ = climul) - redefines procedure cmulQ internally as climul so that 'cmulQ' can handle now `type/cliprod` in polynomials
  - macro(`&c` = climul) - redefines procedure `&c` internally as climul so that `&c` can handle now `type/cliprod` in polynomials
  - macro(`&cQ` = climul) - redefines procedure `&cQ` internally as climul so that `&cQ` can handle now `type/cliprod` in polynomials
  - macro(reversion = clirev) - redefines procedure reversion internally as clirev so that 'reversion' can handle now `type/cliprod` in polynomials
  - macro(LC = LCbig) - redefines procedure LC internally as LCbig so that reversion can handle now `type/cliprod` in polynomials
  - macro(RC = RCbig) - redefines procedure RC internally as RCbig so that reversion can handle now `type/cliprod` in polynomials

- There are two environmental variables in 'Bigebra' package. For more information see Bigebra:-help.

  - _CLIENV[_SILENT] - default value is 'false', while possible values are 'true' or 'false'. It controls displaying warning messages when 'Bigebra' is loaded
  - _CLIENV[_QDEF_PREFACTOR] - default value is -1: it controls prefactor in gswitch from Bigebra package.

- The following environmental variables are used in 'Octonion' package:

  - _octbasis = [Id, e1, e2, e3, e4, e5, e6, e7] - This list contains default octonion basis from Cl(0,7). That is, octonions are treated as paravectors in Cl(0,7)

  - _pureoctbasis = [e1, e2, e3, e4, e5, e6, e7] - This list contains default pure octonion basis from Cl(0,7), That is, pure vector part is a 1-vector in Cl(0,7). See also purevectorpart in the 'Octonion' package.

- _default_Fano_triples = [[1,3,7], [1,2,4], [1,5,6], [2,3,5], [2,6,7], [3,4,6], [4,5,7]] - This list contains default Fano triples (see `type/Fano_triples`) that define octonionic multiplication omul in the non-associative algebra of octonions.

- _default_squares = [-Id, -Id, -Id, -Id, -Id, -Id, -Id] - List that keeps default square values of the unit basis octonions, that is omul(e1,e1) = -Id, omul(e2,e2) = -Id,..., omul(e7,e7) = -Id.

- _default_Clifford_product has value changed to 'cmulNUM' upon loading this package. This is because procedure cmulNUM is more efficient than cmulRS when the bilinear form B is numeric which is the case here since quaternions require B to be a 3 by 3 identity matrix.

- There are no additional environmental variables in the 'GTP' package.
- **Note that 'Bigebra' for Maple 6 needs to be loaded with the command >with(Bigebra):**

Examples:

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

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


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


`>>> Global variables defined in Cliplus:-setup are now available and have thes
e values:  <<<`
`************* Start *************`
macro(Clifford:-cmul = climul)
macro(Clifford:-cmulQ = climul)
macro(`&c` = climul)
macro(`&cQ` = climul)
macro(Clifford:-reversion = clirev)
macro(Clifford:-LC = LCbig)
macro(Clifford:-RC = RCbig)
```

```
`Warning, new definitions for type/climon and type/clipolynom now include &C`
`************* End *************`



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



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


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

```
>
```

Example 1: Changing value of dim_V:

```
> dim_V; #the default value is (max) 9
```

$$9$$

```
> p1:=e1we4we3weiwej-2*elwekweswet;maxgrade(p1);
  p2:=e1we4we3weiwej-2*elwekweswet+elwekwes+epwekweswetwey;maxgra
  de(p2);
```

$$p1 := e1we4we3weiwej - 2\ elwekweswet$$

$$5$$

$$p2 := e1we4we3weiwej - 2\ elwekweswet + elwekwes + epwekweswetwey$$

$$5$$

```
> out1:=wedge(p1,p2);
```

$$out1 := -4\ e1we3we4weiwejwekwelweswet + e1we3we4weiwejwekwelwes$$

Observe, that 'wedge' has cut out from its output result of wedge(e1we4we3weiwej,epwekweswetwey) = e1we4we3weiwejwepwekweswetwey (up to a permutation sign) since the latter is a monomial of grade 10 which does not exist in the Grassmann algebra /\ V when dim(V) = 9. Observe that one of the terms in the last output is of grade 8 and the other is of grade 9. If we decrease the value of dim_V by either making an assignment or by defining a bilinear form 'B' of size smaller than 9, more terms will be cut out from the output:

```
> dim_V:=8;
```

$$dim\_V := 8$$

```
> out2:=wedge(p1,p2);
```

$$out2 := e1we3we4weiwejwekwelwes$$

Observe that output 'out2' no longer contains term '-4*e1we3we4weiwejwekwelweswet' since it is of grade 9 which does not exist in /\V when dim(V) = 8. It only contains term of grade 8. No warning message is printed since the change of 'dim_V' was done by the user. If the user assigns a bilinear form B of smaller size now than 8, result will be 0:

```
> B:=matrix(3,3,[]):
> wedge(p1,p2);
Warning, since B has been (re-)assigned, value of dim_V has been reduced by 'we
dge' to 3
Error, (in Clifford:-wedge) argument(s) contain(s) index larger then current
value of dim_V which is now 3. To complete computation, increase value of dim_V
or assign square matrix of size at least 4 by 4 to bilinear form B
```

The above error message is produced by 'wedge' and is self-explanatory. The warning message is not printed at the second attempt to compute wedge(p1,p2) since the value of dim_V has not been changed. Let's assign new values to p1 and p2:

```
> p1:=2+e1we2+e3we2-e1we3we2;p2:=2+e3+e1we3we2-e3we2;
```

$$p1 := 2 + e1we2 + e3we2 - e1we3we2$$

$$p2 := 2 + e3 + e1we3we2 - e3we2$$

```
> out3:=wedge(p1,p2);
```

$$out3 := 4\ Id + 2\ e1we2 + 2\ e3 + e1we2we3$$

```
> dim_V;
```

$$3$$

Observe, that since there is no index in p1 or p2 that would be larger than 3, computation can proceed and all terms in the output out3 are listed since none of them is of grade higher than 3. If we now decrease the size of 'dim_V' to 2 by re-assigning B, a warning message will be printed and the term of grade 3 will be dropped from out3. The warning message is printed because the default value of '_warnings_flag' is true. To suppress printing of this warning message, change the value of _warnings_flag to 'false':

```
> _warnings_flag;
```

$$true$$

```
> p1:=2+eiwejwek-e1we2;p2:=elwem-e1we2+3;
```

$$p1 := 2 + eiwejwek - e1we2$$

$$p2 := elwem - e1we2 + 3$$

```
> B:=matrix(2,2,[]):
> out4:=wedge(p1,p2);
Warning, since B has been (re-)assigned, value of dim_V has been reduced by 'we
dge' to 2
```

$$out4 := 2\ elwem - 5\ e1we2 + 6\ Id$$

```
> dim_V;
```

$$2$$

Observe that the value of dim_V has been reduced by 'wedge' from 3 to 2. Observe also that if we increase the size of 'B' to 3, additional terms in out4 above will appear:

```
> B:=matrix(3,3,[]):
> out5:=wedge(p1,p2);
```

$$out5 := 2\ elwem - 5\ e1we2 + 6\ Id + 3\ eiwejwek$$

In order to see all terms in wedge(p1,p2), make sure that B is either unassigned and value of 9 is assigned to dim_V, or the size of B is increased to 9.

```
> B:='B':dim_V;
```

$$3$$

```
> dim_V:=9:
> out6:=wedge(p1,p2);
```

$$out6 := 2\ elwem + eiwejwekwelwem - e1we2welwem - 5\ e1we2 - e1we2weiwejwek + 6\ Id + 3\ eiwejwek$$

```
> dim_V;
```

$$9$$

```
>
```

Example 2: Selecting Clifford product and value of the global variable _default_Clifford_product.

Upon loading CLIFFORD, the value of _default_Clifford_product is set to 'cmulRS':

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

$$Clifford{:-}cmulRS$$

Internally, procedure Clifford:-cmul calls on procedure Clifford:-cmulgen which in turn uses Clifford:-cmulRS on any two Clifford multivectors (with a required third parameter of type name, symbol, matrix or array). I order to see that indeed 'cmulRS' is used internally, let's cause an error message to appear as follows:

```
> cmulgen(e1,e2);
Error, (in Clifford:-cmulRS) exactly three arguments are needed
```

The error points to the lack of the third required argument in 'cmulRS'. The error disappears when a third argument is used:

```
> cmulgen(e1,e2,H);
```

$$e1we2 + H_{1,2}\ Id$$

The value of _default_Clifford_product can be changed to one of the other three possible values either by direct assignment or with the help of procedure useproduct:

```
> useproduct(cmulNUM); # changing to Chevalley's recursive
  definition
Warning, cmul will use cmulNUM; for help see pages ?cmul, ?Clifford:-intro, or
?cmulNUM
```

Like above, let's check that indeed procedure Clifford:-cmulNUM is now used internally by causing another error to appear:

```
> _default_Clifford_product;
```

$$Clifford{:-}cmulNUM$$

```
> cmulgen(e1,e2);
Error, (in Clifford:-cmulNUM) exactly three arguments are needed
```

Again, the error this time points to the lack of the third required argument in 'cmulNUM'. The error disappears when a third argument is used:

```
> cmulgen(e1,e2,K);
```

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

We can also change the value of _default_Clifford_product to a generic dummy name cmulgen and a appropriate warning is again printed using procedure useproduct:

```
> useproduct(cmulgen); # changing to a generic name which is also
  a dummy procedure in CLIFFORD
Warning, cmul will use cmulgen; for help see pages ?cmul, ?Clifford:-intro, or
?cmulgen
```

Then, we get:

```
> cmulgen(e1,e2);
Warning, to assign Clifford product, execute 'useproduct' with argument cmulRS,
cmulNUM, or cmul_user_defined first
```

$$Clifford{:}\text{-}cmulgen(e1, e2)$$

```
> useproduct(cmulNUM); # changing to cmulNUM
  cmul(e1,e2);
Warning, cmul will use cmulNUM; for help see pages ?cmul, ?Clifford:-intro, or
?cmulNUM

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

$$e1we2 + B_{1,2}\, Id$$

Printing of warnings can be stopped by assigning false to _warnings_flag:

```
> _warnings_flag:=false:
  cmul(e1,e2);
```

$$B_{1,2}\, Id + e1we2$$

The name of the bilinear form can be passed on this way:

```
> cmul[K](e1,e2);
```

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

However, when _default_Clifford_product is set to generic 'cmulgen' or to some undefined yet name 'cmul_user_defined', obviously 'cmul' is not fully functional:

```
> cmul(e1,e2,e3);
```

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

```
> useproduct(cmul_user_defined);
Warning, no computations with cmul can be peformed yet since cmul_user_defined
has not been defined as procedure. Select cmulRS, cmulNUM, or a new procedure
as argument to useproduct.
```

```
> cmul(e1,e2,e3);
Error, (in Clifford:-cmul) global variable _default_Clifford_product must be
```

```
assigned a procedure so that 'cmul' could proceed beyond this point. Sorry. For
help see ?cmul.
```

Changing back to 'cmulRS' or 'cmulNUM' yields correct answer and 'cmul' becomes fully functional:

```
> useproduct(cmulRS):
Warning, cmul will use cmulRS; for help see pages ?cmul, ?Clifford:-intro, or
?cmulRS
```

```
> cmul(e1,e2,e3);
```

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

```
> cmul[K](e1-2,2*e2+e4,e3,e5);
```

$$(-2\, K_{1,3}\, K_{2,5} + 2\, K_{2,3}\, K_{1,5} + K_{1,4}\, K_{3,5} - K_{1,3}\, K_{4,5} + K_{4,3}\, K_{1,5} + 2\, K_{1,2}\, K_{3,5})\, Id$$
$$- (2\, K_{2,5} + K_{4,5})\, e1we3 + 2\, K_{1,5}\, e2we3 + 2\, K_{3,5}\, e1we2 - 4\, K_{3,5}\, e2 + 2\, (2\, K_{2,5} + K_{4,5})\, e3$$
$$- 2\, K_{3,5}\, e4 - 2\, (2\, K_{2,3} + K_{4,3})\, e5 + K_{3,5}\, e1we4 - K_{1,5}\, e3we4 - 4\, e2we3we5$$
$$+ (2\, K_{2,3} + K_{4,3})\, e1we5 - 2\, K_{1,3}\, e2we5 + (2\, K_{1,2} + K_{1,4})\, e3we5 + 2\, e1we2we3we5$$
$$- e1we3we4we5 - K_{1,3}\, e4we5 + 2\, e3we4we5$$

```
>
```

Finally, a user can define his/her own Clifford multiplication procedure 'cmul_user_defined' which can be used internally by Clifford:-cmul] internally. That new procedure is expected to have three arguments: x, y, and a name where x and y are arbitrary elements in Cl(B) and a name is of type 'name', 'symbol', 'array', or 'matrix'. The output of this procedure must be an element in Cl(B) of type 'cliscalar', 'clibasmon', 'climon', or 'clipolynom'. User will be expected then to write his/her help page for that new function and add it to the library of all CLIFFORD help pages. For example, let's define a new Clifford "multiplication" which just returns the wedge product of the two elements x and y:

```
> cmul_user_defined:=proc(x,y,lname) return wedge(x,y) end proc:
```

Let's define two elements in Cl(B):

```
> p1:=e1-2*e4we5;
  p2:=e2-e9+Pi*e3;
```

$$p1 := e1 - 2\, e4we5$$
$$p2 := e2 - e9 + \pi\, e3$$

```
> cmul_user_defined(p1,p2);
```

$$e1we2 - 2\, e2we4we5 - e1we9 + 2\, e4we5we9 + \pi\, e1we3 - 2\, \pi\, e3we4we5$$

```
> useproduct(cmul_user_defined);
Warning, cmul will use cmul_user_defined; for help see pages ?cmul,
?Clifford:-intro, or ?cmul_user_defined
```

```
> cmul(p1,p2);
```

$$\pi\, e1we3 + e1we2 - 2\, \pi\, e3we4we5 - 2\, e2we4we5 + 2\, e4we5we9 - e1we9$$

Another example, in which the bilinear form entered as the third parameter will be used, could be the following

```
> cmul_user_defined:=proc(x,y,lname) return
  LC(vectorpart(x,1),vectorpart(y,1),lname) end proc:
> cmul_user_defined(p1,p2,K);
```

$$K_{1,2}\,Id - K_{1,9}\,Id + \pi\,K_{1,3}\,Id$$

```
> cmul(p1,p2);
```

$$(B_{1,2} - B_{1,9} + \pi\,B_{1,3})\,Id$$

```
> cmul[P](e1,e2);
```

$$P_{1,2}\,Id$$

```
>
```

Example 3: Setting value of _prolevel.

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

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


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


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



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

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

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

```
> cliparse(e1we); #Intentional error
Error, (in Clifford:-cliparse) check spelling of e1we or define it as a
constant or an alias
```

```
> _prolevel:=true:
> cliparse(e1we);
```

$$true$$

```
>
```

Example 4: Environmental variables in other packages:

```
> restart:with(Clifford):with(Cliplus):with(Bigebra):
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
> CLIFFORD_ENV();
```

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



`>>> Global variables defined in Cliplus:-setup are now available and have thes
e values:  <<<`
`************* Start *************`
macro(Clifford:-cmul = climul)
```

```
macro(Clifford:-cmulQ = climul)
macro(`&c` = climul)
macro(`&cQ` = climul)
macro(Clifford:-reversion = clirev)
macro(Clifford:-LC = LCbig)
macro(Clifford:-RC = RCbig)
`Warning, new definitions for type/climon and type/clipolynom now include &C`
`************* End *************`



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



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

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

```
>
> with(Octonion);
```

[Φ, *associator*, *commutator*, *def_omultable*, *o_conjug*, *oinv*, *omul*, *omultable*, *onorm*, *oversion*, *purevectorpart*, *realpart*]

```
> CLIFFORD_ENV();
```

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

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



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



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


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

```
>
> with(GTP);
```

$$[cmulB, gbasis, gcollect, gprod, grade, gradedprod, tensorrank]$$

```
> CLIFFORD_ENV();
```

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

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



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



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

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

[>

See Also: Clifford:-maxgrade, Clifford:-`type/clibasmon`, Clifford:-reorder, Clifford:-wedge, Clifford:-makeclibasmon

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-clilinear - make a procedure linear with respect to Clifford scalars

Calling Sequence:

clilinear(a1, K);
clilinear(a1, K, Ps);

Parameters:

a1 - expression of the type 'cliscalar', 'clibasmon', 'climon' or 'clipolynom'
K - expression of the type 'name' or 'procedure'
Ps - (optional) sequence of parameters that can be passed on to K

Description:

- Procedure 'clilinear' makes any previously defined procedure K linear with respect to Clifford scalars of the type 'cliscalar' in the first argument which is expected to be of the type 'cliscalar' or 'clipolynom'. See `type/cliscalar` and `type/clipolynom` for more help.
- The name of the procedure to be made linear is entered as the second argument.
- This procedure also distributes operators with respect to elements of `type/cliprod` provided additional package 'Cliplus' is loaded.
- Any additional arguments are passed on to the procedure K.

Examples:

```
> restart:with(Clifford):
> clilinear(3*Pi+2*e2we3-e4we3,K);
  clilinear(3*Pi+2*e2we3-e4we3,K,P1);
  clilinear(3*Pi+2*e2we3-e4we3,K,P1,P2);
```

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

$$3\,\pi\,\mathrm{K}(Id) + 2\,\mathrm{K}(e2we3) - \mathrm{K}(e4we3)$$

$$3\,\pi\,\mathrm{K}(Id, P1) + 2\,\mathrm{K}(e2we3, P1) - \mathrm{K}(e4we3, P1)$$

$$3\,\pi\,\mathrm{K}(Id, P1, P2) + 2\,\mathrm{K}(e2we3, P1, P2) - \mathrm{K}(e4we3, P1, P2)$$

```
> clilinear(1+3*e1+a*e1we2+2*e3-alpha+Pi*`&C`(e1,e3,e4),H);
```

$$\mathrm{H}(Id) + 3\,\mathrm{H}(e1) + a\,\mathrm{H}(e1we2) + 2\,\mathrm{H}(e3) - \alpha\,\mathrm{H}(Id) + \pi\,\mathrm{H}(\&\mathrm{C}(e1, e3, e4))$$

```
> clilinear((2*e1+e3)/sqrt(a^2+1),K);
  clilinear((2*e1+e3)/sqrt(a^2+1),K,P1);
  clilinear((2*e1+e3)/sqrt(a^2+1),K,P1,P2,P3);
```

$$\frac{2\,\mathrm{K}(e1)}{\sqrt{a^2+1}} + \frac{\mathrm{K}(e3)}{\sqrt{a^2+1}}$$

$$\frac{2\,\mathrm{K}(e1, P1)}{\sqrt{a^2+1}} + \frac{\mathrm{K}(e3, P1)}{\sqrt{a^2+1}}$$

$$\frac{2\,\mathrm{K}(e1, P1, P2, P3)}{\sqrt{a^2+1}} + \frac{\mathrm{K}(e3, P1, P2, P3)}{\sqrt{a^2+1}}$$

[>

See Also: Clifford:-clibilinear, Clifford:-`type/clipolynom`, Clifford:-`type/cliscalar`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-`type/climatrix` - define a new type: a Clifford matrix

Calling Sequence:

type(m, climatrix);

Parameters:

m - an algebraic expression of type 'matrix'

Description:

- A matrix M is of type 'climatrix' if it contains at least one entry of the type 'clipolynom' (see `type/clipolynom` for more help on this type).
- Note that anything in Maple that has been defined via the procedure linalg[matrix] is of the standard Maple type 'matrix' including matrices with entries in a Clifford algebra. Since a matrix with numerical entries is not of the type 'climatrix', this procedure allows one to distinguish such matrix from those that do have at least one entry in a Clifford algebra.
- Matrices of the type 'matrix' but not of the 'climatrix' may be multiplied using standard Maple matrix multiplication operator `&*`.
- Matrices of the type 'climatrix' must be multiplied using the procedure 'rmulm' (see rmulm for more help).
- See also related types 'diagmatrix' (`type/diagmatrix`), 'symmatrix' (`type/symmatrix`), and 'antisymmatrix' (`type/antisymmatrix`).

Examples:

```
> restart:with(Clifford):
  M:=linalg[matrix](2,2,[e1,e3we4+e3,e4,2-e2]);
```

$$M := \begin{bmatrix} e1 & e3we4 + e3 \\ e4 & 2 - e2 \end{bmatrix}$$

```
> type(M,climatrix);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$true$$

```
> type(M,matrix);
```

$$true$$

```
> M:=linalg[matrix](2,2,[1,a+b,c,2-f]);
```

$$M := \begin{bmatrix} 1 & a + b \\ c & 2 - f \end{bmatrix}$$

```
> type(M,climatrix);
```

$$false$$

```
> type(M,matrix);
```

$$true$$

```
> M:=linalg[matrix](2,2,[1,2,3,4]);
```

$$M := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
> type(M,climatrix);
```

$$false$$

```
> type(M,matrix);
```

$$true$$

```
>
```

See Also: Clifford:-`type/diagmatrix`, Clifford:-`type/antisymmatrix`, Clifford:-`type/symmatrix`, Clifford:-rmulm

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-climinpoly - compute minimal polynomial of any Clifford polynomial

Calling Sequence:

climinpoly(p);climinpoly[K](p);
climinpoly(p,'powers');climinpoly[K](p,'powers');
climinpoly(p,'horner');climinpoly[K](p,'horner');
climinpoly(p, 'powers','horner');climinpoly[K](p, 'powers','horner');
climinpoly(p,'horner','powers');climinpoly[K](p,'horner','powers');

Parameters:

p - an expression of type 'clipolynom'
'powers', 'horner' - (optional) symbols
K - (optional) index of type name, symbol, array, matrix, or
`&*`(numeric,{name,symbol,array,matrix})

Description:

- Procedure 'climinpoly' computes the minimal polynomial m(x) of any element p of type 'clipolynom' in Cl(B). It returns polynomial m(x) in one indeterminate 'x'.
- It can take optional index K in which case computations are performed in Cl(K).
- By definition, the minimal polynomial of p in Cl(B) is the monic polynomial of lowest degree satisfied by p.
- When optional parameter 'horner' is used, polynomial m(x) is returned in horner form. For more information, check `convert/horner`.
- .When optional parameter 'powers' is used, the procedure returns a list with two elements:
  (1) the first element in the list is the minimal polynomial m(x) of some degree d (returned in the horner form if option 'horner' is also used);
  (2) the second element in the list is a list containing consecutive, linearly independent powers of p, i.e., [Id, p, p^2, ...,p^(d-1)] where p^k = cmul(p,$k), for any k between 1 and d-1.
- Procedure subs_clipolynom allows one to substitute into any polynomial t(x) the actual Clifford polynomial p and expand it. If t(x) is the minimal polynomial of p, this procedure can be used to check that p satisfies t(x).
- This procedure is used by the procedure 'sexp' to exponentiate p modulo m(x) (see sexp for more information).
- Warning message about value of dim_V being reduced is produced by the procedure wedge. See also CLIFFORD_ENV for more help.

Examples:

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

Example 1: Computations in Cl(B) where B is unspecified.

```
> p:=2*t-2*e1we2+4*e1;
```

$$p := 2\,t - 2\,e1we2 + 4\,e1$$

The minimal polynomial of p is computed next. It has t and entries of B as parameters. If we a different name for the index, we get:

```
> pp:=climinpoly(p);climinpoly[K](p);climinpoly[-K](p);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$pp :=$$

$$x^2 + 4\,t^2 - 4\,t\,B_{2,1} + 4\,t\,B_{1,2} - 4\,B_{2,1}\,B_{1,2} + 4\,B_{2,2}\,B_{1,1} - (4\,t - 2\,B_{2,1} + 2\,B_{1,2})\,x - 16\,B_{1,1}$$

$$x^2 + 4\,t^2 - 4\,t\,K_{2,1} + 4\,t\,K_{1,2} - 4\,K_{2,1}\,K_{1,2} + 4\,K_{2,2}\,K_{1,1} - (4\,t - 2\,K_{2,1} + 2\,K_{1,2})\,x - 16\,K_{1,1}$$

$$x^2 + 4\,t^2 + 4\,t\,K_{2,1} - 4\,t\,K_{1,2} - 4\,K_{2,1}\,K_{1,2} + 4\,K_{2,2}\,K_{1,1} - (4\,t + 2\,K_{2,1} - 2\,K_{1,2})\,x + 16\,K_{1,1}$$

```
> pphorner:=climinpoly(p,'horner');
```

$$pphorner :=$$

$$-4\,B_{2,1}\,B_{1,2} - 16\,B_{1,1} + 4\,B_{2,2}\,B_{1,1} + (-4\,B_{2,1} + 4\,B_{1,2} + 4\,t)\,t + (-4\,t + 2\,B_{2,1} - 2\,B_{1,2} + x)\,x$$

The above is the minimal polynomial of p. It has t and entries of B as parameters.When used with the second optional argument 'powers', additional list of linearly independent powers of p is included in the output.

```
> climinpoly(p,powers);climinpoly[K](p,powers);
```

$$[x^2 + 4\,t^2 - 4\,t\,B_{2,1} + 4\,t\,B_{1,2} - 4\,B_{2,1}\,B_{1,2} + 4\,B_{2,2}\,B_{1,1} - (4\,t - 2\,B_{2,1} + 2\,B_{1,2})\,x - 16\,B_{1,1},$$
$$[Id, 2\,t\,Id - 2\,e1we2 + 4\,e1]]$$

$$[x^2 + 4\,t^2 - 4\,t\,K_{2,1} + 4\,t\,K_{1,2} - 4\,K_{2,1}\,K_{1,2} + 4\,K_{2,2}\,K_{1,1} - (4\,t - 2\,K_{2,1} + 2\,K_{1,2})\,x - 16\,K_{1,1},$$
$$[Id, 2\,t\,Id - 2\,e1we2 + 4\,e1]]$$

It is also possible to use both options 'powers' and 'horner', in any order:

```
> climinpoly(p,powers,horner);
```

$$[$$
$$-4\,B_{2,1}\,B_{1,2} - 16\,B_{1,1} + 4\,B_{2,2}\,B_{1,1} + (-4\,B_{2,1} + 4\,B_{1,2} + 4\,t)\,t + (-4\,t + 2\,B_{2,1} - 2\,B_{1,2} + x)\,x$$
$$, [Id, 2\,t\,Id - 2\,e1we2 + 4\,e1]]$$

```
> climinpoly(p,horner,powers);
```

$$[$$
$$-4\,B_{2,1}\,B_{1,2} - 16\,B_{1,1} + 4\,B_{2,2}\,B_{1,1} + (-4\,B_{2,1} + 4\,B_{1,2} + 4\,t)\,t + (-4\,t + 2\,B_{2,1} - 2\,B_{1,2} + x)\,x$$
$$, [Id, 2\,t\,Id - 2\,e1we2 + 4\,e1]]$$

We can check that pp is indeed satisfied by p:

```
> subs_clipolynom(p,pp);
```

$$0$$

or, in the longer form returning the minimal polynomial in Horner form with the indeterminate replaced by p:

```
> subs_clipolynom(p,pp,horner);
```

$$(-4\,B_{2,1}\,B_{1,2} - 16\,B_{1,1} + 4\,B_{2,2}\,B_{1,1} - 4\,t\,B_{2,1} + 4\,t\,B_{1,2} + 4\,t^2)\,Id + Clifford{:}\text{-}cmul(-2\,(t - B_{2,1} + B_{1,2})\,Id - 2\,e1we2 + 4\,e1,\ 2\,t\,Id - 2\,e1we2 + 4\,e1)$$

```
> expand(%);
```

$$0$$

Example 2. Computations in Cl(3,1). We will use cmulNUM instead of cmulRS.

```
> useproduct(cmulNUM);
  B:=linalg[diag](1,1,1,-1):
Warning, cmul will use cmulNUM; for help see pages ?cmul, ?Clifford:-intro, or
?cmulNUM
```

```
> p:=2+2*e4-e1we2+e3;
```

$$p := 2 + 2\,e4 - e1we2 + e3$$

```
> pp:=climinpoly(p);climinpoly(p,horner);climinpoly(p,powers,horn
  er);
```

$$pp := x^4 - 8\,x^3 + 32\,x^2 - 64\,x + 52$$

$$52 + (-64 + (32 + (-8 + x)\,x)\,x)\,x$$

$$[52 + (-64 + (32 + (-8 + x)\,x)\,x)\,x,\ [Id,\ 2\,Id + 2\,e4 - e1we2 + e3,\ 8\,e4 - 4\,e1we2 + 4\,e3 - 4\,e1we2we4 - 2\,e1we2we3,\ 12\,e4 + 6\,e3 - 24\,e1we2we4 - 12\,e1we2we3 - 16\,Id - 2\,e1we2]]$$

```
> subs_clipolynom(p,pp,horner);
```

$$52\,Id + Clifford{:}\text{-}cmul(-64\,Id + Clifford{:}\text{-}cmul(32\,Id + Clifford{:}\text{-}cmul(-6\,Id + 2\,e4 - e1we2 + e3,\ 2\,Id + 2\,e4 - e1we2 + e3),\ 2\,Id + 2\,e4 - e1we2 + e3),\ 2\,Id + 2\,e4 - e1we2 + e3)$$

```
> eval(%);
```

$$0$$

We use procedure 'sexp' to exponentiate p up to and including order n entered as a second argument.

```
> s:=time():p1:=sexp(p,5);time()-s;
```

$$p1 := -\frac{19\,Id}{10} + \frac{71\,e4}{15} + \frac{7\,e1we2}{10} + \frac{71\,e3}{30} - \frac{26\,e1we2we4}{3} - \frac{13\,e1we2we3}{3}$$

$$0.125$$

```
> s:=time():p2:=cexp(p,5);time()-s;
```

$$p2 := -\frac{19\,Id}{10} + \frac{71\,e4}{15} + \frac{7\,e1we2}{10} + \frac{71\,e3}{30} - \frac{26\,e1we2we4}{3} - \frac{13\,e1we2we3}{3}$$

$$0.281$$

Thus, the procedure 'sexp' was many times faster, but the results are the same:

```
> p1-p2;
```

$$0$$

Example 3: Computations in Cl(2,2).

```
> B:=linalg[diag](1,1,-1,-1):
  p:=e1we2+e1we3+e4: #define B and p
> pp:=climinpoly(p); #finding the minimal polynomial of p
```

$$pp := x^4 + 2\,x^2 + 1$$

```
> subs_clipolynom(p,pp,horner);
```

$$Id + Clifford{:}\text{-}cmul(Clifford{:}\text{-}cmul(2\,Id + Clifford{:}\text{-}cmul(e1we2 + e1we3 + e4, e1we2 + e1we3 + e4), e1we2 + e1we3 + e4), e1we2 + e1we3 + e4)$$

```
> eval(%);
```

$$0$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 1.518000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

See Also: Clifford:-subs_clipolynom, Clifford:-cexp, Clifford:-sexp, Clifford:-wexp, Clifford:-cexpQ, Clifford:-`type/clipolynom`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-`type/climon` - define a new type: a Clifford monomial

Calling Sequence:

type(p, climon);

Parameters:

p - an algebraic expression of type 'algebraic'

Description:

- A *Clifford monomial* is essentially any Grassmann basis monomial (of the type 'clibasmon') multiplied by a Clifford scalar (of the type 'cliscalar'). See `type/clibasmon` and `type/cliscalar` for more information.
- The procedure returns 'true' or 'false' depending whether the input p is or is not of the type 'clibasmon'.
- Permutations of the indices also yield elements if this type.
- Notice that expressions of the form a*`&C`(e1,e3) where a is of `type/cliscalar` are not Clifford monomials unless package 'Cliplus' is loaded. **That is, definitions of `type/climon` and type/clipolynom` are changed when 'Cliplus' is loaded.** When such expressions are found in the argument and 'Cliplus' has not been loaded, a warning message is printed. To find out more on CLIFFORD's environmental variables, check CLIFFORD_ENV.
- Expressions `&C`(e1,e2) and `&C`[K](e1,e2,e3) denote unevaluated Clifford product and are of `type/cliprod`.

Examples:

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

true

```
> type(2*e1we2,climon);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

true

```
> type(2*e2we1,climon);
```

true

```
> type(`&C`(e1,e2),cliprod);
```

true

```
> type(`&C`[K](e1,e2),cliprod);
```

true

```
> type(`&C`[-K](e1,e2),cliprod);
```

true

```
> type(2*`&C`(e1,e2),climon);
```

true

```
> _warnings_flag:=false;
```

$$\_warnings\_flag := false$$

```
> type(2*`&C`(e1,e2),climon);
```

$$true$$

```
> type(2*e1we2+3,climon);
```

$$false$$

```
> type(3*Id+eiwej,climon);
```

$$false$$

```
> type(3*cos(alpha)*Id,climon);
```

$$true$$

```
> type(3*cos(alpha),climon);
```

$$false$$

```
> type(1,climon);
```

$$false$$

```
> type(e1/sqrt(a^2+1),clibasmon);
```

$$false$$

```
> type(e1/sqrt(a^2+1),climon);
```

$$true$$

```
> type(e1/sqrt(a^2+1),clipolynom);
```

$$false$$

```
> type((e1+2*e2)/sqrt(a^2+1),clibasmon);
```

$$false$$

```
> type((e1+2*e2)/sqrt(a^2+1),climon);
```

$$false$$

```
> type((e1+2*e2)/sqrt(a^2+1),clipolynom);
```

$$true$$

```
> with(Cliplus):
> type(2*`&C`(e1,e2),climon);
  type(`&C`(e1,e2),climon);
```

$$true$$

$$false$$

```
>
```

See Also: Clifford:-`type/clipolynom`, Clifford:-`type/cliscalar`, Clifford:-`type/clibasmon`, Clifford:-`type/cliprod`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-cliparse - check user's input for syntax errors

Calling Sequence:

cliparse(p);

Parameters:

p - any expression of the type 'anything'

Description:

- Procedure 'cliparse' checks user's input for correct spelling provided the global variable _prolevel is set to true. The default value of _prolevel is set to 'false' by the initialization procedure Clifford:-setup.
- In particular, 'cliparse' it checks the spelling of the basis monomials. It returns 'true' if the input is correct and 'false' if the input is incorrect.
- User can assign 'true' to _prolevel in any worksheet. When _prolevel=true, procedure 'cliparse' always returns 'true' no matter what input.
- The purpose of this feature is to speed up computations: when checking for spelling errors is not done, time needed for computations is reduced significantly (see below).
- The procedure can recognize, in most cases, that strings like x1, x2, a1, a2, etc., are intended to be Maple constants. Thus, these strings no longer need to be defined as constants.
- When unable to decide if the given input is correct, the procedure returns a message asking the user to check the spelling of the first found suspect word, or to define it as a constant, or to alias it. The suspect word is then returned as part of the message.
- The procedure is intended to catch errors like 'e1e2' instead of 'e1we2', or 'e1w2' instead of 'e1we2' in the user's input. If, for example, the basis monomial 'e1we2' has been aliased as 'e12' via the procedure 'makealiases', the procedure returns 'true' for 'e12'. See makealiases for more help.
- The procedure can also detect invalid expressions like 'e6we6', i.e., where indices have been repeated.
- This procedure may also be used to check matrix entries as shown below.
- See also procedure Cliplus:-makeclialiases that creates aliases for unevaluated Clifford products. It becomes available once the additional package 'Cliplus' gets loaded.

Examples:

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

$$false$$

```
> cliparse(e1we2);
```

$$true$$

```
> cliparse(1+e1we2+2*e3we2);
```

$$true$$

```
> cliparse(e1e2+a*e2we3); ###<<<- Intended error message
Error, (in Clifford:-cliparse) check spelling of e1e2 or define it as a
constant or an alias
```

In the above, 'cliparse' has checked the spelling of 'e1e2' and has determined, correctly, that it is most likely an error. It was expecting to see 'e1we2' instead of 'e1e2'. When the value of _prolevel is set to false by the user, 'cliparse' always returns 'true' no matter what input:

```
> _prolevel:=true:
> cliparse(e1e2+a*e2we3);
```

$$true$$

Thus, it is possible to speed up computations in CLIFFORD by not doing expensive parsing of the user's input provided user inputs expressions that make sense. For example, the following results are returned without 'cliparsing':

```
> cmul(e1e2,e1e2); # <<<--- strange result that is wrong
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$e1e2^2\ Id$$

```
> cliparse(a);
```

$$true$$

```
> _prolevel:=false:
> cliparse(x12*e1we2+x23*e2we3+x13*e1we3);
```

$$true$$

```
> makealiases(3):eval(%);
```

$$e12, e21, e13, e31, e23, e32, e123, e132, e213, e231, e312, e321$$

```
> cliparse(2*e12+a*e32+c*e13);
```

$$true$$

```
> M:=linalg[matrix](2,2,[3*e2we3-Id,2+e1,-e4,e5+e6we6]);
```

$$M := \begin{bmatrix} 3\ e23 - Id & 2 + e1 \\ -e4 & e5 + e6we6 \end{bmatrix}$$

```
> map(cliparse,M);  ###<<<- Intended error message
Error, (in Clifford:-cliparse) check spelling of e6we6 or define it as a
constant or an alias
```

```
> M:=linalg[matrix](2,2,[3*e2e3-Id,2+e1,-e4,e5+e6we6]);
```

$$M := \begin{bmatrix} 3\ e2e3 - Id & 2 + e1 \\ -e4 & e5 + e6we6 \end{bmatrix}$$

```
> map(cliparse,M); ###<<<- Intended error message
Error, (in Clifford:-cliparse) check spelling of e2e3 or define it as a
constant or an alias
```

```
> M:=linalg[matrix](2,2,[3*e2we3-Id,2+e1,-e4,e5+e6we7]);
```

$$M := \begin{bmatrix} 3\ e23 - Id & 2 + e1 \\ -e4 & e5 + e6we7 \end{bmatrix}$$

```
> map(cliparse,M);
```

$$\begin{bmatrix} true & true \\ true & true \end{bmatrix}$$

```
> cliparse(e1we23); ###<<<- Intended error message
Error, (in Clifford:-cliparse) check spelling of e1we23 or define it as a
constant or an alias
```

```
> cliparse(e6we7);
```

$$true$$

```
>
```

See Also: Clifford:-`type/clipolynom`, Clifford:-`type/clibasmon`, Clifford:-makealiases

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-`type/clipolynom` - define a new type: a Clifford polynomial

Calling Sequence:

type(p, clipolynom);

Parameters:

p - an algebraic expression of type 'anything'

Description:

- A *Clifford polynomial* is essentially any multivariate polynomial in the basis monomials specified by the procedure 'cbasis' (permutation of the indices allowed) (see cbasis). Thus, it is a sum of elements of the type 'climon' (see `type/climon`).
- Elements of this type are the most general elements in any Clifford algebra.
- Notice that expressions of the form 2*`&C`(e1,e3) + `&C`(e1,e3,e4) are not Clifford polynomials unless package 'Cliplus' is loaded. **That is, definitions of `type/climon` and type/clipolynom` are changed when 'Cliplus' is loaded.** When expression of type 'cliprod' are found among the terms of p and package 'Cliplus' has not been loaded, a warning message is printed. Printing of these warning messages can be suppressed by setting CLIFFORD's environmental variable _warnings_flag to false. To find out more about such variables, try CLIFFORD_ENV.
- Expressions `&C`(e1,e2) and `&C`[K](e1,e2,e3) denote unevaluated Clifford product and are of `type/cliprod`.

Examples:

```
> restart:with(Clifford):_prolevel:=false:_warnings_flag;
```

$$true$$

```
> p:=2*`&C`(e1,e3)+3*`&C`(e2,e3,e4);
```

$$p := 2\,(e1\ \&C\ e3) + 3\ \&C(e2, e3, e4)$$

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

$$true$$

```
> _warnings_flag:=false;
```

$$\_warnings\_flag := false$$

```
> type(p,clipolynom);
```

$$true$$

```
> _warnings_flag:=true;
```

$$\_warnings\_flag := true$$

```
> with(Cliplus);
```

$$[LCbig, RCbig, clibasis, clieval, cliexpand, climul, clirev, dottedcbasis, dwedge, makeclialiases]$$

```
> type(p,clipolynom);
```

true

```
> type(e1we2,clipolynom);
```

false

```
> type(2*e2we1+3+Pi*B[2,3]*Id,clipolynom);
```

true

```
> type(2+Id,clipolynom);
```

true

```
> type(3*Id,clipolynom);
```

false

```
> type(e3we2,clipolynom);
```

false

```
> type(3*cos(alpha),clipolynom);
```

false

```
> type(3*cos(alpha),clipolynom);
```

false

```
> type(e1/sqrt(a^2+1),clibasmon);
```

false

```
> type(e1/sqrt(a^2+1),climon);
```

true

```
> type(e1/sqrt(a^2+1),clipolynom);
```

false

```
> type((e1-2*e1we2)/sqrt(a^2+1),clibasmon);
```

false

```
> type((e1-2*e1we2)/sqrt(a^2+1),climon);
```

false

```
> type((e1-2*e1we2)/sqrt(a^2+1),clipolynom);
```

true

```
> type(2*`&C`(e1,e3)+3*`&C`(e2,e3,e4),clipolynom);
```

true

```
> type(2*`&C`[K](e1,e3)+3*`&C`[K](e2,e3,e4),clipolynom);
```

true

```
>
```

See Also: Clifford:-setup, Clifford:-cliparse, Clifford:-`type/climon`, Clifford:-`type/cliscalar`, Clifford:-`type/clibasmon`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-`type/cliprod` - define a new type: an inert (unevaluated) Clifford product in Cl(B) or Cl(K)

Calling Sequence:

type(f, cliprod);

Parameters:

f - function or anything

Description:

- The procedure checks if a function is of type 'cliprod'.
- The procedure returns 'true' or 'false' depending whether its argument is or is not of the type 'cliprod', that is, whether it is an inert function `&C` or `&C`[K], which represents inert form of Clifford product in Cl(B) or Cl(K), respectively.
- The active form of this Clifford product Clifford:-cmul and Clifford:-`&c`.
- This type is needed in the supplementary packages GTP and Cliplus.

Examples:

```
> restart:with(Clifford):
> e1 &C e2,type(e1 &C e2,cliprod);
  `&C`[B](e1,e2),type(`&C`[B](e1,e2),cliprod);
  `&C`[K](e1,e2),type(`&C`[K](e1,e2),cliprod);
  `&C`[-B](e1,e2),type(`&C`[-B](e1,e2),cliprod);
  `&C`[-2*K](e1,e2),type(`&C`[-2*K](e1,e2),cliprod);
```

$$e1\ \&C\ e2,\ true$$

$$\&C_B(e1, e2),\ true$$

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

$$\&C_{-B}(e1, e2),\ true$$

$$\&C_{-2\,K}(e1, e2),\ true$$

```
> p:=1+2*(e1we2 &C e4)-`&C`(e1,e2,e3);
  p:=1+2*`&C`[B](e1we2,e4)-`&C`[B](e1,e2,e3);
```

$$p := 1 + 2\,(e1we2\ \&C\ e4) - \&C(e1, e2, e3)$$

$$p := 1 + 2\,\&C_B(e1we2, e4) - \&C_B(e1, e2, e3)$$

```
> type(p,cliprod);type(p,clipolynom);
```

$$false$$

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

$$true$$

```
> type(2*(e1 &C e2),cliprod);
```

false

```
> 
```

See Also: GTP:-setup, Cliplus

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-cliremove - remove certain vector subscript in a basis monomial

Calling Sequence:

cliremove(n,s);

Parameters:

n - a positive integer
s - a string or symbol

Description:

- Procedure 'cliremove' removes one vector index from a location specified by the procedure 'ord'.
- Recall that procedure 'ord' specifies location of the vector indices in a Clifford monomial or basis monomial.
- If a Clifford basis monomial is of grade one, i.e., it is a 1-vector, the procedure returns NULL statement.
- This procedure is primarily for internal use.

Examples:

```
> restart:with(Clifford):
> ord(e1we2we3);
```

$$[2, 5, 8]$$

```
> cliremove(2,e1we2we3);
```

$$e2we3$$

```
> cliremove(5,e1we2we3);
```

$$e1we3$$

```
> cliremove(2,e1);
>
```

See Also: Clifford:-reorder, Clifford:-clicollect, Clifford:-`type/climon`, Clifford:-`type/cliscalar`, Clifford:-`type/clibasmon`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-`type/cliscalar` - define a new type: a Clifford scalar

Calling Sequence:

type(p, cliscalar);

Parameters:

p - an algebraic expression of type 'anything'

Description:

- A *Clifford scalar* is any number, function, constant, or an algebraic expression not containing any of the basis monomials or expressions of the type 'clibasmon' (see `type/clibasmon` for more help on this type).
- The types of all expressions that, by default, are considered to be of type 'cliscalar' are collected in an environmental variable _scalartypes which can be displayed using the procedure CLIFFORD_ENV. User can add or delete certain types from that set.
- This means that, for example, 2*Id is not of the type 'cliscalar' while 2*(a+b) is.
- Clifford scalars are essentially all expressions such that, for example, Clifford multiplication procedures 'cmul' and 'cmulQ' may treat as constant factors (see cmul and cmulQ for help on Clifford multiplication).

Examples:

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

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


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


`>>> Global variables defined in Cliplus:-setup are now available and have thes
e values:  <<<`
```

```
`************* Start *************`
macro(Clifford:-cmul = climul)
macro(Clifford:-cmulQ = climul)
macro(`&c` = climul)
macro(`&cQ` = climul)
macro(Clifford:-reversion = clirev)
macro(Clifford:-LC = LCbig)
macro(Clifford:-RC = RCbig)
`Warning, new definitions for type/climon and type/clipolynom now include &C`
`************* End *************`



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



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


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

```
> type(e1we2,cliscalar);
```

$$false$$

```
> type(2*(a+b),cliscalar);
```

$$true$$

```
> type(2*e1+e2we3+2*Pi*B[1,2],cliscalar);
```

$$false$$

```
> type(a*B[1,2],cliscalar);
```

$$true$$

```
> type(a*B[1,2]-B[3,4],cliscalar);
```

$$true$$

```
> type(cos(alpha)*B[1,2],cliscalar);
```

$$true$$

```
> type(cos(alpha)+e1,cliscalar);
```

$$false$$

```
> p:=x^2-4*x^7;
```

$$p := x^2 - 4\,x^7$$

```
> type(p,cliscalar);
```

true

```
> 
```

See Also: Clifford:-cmulQ, Clifford:-cmul

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-clisolve - procedure that solves symbolic equations in Clifford algebra Cl(V,B)

Calling Sequence:

clisolve(eq, pp);
clisolve(eq, S);

Parameters:

eq - left hand side of an equation eq = 0 that is to be solved and viewed as, in the most general case, as a Clifford polynomial of `type/clipolynom`
pp - an element in Cl(V,B), usually the most general element, whose unknown coefficients are of `type/indexed`, and which is to satisfy equation eq = 0
S - list of the unknown coefficients in 'eq' that need to be solved for

Description:

- Procedure 'clisolve' solves equations in Clifford algebra Cl(V,B). Its ability to solve equations is only limited by the ability of Maple' solve command to solve systems of equations.
- This procedure takes two arguments: the first argument is the left hand side of an equation that is to be solved. It is expected to be of `type/clipolynom`, or `type/climon`. The second argument is either a list of the unknowns to be solved for that appear in 'eq', or, an unknown Clifford number pp expressed in terms of the Grassmann basis in Cl(V,B) with some general unknown coefficients which are of `type/index`. Then, the procedure solve for these coefficients.
- The basis elements can be aliased, e.g., 'e1we2' can be aliased as 'e12' with the procedure makealiases.

Examples:

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

Example 1: Let's find the most general idempotent element in the Clifford algebra Cl(2,0) of the Euclidean plane.

```
> B:=diag(1,1);
```

$$B := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

```
> cbas:=cbasis(coldim(B)); #Grassmann basis in Cl(2,0)
```

$$cbas := [Id, e1, e2, e1we2]$$

Let 'pp' be the most general element in Cl(2,0) expressed in terms of the Grassmann basis:

```
> pp:=add(x[i]*cbas[i],i=1..nops(cbas));
```

$$pp := x_1\, Id + x_2\, e1 + x_3\, e2 + x_4\, e1we2$$

A list of the unknown coefficients in 'pp' is then:

```
> ind:=[seq(x[i],i=1..nops(cbas))];
```

$$ind := [x_1, x_2, x_3, x_4]$$

The idempotent equation will then be: cmul(pp,pp) - Id whose left hand side we assign to 'eq':

```
> eq:=cmul(pp,pp)-pp;
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$eq := 2\,x_2\,x_1\,e1 + 2\,x_3\,x_1\,e2 + (x_1^{\,2} + x_2^{\,2} + x_3^{\,2} - x_4^{\,2})\,Id + 2\,x_4\,x_1\,e1we2 - x_1\,Id - x_2\,e1 - x_3\,e2 - x_4\,e1we2$$

Let's solve now equation eq = 0 for [x[1], x[2], x[3], x[4]]:

```
> L:=clisolve(eq,ind);nops(L);
```

$$L := \Big[\{x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 0\}, \{x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 0\},$$
$$\{x_1 = \frac{1}{2}, x_2 = 0, x_3 = 0, x_4 = \frac{1}{2} I\}, \{x_1 = \frac{1}{2}, x_2 = 0, x_3 = 0, x_4 = \frac{-1}{2} I\},$$
$$\{x_3 = x_3, x_4 = \frac{1}{2}\sqrt{-1 + 4\,x_3^{\,2}}, x_1 = \frac{1}{2}, x_2 = 0\}, \{x_3 = x_3, x_4 = -\frac{1}{2}\sqrt{-1 + 4\,x_3^{\,2}}, x_1 = \frac{1}{2}, x_2 = 0\},$$
$$\{x_2 = x_2, x_4 = \frac{1}{2}\sqrt{-1 + 4\,x_2^{\,2}}, x_1 = \frac{1}{2}, x_3 = 0\}, \{x_2 = x_2, x_4 = -\frac{1}{2}\sqrt{-1 + 4\,x_2^{\,2}}, x_1 = \frac{1}{2}, x_3 = 0\},$$
$$\{x_2 = x_2, x_3 = \frac{1}{2}\sqrt{1 - 4\,x_2^{\,2}}, x_1 = \frac{1}{2}, x_4 = 0\}, \{x_2 = x_2, x_3 = -\frac{1}{2}\sqrt{1 - 4\,x_2^{\,2}}, x_1 = \frac{1}{2}, x_4 = 0\},$$
$$\{x_3 = x_3, x_2 = x_2, x_4 = \frac{1}{2}\sqrt{-1 + 4\,x_2^{\,2} + 4\,x_3^{\,2}}, x_1 = \frac{1}{2}\},$$
$$\{x_3 = x_3, x_2 = x_2, x_4 = -\frac{1}{2}\sqrt{-1 + 4\,x_2^{\,2} + 4\,x_3^{\,2}}, x_1 = \frac{1}{2}\}\Big]$$

$$12$$

Thus, the list L contains 12 different types of solutions.

To find out the possible solutions to cmul(pp,pp) = pp, we substitute five different solution sets collected in L into pp:

```
> for i from 1 to nops(L) do pp||i:=subs(L[i],pp) od;
```

$$pp1 := 0$$
$$pp2 := Id$$
$$pp3 := \frac{Id}{2} + \frac{1}{2} I\,e1we2$$
$$pp4 := \frac{Id}{2} - \frac{1}{2} I\,e1we2$$
$$pp5 := \frac{Id}{2} + x_3\,e2 + \frac{1}{2}\sqrt{-1 + 4\,x_3^{\,2}}\,e1we2$$

$$pp6 := \frac{Id}{2} + x_3\, e2 - \frac{1}{2}\sqrt{-1 + 4\, x_3^{\,2}}\; e1we2$$

$$pp7 := \frac{Id}{2} + x_2\, e1 + \frac{1}{2}\sqrt{-1 + 4\, x_2^{\,2}}\; e1we2$$

$$pp8 := \frac{Id}{2} + x_2\, e1 - \frac{1}{2}\sqrt{-1 + 4\, x_2^{\,2}}\; e1we2$$

$$pp9 := \frac{Id}{2} + x_2\, e1 + \frac{1}{2}\sqrt{1 - 4\, x_2^{\,2}}\; e2$$

$$pp10 := \frac{Id}{2} + x_2\, e1 - \frac{1}{2}\sqrt{1 - 4\, x_2^{\,2}}\; e2$$

$$pp11 := \frac{Id}{2} + x_2\, e1 + x_3\, e2 + \frac{1}{2}\sqrt{-1 + 4\, x_2^{\,2} + 4\, x_3^{\,2}}\; e1we2$$

$$pp12 := \frac{Id}{2} + x_2\, e1 + x_3\, e2 - \frac{1}{2}\sqrt{-1 + 4\, x_2^{\,2} + 4\, x_3^{\,2}}\; e1we2$$

Finally, let's verify that pp1, pp2, pp3, pp4, and pp5 satisfy this equation. It is obvious in case of pp1 and pp2, and maybe in the case of pp3, but it is not that obvious in the case of pp4 and pp5:

```
> seq(cmul(pp||i,pp||i)-pp||i,i=1..nops(L));
```

$$0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0$$

It is of course well-known that elements pp3 and pp4 listed above are primitive idempotents in Cl(2,0), that is, they are of `type/primitiveidemp`.

```
> type(pp3,primitiveidemp),type(pp4,primitiveidemp);
```

$$true, true$$

while, of course, the identity element 'Id' in Cl(2,0) is an idempotent but it is not primitive as it is a sum of two primitive idempotents pp3 and pp4, which are mutually annihilating:

```
> type(Id,primitiveidemp);
```

$$false$$

```
> pp3+pp4,cmul(pp3,pp4),cmul(pp4,pp3);
```

$$Id, 0, 0$$

```
>
```

Example 2: Let's attempt now to solve equation cmul(pp,e1+e2) = Id + e2, which we enter as follows:

```
> 'pp'=pp;
  eq:=cmul(pp,e1+e2)-Id-e2;
```

$$pp = x_1\, Id + x_2\, e1 + x_3\, e2 + x_4\, e1we2$$

$$eq := (x_1 + x_4)\, e1 + (x_1 - x_4)\, e2 + (x_2 + x_3)\, Id + (x_2 - x_3)\, e1we2 - Id - e2$$

This time we solve for the unknown 'pp' and the program picks up correctly the unknown indexed coefficients in pp:

```
> L:=clisolve(eq,pp);
```

$$L := \left[\frac{Id}{2} + \frac{e1}{2} + \frac{e2}{2} - \frac{e1we2}{2}\right]$$

Observe that there is only one solution:

```
> psol:=L[1];
```

$$psol := \frac{Id}{2} + \frac{e1}{2} + \frac{e2}{2} - \frac{e1we2}{2}$$

```
> cmul(psol,e1+e2)-Id-e2;
```

$$0$$

```
>
```

Example 3: We will attempt to solve equation: cmul(reversion(pp),pp) = Id.

```
> revpp:=reversion(pp);
```

$$revpp := x_2\, e1 + x_3\, e2 + x_1\, Id - x_4\, e1we2$$

```
> eq:=cmul(revpp,pp)-Id;
```

$$eq := 2\,(x_2\, x_1 - x_4\, x_3)\, e1 + 2\,(x_3\, x_1 + x_4\, x_2)\, e2 + (x_2^{\,2} + x_3^{\,2} + x_1^{\,2} + x_4^{\,2})\, Id - Id$$

```
> L:=clisolve(eq,pp);
```

$$L := [x_2\, e1 + \sqrt{-x_2^{\,2} + 1}\; e2, x_2\, e1 - \sqrt{-x_2^{\,2} + 1}\; e2, x_1\, Id + \sqrt{-x_1^{\,2} + 1}\; e1we2, x_1\, Id - \sqrt{-x_1^{\,2} + 1}\; e1we2, e2, -e2]$$

As above, we introduce aliases 'kappa1' and 'kappa2' to shorten Maple's output:

```
> alias(kappa1=RootOf(_Z^2+x[3]^2-1)):alias(kappa2=RootOf(_Z^2+x[
  4]^2-1)):
> L;
```

$$[x_2\, e1 + \sqrt{-x_2^{\,2} + 1}\; e2, x_2\, e1 - \sqrt{-x_2^{\,2} + 1}\; e2, x_1\, Id + \sqrt{-x_1^{\,2} + 1}\; e1we2, x_1\, Id - \sqrt{-x_1^{\,2} + 1}\; e1we2, e2, -e2]$$

```
> psol1:=L[1];psol2:=L[2];
```

$$psol1 := x_2\, e1 + \sqrt{-x_2^{\,2} + 1}\; e2$$

$$psol2 := x_2\, e1 - \sqrt{-x_2^{\,2} + 1}\; e2$$

Thus, elements psol1, and psol2 solve the given equation as we verify next:

```
> cmul(reversion(L[1]),L[1])-Id;
```

$$x_2^{\,2}\, Id + (-x_2^{\,2} + 1)\, Id - Id$$

```
> cmul(reversion(L[2]),L[2])-Id;
```

$$x_2^{\,2}\, Id + (-x_2^{\,2} + 1)\, Id - Id$$

```
>
```

Example 4: Procedure 'clisolve' could be used to find the symbolic inverse of the given Clifford polynomial or monomial, although there is a dedicated procedure cinv that does just that. For example, let's find some inverses.

```
> p:=e1+e2+2*e1we2;
```

$$p := e1 + e2 + 2\, e1we2$$

```
> pinv:=cinv(p);
```

$$pinv := -\frac{e1}{2} - \frac{e2}{2} - e1we2$$

```
> cmul(p,pinv);
```

$$Id$$

Thus, procedure 'cinv' has found the inverse to be 'pinv' displayed above. Let's try to use more general 'clisolve'.

```
> 'pp'=pp; #the most general element in Cl(2,0)
```

$$pp = x_1\, Id + x_2\, e1 + x_3\, e2 + x_4\, e1we2$$

```
> eq:=cmul(p,pp)-Id; #equation to be solved
```

$$eq := (x_1 - x_4 + 2\, x_3)\, e1 + (x_1 - 2\, x_2 + x_4)\, e2 + (x_2 + x_3 - 2\, x_4)\, Id + (2\, x_1 - x_2 + x_3)\, e1we2 - Id$$

```
> clisolve(eq,pp);
```

$$\left[-\frac{e1}{2} - \frac{e2}{2} - e1we2\right]$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 1.109000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

Thus, there is exactly one solution.

See Also: Clifford:-cbasis, Clifford:-`type/clipolynom`, Clifford:-reversion, Clifford:-cmul

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-clisort - sort given Clifford polynomial

Calling Sequence:

clisort(p);

Parameters:

p - expression of the type 'cliscalar', 'clibasmon', 'climon' or 'clipolynom' or 'matrix'

Description:

- Procedure 'clisort' sorts the given multivariate Clifford polynomial with respect to the Clifford indeterminates found in it.
- It puts scalar coefficients of the type 'cliscalar' in front of the Clifford basis monomials of the type 'clibasmon'. See `type/cliscalar` and `type/clibasmon` for more help.
- This procedure may also be applied to matrices with entries in a Clifford algebra in which case it sorts every matrix entry.

Examples:

```
> restart:bench:=time():with(Clifford):
> clisort(3*Pi+e2we3*2*alpha-e4we3*6/sqrt(a^2+1));
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$2\,\alpha\,e2we3 - \frac{6\,e4we3}{\sqrt{a^2+1}} + 3\,\pi$$

```
> clisort(Id*2*Pi);
```

$$2\,\pi\,Id$$

```
> M:=linalg[matrix](2,2,[e1we2*Pi*a,e3*3,e4,Id*B[1,1]]);
```

$$M := \begin{bmatrix} e1we2\,\pi\,a & 3\,e3 \\ e4 & Id\,B_{1,1} \end{bmatrix}$$

```
> clisort(M);
```

$$\begin{bmatrix} \pi\,a\,e1we2 & 3\,e3 \\ e4 & B_{1,1}\,Id \end{bmatrix}$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 0.093000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

See Also: Clifford:-clicollect, Clifford:-reorder, Clifford:-`type/cliscalar`, Clifford:-`type/clibasmon`, Clifford:-`type/climon`, Clifford:-`type/clipolynom`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-cliterms - find and display basis elements in a Clifford polynomial

Calling Sequence:

cliterms(p);

Parameters:

p - an expression of type 'cliscalar', 'clibasmon', 'climon', 'clipolynom'

Description:

- Procedure 'cliterms' identifies Clifford basis elements in the given Clifford polynomial p of type `type/clibasmon` or `type/cliprod`.
- It displays the basis elements of p in a form of a set.
- If the input contains a scalar term of the type 'cliscalar', then 'Id' is returned for that term. See `type/cliscalar` for more help.
- Recall that when package 'Cliplus' is loaded in, the following two types in CLIFFORD: `type/climon` and `type/clipolynom` get redefined and include now polynomial and monomial terms with expressions of `type/cliprod`. When such expressions are present, 'clicollect' collects with respect to these terms as well.
- When input contains elements of type cliprod, they are counted and returned like clibasmons, however, a warning message is printed. Printing of the warning message can be suppressed by setting global variable _warnings_flag to false. See CLIFFORD_ENV for more help on CLIFFORD's environmental variables.

Examples:

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

$$true$$

```
> cliterms(3*Pi+2*e2we3-e2+2*alpha*e1we2we3);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$\{Id, e2we3, e2, e1we2we3\}$$

```
> cliterms(1+e1+e2we1+&C(e1,e2));
```

$$\{Id, e2we1, e1, e1\ \&C\ e2\}$$

```
> _warnings_flag:=false:
> cliterms(1+e1+e2we1+&C(e1,e2));
```

$$\{Id, e2we1, e1, e1\ \&C\ e2\}$$

```
> type(1+e1+e2we1+&C(e1,e2),clipolynom);
```

$$true$$

```
> cliterms(1+e1+e2we1+&C(e1,e2));
```

$$\{Id, e2we1, e1, e1\ \&C\ e2\}$$

```
> cliterms(e1/sqrt(a^2+1));
```

$$\{e1\}$$

```
> cliterms((e1-2*e2+4)/sqrt(a^2+1));
```

$$\{Id, e2, e1\}$$

```
> cliterms(3*`&C`[K](e1,e3));
```

$$\{\&C_K(e1, e3)\}$$

```
> cliterms(3*`&C`[K](e1,e3)+2*`&C`[K](e1,e3,e4));
```

$$\{\&C_K(e1, e3), \&C_K(e1, e3, e4)\}$$

```
>
```

See Also: Clifford:-displayid, Clifford:-`type/clipolynom`, Clifford:-`type/cliprod`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-cmul - Clifford product in the Clifford algebra Cl(B) and its infix form '&c'

Calling Sequence:

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

p1 &c p2 &c ... &c pn;
&c[K](p1,p2,...,pn);
&c["K"](p1,p2,...pn) (when K is assigned a matrix)

Parameters:

p1, p2, ..., pn - expressions of the type 'cliscalar' or 'clipolynom'
K - (optional) index of type name, symbol, matrix, or array, or
`&*`(numeric,{name,symbol,array,matrix})

Description:

- Procedure 'cmul' and its infix form '&c' give the Clifford multiplication in the Clifford algebra of an arbitrary bilinear form B when no optional index is specified. If index K is specified, then computations takes place in Cl(K). For example, it is often desirable to compute Clifford product in Cl(g) where g is the symmetric part of B.
- The bilinear form B is totally arbitrary: symbolic, undefined, symmetric, diagonal, with or without an antisymmetric part, numeric.
- Procedure 'cmul' works in all dimensions 1 through 9. When the bilinear form is diagonal, use 'cmulQ' which gives Clifford multiplication in orthogonal Clifford algebras. See cmulQ for more help.
- Via the procedure 'rmulm', this multiplication can also be applied to matrices with entries in the Clifford algebra Cl(B). See rmulm for more help. Clifford multiplication 'cmul' can be based on the recursive use of the following formula:

  xu = cmul(x,u) = x &c u = wedge(x, u) + LC(x, u) = x &w u + LC(x, u)

  or on the Rota-Stein cliffordization technique. In the first case, 'cmul' internally uses procedure cmulNUM while in the second case it uses cmulRS. Selection which product is used can be made by the user using procedure useproduct. The default is 'cmulRS'. See also cmulgen which is the generic name for the Clifford product used by CLIFFORD. To find out which product is currently use, execute procedure CLIFFORD_ENV which displays all values of environmental variables used by CLIFFORD, or, see the value currently assigned to the global variable _default_Clifford_product.
- In the above, x is a 1-vector and u is any element in the algebra Cl(Q), wedge(x,u) = x &w u denotes the wedge or exterior product between x and u, and LCQ(x, u) denotes the left contraction of u by x. See wedge and LCQ for more help.

- Through this formula, the Clifford algebra Cl(B) becomes a subalgebra of an algebra of endomorphisms of an exterior algebra over V.
- Clifford inverse in Cl(B) is computed with cinv. In general, Clifford inverse of p in Cl(K) is computed with cinv(p,K).
- To speed up computations, set the global variable _prolevel to 'true'. To find out more, see help page on cliparse and `type/clipolynom`.
- See also procedure cmulB in the package 'GTP' for the Clifford product in the Clifford algebra Cl(B). The difference between 'cmul' and 'cmulB' is that 'cmul' uses one bilinear form B which is globally defined. On the other hand, 'cmulB' needs some bilinear form B to be specified as one of its inputs. This allows one to use 'cmulB' with different bilinear forms B in the same worksheet.
- Upon loading of a supplementary package 'Cliplus', this procedure can then apply Clifford product to polynomials containing unevaluated Clifford products, that is, expressions of `type/cliprod` like `&C`(e1,e3,e4), or `&C`[K](e1,e2), etc.
- In the ampersand version &c of 'cmul', when index K has been assigned a matrix, one must use double quotes as in &c["K"](p1,p2,...,pn) to stop premature evaluation of K to a matrix by Maple. Enclosing K in the double quotes is not needed in 'cmul'.
- To see how to define a new "Clifford product", go to cmul_user_defined for some help.

Examples:

```
> restart:bench:=time():
  with(Clifford):eval(makealiases(6,'ordered')):
  s:=time():
```

```
> _prolevel; #default value is 'false'
  _default_Clifford_product;
```

$$false$$

$$Clifford{:}\text{-}cmulRS$$

```
>
```

Example 1: Let's see simple computations involving Clifford product and a symbolic, unassigned bilinear form B:

```
> assigned(B);
```

$$false$$

```
> cmul(3*Pi*e2we1we3,e4+3*e5,e6,e4+3*e5);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$3\,\pi\,(B_{4,6}\,B_{2,4} - B_{2,6}\,B_{4,4} + B_{2,4}\,B_{6,4} + 3\,B_{2,5}\,B_{6,4} - 3\,B_{2,6}\,B_{4,5} + 9\,B_{5,6}\,B_{2,5} - 9\,B_{2,6}\,B_{5,5} + 3\,B_{5,6}\,B_{2,4} + 3\,B_{4,6}\,B_{2,5} + 9\,B_{2,5}\,B_{6,5} + 3\,B_{2,4}\,B_{6,5} - 3\,B_{2,6}\,B_{5,4})\,e13 - 3\,\pi\,(9\,B_{5,6}\,B_{1,5} - 9\,B_{1,6}\,B_{5,5} + B_{4,6}\,B_{1,4} + 3\,B_{1,4}\,B_{6,5} + 9\,B_{1,5}\,B_{6,5} + 3\,B_{4,6}\,B_{1,5} + 3\,B_{1,5}\,B_{6,4} - 3\,B_{1,6}\,B_{5,4} + B_{1,4}\,B_{6,4} - 3\,B_{1,6}\,B_{4,5} + 3\,B_{5,6}\,B_{1,4} - B_{1,6}\,B_{4,4})\,e23 - 3\,\pi\,(-3\,B_{3,6}\,B_{5,4} + 9\,B_{3,5}\,B_{6,5}$$

$$+ B_{3,4} B_{6,4} - B_{3,6} B_{4,4} + 3 B_{5,6} B_{3,4} - 9 B_{3,6} B_{5,5} + 9 B_{5,6} B_{3,5} + 3 B_{3,4} B_{6,5} - 3 B_{3,6} B_{4,5}$$
$$+ 3 B_{4,6} B_{3,5} + 3 B_{3,5} B_{6,4} + B_{4,6} B_{3,4}) e12 - 3 \pi (B_{4,6} + 3 B_{6,5} + B_{6,4} + 3 B_{5,6}) e1234$$
$$- 9 \pi (B_{4,6} + 3 B_{6,5} + B_{6,4} + 3 B_{5,6}) e1235 + 3 \pi (B_{4,4} + 9 B_{5,5} + 3 B_{4,5} + 3 B_{5,4}) e1236$$

One can also enter B explicitly. In place of `B`, one can use name of another form, or a numeric multiple of a name:

```
> 'cmul[B](ei,ej,ek)'=cmul[B](ei,ej,ek);
  '&c[B](ei,ej,ek)'=&c[B](ei,ej,ek);
  'cmul[K](ei,ej,ek)'=cmul[K](ei,ej,ek);
  '&c[K](ei,ej,ek)'=&c[K](ei,ej,ek);
  'cmul[g](ei,ej,ek)'=cmul[g](ei,ej,ek);
  '&c[g](ei,ej,ek)'=&c[g](ei,ej,ek);
  'cmul[-B](ei,ej,ek)'=cmul[-B](ei,ej,ek);
  '&c[-B](ei,ej,ek)'=&c[-B](ei,ej,ek);
  'cmul[-K](ei,ej,ek)'=cmul[-K](ei,ej,ek);
  '&c[-K](ei,ej,ek)'=&c[-K](ei,ej,ek);
  'cmul[-g](ei,ej,ek)'=cmul[-g](ei,ej,ek);
  '&c[-g](ei,ej,ek)'=&c[-g](ei,ej,ek);
```

$$Cliplus{:}\text{-}climul_{B}(ei, ej, ek) = eiwejwek - B_{i,k}\, ej + B_{i,j}\, ek + B_{j,k}\, ei$$
$$Cliplus{:}\text{-}climul([B](ei, ej, ek)) = eiwejwek - B_{i,k}\, ej + B_{i,j}\, ek + B_{j,k}\, ei$$
$$Cliplus{:}\text{-}climul_{K}(ei, ej, ek) = eiwejwek - K_{i,k}\, ej + K_{i,j}\, ek + K_{j,k}\, ei$$
$$Cliplus{:}\text{-}climul([K](ei, ej, ek)) = eiwejwek - K_{i,k}\, ej + K_{i,j}\, ek + K_{j,k}\, ei$$
$$Cliplus{:}\text{-}climul_{g}(ei, ej, ek) = eiwejwek - g_{i,k}\, ej + g_{i,j}\, ek + g_{j,k}\, ei$$
$$Cliplus{:}\text{-}climul([g](ei, ej, ek)) = eiwejwek - g_{i,k}\, ej + g_{i,j}\, ek + g_{j,k}\, ei$$
$$Cliplus{:}\text{-}climul_{-B}(ei, ej, ek) = eiwejwek + B_{i,k}\, ej - B_{i,j}\, ek - B_{j,k}\, ei$$
$$Cliplus{:}\text{-}climul([-B](ei, ej, ek)) = eiwejwek + B_{i,k}\, ej - B_{i,j}\, ek - B_{j,k}\, ei$$
$$Cliplus{:}\text{-}climul_{-K}(ei, ej, ek) = eiwejwek + K_{i,k}\, ej - K_{i,j}\, ek - K_{j,k}\, ei$$
$$Cliplus{:}\text{-}climul([-K](ei, ej, ek)) = eiwejwek + K_{i,k}\, ej - K_{i,j}\, ek - K_{j,k}\, ei$$
$$Cliplus{:}\text{-}climul_{-g}(ei, ej, ek) = eiwejwek + g_{i,k}\, ej - g_{i,j}\, ek - g_{j,k}\, ei$$
$$Cliplus{:}\text{-}climul([-g](ei, ej, ek)) = eiwejwek + g_{i,k}\, ej - g_{i,j}\, ek - g_{j,k}\, ei$$

```
> e1 &c e2 &c e3; e1 &cQ e1 &cQ e3; #the infix forms of 'cmul'
  and 'cmulQ'
  &c(e1,e2,e3); &cQ(e1,e1,e3); #the infix forms of 'cmul' and
  'cmulQ'
  &c[K](e1,e2,e3); &cQ[K](e1,e1,e3); #the infix forms of 'cmul'
  and 'cmulQ'
  &c[-K](e1,e2,e3); &cQ[-K](e1,e1,e3); #the infix forms of 'cmul'
```

```
and 'cmulQ'
```

$$e123 + B_{2,3}\, e1 - B_{1,3}\, e2 + B_{1,2}\, e3$$

$$B_{1,1}\, e3$$

$$e123 + B_{2,3}\, e1 - B_{1,3}\, e2 + B_{1,2}\, e3$$

$$B_{1,1}\, e3$$

$$e123 + K_{2,3}\, e1 - K_{1,3}\, e2 + K_{1,2}\, e3$$

$$K_{1,1}\, e3$$

$$e123 - K_{2,3}\, e1 + K_{1,3}\, e2 - K_{1,2}\, e3$$

$$-K_{1,1}\, e3$$

```
> (4+e1) &c (eiwej+4*alpha*Pi*Id);
```

$$B_{1,i}\, ej - B_{1,j}\, ei + 16\, \alpha\, \pi\, Id + 4\, \alpha\, \pi\, e1 + 4\, eiwej + e1weiwej$$

```
> cmul(ei,ej)+cmul(ej,ei);clicollect(%); #equals 0 when
  B[i,j]=-B[j,i]
```

$$B_{i,j}\, Id + B_{j,i}\, Id$$

$$(B_{i,j} + B_{j,i})\, Id$$

```
> clicollect(e1 &c e2 + e2 &c e1); #equals 0 when B[1,2]=-B[2,1]
```

$$(B_{1,2} + B_{2,1})\, Id$$

```
> clicollect(e1 &cQ e2 + e2 &cQ e1); #equals 0
```

$$(B_{1,2} + B_{2,1})\, Id$$

```
> e1 &c e1, e1 &c e2, e1 &cQ e1, e1 &cQ e2;
```

$$B_{1,1}\, Id,\ B_{1,2}\, Id + e12,\ B_{1,1}\, Id,\ B_{1,2}\, Id + e12$$

```
>
```

Example 2: Now, the same computations as in Example 1 above but performed with an arbitrary symbolic form g of `type/name` used as an index. For example, g could be the symmetric part of B.

```
> type(g,name);
```

$$true$$

```
> cmul[g](3*Pi*e2we1,e4+3*e5,e6,2*e4); # long notation using
  'cmul'
```

$$\begin{aligned}
&-18\, \pi\, e12456 - 6\, \pi\, (g_{6,4} + g_{4,6} + 3\, g_{5,6})\, e124 - 18\, \pi\, g_{6,4}\, e125 + 6\, \pi\, (3\, g_{5,4} + g_{4,4})\, e126 \\
&\quad - 18\, \pi\, g_{2,6}\, e145 + 18\, \pi\, g_{2,5}\, e146 - 18\, \pi\, g_{2,4}\, e156 + 18\, \pi\, g_{1,6}\, e245 - 18\, \pi\, g_{1,5}\, e246 \\
&\quad + 18\, \pi\, g_{1,4}\, e256 \\
&\quad - 6\, \pi\, (g_{4,6}\, g_{2,4} + 3\, g_{5,6}\, g_{2,4} + g_{2,4}\, g_{6,4} + 3\, g_{2,5}\, g_{6,4} - g_{2,6}\, g_{4,4} - 3\, g_{2,6}\, g_{5,4})\, e1 \\
&\quad + 6\, \pi\, (3\, g_{1,5}\, g_{6,4} + g_{4,6}\, g_{1,4} + g_{1,4}\, g_{6,4} - 3\, g_{1,6}\, g_{5,4} + 3\, g_{5,6}\, g_{1,4} - g_{1,6}\, g_{4,4})\, e2 \\
&\quad + 18\, \pi\, (-g_{2,5}\, g_{1,6} + g_{1,5}\, g_{2,6})\, e4 - 18\, \pi\, (-g_{2,4}\, g_{1,6} + g_{1,4}\, g_{2,6})\, e5
\end{aligned}$$

$$+ 18\,\pi\,(g_{2,5}\,g_{1,4} - g_{1,5}\,g_{2,4})\,e6$$

```
> &c[-g](3*Pi*e2we1,e4+3*e5,e6,2*e4); # short notation using &c
```

$$-18\,\pi\,e12456 + 6\,\pi\,(g_{6,4} + g_{4,6} + 3\,g_{5,6})\,e124 + 18\,\pi\,g_{6,4}\,e125 - 6\,\pi\,(3\,g_{5,4} + g_{4,4})\,e126$$
$$+ 18\,\pi\,g_{2,6}\,e145 - 18\,\pi\,g_{2,5}\,e146 + 18\,\pi\,g_{2,4}\,e156 - 18\,\pi\,g_{1,6}\,e245 + 18\,\pi\,g_{1,5}\,e246$$
$$- 18\,\pi\,g_{1,4}\,e256$$
$$- 6\,\pi\,(g_{4,6}\,g_{2,4} + 3\,g_{5,6}\,g_{2,4} + g_{2,4}\,g_{6,4} + 3\,g_{2,5}\,g_{6,4} - g_{2,6}\,g_{4,4} - 3\,g_{2,6}\,g_{5,4})\,e1$$
$$+ 6\,\pi\,(3\,g_{1,5}\,g_{6,4} + g_{4,6}\,g_{1,4} + g_{1,4}\,g_{6,4} - 3\,g_{1,6}\,g_{5,4} + 3\,g_{5,6}\,g_{1,4} - g_{1,6}\,g_{4,4})\,e2$$
$$+ 18\,\pi\,(-g_{2,5}\,g_{1,6} + g_{1,5}\,g_{2,6})\,e4 - 18\,\pi\,(-g_{2,4}\,g_{1,6} + g_{1,4}\,g_{2,6})\,e5$$
$$+ 18\,\pi\,(g_{2,5}\,g_{1,4} - g_{1,5}\,g_{2,4})\,e6$$

```
> (4+e1) &c (eiwej+4*alpha*Pi*Id);
```

$$B_{1,i}\,ej - B_{1,j}\,ei + 16\,\alpha\,\pi\,Id + 4\,\alpha\,\pi\,e1 + 4\,eiwej + e1weiwej$$

```
> cmul[B](ei,ej)+cmul[B](ej,ei);clicollect(%); #equals 0 when
  B[i,j]=-B[j,i]
  cmul[-B](ei,ej)+cmul[-B](ej,ei);clicollect(%); #equals 0 when
  B[i,j]=-B[j,i]
```

$$B_{i,j}\,Id + B_{j,i}\,Id$$
$$(B_{i,j} + B_{j,i})\,Id$$
$$-B_{i,j}\,Id - B_{j,i}\,Id$$
$$(-B_{i,j} - B_{j,i})\,Id$$

```
> clicollect(e1 &c e2 + e2 &c e1); #equals 0 when B[1,2]=-B[2,1]
  clicollect(&c[B](e1,e2) + &c[B](e2,e1)); #equals 0 when
  B[1,2]=-B[2,1]
```

$$(B_{1,2} + B_{2,1})\,Id$$
$$(B_{1,2} + B_{2,1})\,Id$$

```
> clicollect(e1 &cQ e2 + e2 &cQ e1); #equals 0
```

$$(B_{1,2} + B_{2,1})\,Id$$

```
> e1 &c e1, e1 &c e2, e1 &cQ e1, e1 &cQ e2;
```

$$B_{1,1}\,Id,\ B_{1,2}\,Id + e12,\ B_{1,1}\,Id,\ B_{1,2}\,Id + e12$$

```
>
```

Example 3: Same computations as in Example 3 except that we assign a matrix to B.

```
> B:=linalg[diag](1$6);
```

$$B := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

```
> cmul(3*Pi*e2we1we3,e4+3*e5,e6,e4+3*e5);
  cmul[B](3*Pi*e2we1we3,e4+3*e5,e6,e4+3*e5);
```

$$30\,\pi\,e1236$$

$$30\,\pi\,e1236$$

Since B has been assigned a matrix, we must use double quotes around the index in `&c` to prevent premature evaluation of the index by Maple as shown below:

```
> &c[B](3*Pi*e2we1we3,e4+3*e5,e6,e4+3*e5);
Error, (in Clifford:-cliparse) check spelling of sparse or define it as a
constant or an alias
```

```
> &c['B'](3*Pi*e2we1we3,e4+3*e5,e6,e4+3*e5);
Error, (in Clifford:-cliparse) check spelling of sparse or define it as a
constant or an alias
```

```
> &c[''B''](3*Pi*e2we1we3,e4+3*e5,e6,e4+3*e5);
```

$$30\,\pi\,e1236$$

```
>
```

Example 4: Let's see how to multiply basis monomials in Cl(3):

```
> clibasis:=cbasis(3);B:=linalg[diag](1,1,1):
```

$$clibasis := [Id, e1, e2, e3, e12, e13, e23, e123]$$

```
> M:=matrix(8,8,(i,j)->cmul(clibasis[i],clibasis[j])):
> evalm(M);
```

$$\begin{bmatrix} Id & e1 & e2 & e3 & e12 & e13 & e23 & e123 \\ e1 & Id & e12 & e13 & e2 & e3 & e123 & e23 \\ e2 & -e12 & Id & e23 & -e1 & -e123 & e3 & -e13 \\ e3 & -e13 & -e23 & Id & e123 & -e1 & -e2 & e12 \\ e12 & -e2 & e1 & e123 & -Id & -e23 & e13 & -e3 \\ e13 & -e3 & -e123 & e1 & e23 & -Id & -e12 & e2 \\ e23 & e123 & -e3 & e2 & -e13 & e12 & -Id & -e1 \\ e123 & e23 & -e13 & e12 & -e3 & e2 & -e1 & -Id \end{bmatrix}$$

```
>
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 1.281000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

Thus, the above is the multiplication table for the basis monomials in Cl(3).

See Also: Clifford:-cmulQ, Clifford:-rmulm, Clifford:-`type/clipolynom`, Clifford:-`type/climon`, Clifford:-`type/clibasmon`, Clifford:-`type/cliscalar`, Clifford:-cmulNUM, Clifford:-cmulRS, Clifford:-cmulgen, Clifford:-useproduct

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-cmul_user_defined - user defined procedure for Clifford product that will be used by Clifford:-cmul

Calling Sequence:

cmul_user_defined(p1,p2,K);

Parameters:

p1, p2 - elements in a Clifford algebra Cl(B) of type 'cliscalar', 'clibasmon', 'climon', 'clipolynom', or some other new type defined by user
K - required argument of type 'name', 'symbol', 'matrix', or 'array'

Description:

- Name 'cmul_user_defined' is one of the allowed names that procedure 'useproduct' accepts. It allows user to select which procedure, that gives Clifford product, is used in 'cmul' internally.
- This is a generic name for the Clifford product that user may have developed. It is expected to be a procedure that takes exactly three arguments: p1, p2, and K, where p1 and p2 are, for example, Clifford polynomials, while K is of type name, symbol, matrix, array (see some examples below).
- Thus, since the user can really change the internally used procedure in 'cmul', this procedure allows one to test a new code for Clifford product, or defined new Clifford products.

Examples:

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

Upon loading CLIFFORD, one of the environmental variables displayed by CLIFFORD_ENV is _default_Clifford_product. The default value of that variable is 'cmulRS' which gives the Clifford product computed using the Rota-Stein cliffordization technique (see package Bigebra for more help on the subject).

```
> CLIFFORD_ENV();
```

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

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


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



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



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


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

If user wants to create a new Clifford product, it can be assigned to a name _user_defined_Clifford_product. Of course, that generic name can be aliased to some shorter name. Here we just show an example where no procedure has been assigned yet to cmul_user_defined:

```
> useproduct(cmul_user_defined);
Warning, no computations with cmul can be peformed yet since cmul_user_defined
has not been defined as procedure. Select cmulRS, cmulNUM, or a new procedure
as argument to useproduct.
```

Example 1: For example, we could define the following new Clifford product:

```
> cmul_user_defined:=proc(p1::{cliscalar,clibasmon,climon,clipoly
  nom},

  p2::{cliscalar,clibasmon,climon,clipolynom},
                                K::{name,symbol,matrix,array}) local
```

```
a1,a2,p:
a1:=scalarpart(p1)+vectorpart(p1,1):#assign to a1 only the
scalar and the 1-vector parts of p1
a2:=scalarpart(p1)+vectorpart(p2,1):#assign to a2 only the
scalar and the 1-vector parts of p2
return LC(a1,a2,K)
end proc:
```

As it can be guessed from the above code, the result of cmul_user_defined(p1,p2,K) is just the left contraction of the scalar and 1-vector part of p2 by the scalar and the 1-vector part of p1 where the contraction is taken with respect to some form K.

```
> useproduct(cmul_user_defined);
Warning, cmul will use cmul_user_defined; for help see pages ?cmul,
?Clifford:-intro, or ?cmul_user_defined
```

```
> cmul(e1,e2+e3we4);
```

$$B_{1,2}\, Id$$

```
>
```

See Also: Clifford:-`type/clipolynom`, Clifford:-`type/climon`, Clifford:-`type/clibasmon`, Clifford:-`type/cliscalar`, Clifford:-useproduct, Clifford:-cmulNUM, Clifford:-cmulRS, Clifford:-cmulgen, Clifford:-vectorpart, Clifford:-scalarpart

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-cmulgen - place holder for a procedure that will be used by Clifford:-cmul] to compute Clifford product, or, it can be used directly

Calling Sequence:

cmulgen(p1,p2,K);

Parameters:

p1,p2 - elements in Cl(B) of type 'cliscalar', 'clibasmon', 'climon', or 'clipolynom'
K - required argument of type 'name', 'symbol', 'matrix', or 'array', or
`&*`(numeric,{name,symbol,matrix,array})

Description:

- This procedure is a 'dummy procedure' which checks if the currently assigned name to the global variable _default_Clifford_product is one of 'cmulNUM', 'cmulRS', 'cmulgen', or 'cmul_user_defined'. It is called upon by 'cmul' when it needs to compute a product between two Clifford elements p1 and p2 in Cl(K).
- Procedure 'cmul' is then a wrapper functions which extends 'cmulNUM', 'cmulRS', and 'cmul_user_defined' to more than just two polynomials.

Examples:

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

Upon loading CLIFFORD, one of the environmental variables displayed by CLIFFORD_ENV is _default_Clifford_product. The default value of that variable is 'cmulRS' which gives the Clifford product computed using the Rota-Stein cliffordization technique (see package Bigebra for more help on the subject).

```
> CLIFFORD_ENV();
```

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

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


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



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



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


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

To see that indeed procedure 'cmulRS' is used for internal computations, let's cause an error message in 'cmulgen':

```
> _default_Clifford_product;
  cmulgen(e1,e2);  ###<<<- Intended error
```

$$Clifford{:}\text{-}cmulNUM$$

```
Error, (in Clifford:-cmulNUM) exactly three arguments are needed
```

```
> cmulgen(e1,e2,B);
  cmulgen(e1,e2,-B);
```

$$e1we2 + B_{1,2}\, Id$$

$$e1we2 - B_{1,2}\, Id$$

Hence, 'cmulgen' holds the name 'cmulRS'. Since 'cmulgen' applies internally name currently assigned to the global variable _default_Clifford_product to its arguments, and since 'cmulRS'

(like 'cmulNUM') requires three arguments, the above error message has been returned.

It is possible to change the value of _default_Clifford_product to one of the following: 'cmulNUM', cmulgen, or _user_defined_Clifford_product. When the change is made, a warning message is printed as follows:

```
> _default_Clifford_product; #current value of the global
  variable
  useproduct(cmulNUM); #selecting 'cmulNUM'
  _default_Clifford_product; #current value of the global
  variable
```

Clifford:-cmulNUM

```
Warning, cmul will use cmulNUM; for help see pages ?cmul, ?Clifford:-intro, or
?cmulNUM
```

Clifford:-cmulNUM

```
> _default_Clifford_product;
  cmulgen(e1,e2);  ###<<<- Intended error
```

Clifford:-cmulNUM

```
Error, (in Clifford:-cmulNUM) exactly three arguments are needed
```

Since now we wanted to use 'cmulNUM', the above shows that instead of 'cmulRS', 'cmulNUM' is now used internally to compute Clifford product.

Let's assign now the generic name 'cmulgen' to the global variable

```
> useproduct(cmulgen);
Warning, cmul will use cmulgen; for help see pages ?cmul, ?Clifford:-intro, or
?cmulgen
```

```
> cmulgen(e1,e2);
Warning, to assign Clifford product, execute 'useproduct' with argument cmulRS,
cmulNUM, or cmul_user_defined first
```

Clifford:-cmulgen(e1, e2)

Of course nothing has happened since _default_Clifford_product = cmulgen which is a dummy variable.

```
> _default_Clifford_product;
```

Clifford:-cmulgen

```
>
```

If user wants to create a new Clifford product, it can be assigned to a name _user_defined_Clifford_product. Of course, that generic name can be aliased to some shorter name. Go to cmul_user_defined for more help.

See Also: Clifford:-`type/clipolynom`, Clifford:-`type/climon`, Clifford:-`type/clibasmon`, Clifford:-`type/cliscalar`, Clifford:-cmulNUM, Clifford:-cmulRS, cmul_user_defined,

Clifford:-vectorpart, Clifford:-scalarpart

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-cmulNUM - Clifford product for two Grassmann basis monomials and (arbitrary but preferably) sparse numeric bilinear form

Calling Sequence:

c3 := cmuNUM(c1,c2,B)

Parameters:

- c1, c2- Clifford monomials (elements of the type: type/clibasmon)
- B - bilinear form of type `name`, `symbol`, `matrix`, `array`, or `&*`(numeric,{name,symbol,array,matrix})

Output:

- c1 : a Clifford polynom

Description:

- Since the Clifford product provides the main functionality of the CLIFFORD package, the best available mathematics has been incorporated in its code. The user normally does not use the internal functions cmulRS and cmulNUM but the wrapper function `&c`[K](arg1,arg2,...) which allows one to pass the name of a bilinear form K. The wrapper function can also act on any number of arguments of type clipolynom (since the Clifford product is associative this makes sense) and on a much wider class of types including Clifford matrices of type climatrix or Clifford polynomials in other bases, see Cliplus.
- There is a facility to use one of the internal functions (cmulRS or cmulNUM) when knowledge of the bilinear form allows one to decide which procedure might yield better performance. Moreover, the user can supply a `product` function (not necessarily a Clifford product) acting on two clibasmons, that is, monomials of type clibasmon. The wrapper uses an appropriate function, cmulRS, cmulNUM, or other, which has been selected by the user via a special function Clifford:-useproduct.
- There are two internal Clifford multiplications which are appropriate for different purposes. While cmulNUM is fast on sparse numeric matrices and on numeric matrices in general for dimensions dim_V >= 5, cmulRS was designed for symbolic calculations, but since it computes reasonably well in the numeric sparse case up to dim_V = 5, it was chosen as the default product of the package. Both internal multiplication procedures, cmulRS and cmulNUM, take two Clifford monomials of type clibasmon as input together with a third argument of type name/symbol/matrix/array which represents chosen bilinear form.
- To see how to define a new "Clifford product", go to cmul_user_defined for some help.
- The evaluation of Clifford products in a Grassmann basis is quite involved and normally is done by a recursive process that involves Chevalley deformation. This approach has been employed in

cmulNUM. However, Hopf algebraic methods can be applied also, see cmulRS. Unfortunately, during the recursive evaluation many unnecessary terms are calculated which later cancel out at the next recursive call. This feature, while being beneficial when the bilinear form is sparse numeric since it cuts out many branches of the recursion quite early, prevents fast evaluation in the symbolic case where in general all terms might be non-zero. From this observation, the two possibilities to evaluate the Clifford product have emerged.

- We introduce the Chevalley deformation and the Clifford map to explain the algorithm used in cmulNUM. The Clifford map is defined as (x in V, u in /\V)

  γ _*x* (u) = LC(x,u,B) + x /\ u = x _|B u + x /\ u
  γ _*x* γ _*y* = γ _(*x wedge y*) + B(x,y)* γ _*Id*
  γ _(*a x* + *b y*) = a* γ _*x* + b* γ _*y*

  One knows how to compute with the wedge x /\ u and the contraction LC(x,u,B) = x _|B u, (see the help page Clifford:-LC for more information on the left contraction) remember the following relations for x,y, in V and u,v,w, in /\V:

  (1) x _|B y = B(x,y)
  (2) x _|B (u /\ v) = (x _|B u) /\ v + u^ /\ (x _|B v)
  (3) (u /\ v) _|B w = u _|B (v _|B w)

  where u^ is the Grassmann grade involution. Hence we can use the Clifford map γ _*x* (Chevalley deformation of the Grassmann algebra) to define a Clifford product of a one-vector and a multivector. Analogous formulas can be given for a right Clifford map using the right contraction. See Clifford:-RC for help on right contraction. Below, the right contraction is denoted by |_ .

- The Clifford product '&c' can now be defined as follows: We have to split off a single element from the first factor of a product of two Grassmann basis monomials and then use Chevalley's Clifford map (parentheses ( ) are employed for better readability and to show precedence of the involved operations, usually contractions have a higher precedence as wedge products as Clifford product to allow dropping parentheses):

  (eaw..webwec) &c (efw..weg) = (eaw..wewb &c ec) &c (efw..weg) - (eaw..web B |_ ec) &c (efw..weg)
  = (eaw..wewb) &c (ec _|B (efw..weg) + ecwefw..weg) - (eaw..web B |_ ec) &c (efw..weg)

  e.g. for (e1we2) &c (e3we4) we have

  (e1we2) &c (e3we4) = (e1 &c e2) &c (e3we4) - (B(e1,e2) Id) &c (e3we4)
  = e1 &c (B(e2,e3) e4 - B(e2,e4) e3) + e2we3we4) - B(e1,e2) (Id) &c (e3we4)

a second recursion of the process gives now

= B(e2,e3)B(e1,e4) - B(e2,e4)B(e1,e3) + B(e2,e3) e1we4 - B(e2,e4) e1we3
+ **B(e1,e2)e3we4** - B(e1,e3) e2we4 + B(e1,e4) e2we3 + e1we2we3we4 - **B(e1,e2)e3we4**

with the bold terms canceling out. Note that the last term in the right hand side was generated (superfluously) in the first step of the recursion.

- The Clifford product can be derived from the above recursion. The induction starts with a left factor of grade one or grade zero which is trivial, i.e. Id &c eaw..web = eaw..web. In the case that the left factor is of grade one we use the Clifford product expressed by the Clifford map of Chevalley, i.e. ea &c ebw..wec = ea _|B ebw..wec + eawebw..wec. We make a complete induction in the following way. If the left factor is of higher grade, say n, one application of the recursion yields Clifford products where the left factor side is of grade either n-1 or n-2, hence the recursion stops after at most n-1 steps.
- A disadvantage of the recursive approach is that additional terms are produced by shifting Grassmann wedge products into Clifford products to swap one factor to the right. These terms cancel out, but this process increases unnecessarily computing time.
- An advantage of the recursive approach occurs when the bilinear form B is numeric and sparse (many zeros in the matrix of B). In this case after any recursion many terms are dropped (since Maple simplifies such expressions immediately) and only a few remaining terms enter the next step of the recursion. If the dimension of V is large i.e. dim_V >=6, sparse matrices benefit drastically from this process over the Hopf algebraic approach of cmulRS which computes all terms without benefiting from the special (sparse numeric) form of the bilinear form.
- One could think about shifting factors from right to left, however this works out in the same way. Moreover, if the grade of the left factor n is greater than the grade m of the right factor, then the recursion stops also (since the terms evaluate to zero) after at most n-1 steps, so no increase in performance can be gained this way.
- For clarity and to show our approach we display the (simplified) algorithm of cmulNUM:

```
cmulNUM(x,y,B)   ## x,y, clibasmons and B a bilinear form
  begin
  if x = 0 or y = 0 then RETURN(0) fi   # first assumption about the induction
  if y = Id then RETURN(x) fi           # Id is unit
  lstx <- extract indices from x
  NX <- |x|
  if   NX = 0 then RETURN(coefficient_of(x)*y)   # handles terms like a*Id &c u
  elif NX = 1 then                          # first basis of recursion
    lsty <- extract indices from y
```

```
  RETURN( reorder -> (
    makeclibasmon(cat->(lstx[1],lsty))
   +sum i in 1..|lsty| ((-1)^(i-1)*B[lstx[1],lsty[i]]*makeclibasmon(lsty minus lsty[i]))
  ));
elif NX = 2 then                          # second basis of recursion
  x1 <- first vector of x
  x2 <- second vector of x
  RETURN(clibilinear -> cmulNUM(x1,cmulNUM(x2,y,B),B)-B[lstx]*y)
fi                                        # and the actual recursion for the remaining cases
xn <- last vector of x
xr <- first to prelast vectors
RETURN(clibilinear -> ( cmulNUM(xr,cmulNUM(xn,xr,B),B) )
        -add (i in 1..NX-1) ((-1)^(i)*B[lst[NX-i],lstx[NX]]*
        *cmulNUM(makeclibasmon(lstx minus lstx[i]),y,B)))
end cmulNUM
```

The function clibilinear is used to make functions bilinear which can act on basis monomials only. A reasonable amount of computation is hidden in this statement.

- **References:**
  [1] R. Ablamowicz, B. Fauser, Effective algorithms for evaluating Clifford products, pre-print 2003

Examples:

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

We check for some special cases and use arbitrary bilinear forms, e.g. B, K, T ...

```
> cmulNUM(0,e1we2,K);
  cmulNUM(Id,Id,K);
  cmulNUM(Id,ei,K);
  cmulNUM(ei,Id,K);
  cmulNUM(e1,e2,K);
  cmulNUM(ei,ej,B);
  cmulNUM(ei,ej,-K);
```

$$0$$

$$Id$$

$$ei$$

$$ei$$

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

$$eiwej + B_{i,j}\, Id$$

$$eiwej - K_{i,j}\, Id$$

Now let us go for more complicated monomials:

```
> cmulNUM(e1we2,e2we3,B);
  cmulNUM(e1we2we3,e1we2we3,T);
  cmulNUM(e1we2we3,e4we5we6,-T);
```

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

$$B_{2,2}\, e1we3 + B_{2,2}\, B_{1,3}\, Id - B_{2,3}\, e1we2 - B_{2,3}\, B_{1,2}\, Id - B_{1,2}\, e2we3$$

$$\begin{aligned}
&T_{3,1}\, T_{2,2}\, e1we3 + T_{3,1}\, T_{2,2}\, T_{1,3}\, Id - T_{3,1}\, T_{2,3}\, e1we2 - T_{3,1}\, T_{2,3}\, T_{1,2}\, Id - T_{3,1}\, T_{1,2}\, e2we3 \\
&\quad + T_{3,2}\, T_{1,1}\, e2we3 + T_{3,2}\, T_{1,3}\, e1we2 - T_{3,2}\, T_{2,1}\, e1we3 - T_{3,2}\, T_{2,1}\, T_{1,3}\, Id + T_{3,2}\, T_{2,3}\, T_{1,1}\, Id \\
&\quad + T_{3,3}\, T_{2,1}\, e1we2 + T_{3,3}\, T_{2,1}\, T_{1,2}\, Id - T_{3,3}\, T_{2,2}\, T_{1,1}\, Id - T_{3,3}\, T_{1,2}\, e1we2 - T_{2,3}\, T_{1,1}\, e2we3 \\
&\quad + T_{2,3}\, T_{1,2}\, e1we3 + T_{1,3}\, T_{2,1}\, e2we3 - T_{1,3}\, T_{2,2}\, e1we3 \\
&-T_{2,6}\, T_{1,5}\, e3we4 + T_{3,6}\, T_{2,4}\, e1we5 + T_{2,4}\, T_{1,5}\, e3we6 + T_{2,6}\, T_{1,4}\, e3we5 - T_{2,4}\, T_{1,6}\, e3we5 \\
&\quad - T_{3,6}\, T_{2,5}\, e1we4 + T_{3,6}\, T_{1,5}\, e2we4 - T_{2,5}\, T_{1,4}\, e3we6 - T_{3,4}\, T_{1,5}\, e2we6 + T_{3,5}\, T_{2,6}\, e1we4 \\
&\quad + T_{2,5}\, T_{1,6}\, e3we4 - T_{3,4}\, T_{2,6}\, e1we5 + T_{3,4}\, T_{2,5}\, e1we6 + T_{1,5}\, e2we3we4we6 \\
&\quad + e1we2we3we4we5we6 + T_{3,5}\, T_{1,4}\, e2we6 + T_{3,4}\, T_{1,6}\, e2we5 - T_{3,5}\, T_{2,4}\, e1we6 \\
&\quad - T_{3,6}\, T_{1,4}\, e2we5 - T_{3,5}\, T_{1,6}\, e2we4 - T_{1,4}\, e2we3we5we6 - T_{1,6}\, e2we3we4we5 \\
&\quad + T_{2,4}\, e1we3we5we6 - T_{2,5}\, e1we3we4we6 + T_{2,6}\, e1we3we4we5 + T_{3,5}\, T_{2,4}\, T_{1,6}\, Id \\
&\quad - T_{3,5}\, T_{2,6}\, T_{1,4}\, Id - T_{3,4}\, T_{2,5}\, T_{1,6}\, Id + T_{3,4}\, T_{2,6}\, T_{1,5}\, Id - T_{3,6}\, T_{2,4}\, T_{1,5}\, Id + T_{3,6}\, T_{2,5}\, T_{1,4}\, Id \\
&\quad + T_{3,5}\, e1we2we4we6 - T_{3,4}\, e1we2we5we6 - T_{3,6}\, e1we2we4we5
\end{aligned}$$

To perform some benchmarks, we need some random Grassmann basis monomials:

```
> dim_V:=9:
  rd:=rand(0..dim_V):
  rdmonom:=proc(N) makeclibasmon([op({seq(rd(),i=1..N)} minus {0}
  )]) end:
  rdmonom(rd()),rdmonom(rd()),rdmonom(rd());
```

$$e1we3we4we5we9,\ Id,\ e1we2we3we4we7we9$$

We iterate over MAX basis monomials and compute their mutual products. Since we expect cmulNUM to be faster as cmulRS in the sparse numeric case, we explore this case here, for a benchmark with a symbolic matrix B see Clifford:-cmulRS] .

```
> B:=linalg[randmatrix](dim_V,dim_V,sparse,entries=rand(-1..1));
```

$$B := \begin{bmatrix} 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix}$$

Test the bilinear form (third line):

```
> seq(cmulNUM(e3,cat(e,i),B),i=1..7);
```

$$-e1we3, -e2we3, 0, e3we4, e3we5, e3we6, e3we7$$

And benchmark with this bilinear form the two cmul functions:

```
> MAX:=5:
  monlist:=[seq(rdmonom(rd()),i=1..MAX)];
  tm:=time():
  for i from 1 to MAX do
  for j from 1 to MAX do
    cmulRS(monlist[i],monlist[j],B);
  end do:
  end do:
  tm:=time()-tm:
  printf("computation with cmulRS needed %f seconds",tm);
  tm:=time():
  for i from 1 to MAX do
  for j from 1 to MAX do
    cmulNUM(monlist[i],monlist[j],B);
  end do:
  end do:
  tm:=time()-tm:
  printf("computation with cmulNUM needed %f seconds",tm);
```

$$monlist := [e1we2we3we8we9, Id, Id, e1we3, e1we2we6]$$

```
computation with cmulRS needed 0.094000 seconds
computation with cmulNUM needed 0.281000 seconds
```

Hence we find the desired result. The relative factor of performance between these functions changes with the randomly chosen basis monomials and due to the garbage collection in Maple, etc., but we find roughly a factor 1.4 in this setting, which is reasonable (since we had a mixture of higher and lower grades and only the higher terms compute really faster).

Note that one can increase speed in the computation of Clifford products in the BIGEBRA package by using it in the Bigebra:-`&map` function. This is possible since the tensor product is multilinear and the entries are reduced to be only clibasmons, however **the tensor product has to be evaluated.**

```
> with(Bigebra):
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[help]
> t1:=&t(a*e1+e2,b*e1we2,e3);
```

$$t1 := a\, b\, \&t(e1, e1we2, e3) + b\, \&t(e2, e1we2, e3)$$

```
> &map(t1,1,cmulRS,B);
  &map(t1,2,cmulRS,B);
```

$$0$$

$$a\, b\, ((e1\ \&t\ e1we2we3) + (e1\ \&t\ e2)) + b\, ((e2\ \&t\ e1we2we3) + (e2\ \&t\ e2))$$

```
> ## ERROR: Note that prefactors are suppressed in unevaluated
  &t's !!!!!
  &map('`&t`'(a*e1,b*e2),1,cmulRS,B);
  &map(&t(a*e1,b*e2),1,cmulRS,B);
```

$$\&t(e1we2)$$

$$a\, b\, \&t(e1we2)$$

```
>
```

See Also: Clifford:-cmulRS, Clifford:-`&c`, `type/clibasmon`, Bigebra:-&map

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-cmulQ - Clifford product in an orthogonal Clifford algebra Cl(Q) and its infix form '&cQ'

Calling Sequence:

cmulQ(p1,p2,...pn);
cmulQ[K](p1,p2,...pn);

p1 &cQ p2 &cQ ... &cQ pn;
&cQ(p1,p2,p3,...,pn);
&cQ[K](p1,p2,p3,...,pn);
&cQ["K"](p1,p2,p3,...,pn); (when K has been assigned a matrix)

Parameters:

p1, p2, ..., pn - expressions of the type 'cliscalar' or 'clipolynom'

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

Description:

- Procedure 'cmulQ' and its infix form '&cQ' are special versions of 'cmul' and '&c' respectively. It gives the Clifford multiplication in the Clifford algebra of the quadratic form Q related to the symmetric part g of B as Q(x) = g(x, x) = B(x, x) where B = g + A and A is the alternating part of B (it could be identically equal to zero).
- If an optional index K is used, computations are performed w.r.t. to the diagonal entries of K in Cl(K).
- Via the procedure 'rmulm', this multiplication can also be applied to matrices with entries in an orthogonal Clifford algebra Cl(Q). See rmulm for more help.
- Clifford multiplication 'cmulQ' is based on the recursive use of the following formula:

$$xu = \text{wedge}(x, u) + \text{LCQ}(x, u) = x \,\&w\, u + \text{LCQ}(x, u)$$

- In the above, x is a 1-vector and u is any element in the algebra Cl(Q), wedge(x,u) = x &w u denotes the wedge or exterior product between x and u, and LCQ(x, u) denotes the left contraction of u by x. See LCQ for more help.
- Through this formula, the Clifford algebra Cl(Q) becomes a subalgebra of an algebra of endomorphisms of an exterior algebra over V.
- This procedure is multilinear. The infix form is given by, e.g., cmulQ(e1,e2,e3) = e1 &cQ e2 &cQ e3, etc..
- When the bilinear form B is not diagonal, use the full version of Clifford multiplication called

'cmul' and its infix form '&c'. See cmul or &c for more help.

- To multiply matrices with entries in a Clifford algebra and apply 'cmulQ' to the matrix entries use rmulm.
- To speed up computations, set the global variable _prolevel to 'true'. To find out more, see help page on cliparse and `type/clipolynom`.
- It is possible to assign a new name or a matrix to B and use it instead of the symbol B. See Example 2 below.
- NOTE: When the index has been assigned a matrix, put double quotes around the name of the matrix when using the ampersand version `&cQ` of 'cmulQ'.

Examples:

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

$$false$$

Example 1: Some computations with 'cmul' and 'cmulQ' with B being purely symbolic:

```
> cmulQ(3*Pi*e2we1we3,e4+3*e5,e6);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$-3\,\pi\, e1we2we3we4we6 - 9\,\pi\, e1we2we3we5we6$$

```
> cmul(ei,ej,ek);cmulQ(ei,ej,ek);
  cmul[K](ei,ej,ek);cmulQ[K](ei,ej,ek);
  cmul[-K](ei,ej,ek);cmulQ[-K](ei,ej,ek);
```

$$B_{j,k}\, ei - B_{i,k}\, ej + B_{i,j}\, ek + eiwejwek$$

$$B_{j,k}\, ei - B_{i,k}\, ej + B_{i,j}\, ek + eiwejwek$$

$$K_{j,k}\, ei - K_{i,k}\, ej + K_{i,j}\, ek + eiwejwek$$

$$K_{j,k}\, ei - K_{i,k}\, ej + K_{i,j}\, ek + eiwejwek$$

$$-K_{j,k}\, ei + K_{i,k}\, ej - K_{i,j}\, ek + eiwejwek$$

$$-K_{j,k}\, ei + K_{i,k}\, ej - K_{i,j}\, ek + eiwejwek$$

```
> cmul(e1,e2,e3);cmulQ(e1,e2,e3);
```

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

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

```
> cmulQ[K](4+e1,eiwej+4*alpha*Pi*Id);
  cmulQ[-K](4+e1,eiwej+4*alpha*Pi*Id);
```

$$-K_{1,j}\, ei + 16\,\alpha\,\pi\, Id + 4\,\alpha\,\pi\, e1 + e1weiwej + K_{1,i}\, ej + 4\, eiwej$$

$$K_{1,j}\, ei + 16\,\alpha\,\pi\, Id + 4\,\alpha\,\pi\, e1 + e1weiwej - K_{1,i}\, ej + 4\, eiwej$$

```
> cmulQ(ei,ej)+cmulQ(ej,ei);clicollect(%);
```

$$B_{i,j}\, Id + B_{j,i}\, Id$$

$$(B_{i,j} + B_{j,i})\, Id$$

```
> e1 &cQ e2 + e2 &cQ e1;
```

$$B_{1,2}\, Id + B_{2,1}\, Id$$

```
> e1 &cQ e1; e1 &cQ e2we3;
```

$$B_{1,1}\, Id$$

$$-B_{1,3}\, e2 + e1we2we3 + B_{1,2}\, e3$$

```
> &c[K](e1,e2);&cQ[K](e1,e2);
  &c[-K](e1,e2);&cQ[-K](e1,e2);
```

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

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

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

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

Example 2: The same computations as in Example 1 except that instead of B we use a new name K:

```
> cmulQ[K](3*Pi*e2we1we3,e4+3*e5,e6);
```

$$\begin{aligned}
&-9\,\pi\, e1we2we3we5we6 - 3\,\pi\, e1we2we3we4we6 \\
&\quad + 3\,\pi\,(K_{2,4}\,K_{3,6} - K_{3,4}\,K_{2,6} + 3\,K_{2,5}\,K_{3,6} - 3\,K_{3,5}\,K_{2,6})\, e1 \\
&\quad + 3\,\pi\,(3\,K_{3,5}\,K_{1,6} - 3\,K_{1,5}\,K_{3,6} + K_{3,4}\,K_{1,6} - K_{1,4}\,K_{3,6})\, e2 - 3\,\pi\,(K_{1,4} + 3\,K_{1,5})\, e2we3we6 \\
&\quad - 3\,\pi\,(K_{3,4} + 3\,K_{3,5})\, e1we2we6 - 3\,\pi\,(K_{4,6} + 3\,K_{5,6})\, e1we2we3 + 3\,\pi\,K_{3,6}\, e1we2we4 \\
&\quad - 3\,\pi\,K_{2,6}\, e1we3we4 + 3\,\pi\,K_{1,6}\, e2we3we4 + 9\,\pi\,K_{3,6}\, e1we2we5 - 9\,\pi\,K_{2,6}\, e1we3we5 \\
&\quad + 9\,\pi\,K_{1,6}\, e2we3we5 + 3\,\pi\,(K_{2,4} + 3\,K_{2,5})\, e1we3we6 \\
&\quad - 3\,\pi\,(3\,K_{2,5}\,K_{1,6} + K_{2,4}\,K_{1,6} - 3\,K_{1,5}\,K_{2,6} - K_{1,4}\,K_{2,6})\, e3
\end{aligned}$$

```
> cmul[K](ei,ej,ek);cmulQ[K](ei,ej,ek);
  &c[K](ei,ej,ek);&cQ[K](ei,ej,ek);
```

$$K_{j,k}\, ei - K_{i,k}\, ej + K_{i,j}\, ek + eiwejwek$$

$$K_{j,k}\, ei - K_{i,k}\, ej + K_{i,j}\, ek + eiwejwek$$

$$K_{j,k}\, ei - K_{i,k}\, ej + K_{i,j}\, ek + eiwejwek$$

$$K_{j,k}\, ei - K_{i,k}\, ej + K_{i,j}\, ek + eiwejwek$$

```
> cmul[K](e1,e2,e3);cmulQ[K](e1,e2,e3);
  &c[K](e1,e2,e3);&cQ[K](e1,e2,e3);
```

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

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

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

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

```
> cmulQ[-K](4+e1,eiwej+4*alpha*Pi*Id);
  &cQ[-K](4+e1,eiwej+4*alpha*Pi*Id);
```

$$K_{1,j}\, ei + 16\, \alpha\, \pi\, Id + 4\, \alpha\, \pi\, e1 + e1weiwej - K_{1,i}\, ej + 4\, eiwej$$

$$K_{1,j}\, ei + 16\, \alpha\, \pi\, Id + 4\, \alpha\, \pi\, e1 + e1weiwej - K_{1,i}\, ej + 4\, eiwej$$

```
> cmulQ(ei,ej)+cmulQ(ej,ei);&cQ(ei,ej)+&cQ(ej,ei);clicollect(%);
```

$$B_{i,j}\, Id + B_{j,i}\, Id$$

$$B_{i,j}\, Id + B_{j,i}\, Id$$

$$(B_{i,j} + B_{j,i})\, Id$$

Example 3: If B is assigned a matrix, 'cmulQ' computes only with diagonal entries of the matrix: whereas 'cmul' computes with all entries.

```
> B:=matrix(2,2,[a,b,-b,c]);
```

$$B := \begin{bmatrix} a & b \\ -b & c \end{bmatrix}$$

```
> cmulQ(e1,e2+e1);
  &cQ[B](e1,e2+e1); ##<-- Error message
```

$$(b + a)\, Id + e1we2$$

```
Error, (in Cliplus:-climul) invalid input: Cliplus:-clieval[array] expects its
1st argument, a, to be of type algebraic, but received 1 .. 2
```

```
> &cQ['B'](e1,e2+e1); ##<-- Error message
Error, (in Cliplus:-climul) invalid input: Cliplus:-clieval[array] expects its
1st argument, a, to be of type algebraic, but received 1 .. 2
```

```
> &cQ[''B''](e1,e2+e1); ## Correct answer
```

$$(b + a)\, Id + e1we2$$

```
> cmul[B](e1,e2+e1);
  &c[''B''](e1,e2+e1);
```

$$(b + a)\, Id + e1we2$$

$$(b + a)\, Id + e1we2$$

```
>
```

See Also: Clifford:-cmul, Clifford:-rmulm, Clifford:-`type/clipolynom`, Clifford:-`type/climon`, Clifford:-`type/clibasmon`, Clifford:-`type/cliscalar`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-cmulRS - Clifford product for two Grassmann basis monomials and (arbitrary but preferably) symbolic bilinear form

Calling Sequence:

c3 := cmulRS(c1,c2,B)

Parameters:

- c1, c2- Clifford monomials (elements of the type: `type/clibasmon`)
- B - bilinear form of type `name`, `symbol`, `matrix`, `array` or `&*`(numeric,{name,symbol,array,matrix})

Output:

- c3 : a Clifford polynom

Description:

- Since the Clifford product provides the main functionality of the CLIFFORD package, the best available mathematics has been incorporated in its code. The user normally does not use the internal functions cmulRS and cmulNUM but the wrapper function `&c`[K](arg1,arg2,...) which allows one to pass the name of a bilinear form K. The wrapper function can also act on any number of arguments of type clipolynom (since the Clifford product is associative this makes sense) and on a much wider class of types including Clifford matrices of type climatrix or Clifford polynomials in other bases, see Cliplus.
- There is a facility to use one of the internal functions (cmulRS or cmulNUM) when knowledge of the bilinear form allows one to decide which procedure might yield better performance. Moreover, the user can supply a `product` function (not necessarily a Clifford product) acting on two clibasmons, that is, monomials of type clibasmon. The wrapper uses the appropriate function, cmulRS, cmulNUM, or other, which has been selected by the user via a special function Clifford:-useproduct.
- There are two internal Clifford multiplications which are appropriate for different purposes. While cmulNUM is fast on sparse numeric matrices and on numeric matrices in general for dimensions dim_V >= 5, cmulRS was designed for symbolic calculations. Since cmulRS computes reasonably well in the numeric sparse case up to dim_V = 5, it was chosen as the default product of the package. Both internal multiplication procedures, cmulRS and cmulNUM, take two Clifford monomials of type clibasmon as input together with a third argument of type name/symbol/matrix/array which represents the chosen bilinear form.
- The evaluation of Clifford products in a Grassmann basis is quite involved and normally is done by a recursive process that uses Chevalley deformation, see cmulNUM. However, during the recursive evaluation many unnecessary terms are calculated which later cancel out at the next

recursive call. This feature, while being beneficial when the bilinear form is sparse numeric, since it cuts out many branches of the recursion quite early, prevents fast evaluation in the symbolic case where in general all terms might be non-zero. From this observation, the two possibilities to evaluate the Clifford product have emerged.

- To see how to define a new "Clifford product", go to cmul_user_defined for some help.
- cmulRS is computed using non-iterative Rota-Stein cliffordization, see [1,2] and the help pages of the Bigebra package for further literature. The cliffordization process is based on Hopf algebra theory. The Clifford product is obtained from the Grassmann wedge product and its Grassmann co-product, see Bigebra:-`&gco` , as follows (in tangle notation):

```
Δ |   | Δ
 / \_/ \
 \  B  /
  \   /
   \ /
    | Λ
```

where Λ is the Grassmann exterior wedge product and Δ is the Grassmann exterior co-product, which is obtained from the wedge product by categorical duality, i.e. to every algebra over a linear space V having a product we find a co-algebra having a co-product over the same space by reversing all arrows in all axiomatic commutative diagrams. Note that the co-product splits a single (input) 'factor' x into a tensor product (sum of ordered pairs x_(1)i, x_(2)i) of (output) factors. The main feature is that every such pair multiplies back to the input if the dual operation of multiplication is applied, i.e. x_(1)i /\ x_(2)i = x for each pair i. The 'cup' like part of the tangle decorated with B is the bilinear form B extended to the whole Grassmann algebra, i.e. the map B : /\V x /\ V -> k, with B : V x V -> k as B(x,y) on vectors. Hence, cmulRS computes on x and y for the given B, later extended to polynomials by bilinearity, as follows:

$$\text{cmulRS(x,y,B)} = \sum_{i=1}^{n} \sum_{j=1}^{n} \text{(+/-) x\_(1)i /\ y\_(2)j * B(x\_(2)i , y\_(1)j)}$$

where n is a dummy describing the cardinality of the required split and the sign is due to the needed permutations to arrange the factors.

- The (simplified some further efforts to speed it up are employed in the package) algorithm of cmulRS is as follows:

```
cmulRS(x,y,B)  [x,y two Grassmann monomials, B the name,... of a bilinear form]
  begin
  lstx <- list of indices from x
  lsty <- list of indices from y
  NX <- |lstx|
```

```
  NY <- |lsty|
  funx <- a function which maps the integers 1..NX onto the elements of lstx keeping their
order
  funy <- a function which maps the integers 1..NY onto the elements of lsty keeping their
order
  ## this will allow to calculate with arbitrary indices and to compute the necessary signs
with some ease
  psetx <- power set of {1..NX} # actually a list in a certain order
                                # the ith and (2^NX+1-i)th element are disjoint adding up to the set
{1..NX}
  psety <- power set of {1..NY} # actually a list in a certain order
                                # the ith and (2^NY+1-i)th element are disjoint adding up to the set
{1..NY}
  ## for faster computation we sort this power sets by grade
  ## we compute the sign for any term in the power set (that's tricky)
  psetx <- sort psetx by grade
  psety <- sort psety by grade
  pSgnx <- sum_(i in psetx) (-1)^sum_(j in psetx[i]) (psetx[i][j]-j)
  pSgny <- sum_(i in psety) (-1)^sum_(j in psety[i]) (psety[i][j]-j)
  ## now we need a subroutine for the cup tangle computing the bilinear form
  cup(x,y,B)
    begin
    if |x| <> |y| then RETURN(0) fi
    if |x| = 0 then RETURN(1) fi
    if |x| = 1 then RETURN(B[x[1],y[1]]) fi
    RETURN(  sum_(j in 1..|x|)
               (-1)^(j-1)*B(x[1],y[j])*
               *cup(x[2..-1],y/y[j],B)    )
    end cup
  ## now we compute the double sum, to gain efficiency we do this grade wise
  ## note that there are r over NX r-vectors in psetx, analogous for psety
  max_grade <- |lstx <- convert_to_set union lsty <- convert_to_set|
  res <- 0, pos1 <- 0
  for j from 0 to NX       # iterate over all j-vectors of psetx
    begin
    F1 <- N1!/((N1-j)!*j!) # number of terms (N1 over j)
    pos2 <- 0
    for i from 0 to min(N2,max_grade-j) # iterate over all i-vectors of psety
                                  # which do not exceed max_grade (others are zero)
      begin
      F2 <- N2!/((N2-i)!*i!)          # number of terms (N2 over i)
      for n from 1 to F1              # for all j-vectors
        begin
        for m from 1 to F2            # for all i-vectors
```

```
        begin
        res <- res +
          pSgnx[pos1+n]*pSgny[pos2+m]*
          *cup(fun1(psetx[PN1-pos1-n]),fun2(psety[pos2+m]),lname)*
          makeclibasmon -> ([fun1 -> psetx[pos1+n],fun2 -> psety[PN2-pos2-m])])
        end
      end
    pos2 <- pos2+F2
    end
    pos1 <- pos1+F1;
    end
    reorder -> res                    # reorder map all basis elements into standard order
  end cmulRS
```

- It is clear from this algorithm that only such terms are considered which might be non-zero: If all B[i,j] are non-zero and different so that no cancellation takes place between them, all these terms will survive. The combinatorial power of the Hopf algebraic approach is clearly demonstrated with this algorithm and its behavior in benchmarks.
- **References:**
  [1] G.-C. Rota, J.A. Stein, Plethystic Hopf algebras, Proc. Natl. Acad. Sci. USA 91, 1994:13057-13061
  [2] R. Ablamowicz, B. Fauser, Effective algorithms for evaluating Clifford products, pre-print 2003

Examples:

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

We check for some special cases and use arbitrary bilinear forms, e.g. B, K, T ...

```
> cmulRS(0,e1we2,K);
  cmulRS(Id,Id,K);
  cmulRS(Id,ei,-K);
  cmulRS(ei,Id,K);
  cmulRS(e1,e2,-K);
  cmulRS(ei,ej,B);
  cmulRS(ei,ej,-B);
```

$$0$$

$$Id$$

$$ei$$

$$ei$$

$$e1we2 - K_{1,2}\, Id$$

$$eiwej + B_{i,j}\, Id$$

$$eiwej - B_{i,j}\, Id$$

Now let us go for more complicated monomials:

```
> cmulRS(e1we2,e2we3,B);
  cmulRS(e1we2,e2we3,-B);
  cmulRS(e1we2,e1we2we3,T);
  cmulRS(e1we2,e1we2we3,-T);
```

$$(B_{2,2}\, B_{1,3} - B_{2,3}\, B_{1,2})\, Id + B_{2,2}\, e1we3 - B_{2,3}\, e1we2 - B_{1,2}\, e2we3$$

$$(B_{2,2}\, B_{1,3} - B_{2,3}\, B_{1,2})\, Id - B_{2,2}\, e1we3 + B_{2,3}\, e1we2 + B_{1,2}\, e2we3$$

$$(T_{2,1}\, T_{1,2} - T_{2,2}\, T_{1,1})\, e3 - (T_{2,1}\, T_{1,3} - T_{2,3}\, T_{1,1})\, e2 + (T_{2,2}\, T_{1,3} - T_{2,3}\, T_{1,2})\, e1 + T_{2,1}\, e1we2we3 - T_{1,2}\, e1we2we3$$

$$(T_{2,1}\, T_{1,2} - T_{2,2}\, T_{1,1})\, e3 - (T_{2,1}\, T_{1,3} - T_{2,3}\, T_{1,1})\, e2 + (T_{2,2}\, T_{1,3} - T_{2,3}\, T_{1,2})\, e1 - T_{2,1}\, e1we2we3 + T_{1,2}\, e1we2we3$$

To perform some benchmarks, we need some random basis monomials:

```
> rd:=rand(0..7):
  rdmonom:=proc(N) makeclibasmon([op({seq(rd(),i=1..N)} minus {0}
  )]) end:
  rdmonom(rd()),rdmonom(rd()),rdmonom(rd());
```

$$e1we4we6,\ e1we2we3we4we5,\ e7$$

We iterate over MAX basis monomials and compute their mutual products. Since we expect cmulRS to be faster as cmulNUM in the symbolic case, we explore this case here, for a benchmark with a sparse numeric matrix B see Clifford:-cmulNUM .

```
> MAX:=5:
  monlist:=[seq(rdmonom(rd()),i=1..MAX)];
  tm:=time():
  for i from 1 to MAX do
  for j from 1 to MAX do
    cmulRS(monlist[i],monlist[j],B);
  end do:
  end do:
  tm:=time()-tm:
  printf("computation with cmulRS needed %f seconds",tm);
  tm:=time():
  for i from 1 to MAX do
  for j from 1 to MAX do
    cmulNUM(monlist[i],monlist[j],B);
  end do:
  end do:
  tm:=time()-tm:
  printf("computation with cmulNUM needed %f seconds",tm);
```

$$monlist := [e1we2we3we4we7,\ e2,\ e4,\ e1we4we5,\ e6we7]$$

```
computation with cmulRS needed 0.406000 seconds
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
computation with cmulNUM needed 2.047000 seconds
```

Hence we find the desired result. The relative factor of performance between these functions changes with the randomly chosen basis monomials and due to the garbage collection in Maple, etc., but we find roughly a factor 8 in this setting, which is reasonable.

Note that one can gain speed in the computation of Clifford products in the BIGEBRA package by using it in the Bigebra:-`&map` function. This is possible since the tensor product is multilinear and the entries are reduced to be only clibasmons, however **the tensor product has to be evaluated.**

```
> with(Bigebra):
Increase verbosity by infolevel[`function`]=val -- use online help > ?Bigebra[h
elp]
> t1:=&t(a*e1+e2,b*e1we2,e3);
```

$$t1 := a\,b\,\&t(e1, e1we2, e3) + b\,\&t(e2, e1we2, e3)$$

```
> &map(t1,1,cmulRS,B);
  &map(t1,2,cmulRS,B);
```

$$a\,b\,(B_{1,1}\,(e2\,\&t\,e3) - B_{1,2}\,(e1\,\&t\,e3)) + b\,(B_{2,1}\,(e2\,\&t\,e3) - B_{2,2}\,(e1\,\&t\,e3))$$

$$a\,b\,((e1\,\&t\,e1we2we3) + B_{2,3}\,(e1\,\&t\,e1) - B_{1,3}\,(e1\,\&t\,e2)) + b\,((e2\,\&t\,e1we2we3) + B_{2,3}\,(e2\,\&t\,e1) - B_{1,3}\,(e2\,\&t\,e2))$$

```
> ## ERROR: Note that prefactors are suppressed in unevaluated
  &t's !!!!!
  &map('`&t`'(a*e1,b*e2,e3),1,cmulRS,B);
  &map(&t(a*e1,b*e2,e3),1,cmulRS,B);
```

$$(e1we2\,\&t\,e3) + B_{1,2}\,(Id\,\&t\,e3)$$

$$a\,b\,((e1we2\,\&t\,e3) + B_{1,2}\,(Id\,\&t\,e3))$$

```
>
```

See Also: Clifford:-cmulNUM, Clifford:-`&c`, Clifford:-`type/clibasmon`, Bigebra:-`&map`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-cocycle - finds an element x in Cl(V,B) connecting two arguments p1 and p2

Calling Sequence:

cocycle(p1, p2);
cocycle(p2, p2, a3);
cocycle(p1, p2, a3, a4);

Parameters:

p1, p2 - elements in Cl(V,B) of type "clibasmon', 'climon', or 'clipolynom'
a3 - list of elements in Cl(V,B) of type "clibasmon', 'climon', or 'clipolynom' (optional)
a4 - argument of type 'symbol' (optional)

Description:

- Procedure 'cocycle' tries to find an element x in the smallest Clifford algebra such that cmul(x, p1) = cmul(p2, x) where p1 and p2 are the first two arguments of type 'clibasmon', 'climon', or 'clipolynom'.
- If only two arguments p1 and p2 are passed to the procedure, element x belongs to the Clifford algebra over the lowest dimension dim = max(maxindex(p1), maxindex(p2)).
- If three arguments are used with the third argument being a list of elements of type 'clibasmon', 'climon', or 'clipolynom', then x is searched from among the set generated additively and multiplicatively by p1, p2, and the elements in the third list a3.
- If the fourth argument a4 is used, then the third argument is expected to be a list of elements of type 'clibasmon', in which case the procedure searches for all x from that list that satisfy cmul(x, p1) = cmul(p2, x).

Examples:

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

Example 1: Let's try to find an element x in the Clifford algebra Cl(V,B) where dim(V) = 3, p1 = e3, and p2 = -e3, such that cmul(x, p1) = cmul(p2, x).

```
> p1:=e3:p2:=-e3:
> x:=cocycle(p1,p2);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$x := \left(\frac{1}{2}\_X_6 B_{1,3} - \frac{1}{2}\_X_6 B_{3,1} + \frac{1}{2}\_X_7 B_{2,3} - \frac{1}{2}\_X_7 B_{3,2}\right) Id + \_X_2\, e1 + \_X_3\, e2 - \frac{1}{2}\frac{(\_X_2 B_{1,3} + \_X_3 B_{2,3} + \_X_2 B_{3,1} + \_X_3 B_{3,2})\, e3}{B_{3,3}} + \_X_6\, e1we3 + \_X_7\, e2we3$$

```
> cmul(x,p1)-cmul(p2,x);
```

$$0$$

```
>
```

Example 2: Let's try to find in Cl(3,0) x that connects two polynomials p1 and p2:

```
> B:=linalg[diag](1,1,1):
> p1:=1+2*e1-e1we3;p2:=3*e2+e2we3;
```

$$p1 := 1 + 2\,e1 - e1we3$$

$$p2 := 3\,e2 + e2we3$$

```
> cocycle(p1,p2);
```

$$0$$

```
> cocycle(1+2*e1-e1we3,3*e2+e2we3, [e1we2+Id,e1we2we3,e3]);
```

$$0$$

```
> cocycle(1+2*e1-e1we3,1+2*e1-e1we3,
  [Id,e1we2,e1we2we3],'clibasmon');
```

$$[Id, e1we2we3]$$

The last result should not be a surprise since the center of Cl(3,0) is two dimensional.

```
>
```

See Also: Clifford:-clisolve, Clifford:-`type/clibasmon`, Clifford:-`type/climon`, Clifford:-`type/clipolynom`, Clifford:-cmul

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-commutingelements - find commuting basis monomials of square 1

Calling Sequence:

commutingelements(L);

Parameters:

L - list of basis monomials of type 'list(clibasmon)'

Description:

- Procedure 'commutingelements' finds, in the given list of Clifford basis monomials of the type 'clibasmon', commuting elements other than the identity element 'Id' with the square 1 mod Id. See `type/clibasmon` for more help.
- The procedure may be used in several ways.

- to create primitive idempotents in the given Clifford algebra,
- to check if the given idempotent is primitive (see `type/primitiveidemp` for more help),
- to factor the given idempotent into a product of simple idempotents (see factoridempotent for more help).

- In a canonical basis for Cl(Q) there are sets of k = q - RHnumber(q-p) basis monomials which will commute and have square 1 mod Id (these sets need not be mutually disjoint). Here [p,q] is the signature of the quadratic form Q and 'RHnumber' is the Radon-Hurwitz function (see RHnumber for more help).
- Each such set of k commuting basis monomials generates a group of order 2^k.
- Each of the commuting elements in the given set may be used to define a simple idempotent in Cl(Q). Any product of these k simple idempotents is then a *primitive* idempotent f in Cl(Q) needed to generate a minimal ideal in Cl(Q).
- If the list L is too small to find k such commuting elements in it, the procedure returns as many generators it can find. If none are found, an empty list is returned.
- For this procedure to work, the bilinear form B must be defined first.

Examples:

```
> restart:with(Clifford):
> clibasis:=cbasis(4):B:=linalg[diag](1,1,1,-1):
> L1:=commutingelements(clibasis);#find a set of commuting
  elements
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$L1 := [e1, e2we4]$$

```
> L2:=commutingelements(remove(member,clibasis,L1));#find another
  set
```

$$L2 := [e2, e1we4]$$

```
> L3:=commutingelements(remove(member,clibasis,[op(L1),op(L2)]));
  #find another set
```

$$L3 := [e3, e1we3we4]$$

We may now define three different primitive idempotents using the above commuting sets:

```
> f1:=cmulQ((1/2)*(1+e1),(1/2)*(1+e2we4));
```

$$f1 := \frac{Id}{4} + \frac{e1we2we4}{4} + \frac{e1}{4} + \frac{e2we4}{4}$$

```
> factoridempotent(f1); #factor f1 and write it as a Clifford
  product
```

$$Clifford{:-}cmulQ\left(\frac{Id}{2} + \frac{e1}{2}, \frac{Id}{2} + \frac{e2we4}{2}\right)$$

```
> f2:=cmulQ((1/2)*(1+e2),(1/2)*(1+e1we4));
```

$$f2 := \frac{Id}{4} + \frac{e1we4}{4} - \frac{e1we2we4}{4} + \frac{e2}{4}$$

```
> factoridempotent(f2); #factor f2 and write it as a Clifford
  product
```

$$Clifford{:-}cmulQ\left(\frac{Id}{2} + \frac{e2}{2}, \frac{Id}{2} + \frac{e1we4}{2}\right)$$

```
> f3:=cmulQ((1/2)*(1+e3),(1/2)*(1+e1we3we4));
```

$$f3 := \frac{Id}{4} - \frac{e1we4}{4} + \frac{e1we3we4}{4} + \frac{e3}{4}$$

```
> f33:=factoridempotent(f3); #factor f3 and write it as a
  Clifford product
```

$$f33 := Clifford{:-}cmulQ\left(\frac{Id}{2} + \frac{e3}{2}, \frac{Id}{2} - \frac{e1we4}{2}\right)$$

NOTE: although the above factorization of f3 does have different factors than f3, this is another factorization of f3 as seen next in the following verification:

```
> evalb(f33=f3);
```

$$true$$

```
> type(f1,primitiveidemp); #verify that f1 is primitive in
  Cl(3,1)
```

$$true$$

```
> type(f2,primitiveidemp); #verify that f2 is primitive in
  Cl(3,1)
```

$$true$$

```
> type(f3,primitiveidemp); #verify that f3 is primitive in
  Cl(3,1)
```

$$true$$

```
> f4:=(1/2)*(1+e1); #a non-primitive idempotent
```

$$f4 := \frac{1}{2} + \frac{e1}{2}$$

```
> type(f4,idempotent);
```

$$true$$

```
> type(f4,primitiveidemp);
```

$$false$$

The following lists either contain non commuting elements, or their elements do not square to 1.

```
> L4:=[e1we2,e1we4];
```

$$L4 := [e1we2, e1we4]$$

```
> commutingelements(L4);
```

$$[e1we4]$$

```
> L5:=[e1we2,e1we3];
```

$$L5 := [e1we2, e1we3]$$

```
> commutingelements(L5);
```

$$[\ ]$$

```
> L6:=[Id,e1,e2];
```

$$L6 := [Id, e1, e2]$$

```
> commutingelements(L6);
```

$$[e1]$$

```
>
```

See Also: Clifford:-factoridempotent, Clifford:-RHnumber, Clifford:-`type/primitiveidemp`, Clifford:-`type/idempotent`, Clifford:-RHnumber, Clifford:-cbasis

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-conjugation - conjugation anti-automorphism in a Clifford algebra Cl(B)

Calling Sequence:

conjugation(p);
conjugation(p,lname);

Parameters:

p - expression of the type 'cliscalar' or 'clipolynom' or 'matrix'
lname - (optional) argument of type name, symbol, matrix, array or `&*`(numeric,{name,symbol,array,matrix})

Description:

- Procedure 'conjugation' calculates conjugation in the Clifford algebra. Note that 'conjugation' is defined as a composition of 'reversion' and 'gradeinv'. See reversion and gradeinv for more help.
- When the antisymmetric part of B is not zero, 'conjugation' does not preserve the multilinear structure of the algebra because it mixes grades, i.e., it does not preserve the gradation of the exterior algebra.
- It is an automorphism of Cl(B) of order 2.
- Conjugation may also be applied to matrices with entries in a Clifford algebra in which case it is applied to each matrix entry.
- When optional argument is used, for example K, conjugation is performed in Cl(K).

Examples:

```
> restart:with(Clifford):
> conjugation((2+alpha)*Id);
```

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

$$2\,Id + \alpha\,Id$$

```
> conjugation(e1we2,B); #note the off-diagonal entries of B
  conjugation(e1we2,-B); #note the off-diagonal entries of B
```

$$B_{2,1}\,Id - B_{1,2}\,Id - e1we2$$

$$-B_{2,1}\,Id + B_{1,2}\,Id - e1we2$$

```
> conjugation(e3we2we1,K); #note the off-diagonal entries of B
  conjugation(e3we2we1,-K); #note the off-diagonal entries of B
```

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

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

```
> u:=4+Pi*e1we4+2*e1we2we3:v:=a*e1+3*e2we4+e3:
> uv:=cmul(u,v):
```

Conjugation is an anti-automorphism:

```
> evalb(expand(conjugation(uv))=expand(cmul(conjugation(v),conjug
```

```
ation(u))));
```

$$true$$

```
> conjugation(e1wei);
```

$$B_{i,1}\, Id - B_{1,i}\, Id - e1wei$$

```
> M:=linalg[matrix](2,2,[3+e2we3,3*e2,-4*a*e2we1,e9]);
```

$$M := \begin{bmatrix} 3 + e2we3 & 3\, e2 \\ -4\, a\, e2we1 & e9 \end{bmatrix}$$

```
> conjugation(M,B);
  conjugation(M,-B);
```

$$\begin{bmatrix} 3\, Id + B_{3,2}\, Id - B_{2,3}\, Id - e2we3 & -3\, e2 \\ -4\, a\, B_{1,2}\, Id + 4\, a\, B_{2,1}\, Id - 4\, a\, e1we2 & -e9 \end{bmatrix}$$

$$\begin{bmatrix} 3\, Id - B_{3,2}\, Id + B_{2,3}\, Id - e2we3 & -3\, e2 \\ 4\, a\, B_{1,2}\, Id - 4\, a\, B_{2,1}\, Id - 4\, a\, e1we2 & -e9 \end{bmatrix}$$

```
> B:=linalg[randmatrix](3,3);
```

$$B := \begin{bmatrix} -7 & 22 & -55 \\ -94 & 87 & -56 \\ 0 & -62 & 97 \end{bmatrix}$$

```
> conjugation(M,B);
  conjugation(M,-B);
```

$$\begin{bmatrix} -3\, Id - e2we3 & -3\, e2 \\ -464\, a\, Id - 4\, a\, e1we2 & -e9 \end{bmatrix}$$

$$\begin{bmatrix} 9\, Id - e2we3 & -3\, e2 \\ 464\, a\, Id - 4\, a\, e1we2 & -e9 \end{bmatrix}$$

```
>
```

See Also: Clifford:-reversion, Clifford:-gradeinv, Clifford:-`type/clipolynom`, Clifford:-`type/climon`, Clifford:-`type/clibasmon`, Clifford:-`type/cliscalar`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-ddfmatrix, Clifford:-cdfmatrix - decompose a double field matrix and create a double field matrix

Calling Sequence:

ddfmatrix(M) - decompose a matrix of type 'dfmatrix' into a list L of two square matrices,
cdfmatrix(L) - create a matrix M of type 'dfmatrix' from a list L containing two n x n matrices,

Parameters:

M - a matrix of `type/dfmatrix`
L - list containing two square matrices of the same size

Description:

- A matrix M is of `type/dfmatrix' if it is a square matrix whose entries are two element lists in a double field. These matrices arise when faithful matrix representations of semi-simple Clifford algebras are calculated. Use all_sigs to display signatures of semisimple Clifford algebras in dimensions 1 through 9.
- Faithful spinor representations of semisimple Clifford algebras have been precomputed. They are stored in a library file and can be retrieved with the procedure clidata.
- 'cdmatrix' is equivalent to procedure `convert/dfmatrix` except it is used differently.

Examples:

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

Let's display signatures of semi-simple Clifford algebras in dimensions 1 through 9 which are isomorphic to rings of matrices over double real field R+R:

```
> all_sigs(1..9,'real','semisimple');
```

$$[[0, 7], [1, 0], [1, 8], [2, 1], [3, 2], [4, 3], [5, 4], [9, 0]]$$

Let's display signatures of semi-simple Clifford algebras in dimensions 1 through 9 which are isomorphic to rings of matrices over double quaternionic field H+H:

```
> all_sigs(1..9,'quat','semisimple');
```

$$[[0, 3], [1, 4], [2, 5], [3, 6], [5, 0], [6, 1], [7, 2]]$$

There are no semisimple Clifford algebras that would be isomorphic to rings of matrices of double complex field C+C:

```
> all_sigs(1..9,'complex','semisimple');
```

$$[\ ]$$

More information about Clifford algebra Cl(Q) of the quadratic form of signature (2,1) can be found using procedure clidata as follows:

```
> clidata([2,1]);
```

$$\left[real, 2, semisimple, 'cmulQ'\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e2we3}{2}\right), [Id, e2], [Id], [Id, e2]\right]$$

Example 1: Let's view matrices m[i], i=1..3, representing 1-vectors {e1,e2,e3} in Cl(2,1) = Mat(2,2,R+R). These matrices are of type `type/dfmatrix` and have been precomputed. They can be displayed with the procedure matKrepr:

```
> pq:=[2,1];
  B:=diag(1$pq[1],-1$pq[2]);
```

$$pq := [2, 1]$$

$$B := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

```
> L:=matKrepr();
```

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

$$L := \left[e1 = \begin{bmatrix} [1, -1] & [0, 0] \\ [0, 0] & [-1, 1] \end{bmatrix}, e2 = \begin{bmatrix} [0, 0] & [1, 1] \\ [1, 1] & [0, 0] \end{bmatrix}, e3 = \begin{bmatrix} [0, 0] & [-1, -1] \\ [1, 1] & [0, 0] \end{bmatrix} \right]$$

Let's assign these matrices to m[i], i=1..3:

```
> for i from 1 to nops(L) do m[i]:=rhs(L[i]) od;
```

$$m_1 := \begin{bmatrix} [1, -1] & [0, 0] \\ [0, 0] & [-1, 1] \end{bmatrix}$$

$$m_2 := \begin{bmatrix} [0, 0] & [1, 1] \\ [1, 1] & [0, 0] \end{bmatrix}$$

$$m_3 := \begin{bmatrix} [0, 0] & [-1, -1] \\ [1, 1] & [0, 0] \end{bmatrix}$$

All matrices m[i], i=1..3, are of type 'dfmatrix':

```
> for i from 1 to 3 do type(m[i],dfmatrix) od;
```

$$true$$

$$true$$

$$true$$

We can now decompose each matrix into an ordered pair of two ordinary real matrices with the procedure 'ddfmatrix':

```
> L1:=ddfmatrix(m[1]);L2:=ddfmatrix(m[2]);L3:=ddfmatrix(m[3]);
```

$$L1 := \left[\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \right]$$

$$L2 := \left[\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \right]$$

$$L3 := \left[\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \right]$$

Then, we can create back matrices m[1], m[2], m[3] over R+R with the procedure 'cdfmatrix]:

```
> m1:=cdfmatrix(L1);m2:=cdfmatrix(L2);m3:=cdfmatrix(L3);
```

$$m1 := \begin{bmatrix} [1, -1] & [0, 0] \\ [0, 0] & [-1, 1] \end{bmatrix}$$

$$m2 := \begin{bmatrix} [0, 0] & [1, 1] \\ [1, 1] & [0, 0] \end{bmatrix}$$

$$m3 := \begin{bmatrix} [0, 0] & [-1, -1] \\ [1, 1] & [0, 0] \end{bmatrix}$$

or with the procedure `convert/dfmatrix`:

```
> convert(L1,dfmatrix);convert(L2,dfmatrix);convert(L3,dfmatrix);
```

$$\begin{bmatrix} [1, -1] & [0, 0] \\ [0, 0] & [-1, 1] \end{bmatrix}$$

$$\begin{bmatrix} [0, 0] & [1, 1] \\ [1, 1] & [0, 0] \end{bmatrix}$$

$$\begin{bmatrix} [0, 0] & [-1, -1] \\ [1, 1] & [0, 0] \end{bmatrix}$$

Example 2: Let's view matrices m[i], i=1..3, representing 1-vectors {e1,e2,e3} in Cl(0,3) = Mat(1,1,H+H). These matrices are of type `type/dfmatrix` and have been precomputed. They can be displayed with the procedure matKrepr:

```
> pq:=[0,3];
  B:=diag(1$pq[1],-1$pq[2]);
```

$$pq := [0, 3]$$

$$B := \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

```
> L:=matKrepr();
```

$$L := [e1 = [[e1, e1]], e2 = [[e2, e2]], e3 = [[-e1we2, e1we2]]]$$

Let's assign these matrices to m[i], i=1..3:

```
> for i from 1 to nops(L) do m[i]:=rhs(L[i]) od;
```

$$m_1 := [[e1, e1]]$$

$$m_2 := [[e2, e2]]$$

$$m_3 := [[-e1we2, e1we2]]$$

All matrices m[i], i=1..3, are of type 'dfmatrix':

```
> for i from 1 to nops(L) do type(m[i],dfmatrix) od;
```

$$true$$

$$true$$

$$true$$

```
> L1:=ddfmatrix(m[1]);L2:=ddfmatrix(m[2]);L3:=ddfmatrix(m[3]);
```

$$L1 := [[e1], [e1]]$$

$$L2 := [[e2], [e2]]$$

$$L3 := [[-e1we2], [e1we2]]$$

```
> m1:=cdfmatrix(L1);m2:=cdfmatrix(L2);m3:=cdfmatrix(L3);
```

$$m1 := [[e1, e1]]$$

$$m2 := [[e2, e2]]$$

$$m3 := [[-e1we2, e1we2]]$$

>

Algebraic operations on these matrices such as addition and multiplication can be performed with procedures Clifford:-adfmatrix and Clifford:-mdfmatrix.

See Also: Clifford:-`type/dfmatrix`, Clifford:-adfmatrix, Clifford:-mdfmatrix, Clifford:-spinorKrepr, Clifford:-matKrepr

Last modified: December 20, 2007, 2005, RA/BF.

Function: Clifford:-`type/dfmatrix`, Clifford:-`convert/dfmatrix`

Calling Sequence:

type(M,dfmatrix) - yields 'true' or 'false' depending whether M is a matrix over a double field or not
convert(L[m1,m2],dfmatrix) - converts a list of two matrices into a matrix of type 'dfmatrix'

Parameters:

M - a square matrix
m1, m2 - two square matrices of the same size

Description:

- A matrix M is of type 'dfmatrix' if it is a square matrix whose entries are two element lists. These matrices arise when faithful matrix representations of semi-simple Clifford algebras are calculated. Use all_sigs to display signatures of semisimple Clifford algebras in dimensions 1 through 9.
- Faithful spinor representations of semisimple Clifford algebras have been precomputed. They are stored in a library file and can be retrieved with the procedure clidata.
- Procedure `convert/dfmatrix` takes either a list with two square matrices and converts them into a double field matrix of type 'dfmatrix'. The reverse operation of decomposing a matrix over a double field into a pair of square matrices is accomplished by ddfmatrix.

Examples:

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

Let's display signatures of semi-simple Clifford algebras in dimensions 1 through 9 which are isomorphic to rings of matrices over double real field R+R:

```
> all_sigs(1..9,'real','semisimple');
```

$$[[0, 7], [1, 0], [1, 8], [2, 1], [3, 2], [4, 3], [5, 4], [9, 0]]$$

Let's display signatures of semi-simple Clifford algebras in dimensions 1 through 9 which are isomorphic to rings of matrices over double quaternionic field H+H:

```
> all_sigs(1..9,'quat','semisimple');
```

$$[[0, 3], [1, 4], [2, 5], [3, 6], [5, 0], [6, 1], [7, 2]]$$

There are no semisimple Clifford algebras that would be isomorphic to rings of matrices of double complex field C+C:

```
> all_sigs(1..9,'complex','semisimple');
```

$$[\ ]$$

More information about Clifford algebra Cl(Q) of the quadratic form of signature (2,1) can be found using procedure clidata as follows:

```
> clidata([2,1]);
```

$$\left[real, 2, semisimple, 'cmulQ'\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e2we3}{2}\right), [Id, e2], [Id], [Id, e2]\right]$$

Example 1: Let's view matrices m[i], i=1..3, representing 1-vectors {e1,e2,e3} in Cl(2,1) = Mat(2,2,R+R). These matrices are of type `type/dfmatrix` and have been precomputed. They can be displayed with the procedure matKrepr:

```
> pq:=[2,1];
  B:=diag(1$pq[1],-1$pq[2]);
```

$$pq := [2, 1]$$

$$B := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

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

$$L := \left[e1 = \begin{bmatrix} [1, -1] & [0, 0] \\ [0, 0] & [-1, 1] \end{bmatrix}, e2 = \begin{bmatrix} [0, 0] & [1, 1] \\ [1, 1] & [0, 0] \end{bmatrix}, e3 = \begin{bmatrix} [0, 0] & [-1, -1] \\ [1, 1] & [0, 0] \end{bmatrix}\right]$$

Let's assign these matrices to m[i], i=1..3:

```
> for i from 1 to nops(L) do m[i]:=rhs(L[i]) od;
```

$$m_1 := \begin{bmatrix} [1, -1] & [0, 0] \\ [0, 0] & [-1, 1] \end{bmatrix}$$

$$m_2 := \begin{bmatrix} [0, 0] & [1, 1] \\ [1, 1] & [0, 0] \end{bmatrix}$$

$$m_3 := \begin{bmatrix} [0, 0] & [-1, -1] \\ [1, 1] & [0, 0] \end{bmatrix}$$

All matrices m[i], i=1..5, are of type 'dfmatrix':

```
>
```

Matrices m1, m2, and m2 can be decomposed into a pair of matrices and the composed back as follows:

```
> ddfmatrix(m[1]);
```

$$\left[\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}\right]$$

```
> convert(%,dfmatrix);
```

$$\begin{bmatrix} [1, -1] & [0, 0] \\ [0, 0] & [-1, 1] \end{bmatrix}$$

All matrices m1, m2 ,m3 are of type 'dfmatrix':

```
> for i from 1 to 5 do type(m[i],dfmatrix) od;
```

true

true

true

false

false

Example 2: Let's view matrices m[i], i=1..3, representing 1-vectors {e1,e2,e3} in Cl(0,3) = Mat(1,1,H+H). These matrices are of type `type/dfmatrix` and have been precomputed. They can be displayed with the procedure matKrepr:

```
> pq:=[0,3];
  B:=diag(1$pq[1],-1$pq[2]);
```

$$pq := [0, 3]$$

$$B := \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

```
> L:=matKrepr();
```

$$L := [e1 = [[e1, e1]], e2 = [[e2, e2]], e3 = [[-e1we2, e1we2]]]$$

Let's assign these matrices to m[i], i=1..3:

```
> for i from 1 to nops(L) do m[i]:=rhs(L[i]) od;
```

$$m_1 := [[e1, e1]]$$

$$m_2 := [[e2, e2]]$$

$$m_3 := [[-e1we2, e1we2]]$$

All matrices m[i], i=1..3, are of type 'dfmatrix':

```
> for i from 1 to nops(L) do type(m[i],dfmatrix) od;
```

$$true$$

$$true$$

$$true$$

Algebraic operations on these matrices such as addition and multiplication can be performed with procedures Clifford:-adfmatrix and Clifford:-mdfmatrix.

```
>
```

See Also: Clifford:-ddfmatrix, Clifford:-cdfmatrix, Clifford:-adfmatrix, Clifford:-mdfmatrix, Clifford:-spinorKrepr, Clifford:-matKrepr

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-`type/diagmatrix` - define a new type: a diagonal matrix

Calling Sequence:

type(m, diagmatrix);

Parameters:

m - a name type 'matrix'

Description:

- The procedure checks if the matrix m is diagonal.
- The procedure returns 'true' or 'false' depending whether its argument is or is not of the type 'diagmatrix'.
- See also related types 'symmatrix' (`type/symmatrix`), 'antisymmatrix' (`type/antisymmatrix`), and 'climatrix' (`type/climatrix`).

Examples:

```
> restart:with(Clifford):
> B:=linalg[diag](1,1,-1);
```

$$B := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

```
> type(B,diagmatrix);
```

$$true$$

```
> type(B,symmatrix);
```

$$true$$

```
> type(B,antisymmatrix);
```

$$false$$

```
> type(B,climatrix);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$false$$

```
> B:=linalg[matrix](2,2,[a,b,-b,c]);
```

$$B := \begin{bmatrix} a & b \\ -b & c \end{bmatrix}$$

```
> type(B,diagmatrix);
```

$$false$$

```
> type(B,symmatrix);
```

$$false$$

```
> type(B,antisymmatrix);
```

$$false$$

```
> type(B,climatrix);
```

$$false$$

```
> BA:=linalg[matrix](2,2,[0,b,-b,0]);
```

$$BA := \begin{bmatrix} 0 & b \\ -b & 0 \end{bmatrix}$$

```
> type(BA,diagmatrix);
```

$$false$$

```
> type(BA,symmatrix);
```

$$false$$

```
> type(BA,antisymmatrix);
```

$$true$$

```
> type(BA,climatrix);
```

$$false$$

```
> cliB:=linalg[matrix](2,2,[1+e1,2*e1we2-e2we3,e4,e5]);
```

$$cliB := \begin{bmatrix} 1 + e1 & 2\, e1we2 - e2we3 \\ e4 & e5 \end{bmatrix}$$

```
> type(cliB,diagmatrix);
```

$$false$$

```
> type(cliB,symmatrix);
```

$$false$$

```
> type(cliB,antisymmatrix);
```

$$false$$

```
> type(cliB,climatrix);
```

$$true$$

```
>
```

See Also: Clifford:-`type/climatrix`, Clifford:-`type/symmatrix`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-diagonalize - diagonalize a symmetric matrix

Calling Sequence:

diagonalize(M);

Parameters:

M - symmetric matrix

Description:

- Procedure 'diagonalize' is intended to diagonalize a square symmetric matrix, that is, a matrix of `type/symmatrix`.
- This procedure can be used to find the diagonal form of the symmetric part of the bilinear form (matrix) B. Then, procedure can be used to find signature of the symmetric part of B.

Examples:

```
> restart:with(Clifford):
> B:=matrix(2,2,[0,1,1,0]);
```

$$B := \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

```
> diagonalize(B);
```

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

```
> Bsignature(diagonalize(B)); #signature of B
```

$$[1, 1]$$

Define a 3 x 3 matrix and diagonalize it:

```
> B:=matrix(3,3,[0,0,1/2,0,1/2,0,1/2,0,0]);
```

$$B := \begin{bmatrix} 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 \end{bmatrix}$$

```
> diagonalize(B); #diagonalized form of B
```

$$\begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{-1}{2} \end{bmatrix}$$

```
> Bsignature(diagonalize(B)); #signature of B
```

$$[2, 1]$$

This procedure can also diagonalize simple symbolic matrices:

```
> B:=matrix(2,2,[a,b,b,a]);
```

$$B := \begin{bmatrix} a & b \\ b & a \end{bmatrix}$$

```
> diagonalize(B);
```

$$\begin{bmatrix} b+a & 0 \\ 0 & -b+a \end{bmatrix}$$

```
>
```

See Also: Clifford:-Bsignature

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-displayid - display Clifford scalars with the unit 'Id'

Calling Sequence:

displayid(p);

Parameters:

p - an expression of type 'algebraic' or 'matrix'

Description:

- Procedure 'displayid' replaces a user-entered Clifford scalar with that scalar times the algebra unit element 'Id'.
- This procedure is used by some other procedures in the package to check user's input and to display 'Id' in the returned results.
- Since this procedure is used to check user's input, user may enter Clifford scalars without the unit 'Id'. See `type/cliscalar` for more help.
- This procedure may be also applied to matrices with entries in a Clifford algebra.

Examples:

```
> restart:with(Clifford):
> displayid(3);
```

$$3\ Id$$

```
> displayid(e1+2*Pi);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$e1 + 2\,\pi\,Id$$

```
> cmulQ(1+e1,2+e2);
```

$$(2 + B_{1,2})\,Id + e1we2 + 2\,e1 + e2$$

```
> p:=a+2*e2we3-(1/(a+b))*e2we3+e4/(c+d);
```

$$p := a + 2\,e2we3 - \frac{e2we3}{a+b} + \frac{e4}{c+d}$$

```
> displayid(p);
```

$$a\,Id + 2\,e2we3 - \frac{e2we3}{a+b} + \frac{e4}{c+d}$$

```
> p:=p/(a+b+c);
```

$$p := \frac{a + 2\,e2we3 - \frac{e2we3}{a+b} + \frac{e4}{c+d}}{a+b+c}$$

```
> displayid(p);
```

$$\frac{a\,Id}{a+b+c}+\frac{2\,e2we3}{a+b+c}-\frac{e2we3}{(a+b+c)(a+b)}+\frac{e4}{(a+b+c)(c+d)}$$

```
> M:=linalg[matrix](2,2,[2+e3we2,4-5*Id+e2we3,2,9]);
```

$$M := \begin{bmatrix} 2+e3we2 & 4-5\,Id+e2we3 \\ 2 & 9 \end{bmatrix}$$

```
> displayid(M);
```

$$\begin{bmatrix} 2\,Id+e3we2 & -Id+e2we3 \\ 2\,Id & 9\,Id \end{bmatrix}$$

```
> displayid(M);
```

$$\begin{bmatrix} 2\,Id+e3we2 & -Id+e2we3 \\ 2\,Id & 9\,Id \end{bmatrix}$$

```
> reorder(M);
```

$$\begin{bmatrix} 2-e2we3 & 4-5\,Id+e2we3 \\ 2 & 9 \end{bmatrix}$$

```
>
```

See Also: Clifford:-cbasis

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-`type/evenelement` - define a new type: an even element in Cl(B)

Calling Sequence:

type(p, evenelement);

Parameters:

p - an algebraic expression of type 'clipolynom'

Description:

- A *even* element in Cl(B) is a Clifford polynomial which contains only even grades, or which is invariant under grade involution in Cl(B) (see gradeinv for more help).
- It is well known that the even elements in Cl(B) form an even Clifford subalgebra. Recall that Grassmann basis for the even subalgebra of Cl(B) may be formed using cbasis with option 'even'.
- The procedure returns 'true' or 'false' depending whether the input p is or is not of the type 'evenelement'.

Examples:

```
> restart:with(Clifford):
> type(2+2*e1+e1we2,evenelement);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$false$$

```
> p1:=2*e2we1+3+Pi*B[2,3]*Id;p2:=a+2*e2we3;
```

$$p1 := 2\ e2we1 + 3 + \pi\ B_{2,\,3}\ Id$$

$$p2 := a + 2\ e2we3$$

```
> type(p1,evenelement),type(p2,evenelement);
```

$$true,\ true$$

```
> type(cmul(p1,p2),evenelement);
```

$$true$$

```
> p3:=e1+b*e2we3we4;type(p3,oddelement);
```

$$p3 := e1 + b\ e2we3we4$$

$$true$$

```
> type(cmul(p3,p3),oddelement);
```

$$false$$

```
> type(cmul(p3,p3),evenelement);
```

$$true$$

```
> type(cmul(p1,p3),oddelement);
```

$$true$$

```
>
```

See Also: Clifford:-`type/clipolynom`, Clifford:-gradeinv, Clifford:-`type/oddelement`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-extract - extract vector indices from a basis monomial

Calling Sequence:

extract(a1,a2);

Parameters:

a1 - an expression of the type 'cliscalar' or 'climon'
a2 - an optional string

Description:

- Procedure 'extract' extracts indices of a basis monomial and returns them as a list of strings. If necessary, the indices can be returned as a list of integers if a string 'integers' is entered as a second argument.
- Indices in the monomial may be integers, one-letter strings, or they could be mixed.
- Note that extract(Id) = extract(cliscalar) = extract('cliscalar'*Id) = [] results in an empty list: no vector indices are returned.
- Reverse of this procedure is makeclibasmon: procedure that creates a Grassmann monomial of type clibasmon from a list of indices. When the list is empty, the identity element Id of Cl(B) is returned. Procedure reorder can then be used to sort the indices and return an appropriate sign for the sorting permutation. See permsign for more information.

Examples:

```
> restart:with(Clifford):
> L:=extract(e1we2we3);
```

$$L := [1, 2, 3]$$

```
> map(type,%,symbol);
```

$$[true, true, true]$$

```
> makeclibasmon(L);
```

$$e1we2we3$$

```
> extract(e1we2we3,'integers');
```

$$[1, 2, 3]$$

```
> makeclibasmon(%);
```

$$e1we2we3$$

```
> extract(5*ejweiwek);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$[j, i, k]$$

```
> makeclibasmon(%);
```

$$ejweiwek$$

```
> reorder(%);
```

$$-eiwejwek$$

```
> extract(3*Pi);
  makeclibasmon(%);
```

$$[\ ]$$

$$Id$$

```
> extract(3*Id);
  makeclibasmon(%);
```

$$[\ ]$$

$$Id$$

```
> extract(e1wej);
  makeclibasmon(%);
```

$$[1, j]$$

$$e1wej$$

```
> extract(e1wej,'integers');
```

$$[1, j]$$

```
>
```

See Also: Clifford:-reorder, Clifford:-makeclibasmon, Clifford:-makealiases, Clifford:-clicollect, Clifford:-`type/climon`, Clifford:-`type/cliscalar`, Clifford:-`type/clibasmon`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-factoridempotent - factor an idempotent into a product of simple idempotents

Calling Sequence:

factoridempotent(e);

Parameters:

e - Clifford polynomial of type 'idempotent'

Description:

- Procedure 'factoridempotent' can factor the given idempotent e into a product of N elements of the type (1/2)*(Id+e[i]), i=1..N, where {e[i], i = 1..N} is a set of commuting basis monomials with square 1 mod Id in the standard (canonical) basis of Cl(Q).
- It is known that when N = q - RHnumber(q-p) then e is primitive. See `type/primitiveidemp` for more help.
- In a canonical basis for Cl(Q) there are sets of N = q - RHnumber(q-p) basis monomials which will commute and have square 1 mod Id (these sets need not be mutually disjoint). Here [p,q] is the signature of the quadratic form Q and 'RHnumber' is the Radon-Hurwitz function (see RHnumber for more help).
- Each such set of N commuting basis monomials generates a group of order 2^N.
- Each of the commuting elements in the given set may be used to define a simple idempotent in Cl(Q). Any product of these N simple idempotents is then a *primitive* idempotent f in Cl(Q) needed to generate a minimal ideal in Cl(Q).
- For this procedure to work, the bilinear form B must be defined as a matrix.

Examples:

```
> restart:with(Clifford):
> clibasis:=cbasis(4):B:=linalg[diag](1,1,-1,-1):
> L1:=commutingelements(clibasis);#find a set of commuting elements
```

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

$$L1 := [e1, e2we3]$$

```
> L2:=commutingelements(remove(member,clibasis,L1));#find another set
```

$$L2 := [e2, e1we3]$$

```
> L3:=commutingelements(remove(member,clibasis,[op(L1),op(L2)]));#find another set
```

$$L3 := [e1we4, e1we2we4]$$

We may now define three different primitive idempotents using the above commuting sets:

```
> f1:=cmulQ((1/2)*(1+e1),(1/2)*(1+e2we3));
```

$$f1 := \frac{Id}{4} + \frac{e2we3}{4} + \frac{e1}{4} + \frac{e1we2we3}{4}$$

```
> factoridempotent(f1); #factor f1 and write it as a Clifford
  product
```

$$Clifford{:}\text{-}cmulQ\left(\frac{Id}{2} + \frac{e1}{2}, \frac{Id}{2} + \frac{e2we3}{2}\right)$$

```
> f2:=cmulQ((1/2)*(1+e2),(1/2)*(1+e1we3));
```

$$f2 := \frac{Id}{4} + \frac{e1we3}{4} + \frac{e2}{4} - \frac{e1we2we3}{4}$$

```
> factoridempotent(f2); #factor f2 and write it as a Clifford
  product
```

$$Clifford{:}\text{-}cmulQ\left(\frac{Id}{2} + \frac{e2}{2}, \frac{Id}{2} + \frac{e1we3}{2}\right)$$

```
> f3:=cmulQ((1/2)*(1+e1we4),(1/2)*(1+e1we2we4));
```

$$f3 := \frac{Id}{4} + \frac{e1we4}{4} + \frac{e1we2we4}{4} - \frac{e2}{4}$$

```
> f33:=factoridempotent(f3); #factor f3 and write it as a
  Clifford product
```

$$f33 := Clifford{:}\text{-}cmulQ\left(\frac{Id}{2} - \frac{e2}{2}, \frac{Id}{2} + \frac{e1we4}{2}\right)$$

NOTE: although the above factorization of f3 does have different factors than f3, this is another factorization of f3 as seen next in the following verification:

```
> evalb(f33=f3);
```

$$true$$

```
> type(f1,primitiveidemp); #verify that f1 is primitive in
  Cl(2,2)
```

$$true$$

```
> type(f2,primitiveidemp); #verify that f2 is primitive in
  Cl(2,2)
```

$$true$$

```
> type(f3,primitiveidemp); #verify that f3 is primitive in
  Cl(2,2)
```

$$true$$

```
> f4:=(1/2)*(1+e1); #a non-primitive idempotent
```

$$f4 := \frac{1}{2} + \frac{e1}{2}$$

```
> type(f4,primitiveidemp); #verify that f4 is not primitive in
  Cl(2,2)
```

$$false$$

```
> factoridempotent(f4);
```

$$\frac{Id}{2} + \frac{e1}{2}$$

```
>
```

See Also: Clifford:-cbasis, Clifford:-commutingelements, Clifford:-RHnumber, Clifford:-`type/primitiveidemp`, Clifford:-`type/idempotent`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-`type/fieldelement` - define a new type: a basis Clifford monomial with square 1 or -1 mod f

Calling Sequence:

type(a, fieldelement);

Parameters:

a - an algebraic expression of type 'algebraic'

Description:

- This procedure checks which elements in a real basis for a minimal left Cl(Q)f or right fCl(Q) ideal square to 1 or -1 modulo the primitive idempotent f. By definition, these elements are of the type *'fieldelement'*.
- The primitive idempotent f is declared as a global variable in this procedure and it is expected to have been defined. Thus, letter 'f' is reserved for such an idempotent (see `type/idempotent` and `type/primitiveidemp`).
- Essentially this procedure calls 'squaremodf' (see squaremodf) with the arguments a and f which in turn computes the square of a modulo f. That square is either 1, -1 or 0 if the element is nilpotent (see `type/nilpotent` for more help).
- This procedure is used by 'Kfield' to find those basis elements in S and their generators which give a basis for the field K of the spinor space S (see Kfield for more help).
- The procedure returns 'true' or 'false' depending whether its argument is or is not of the type 'fieldelement'.

Examples:

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

Example 1: Let's check which elements in S=Cl(3,0)f are of the type 'fieldelement'. Notice that the field K for this signature is isomorphic to the complex numbers C:

```
> B:=linalg[diag](1,1,1):
> data:=clidata();
```

$$data := \left[complex, 2, simple, \frac{Id}{2} + \frac{e1}{2}, [Id, e2, e3, e2we3], [Id, e2we3], [Id, e2]\right]$$

```
> f:=data[4]; # a primitive idempotent in Cl(3,0)
```

$$f := \frac{Id}{2} + \frac{e1}{2}$$

```
> sbasis_left:=minimalideal(cbasis(3),f,'left');
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$sbasis\_left :=$

$$\left[\left[\frac{Id}{2}+\frac{e1}{2}, \frac{e2}{2}-\frac{e1we2}{2}, \frac{e3}{2}-\frac{e1we3}{2}, \frac{e2we3}{2}+\frac{e1we2we3}{2}\right], [Id, e2, e3, e2we3], left\right]$$

By using 'squaremodf' we may find which elements square to 1 or -1 modulo f in Cl(3,0)f:

```
> map(squaremodf,sbasis_left[1],f);
```

$$[1, 0, 0, -1]$$

Thus the first and the fourth element in sbasis_left[1] should be of the type 'fieldelement':

```
> map(type,sbasis_left[1],fieldelement);
```

$$[true, false, false, true]$$

The generators 'Id' and 'e2we3' of these two basis elements give a basis for K = C.

```
>
```

Example 2: Let's check which elements in S=Cl(1,3)f are of the type 'fieldelement'. Notice that the field K for this signature is isomorphic to the quaternion division ring H:

```
> B:=linalg[diag](1,-1,-1,-1):
> data:=clidata();
```

$$data := \left[quaternionic, 2, simple, \frac{Id}{2}+\frac{e1we4}{2}, [Id, e1, e2, e3, e1we2, e1we3, e2we3, e1we2we3], [Id, e2, e3, e2we3], [Id, e1]\right]$$

```
> f:=data[4];
```

$$f := \frac{Id}{2}+\frac{e1we4}{2}$$

```
> sbasis_left:=minimalideal(cbasis(4),f,'left');
```

$$sbasis\_left := \left[\left[\frac{Id}{2}+\frac{e1we4}{2}, \frac{e1}{2}+\frac{e4}{2}, \frac{e2}{2}-\frac{e1we2we4}{2}, \frac{e3}{2}-\frac{e1we3we4}{2}, \frac{e1we2}{2}-\frac{e2we4}{2}, \frac{e1we3}{2}-\frac{e3we4}{2}, \frac{e2we3}{2}+\frac{e1we2we3we4}{2}, \frac{e1we2we3}{2}+\frac{e2we3we4}{2}\right], [Id, e1, e2, e3, e1we2, e1we3, e2we3, e1we2we3], left\right]$$

By using 'squaremodf' we may find which elements square to 1 or -1 modulo f in Cl(1,3)f:

```
> map(squaremodf,sbasis_left[1],f);
```

$$[1, 0, -1, -1, 0, 0, -1, 0]$$

Thus the first, the third, the fourth, and the seventh element in sbasis_left[1] should be of the type 'fieldelement':

```
> map(type,sbasis_left[1],fieldelement);
```

$$[true, false, true, true, false, false, true, false]$$

Notice that the generators of these elements are 'Id', 'e2', 'e3', and 'e2we3', and that these generators are listed in clidata()[6] (or data[6]) above. Hence, [Id,e2,e3,e2we3] provide a basis for K = H as it can be verified with the procedure 'type/purequatbasis' which checks if the elements of a list are a basis for pure quaternions. See purequatbasis for more help.

```
> type([e2,e3,e2we3],purequatbasis);
```

$$true$$

```
>
```

Example 3: Let's check which elements in S=Cl(3,1)f are of the type 'fieldelement'. Notice that the field K for this signature is isomorphic to the reals R:

```
> B:=linalg[diag](1,1,1,-1):
> data:=clidata();
```

$$data := \left[real, 4, simple, 'Clifford{:}\text{-}cmulQ'\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e3we4}{2}\right), [Id, e2, e3, e2we3], [Id], [Id, e2, e3, e2we3]\right]$$

```
> f:=data[4];
```

$$f := Clifford{:}\text{-}cmulQ\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e3we4}{2}\right)$$

```
> sbasis_left:=minimalideal(cbasis(4),f,'left');
```

$$sbasis\_left := \left[\left[\frac{Id}{4}+\frac{e1}{4}+\frac{e3we4}{4}+\frac{e1we3we4}{4}, \frac{e2}{4}-\frac{e1we2}{4}+\frac{e2we3we4}{4}-\frac{e1we2we3we4}{4}, \frac{e3}{4}-\frac{e1we3}{4}+\frac{e4}{4}-\frac{e1we4}{4}, \frac{e2we3}{4}+\frac{e1we2we3}{4}+\frac{e2we4}{4}+\frac{e1we2we4}{4}\right], [Id, e2, e3, e2we3], left\right]$$

By using 'squaremodf' we may find which elements square to 1 or -1 modulo f:

```
> map(squaremodf,sbasis_left[1],f);
```

$$[1, 0, 0, 0]$$

Thus, only the first element in sbasis_left[1] should be of the type 'fieldelement':

```
> map(type,sbasis_left[1],fieldelement);
```

$$[true, false, false, false]$$

```
>
```

Example 4: Testing error messages:

```
> f:='f';
```

$$f := f$$

```
> map(type,sbasis_left[1],fieldelement);
Error, (in type/fieldelement) primitive idempotent f has not been assigned yet
> f:=e1we2;
```

$$f := e1we2$$

```
> map(type,sbasis_left[1],fieldelement);
Error, (in type/fieldelement) invalid input: type/primitiveidemp expects its
1st argument, f, to be of type idempotent, but received e1we2
> f:=data[4];
```

$$f := Clifford{:}\text{-}cmulQ\left(\frac{Id}{2} + \frac{e1}{2}, \frac{Id}{2} + \frac{e3we4}{2}\right)$$

```
> map(type,sbasis_left[1],fieldelement);
```

$$[true, false, false, false]$$

```
>
```

See Also: Clifford:-`type/primitiveidemp`, Clifford:-`type/idempotent`, Clifford:-`type/nilpotent`, Clifford:-Kfield, Clifford:-squaremodf

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-find1str - find locations of a string of length one in another string of length at least one

Calling Sequence:

find1str(string1,string2)

Parameters:

string1 - string of length one
string2 - string of length at least one

Description:

- The procedure 'find1str' returns a set of positions at which string of length one occurs in another string of length at least one. Returned set may be empty if the first string does not appear in the second string.
- This procedure is used by extract , ord , `type/clibasmon` and cliparse.

Examples:

```
> restart:with(Clifford):
  find1str(e,e1we2we3); find1str(w,e1we2);
```

$$\{1, 4, 7\}$$

$$\{3\}$$

```
> find1str(f,e1we3);
```

$$\{0\}$$

```
> find1str(f,g);
```

$$\{0\}$$

```
>
```

See Also: Clifford:-extract, Clifford:-ord, Clifford:-`type/climon`, Clifford:-`type/cliscalar`, Clifford:-`type/clibasmon`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-findbasis - find a basis in a linear subspace of a Clifford algebra

Calling Sequence:

findbasis(list1,list2);

Parameters:

list1 - a list of Clifford polynomials spanning a vector subspace in a Clifford algebra of type 'list(clipolynom)'
list2 - (optional) a list of basis monomials in the Clifford algebra

Description:

- Procedure 'findbasis' finds a basis in a linear vector space spanned by a set of Clifford polynomials entered as the first list.
- The procedure is used, for example, when finding a basis for a spinor space S considered as a minimal left or right ideal in Cl(B) generated by a primitive idempotent f.
- To speed up computations, it is advisable to enter a standard Clifford basis for Cl(B) in the form of a list of basis monomials as the second argument.
- If only one list is specified, 'findbasis' determines a suitable Clifford basis itself. It creates a Clifford basis by finding the largest index among the polynomials in the first list.

Examples:

```
> restart:with(Clifford):B:=linalg[diag](1,1,1):
> findbasis([2*e1+e2,e2+e1we2,e1we2],[Id,e1,e2,e1we2]);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$[2\,e1+e2,\ e2+e1we2,\ e1we2]$$

```
> findbasis([2*e1+e2,e2+e1we2,e1we2]);
```

$$[2\,e1+e2,\ e2+e1we2,\ e1we2]$$

```
> findbasis([2*e1,e2,e2+e1we2,e1we2+e1]);
```

$$[2\,e1,\ e2,\ e2+e1we2]$$

```
> findbasis([e1,Id,e2]);
```

$$[e1,\ Id,\ e2]$$

```
> findbasis([e1-e2,e1+e2,2*e1+e2]);
```

$$[e1-e2,\ e1+e2]$$

```
> findbasis([2*e1,e1]);
```

$$[2\,e1]$$

```
>
```

See Also: Clifford:-spinorKrepr, Clifford:-spinorKbasis, Clifford:-minimalideal, Clifford:-matKrepr, Clifford:-Kfield, Clifford:-`type/clipolynom`, Clifford:-`type/climon`, Clifford:-`type/clibasmon`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-`type/gencomplex` - define a new type: a generalized complex element in Cl(B)

Calling Sequence:

type(p, gencomplex);

Parameters:

p - an algebraic expression of type 'clibasmon', 'climon', 'clipolynom'

Description:

- A *generalized complex* element in Cl(B) is a Clifford polynomial which belongs to a subalgebra A of Cl(B) isomorphic to complex numbers C.
- In order for this procedure to work, the bilinear form B must be assigned since whether or not an element of Cl(B) is of this type depends on B.
- This procedure is used by 'cinv' (see cinv for more help).
- The procedure returns 'true' or 'false' depending whether the input p is or is not of the type 'gencomplex'.

Examples:

```
> restart:with(Clifford):
> type(2+2*e1+e1we2,gencomplex); #testing an error message
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
Error, (in type/gencomplex) can't check type since B is not assigned a matrix
```

```
> B:=linalg[diag](1,1,-1,-1):
> p:=2*Id-3*e1we2;
```

$$p := 2\,Id - 3\,e1we2$$

```
> type(p,gencomplex);
```

$$true$$

Notice that a generalized complex conjugate of p in A may be defined this way:

```
> gencc:=p->displayid(2*vectorpart(p,0)-p);
```

$$gencc := p \rightarrow Clifford\text{:-}displayid(2\,Clifford\text{:-}vectorpart(p, 0) - p)$$

```
> pc:=gencc(p);
```

$$pc := 2\,Id + 3\,e1we2$$

Generalized norm is then:

```
> gencnorm:=p->scalarpart(cmul(p,gencc(p)));
```

$$gencnorm := p \rightarrow Clifford\text{:-}scalarpart(\mathrm{climul}(p, \mathrm{gencc}(p)))$$

```
> gencnorm(p);
```

$$13$$

Thus, the inverse of p is then:

```
> gencinv:=p->gencc(p)/gencnorm(p);
```

$$gencinv := p \rightarrow \frac{\mathrm{gencc}(p)}{\mathrm{gencnorm}(p)}$$

```
> pinv:=gencinv(p);
```

$$pinv := \frac{2\,Id}{13} + \frac{3\,e1we2}{13}$$

```
> type(pinv,gencomplex);
```

$$true$$

Let's verify this:

```
> cmul(pinv,p),cmul(p,pinv);
```

$$Id, Id$$

```
>
```

See Also: Clifford:-cinv, Clifford:-`type/genquaternion`, Clifford:-`type/clipolynom`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-`type/genquatbasis` - procedure which checks if the given list or set gives basis elements of a skew field isomorphic to the ring of quaternions

Calling Sequence:

type(L,genquatbasis);

Parameters:

L - a list or a set of four algebraic expression of type 'clipolynom'

Description:

- Procedure *genquatbasis* checks if the given list or set of four different elements of type 'clipolynom' in Cl(B) gives a basis for a four dimensional subalgebra H' of Cl(B) isomorphic to the ring of quaternions H.
- Among the four elements of L one must be of type 'idempotent'.
- For example, elements of the type 'genquaternion' belong to such subalgebras. See `type/genquaternion` for more help.

Examples:

```
> restart:with(Clifford):
> type(2+2*e1+e1we2+e2,genquaternion); #testing an error message
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
Error, (in type/genquaternion) square matrix must be assigned to B
> B:=linalg[diag](1,-1,-1,-1):
> q:=2-3*e2+e3-e2we3;
```

$$q := 2 - 3\,e2 + e3 - e2we3$$

```
> type(q,genquaternion);
```

$$true$$

```
> S:=cliterms(q);
```

$$S := \{Id, e2we3, e2, e3\}$$

```
> type(S,genquatbasis);
```

$$true$$

```
>
```

See Also: Clifford:-`type/gencomplex`, Clifford:-cliterms

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-`type/genquaternion` - define a new type: a generalized quaternionic element in Cl(B)

Calling Sequence:

type(p, genquaternion);

Parameters:

p - an algebraic expression of type 'clipolynom'

Description:

- A *generalized quaternionic* element in Cl(B) is a Clifford polynomial which belongs to a subalgebra A of Cl(B) isomorphic to the division ring H of quaternions.
- In order for this procedure to work, the bilinear form B must be assigned since whether or not an element of Cl(B) is of this type depends on B.
- This procedure is used by 'cinv' (see cinv for more help).
- The procedure returns 'true' or 'false' depending whether the input p is or is not of the type 'genquaternion'.
- There is also another type available called 'quaternion'. Elements of this type belong to the even subalgebra of Cl(3). See `type/quaternion` for more help.

Examples:

```
> restart:with(Clifford):
> type(2+2*e1+e1we2+e2,genquaternion); #testing an error message
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
Error, (in type/genquaternion) square matrix must be assigned to B
> B:=linalg[diag](1,-1,-1,-1):
> q:=2-3*e2+e3-e2we3;
```

$$q := 2 - 3\,e2 + e3 - e2we3$$

```
> type(q,genquaternion);
```

$$true$$

Notice that a generalized quaternion conjugate of p in A may be defined this way:

```
> genqc:=p->displayid(2*vectorpart(p,0)-p);
```

$$genqc := p \rightarrow Clifford\text{:-}displayid(2\,Clifford\text{:-}vectorpart(p, 0) - p)$$

```
> qc:=genqc(q);
```

$$qc := 2\,Id + 3\,e2 - e3 + e2we3$$

Generalized norm is then:

```
> genqnorm:=p->scalarpart(displayid(cmul(p,genqc(p))));
```

$$genqnorm := p \rightarrow Clifford\text{:-}scalarpart(Clifford\text{:-}displayid(\mathrm{climul}(p, \mathrm{genqc}(p))))$$

```
> genqnorm(q);
```

$$15$$

Thus, the inverse of q is then:

```
> genqinv:=p->genqc(p)/genqnorm(p);
```

$$genqinv := p \rightarrow \frac{\mathrm{genqc}(p)}{\mathrm{genqnorm}(p)}$$

```
> qinv2:=genqinv(q);
```

$$qinv2 := \frac{2\,Id}{15} + \frac{e2}{5} - \frac{e3}{15} + \frac{e2we3}{15}$$

Let's verify this:

```
> cmul(qinv2,q),cmul(q,qinv2);
```

$$Id, Id$$

```
>
```

See Also: Clifford:-`type/quaternion`, Clifford:-`type/genquatbasis`, Clifford:-qinv, Clifford:-cinv, Clifford:-`type/gencomplex`, Clifford:-`type/clipolynom`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-gradeinv - grade involution in a Clifford algebra Cl(B)

Calling Sequence:

gradeinv(p);

Parameters:

p - expression of the type 'cliscalar', 'clibasmon', 'climon', or 'clipolynom' or 'matrix'

Description:

- Procedure 'gradeinv' is the grade involution in a Clifford algebra Cl(B), i.e., it reverses signs of odd elements and leaves signs of even elements unchanged.
- It is an automorphism of Cl(B) of order 2.
- This procedure is linear.
- Procedure 'gradeinv' may also be applied to matrices with entries in a Clifford algebra in which case it is applied to each matrix entry.

Examples:

```
> restart:with(Clifford):
> gradeinv(3*Pi*e2we1we3+4-e2we1);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$-3\,\pi\,e2we1we3 + 4\,Id - e2we1$$

```
> gradeinv(a*B[i,j]*ejwei+3*ekwei+el);
```

$$a\,B_{i,j}\,ejwei + 3\,ekwei - el$$

```
> gradeinv(4+e1+eiwej+2*e1we2we3+e1we2we3we4);
```

$$4\,Id - e1 + eiwej - 2\,e1we2we3 + e1we2we3we4$$

```
> gradeinv(4*alpha*Pi);
```

$$4\,\alpha\,\pi\,Id$$

```
> gradeinv(gradeinv(4*Id+8*e1-e2we3));
```

$$4\,Id + 8\,e1 - e2we3$$

```
> M:=linalg[matrix](2,2,[2+e2we3,3-e2we3we4,e4,e5-e6*Pi]);
```

$$M := \begin{bmatrix} 2 + e2we3 & 3 - e2we3we4 \\ e4 & e5 - e6\,\pi \end{bmatrix}$$

```
> gradeinv(M);
```

$$\begin{bmatrix} 2\,Id + e2we3 & 3\,Id + e2we3we4 \\ -e4 & -e5 + e6\,\pi \end{bmatrix}$$

```
>
```

See Also: Clifford:-`type/evenelement`, Clifford:-`type/oddelement`, Clifford:-conjugation, Clifford:-reversion, Clifford:-`type/clipolynom`, Clifford:-`type/climon`, Clifford:-`type/clibasmon`, Clifford:-`type/cliscalar`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-`type/idempotent` - define a new type: an idempotent element in a Clifford algebra

Calling Sequence:

type(u, idempotent);
type([u,lname],idempotent);

Parameters:

u - an algebraic expression of type 'cliscalar', 'clibasmon', 'climon', or 'clipolynom'
lname - (optional) argument entered as a second element in a list of type name, symbol, matrix, array, or `&*`(numeric,{name,symbol,array,matrix})

Description:

- Definition of type `idempotent`. When the first argument is just u, the procedure verifies whether or not u is idempotent in the Clifford algebra Cl(B).
- When the first argument is a list, e.g., [u,K] or [u,-K], then the procedure checks if u is idempotent in Cl(K) or Cl(-K), respectively.
- Recall that an element e of Cl(B) is an *idempotent* if e^2 = cmul(e,e) = e where 'cmul' denotes Clifford multiplication in Cl(B). See cmul for more help.
- It is expected that a matrix of the bilinear form B has been specified since, in general, whether or not an element u of Cl(B) is idempotent depends on B. Thus, when the bilinear form B is not assigned, an error message is returned.
- A related type is 'primitiveidemp' which is a primitive idempotent in the given algebra (see Clifford:-`type/primitiveidemp` for more help).
- Recall that an idempotent e is *primitive* if it cannot be written as a sum of two mutually annihilating idempotents. This means that e cannot be written as e' + e" where cmul(e',e') = e', cmul(e",e") = e", and cmul(e',e") = cmul(e",e') = 0.
- See also Radon-Hurwitz function 'RHnumber' (see RHnumber) and a procedure 'factoridempotent' (see factoridempotent).

Examples:

```
> restart:bench:=time():with(Clifford):B:=linalg[diag](1,1,1):
> clidata();
```

$$\left[complex, 2, simple, \frac{Id}{2}+\frac{e1}{2}, [Id, e2, e3, e2we3], [Id, e2we3], [Id, e2]\right]$$

```
> f:=eval(clidata()[4]):'f'=f;
```

$$f=\frac{Id}{2}+\frac{e1}{2}$$

```
> factoridempotent(f);
```

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

$$\frac{Id}{2}+\frac{e1}{2}$$

Testing implicitly in Cl(B):

```
> type((1/2)*(e1+e1we3),idempotent);
  type((1/2)*(1+e3),idempotent);
  type((1/2)*(Id+e1),idempotent);
```

$$false$$

$$true$$

$$true$$

Testing explicitly in Cl(B):

```
> type([(1/2)*(e1+e1we3),B],idempotent);
  type([(1/2)*(1+e3),B],idempotent);
  type([(1/2)*(Id+e1),B],idempotent);
```

$$false$$

$$true$$

$$true$$

Testing explicitly in Cl(-B):

```
> type([(1/2)*(e1+e1we3),-B],idempotent);
  type([(1/2)*(1+e3),-B],idempotent);
  type([(1/2)*(Id+e1),-B],idempotent);
```

$$false$$

$$false$$

$$false$$

```
> B:='B':  #unassigning B
> type((1/2)*(1+e2),idempotent); #testing an error message
Error, (in type/idempotent) matrix must be assigned to B
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 0.792000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

See Also: Clifford:-`type/primitiveidemp`, Clifford:-factoridempotent, Clifford:-commutingelements, Clifford:-RHnumber, Clifford:-`type/nilpotent`, Clifford:-`type/clipolynom`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-isproduct - determine whether the given Clifford polynomial is a product of 1-vectors in Cl(Q)

Calling Sequence:

isproduct(p);

isproduct(p,s);

Parameters:

p - expression of the type 'clipolynom'
s - (optional) symbol expected to be either 'all' or 'any'

Description:

- Procedure 'isproduct' can determine whether the given Clifford polynomial p can be written as a Clifford product of 1-vectors.
- If p can be factored into a product of 1-vectors then the factorization is not unique.
- Recall that if p is factorable, and if p is invertible then p belongs to the *Lipschitz group* (formerly called the *Clifford group*).
- In order to factor p, the procedure sets up an appropriate system of equations. In order to simplify the problem as much as possible, first it tries to find common 1-vector factors in all monomial terms of p. Then, it sets the system of equations and tries to solve is. If no solution is found, 'false' is returned.
- When used without the optional symbol 'all' or 'any', the procedure returns 'true' or 'false' depending whether p can be factored or not.
- When used with the string 'any', it will return 'false' if p is not factorable. If p is factorable, it will return a list consisting of two elements: the first element is 'true' while the second is a list of 1-vectors whose Clifford product gives p. Remember that these vectors are not unique.
- If the procedure is unable to find specific vectors, it will return general vectors.
- Notice that repeated use of the procedure with the string 'any' usually results in a different particular solution to the factorization problem (provided that p is factorable). This is because random substitutions are made by the procedure to find specific solutions.
- When used with the string 'all', it will return 'false' if is not factorable. If p is factorable, it will return a list consisting of two elements: the first element is 'true' while the second is a list of general 1-vectors whose Clifford product gives p. In some obvious cases, e.g., p = 2*e1we2we3, it will return vectors [2*e1,e2,e3] rather than a general solution.
- It assumed that the bilinear form B is in fact assigned a diagonal matrix.
- Notice that when the option 'all' is used, the general solution returned contains usually a few free parameters like x12, x24, etc. Values for these parameters may be of course chosen by the user.

- See also procedure isVahlenmatrix for more use of 'isproduct'.
- To speed up computations, set global variable _prolevel to true.

Examples:

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

Example 1: Simple cases:

```
> B:=linalg[diag](1,1,-1,-1,1,-1,1,1):
> isproduct(e1+e2we4,'any');
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$false$$

```
> isproduct(-2*Id,'all');
```

$$[true, [-2\, e1, e1]]$$

```
> isproduct(2*Pi*e1we2we3,'any');
```

$$[true, [2\, \pi\, e1, e2, e3]]$$

```
> isproduct(2*e1+3*e4,'all');
```

$$[true, [2\, e1 + 3\, e4]]$$

```
> isproduct(2*e1we2we3,'all');
```

$$[true, [2\, e1, e2, e3]]$$

```
> isproduct(Id+e1we4,'any');
```

$$\left[true, \left[-\frac{4\, e1}{3} - \frac{e4}{3}, -e1 - e4\right]\right]$$

```
> L:=%:cmul(L[2][1],L[2][2]);
```

$$Id + e1we4$$

Example 2: Let's try to see if elements of higher grades are products of one vectors:

```
> p:=cmul(e1+e4,e2-e3); #maximum grade of 2
```

$$p := -e1we3 + e1we2 - e2we4 + e3we4$$

```
> L:=isproduct(p,'all');
```

$$L := \left[true, \left[x14\, e1 - \frac{(1 + x14\, x23)\, e2}{x24} + \frac{(1 + x14\, x23)\, e3}{x24} + x14\, e4, x24\, e1 - x23\, e2 + x23\, e3 + x24\, e4\right]\right]$$

```
> for i from 1 to maxgrade(p) do v||i:=L[2][i] od;
```

$$v1 := x14\, e1 - \frac{(1 + x14\, x23)\, e2}{x24} + \frac{(1 + x14\, x23)\, e3}{x24} + x14\, e4$$

$$v2 := x24\, e1 - x23\, e2 + x23\, e3 + x24\, e4$$

```
> simplify(cmul(v||(1..maxgrade(p)))-p);
```

$$0$$

```
> L:=[2*e1we2we3,(a+b)*e1we2we3we4,-Pi*e1we2we3we4we5];
```

$$L := [2\, e1we2we3, (a+b)\, e1we2we3we4, -\pi\, e1we2we3we4we5]$$

```
> map(isproduct,L,'any');
```

$$[[true, [2\, e1, e2, e3]], [true, [(a+b)\, e1, e2, e3, e4]], [true, [-\pi\, e1, e2, e3, e4, e5]]]$$

```
> p:=cmul(e1+e4,e2-e3,e1-e3); #maximum grade of 3
```

$$p := -e1 - e2 - e4 + e3 - e1we2we3 - e1we2we4 + e1we3we4 - e2we3we4$$

```
> L:=isproduct(p,'any');
```

$$L := \left[true, \left[\frac{7\, e1}{2} + e2 - e3 + \frac{7\, e4}{2}, \frac{e1}{2} + e2 - e3 + \frac{e4}{2}, \frac{e1}{3} - \frac{e2}{3}\right]\right]$$

```
> for i from 1 to maxgrade(p) do v||i:=L[2][i] od;
```

$$v1 := \frac{7\, e1}{2} + e2 - e3 + \frac{7\, e4}{2}$$

$$v2 := \frac{e1}{2} + e2 - e3 + \frac{e4}{2}$$

$$v3 := \frac{e1}{3} - \frac{e2}{3}$$

```
> simplify(cmul(v||(1..maxgrade(p)))-p);
```

$$0$$

Example 3: Polynomials of higher grades can also be handled, but they will take more time. In special cases when common factors can be found, computations will be done faster..

```
> p:=cmul(e1+e4,e2,e3,2*e4,e5);
```

$$p := 2\, e1we2we3we4we5 - 2\, e2we3we5$$

```
> L:=isproduct(p,'any');
```

$$L := \left[true, \left[\frac{5\, e1}{2} - \frac{e4}{2}, -e1 + e4, e2, e3, e5\right]\right]$$

```
> for i from 1 to maxgrade(p) do v||i:=L[2][i] od;
```

$$v1 := \frac{5\, e1}{2} - \frac{e4}{2}$$

$$v2 := -e1 + e4$$

$$v3 := e2$$

$$v4 := e3$$

$$v5 := e5$$

```
> simplify(cmul(v||(1..maxgrade(p)))-p);
```

$$0$$

```
> p:=cmul(e1+e4,e2,e3,2*e4,e5+e4,e6);
```

$$p := -2\, e2we3we5we6 - 2\, e1we2we3we6 - 2\, e2we3we4we6 + 2\, e1we2we3we4we5we6$$

```
> L:=isproduct(p,'any');
```

$$L := \left[true, \left[-36\, e1 - 36\, e4, \frac{8\, e1}{9} + e4 - \frac{e5}{3}, \frac{e1}{2} + \frac{e4}{2} - \frac{e5}{2}, e2, e3, e6 \right] \right]$$

```
> for i from 1 to maxgrade(p) do v||i:=L[2][i] od;
```

$$v1 := -36\, e1 - 36\, e4$$

$$v2 := \frac{8\, e1}{9} + e4 - \frac{e5}{3}$$

$$v3 := \frac{e1}{2} + \frac{e4}{2} - \frac{e5}{2}$$

$$v4 := e2$$

$$v5 := e3$$

$$v6 := e6$$

```
> simplify(cmul(v||(1..maxgrade(p)))-p);
```

$$0$$

```
>
```

Example 4: Special cases:

```
> p:=cmul(e1,e2,e3,2*e4,e7,e8+e7);
```

$$p := 2\, e1we2we3we4we7we8 + 2\, e1we2we3we4$$

```
> L:=isproduct(p,'any');
```

$$L := [\, true, [\, 2\, e8, -e7 + e8, e1, e2, e3, e4 \,] \,]$$

```
> for i from 1 to maxgrade(p) do v||i:=L[2][i] od;
```

$$v1 := 2\, e8$$

$$v2 := -e7 + e8$$

$$v3 := e1$$

$$v4 := e2$$

$$v5 := e3$$

$$v6 := e4$$

```
> simplify(cmul(v||(1..maxgrade(p)))-p);
```

$$0$$

```
> B:=linalg[diag](1,1,1,-1):
> p:=2*e1we2we3;
```

$$p := 2\, e1we2we3$$

```
> isproduct(p);
```

$$true$$

```
> isproduct(p,'any');
```

$$[\, true, [\, 2\, e1, e2, e3 \,] \,]$$

```
> isproduct(p,'all');
```

$$[\, true, [\, 2\, e1, e2, e3 \,] \,]$$

```
> p:=cmul(e1+e2,e2-e4+2*e1); #p is a product of 1-vectors
```

$$p := 3\,Id - e1we2 - e2we4 - e1we4$$

```
> isproduct(p);
```

$$true$$

```
> L:=isproduct(p,'any');
```

$$L := [\,true, [\,e1 + e2, 2\,e1 + e2 - e4\,]\,]$$

Notice that vectors shown in L[2] above are different than the ones used in the definition of p. However, their product still gives p:

```
> v1:=L[2][1]:v2:=L[2][2]:v1 &c v2;evalb(p=%);
```

$$3\,Id - e1we2 - e2we4 - e1we4$$

$$true$$

Since the vectors v1 and v2 above are not isotropic as shown below, they are invertible and since p = v1 &c v2 thus p belongs to the Lipschitz group of Cl(3,1).

```
> cmul(v1,v1),cmul(v2,v2);
```

$$2\,Id, 4\,Id$$

Notice also that when we factor p again, usually different factors are returned:

```
> L:=isproduct(p,'any');
```

$$L := \left[\,true, \left[\,e1 + 3\,e2 + 2\,e4, \frac{2\,e1}{3} + e2 + \frac{e4}{3}\,\right]\right]$$

```
> v1:=L[2][1]:v2:=L[2][2]:v1 &c v2;evalb(p=%);
```

$$3\,Id - e1we2 - e2we4 - e1we4$$

$$true$$

Now, let's try to find a general factorization of p:

```
> L:=isproduct(p,'all');
```

$$L := \left[\,true, \left[\,\frac{e1}{x22} + \frac{2\,e2}{-x24 + x22} + \frac{(x24 + x22)\,e4}{x22\,(-x24 + x22)}, (-x24 + x22)\,e1 + x22\,e2 + x24\,e4\,\right]\right]$$

Notice various undetermined parameters in the above solution. We will verify now that the two vectors in L[2] are factors of the polynomial p defined above:

```
> v1:=L[2][1]:v2:=L[2][2]:simplify(v1 &c v2);evalb(p=%);
```

$$3\,Id - e1we2 - e2we4 - e1we4$$

$$true$$

```
>
```

Example 5: Element p of Cl(3,1) has now two parts of grade 3 and 1. Is p factorable? If it is then p must be a product of three 1-vectors.

```
> p:=2*e1we2we3+e1;
```

$$p := 2\,e1we2we3 + e1$$

```
> L:=isproduct(p,'any');
```

$$L := \left[\,true, \left[\,-3\,e2 + e3, -\frac{e2}{2} - \frac{e3}{2}, e1\,\right]\right]$$

```
> v1:=L[2][1]:v2:=L[2][2]:v3:=L[2][3]:simplify(v1 &c v2 &c v3);
```

```
evalb(p=%);
```

$$e1 + 2\, e1we2we3$$

$$true$$

```
> L:=isproduct(p,'all');
```

$$L := \left[true, \left[\frac{(x22 + 2\, x23)\, e2}{x22^2 + x23^2} - \frac{(2\, x22 - x23)\, e3}{x22^2 + x23^2}, x22\, e2 + x23\, e3, e1 \right] \right]$$

```
> v1:=L[2][1]:v2:=L[2][2]:v3:=L[2][3]:simplify(v1 &c v2 &c v3);
evalb(p=%);
```

$$e1 + 2\, e1we2we3$$

$$true$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 5.814000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

See Also: Clifford:-isVahlenmatrix, Clifford:-`type/climon`, Clifford:-`type/clipolynom`, Clifford:-`type/cliscalar`, Clifford:-`type/clibasmon`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-isVahlenmatrix - determine whether the given 2 x 2 Clifford matrix is a Vahlen matrix

Calling Sequence:

isVahlenmatrix(m);

Parameters:

m - expression of the type 'climatrix' or 'matrix'

Description:

- Procedure 'isVahlenmatrix' determines if the given 2 x 2 Clifford matrix is a Vahlen matrix and it returns 'true' or 'false' accordingly. Any matrix with entries in a Clifford algebra is of `type/climatrix`.
- A *Vahlen matrix* is a 2 x 2 matrix V with entries in a Clifford algebra Cl(p, q) such that if

V := matrix(2, 2, [a, b, c, d]);

- and a, b, c, d are elements in Cl(p, q), then the following conditions must be met:
  1. a, b, c, d are products of 1-vectors,
  2. the pseudodeterminant of V is +1 or -1 (or, Id or -Id in the algebra),
  3. a &c reversion(b), reversion(b) &c d, d &c reversion(c), and reversion(c) &c a are all 1-vectors.
- See reversion for more information on this Clifford algebra anti-automorphism and &c for information about the Clifford product inn Cl(B). For information about the pseudodeterminant see pseudodet.
- Condition (1) above implies that a, b, c, and d are elements of the *Lipschitz group* of R^(p,q). Procedure isproduct is used to determine whether this condition is met.
- Recall that in dimensions n >= 3 sense preserving conformal mappings are restrictions of the *Mobius transformations* and are compositions of rotations, translations, dilations and transversions (called also special conformal transformations). A Mobius transformation in R^(p,q) can be written in the form

x --> (a x + b) (c x + d)^(-1)

- where x is a 1-vector that belongs to R^(p,q), a, b, c, d belong to Cl(p,q), and, the products and the inverse are taken in Cl(p,q). This transformation may be represented by the Vahlen matrix V defined above.
- Rotations, translations, dilations, and transversions will then be represented as follows:

- rotations x --> a x a^(-1) where a is in Spin+(p,q), V = matrix(2, 2, [a, 0, 0, a])
- translations x --> x + b where b is in R^(p,q) , V = matrix(2, 2, [1, b, 0, 1])
- dilations x --> s x where s > 0, V = matrix(2, 2, [sqrt(s), 0, 0,

1/sqrt(s)])

- transversions x --> (x + x^2 c)/(1 + 2 x . c + x^2 c^2), V = matrix(2, 2, [1, 0 , c, 1]),

  where c is in R^(p,q), and x . c is the dot product in R^(3,1).

- For this procedure to work, the bilinear form B must be assigned.
- Since the procedure 'isproduct' is not optimized for polynomials of grade higher than 3, condition (1) can only be checked for monomials of higher grades than 3, but not polynomials.
- **References:**

Pertti Lounesto, *"CLICAL and counter-examples"*, in eds. R. Ablamowicz, P. Lounesto, and J. Parra, *'Clifford Algebras with Symbolic and Numeric Computations'*, Birkhauser, Boston, 1996, pp. 3--30.

Pertti. Lounesto and Arthur Springer, *"Mobius transformations and Clifford algebras of Euclidean and anti-Euclidean spaces,"* in *'Deformations of Mathematical Physics,'* ed. J. Lawrynowicz, Kluwer Academic Publishers, 1989, pp. 79--90.

Pertti Lounesto and Esko Latvamaa, *"Conformal transformations and Clifford algebras,"* Proceedings of the American Mathematical Society, Volume 79, Number 4, August 1980, pp. 533--538.

Johannes Maks, *"Modulo (1,1) periodicity of Clifford algebras and the generalized (anti-)Mobius transformations"*. Thesis, Technische Universiteit Delft, 1989.

Examples:

```
> restart:bench:=time():with(Clifford):_prolevel;
```

$$false$$

```
> eval(makealiases(6)):
```

Example 1: Simple cases in the signature (3,1).

(1) Rotations:

```
> B:=linalg[diag](1,1,1,-1):#bilinear form for the Minkowski
  space
> a:=e1we2; #an element of grade 2 in Spin+(3,1)
```

$$a := e12$$

```
> R:=linalg[matrix](2,2,[a,0,0,a]);
```

$$R := \begin{bmatrix} e12 & 0 \\ 0 & e12 \end{bmatrix}$$

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

$$true$$

(2) Translations:

```
> b:=e1+2*e3; #vector in R^(3,1)
```

$$b := e1 + 2\,e3$$

```
> T:=linalg[matrix](2,2,[1,b,0,1]);
```

$$T := \begin{bmatrix} 1 & e1 + 2\,e3 \\ 0 & 1 \end{bmatrix}$$

```
> isVahlenmatrix(T);
```

$$true$$

(3) Dilations:

```
> delta:=1/4: # a positive parameter
> Dil:=linalg[matrix](2,2,[sqrt(delta),0,0,1/sqrt(delta)]);
```

$$Dil := \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & 2 \end{bmatrix}$$

```
> isVahlenmatrix(Dil);
```

$$true$$

(4) Transversions:

```
> c:=2*e1-e3; # vector in R^(3,1)
```

$$c := 2\,e1 - e3$$

```
> Tv:=linalg[matrix](2,2,[1,0,c,1]);
```

$$Tv := \begin{bmatrix} 1 & 0 \\ 2\,e1 - e3 & 1 \end{bmatrix}$$

```
> isVahlenmatrix(Tv);
```

$$true$$

```
>
```

Example 2: If we now take a product of these four matrices above, we will obtain an element of the conformal group in R^(3,1):

```
> conf:=R &cQm T &cQm Dil &cQm Tv;
```

$$conf := \begin{bmatrix} \frac{e12}{2} + 10\,e23 & 4\,e123 - 2\,e2 \\ -2\,e123 - 4\,e2 & 2\,e12 \end{bmatrix}$$

```
>
```

Since in the product above each matrix appeared exactly <u>once</u>, the diagonal entries of 'conf' must be invertible:

```
> cinv(conf[1,1]); #inverse of conf[1,1]
```

$$-\frac{2\,e12}{401} - \frac{40\,e23}{401}$$

```
> cmul(%,conf[1,1]); #let's verify the inverse of conf[1,1]
```

$$Id$$

```
> cinv(conf[2,2]); #inverse of conf[2,2]
```

$$-\frac{e12}{2}$$

```
> cmul(%,conf[2,2]); #let's verify the inverse of conf[2,2]
```

$$Id$$

However, there are elements in the conformal group of R^(3,1) whose Vahlen matrices do not have invertible elements at all. The following example of such matrix is due to Johannes Maks.

```
>
```

Example 3: (Johannes Maks) Example of a Vahlen matrix W without any invertible entries. Matrix W defined below represents an element in the identity component of the conformal group of R^(3,1).

```
> makealiases(4):eval(%):
> W:=evalm((1/2)*linalg[matrix](2,2,[1-e14,-e1+e4,e1+e4,1+e14]));
```

$$W := \begin{bmatrix} \frac{1}{2} - \frac{e14}{2} & -\frac{e1}{2} + \frac{e4}{2} \\ \frac{e1}{2} + \frac{e4}{2} & \frac{1}{2} + \frac{e14}{2} \end{bmatrix}$$

Notice that the diagonal elements of W are non-trivial idempotents in Cl(3,1), hence they are not invertible in Cl(3,1):

```
> type(W[1,1],idempotent); #element (1,1) of W is an idempotent
```

$$true$$

```
> type(W[2,2],idempotent); #element (2,2) of W is an idempotent
```

$$true$$

```
> cinv(W[1,1]); #trying to compute the inverse of W[1,1] gives an
  error
Warning, testing why entered argument has no inverse...

Error, (in Clifford:-cinv) element 1/2-1/2*e1we4 is an idempotent in signature
[3, 1] and as such it has no inverse
```

Notice also that the off-diagonal elements of W are isotropic vectors in R^(3,1), hence they are also non-invertible:

```
> cmul(W[1,2],W[1,2]);
```

$$0$$

```
> cmul(W[2,1],W[2,1]);
```

$$0$$

```
> cinv(W[1,2]);#trying to compute the inverse of W[1,2] gives an
  error
Warning, testing why entered argument has no inverse...

Error, (in Clifford:-cinv) element -1/2*e1+1/2*e4 is nilpotent in signature [3,
1] and as such it has no inverse
```

Let's now verify that W is a Vahlen matrix:

```
> isVahlenmatrix(W);
```

$$true$$

However, W is an element of the identity component of the conformal group in R^(3,1) since its pseudo-determinant is 1, and since it can be written as a product of a transversion, a translation, and a transversion. Thus, in another words, W is <u>not</u> a product of <u>just one</u> rotation, <u>one</u> translation, <u>one</u> dilation, and/or <u>one</u> transversion:

```
> Tv:=linalg[matrix](2,2,[1,0,(e1+e4)/2,1]);
```

$$Tv := \begin{bmatrix} 1 & 0 \\ \frac{e1}{2} + \frac{e4}{2} & 1 \end{bmatrix}$$

```
> T:=linalg[matrix](2,2,[1,(-e1+e4)/2,0,1]);
```

$$T := \begin{bmatrix} 1 & -\frac{e1}{2} + \frac{e4}{2} \\ 0 & 1 \end{bmatrix}$$

```
> Tv &cQm T &cQm Tv,displayid(W); # W = Tv &cQm T &cQm Tv
```

$$\begin{bmatrix} \frac{1}{2} - \frac{e14}{2} & -\frac{e1}{2} + \frac{e4}{2} \\ \frac{e1}{2} + \frac{e4}{2} & \frac{1}{2} + \frac{e14}{2} \end{bmatrix}, \begin{bmatrix} \frac{Id}{2} - \frac{e14}{2} & -\frac{e1}{2} + \frac{e4}{2} \\ \frac{e1}{2} + \frac{e4}{2} & \frac{Id}{2} + \frac{e14}{2} \end{bmatrix}$$

```
> pseudodet(W); #computing pseudo-determinant of W
```

$$Id$$

Thus, the above computation confirms that W = Tv &cQm T &cQm Tv and that the pseudo-determinant of W is 1.

```
>
```

Example 4: This is another variation of Johannes Maks' example of a Vahlen matrix W without any invertible entries. Matrix W represents an element in the identity component of the conformal group of R^(3,1).

```
> W:=evalm((1/2)*linalg[matrix](2,2,[1-e24,-e2+e4,e2+e4,1+e24]));
```

$$\mathrm{W} := \begin{bmatrix} \frac{1}{2} - \frac{e24}{2} & -\frac{e2}{2} + \frac{e4}{2} \\ \frac{e2}{2} + \frac{e4}{2} & \frac{1}{2} + \frac{e24}{2} \end{bmatrix}$$

Notice that the diagonal elements of W are non-trivial idempotents in Cl(3,1), hence they are not invertible in Cl(3,1):

```
> type(W[1,1],idempotent); #element (1,1) of W is an idempotent
```

$$true$$

```
> type(W[2,2],idempotent); #element (2,2) of W is an idempotent
```

$$true$$

```
> cinv(W[1,1]); #trying to compute the inverse of W[1,1] gives an
  error
Warning, testing why entered argument has no inverse...
```

```
Error, (in Clifford:-cinv) element 1/2-1/2*e2we4 is an idempotent in signature
[3, 1] and as such it has no inverse
```

Notice also that the off-diagonal elements of W are isotropic vectors in R^(3,1), hence they are also non-invertible:

```
> cmul(W[1,2],W[1,2]);
```

$$0$$

```
> cmul(W[2,1],W[2,1]);
```

$$0$$

```
> cinv(W[1,2]);#trying to compute the inverse of W[1,2] gives an
  error
Warning, testing why entered argument has no inverse...

Error, (in Clifford:-cinv) element -1/2*e2+1/2*e4 is nilpotent in signature [3,
1] and as such it has no inverse
```

Let's now verify that W is a Vahlen matrix:

```
> isVahlenmatrix(W);
```

$$true$$

However, W is an element of the identity component of the conformal group in R^(3,1) since its pseudo-determinant is 1, and since it can be written as a product of a transversion, a translation, and a transversion. As before, W is *not* a product of <u>just one</u> rotation, <u>one</u> translation, <u>one</u> dilation, and/or <u>one</u> transversion:

```
> Tv:=linalg[matrix](2,2,[1,0,(e2+e4)/2,1]);
```

$$Tv := \begin{bmatrix} 1 & 0 \\ \frac{e2}{2} + \frac{e4}{2} & 1 \end{bmatrix}$$

```
> T:=linalg[matrix](2,2,[1,(-e2+e4)/2,0,1]);
```

$$T := \begin{bmatrix} 1 & -\frac{e2}{2} + \frac{e4}{2} \\ 0 & 1 \end{bmatrix}$$

```
> Tv &cQm T &cQm Tv,map(displayid,W); # W = Tv &cQm T &cQm Tv
```

$$\begin{bmatrix} \frac{1}{2} - \frac{e24}{2} & -\frac{e2}{2} + \frac{e4}{2} \\ \frac{e2}{2} + \frac{e4}{2} & \frac{1}{2} + \frac{e24}{2} \end{bmatrix}, \begin{bmatrix} \frac{Id}{2} - \frac{e24}{2} & -\frac{e2}{2} + \frac{e4}{2} \\ \frac{e2}{2} + \frac{e4}{2} & \frac{Id}{2} + \frac{e24}{2} \end{bmatrix}$$

```
> pseudodet(W); #computing pseudo-determinant of W
```

$$Id$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 3.670000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

Thus, the above computation confirms that W = Tv &cQm T &cQm Tv and that the pseudo-determinant of W is 1.

See Also: Clifford:-reversion, Clifford:-`type/idempotent`, Clifford:-cmul, Clifford:-`type/climatrix, Clifford:-makealiases, Clifford:-displayid, Clifford:-cinv, Clifford:-`&cQm`, Clifford:-pseudodet, Clifford:-isproduct

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-Kfield - find a basis for a field K of a spinor representation of Cl(Q)

Calling Sequence:

Kfield(list,f);

Parameters:

list - a list of elements of type 'list' and 'string'
f - a primitive idempotent in Cl(Q) used to generate minimal ideal S

Description:

- Procedure 'Kfield' computes a Clifford basis for a field K over which a spinor representation of Cl(Q) in a spinor space S defined as a left Cl(Q)f or right fCl(Q) minimal ideal will be found.
- The field K is isomorphic to the reals, or to the complexes, or to the quaternions according to whether (p-q) mod 8 is 0, 1, 2, or 3, 7, or 4, 5, 6, respectively (here [p,q] is the signature of Q).
- Assuming that the bilinear form B has been defined and is diagonal, the first argument of the procedure is expected to be the same as the output from the procedure 'minimalideal' (see minimalideal for more help).
- The second argument is the idempotent f. It is expected that the idempotent f is the same as the one used by the procedure 'minimalideal' to find a real basis in S. One of the basis elements in S is always f.
- The procedure 'Kfield' eliminates from the list of real basis elements in S those elements that are nilpotent (none of the generators of K may be nilpotent) and leaves only those whose square modulo f is either +1 or -1. These elements are of `type/fieldelement` and are returned as the first list in the output.
- If the primitive idempotent f (see `type/primitiveidemp`) is the same as the one stored under clidata()[4] (see clidata for more help) and if the generators of the real basis in S match those stored under clidata()[5], the procedure uses generators of K stored under clidata()[6] and returns them as the second list in its output.
- To summarize, procedure 'Kfield' returns a list consisting of two lists:

- the first list is an ordered list of expanded basis elements for K in S,

- the second list is an ordered list of basis monomials which generate the first list modulo f.

- There is a one-to-one relationship between the elements of the two ordered list.
- Note: the following three examples are a continuation of the examples from the help page for minimalideal. Parts previously discussed have no Maple output.
- To speed up computations we set global variable _prolevel to true. To find out more about _prolevel, try Clifford:-setup.

- For more on environmental variables such as _shortcut_in_Kfield, _prolevel, etc. see procedure CLIFFORD_ENV.

Examples:

```
> restart:bench:=time():with(Clifford):_shortcut_in_Kfield:=false
  :
```

To shorten output, we will use aliases. Since below we will not exceed dimension 6, we can define ordered aliases first:

```
> eval(makealiases(6,'ordered')):
```

Thus, for example,

```
> e1 &c e2;
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$e12 + B_{1,2}\, Id$$

where e12 = e1e2. When the bilinear form B is diagonal, then e12 = e1e2 = e1we2, etc.

Example 1: Clifford algebra Cl(1,1) is isomorphic to R(2)

```
> dim:=2:B:=linalg[diag](1,-1): #define the bilinear form B for
  Cl(1,1)
  clibasis:=cbasis(dim): #compute a Clifford basis for Cl(B)
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data := \left[real, 2, simple, \frac{Id}{2} + \frac{e12}{2}, [Id, e1], [Id], [Id, e1]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own
  here
> sbasis:=minimalideal(clibasis,f,'left'); #compute a real basis
  in Cl(B)f
```

$$sbasis := \left[\left[\frac{Id}{2} + \frac{e12}{2}, \frac{e1}{2} + \frac{e2}{2}\right], [Id, e1], left\right]$$

Note that the generator for K computed next is just Id (returned as a list [Id]) since K is isomorphic to the real field R (compare the first entry in 'data' above for the current signature (1,1)). Notice also that in S this gives a basis element cmul(Id,f) = f which is returned in the first list in the output below.

```
> Kbasis:=Kfield(sbasis,f); #compute a basis for the field K
```

$$Kbasis := \left[\left[\frac{Id}{2} + \frac{e12}{2}\right], [Id]\right]$$

```
>
```

Example 2: Clifford algebra Cl(2,0) is isomorphic to R(2).

```
> dim:=2:B:=linalg[diag](1,1):#define the bilinear form B for
  Cl(2,0)
  clibasis:=cbasis(dim): #compute a Clifford basis for Cl(B)
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data := \left[real, 2, simple, \frac{Id}{2} + \frac{e1}{2}, [Id, e2], [Id], [Id, e2]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own here
> sbasis:=minimalideal(clibasis,f,'left'); #compute a real basis in Cl(B)f
```

$$sbasis := \left[\left[\frac{Id}{2} + \frac{e1}{2}, \frac{e2}{2} - \frac{e12}{2}\right], [Id, e2], left\right]$$

```
> Kbasis:=Kfield(sbasis,f); #compute a basis for the field K
```

$$Kbasis := \left[\left[\frac{Id}{2} + \frac{e1}{2}\right], [Id]\right]$$

```
>
```

Example 3: Clifford algebra Cl(2,2) is isomorphic to R(4).

```
> dim:=4:B:=linalg[diag](1,1,-1,-1):#define the bilinear form B for Cl(2,2)
  clibasis:=cbasis(dim): #compute a Clifford basis for Cl(B)
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data := \left[real, 4, simple, 'Clifford{:}\text{-}cmulQ'\left(\frac{Id}{2} + \frac{e13}{2}, \frac{Id}{2} + \frac{e24}{2}\right), [Id, e1, e2, e12], [Id], [Id, e1, e2, e12]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own here
> sbasis:=minimalideal(clibasis,f,'left'); #compute a real basis in Cl(B)f
```

$$sbasis := \left[\left[\frac{Id}{4} + \frac{e13}{4} + \frac{e24}{4} - \frac{e1234}{4}, \frac{e1}{4} + \frac{e3}{4} + \frac{e124}{4} - \frac{e234}{4}, \frac{e2}{4} - \frac{e123}{4} + \frac{e4}{4} + \frac{e134}{4}, \frac{e12}{4} - \frac{e23}{4} + \frac{e14}{4} + \frac{e34}{4}\right], [Id, e1, e2, e12], left\right]$$

```
> Kbasis:=Kfield(sbasis,f); #compute a basis for the field K
```

$$Kbasis := \left[\left[\frac{Id}{4} + \frac{e13}{4} + \frac{e24}{4} - \frac{e1234}{4}\right], [Id]\right]$$

```
>
```

Example 4: Clifford algebra Cl(3,0) is isomorphic to C(2).

```
> dim:=3:B:=linalg[diag](1,1,1):#define the bilinear form B for Cl(3,0)
  clibasis:=cbasis(dim): #compute a Clifford basis for Cl(B)
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data := \left[complex, 2, simple, \frac{Id}{2} + \frac{e1}{2}, [Id, e2, e3, e23], [Id, e23], [Id, e2]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own here
> sbasis:=minimalideal(clibasis,f,'left'); #compute a real basis in Cl(B)f
```

$$sbasis := \left[\left[\frac{Id}{2} + \frac{e1}{2}, \frac{e2}{2} - \frac{e12}{2}, \frac{e3}{2} - \frac{e13}{2}, \frac{e23}{2} + \frac{e123}{2}\right], [Id, e2, e3, e23], left\right]$$

Note that the generators for K computed next are Id and e2we3 (returned as a list [Id,e23]) since K is isomorphic to the complex field C (compare the first entry in 'data' above for the current signature (3,0)). Notice also that in S this gives two basis elements cmul(Id,f) =f and cmul(e23,f) which are returned in the first list in the output below.

```
> Kbasis:=Kfield(sbasis,f); #compute a basis for the field K
```

$$Kbasis := \left[\left[\frac{Id}{2} + \frac{e1}{2}, \frac{e23}{2} + \frac{e123}{2}\right], [Id, e23]\right]$$

We verify now that the basis monomial e23 = e2we3 may play the role of the imaginary unit I:

```
> cmul(e2we3,e2we3); #element 'e2we3' in K plays the role of I in C
```

$$-Id$$

```
>
```

Example 5: Clifford algebra Cl(1,3) is isomorphic to H(2).

```
> dim:=4:B:=linalg[diag](1,-1,-1,-1):#define form B for Cl(1,3)
  clibasis:=cbasis(dim): #compute a Clifford basis for Cl(B)
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data := \left[quaternionic, 2, simple, \frac{Id}{2} + \frac{e14}{2}, [Id, e1, e2, e3, e12, e13, e23, e123], [Id, e2, e3, e23], [Id, e1]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own here
> sbasis:=minimalideal(clibasis,f,'left'); #compute a real basis in Cl(B)f
```

$$sbasis := \left[\left[\frac{Id}{2} + \frac{e14}{2}, \frac{e1}{2} + \frac{e4}{2}, \frac{e2}{2} - \frac{e124}{2}, \frac{e3}{2} - \frac{e134}{2}, \frac{e12}{2} - \frac{e24}{2}, \frac{e13}{2} - \frac{e34}{2}, \frac{e23}{2} + \frac{e1234}{2}, \frac{e123}{2} + \frac{e234}{2}\right], [Id, e1, e2, e3, e12, e13, e23, e123], left\right]$$

Note that the generators for K computed next are Id, e2, e3, and e2we3 (returned as a list [Id,e2,e3,e23]) since K is isomorphic to the quaternionic division ring H (compare the first entry in 'data' above for the current signature (1,3)). Notice also that in S this gives three basis elements cmul(Id,f) = f, cmul(e2,f), cmul(e3,f), and cmul(e23,f) which are returned in the first list in the

output below.

```
> Kbasis:=Kfield(sbasis,f); #compute a basis for the field K
```

$$Kbasis := \left[\left[\frac{Id}{2}+\frac{e14}{2}, \frac{e2}{2}-\frac{e124}{2}, \frac{e3}{2}-\frac{e134}{2}, \frac{e23}{2}+\frac{e1234}{2}\right], [Id, e2, e3, e23]\right]$$

We verify now that the basis monomials [Id, e2, e3, e23] provide a basis for K as being isomorphic to H: while, of course, Id plays the role of 1, e2, e3, and e23 play the roles of the pure quaternionic units i, j, and k It is not difficult to see that in the signature (1,3) these four elements have all desired properties. The verification maybe be done as follows:

```
> cmul(Id,Id),cmul(e2,e2),cmul(e3,e3),cmul(e2we3,e2we3);#squares
  first
```

$$Id, -Id, -Id, -Id$$

```
> cmul(e2,e3)+cmul(e3,e2);          #e2 and e3 anticommute
  cmul(e2,e2we3)+cmul(e2we3,e2); #e2 and e2we3 anticommute
  cmul(e3,e2we3)+cmul(e2we3,e3); #e2we3 and e3 anticommute
```

$$0$$
$$0$$
$$0$$

The short way to check that the list [e2,e3,e23] provides a basis for pure quaternions is to verify that it is of the type 'purequatbasis' as follows:

```
> type([e2,e3,e23],purequatbasis);
```

$$true$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 1.765000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

Example 6: Clifford algebra Cl(3,1) is isomorphic to R(4).

```
> restart:with(Clifford):eval(makealiases(4)):
> dim:=4:B:=linalg[diag](1,1,1,-1):#define form B for Cl(1,3)
  clibasis:=cbasis(dim): #compute a Clifford basis for Cl(B)
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data := \left[real, 4, simple, 'cmulQ'\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e34}{2}\right), [Id, e2, e3, e23], [Id], [Id, e2, e3, e23]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own
  here
> sbasis:=minimalideal(clibasis,f,'left'):#compute a real basis
  in Cl(B)f
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
> Kbasis:=Kfield(sbasis,f); #compute a basis for the field K
```

$$Kbasis := \left[\left[\frac{Id}{4} + \frac{e1}{4} + \frac{e34}{4} + \frac{e134}{4}\right], [Id]\right]$$

[>

Comments:

- Using 'Kfield' as shown above is the second step towards the computation of the spinor representation of Cl(Q) in a minimal ideal S over the field K. The first step was to find a real basis for S with 'minimalideal' (see minimalideal). The next step is to use the procedure 'spinorKbasis' to find a K-basis for S over the field K (see spinorKbasis).
- The above examples may be redone for right minimal ideals with the same generators for the field K. This is because K is isomorphic to the intersection of the left and right minimal ideals Cl(B)f and fCl(B).

See Also: Clifford:-`type/fieldelement`, Clifford:-squaremodf, Clifford:-cbasis, Clifford:-`type/purequatbasis`, Clifford:-RHnumber, Clifford:-Bsignature, Clifford:-`type/primitiveidemp`, Clifford:-spinorKrepr, Clifford:-spinorKbasis, Clifford:-matKrepr, Clifford:-minimalideal, Clifford:-`type/clipolynom`, Clifford:-clidata

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-LC - left contraction in Cl(B)

Calling Sequence:

LC(u,v);
LC(u,v,name);

Parameters:

u, v - expressions of the type 'cliscalar', 'clibasmon', 'climon', or 'clipolynom'
name - (optional) parameter of type 'name', 'symbol, 'array', or 'matrix', or `&*`(numeric,{name,symbol,matrix,array})

Description:

- Procedure 'LC' defines a *left contraction* in Cl(B), when no third argument is used or when name=B, between a multivector u that acts from the left on a multivector v. Otherwise, contraction is computed in Cl(K) if name=K.
- Both inputs to this procedure are elements in the Clifford algebra Cl(B) of a totally arbitrary bilinear form B. However, it is possible to use this procedure with a third optional parameter of `type/name`, `type/symbol`, `type/array`, or `type/matrix` in which case indices are appended to this parameter. See example below.
- This procedure is used to define Clifford multiplication 'cmul' in Cl(B). See cmul for more help.
- See procedure CLIFFORD_ENV to display values of environmental variables in 'CLIFFORD' such as dim_V.
- To collect complicated expressions, use clicollect.
- Procedure 'LC' can also be applied to expressions containing terms of `type/cliprod` upon loading the supplementary package Cliplus (see below). Notice that unevaluated Clifford products must be entered this way: `&C`[K](e1,e2) when the optional index is used. When no index is used, enter &C(e1,e2).

Examples:

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

$$9$$

Procedure 'LC' gives the left-contraction u _| v in the Clifford algebra Cl(B) of any element v by any element u entered as a first parameter:

```
> LC(e1,e2),LC(e2,e3),LC(e3,e2),LC(e2,e3);
```

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

```
> LC(2*e1-e3+e4,2*e1we2+e4+3*e3we4);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$4\, B_{1,1}\, e2 - 4\, B_{1,2}\, e1 + 2\, B_{1,4}\, Id + 6\, B_{1,3}\, e4 - 6\, B_{1,4}\, e3 - 2\, B_{3,1}\, e2 + 2\, B_{3,2}\, e1 - B_{3,4}\, Id$$

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

```
> clicollect(LC(2+3*e1-e3+e2we3we4,e1we2we3we4+e3we4-e2-3));
```

$$-(6+3\,B_{1,2}-B_{3,2})\,Id-(-B_{4,2}\,B_{3,4}\,B_{2,3}+B_{4,3}\,B_{3,4}\,B_{2,2}+B_{4,2}\,B_{3,3}\,B_{2,4}-B_{4,3}\,B_{3,2}\,B_{2,4}+B_{4,4}\,B_{3,2}\,B_{2,3}-B_{4,4}\,B_{3,3}\,B_{2,2})\,e1+(B_{4,1}\,B_{3,3}\,B_{2,4}-2-B_{4,1}\,B_{3,4}\,B_{2,3}-B_{4,3}\,B_{3,1}\,B_{2,4}+B_{4,4}\,B_{3,1}\,B_{2,3}+B_{4,3}\,B_{3,4}\,B_{2,1}-B_{4,4}\,B_{3,3}\,B_{2,1})\,e2-(-B_{4,2}\,B_{3,1}\,B_{2,4}+B_{4,1}\,B_{3,2}\,B_{2,4}+3\,B_{1,4}-B_{4,1}\,B_{3,4}\,B_{2,2}+B_{4,2}\,B_{3,4}\,B_{2,1}-B_{4,4}\,B_{3,2}\,B_{2,1}-B_{3,4}+B_{4,4}\,B_{3,1}\,B_{2,2})\,e3+2\,e3we4+(B_{4,3}\,B_{3,1}\,B_{2,2}+3\,B_{1,3}-B_{3,3}-B_{4,2}\,B_{3,1}\,B_{2,3}-B_{4,1}\,B_{3,3}\,B_{2,2}+B_{4,2}\,B_{3,3}\,B_{2,1}+B_{4,1}\,B_{3,2}\,B_{2,3}-B_{4,3}\,B_{3,2}\,B_{2,1})\,e4+(3\,B_{1,1}-B_{3,1})\,e2we3we4+2\,e1we2we3we4-(3\,B_{1,2}-B_{3,2})\,e1we3we4+(3\,B_{1,3}-B_{3,3})\,e1we2we4-(-B_{3,4}+3\,B_{1,4})\,e1we2we3$$

Note these special cases:

```
> LC(0,2);
```

$$0$$

```
> p:=a*Id+2*e1we3;
```

$$p := a\,Id+2\,e1we3$$

```
> LC(3,p);LC(3*Id,p);
```

$$3\,a\,Id+6\,e1we3$$

$$3\,a\,Id+6\,e1we3$$

```
> LC(2,3);
```

$$6$$

```
> LC(2*Id,3);
```

$$6\,Id$$

```
> LC(2*Id,3*Id);
```

$$6\,Id$$

Example 1: Simple computations:

```
> p:=2*e1we2;q:=2+e2+e2we3+3*e1we2we3;
  u:=2*e2we3-e1+e3+a*e2we3;v:=-2*e2-3*e1+e4we3+c*e2we3;
  x:=e1-2*e2+e3;
  y:=-e2+3*e1-e4;
```

$$p := 2\,e1we2$$

$$q := 2+e2+e2we3+3\,e1we2we3$$

$$u := 2\,e2we3-e1+e3+a\,e2we3$$

$$v := -2\,e2-3\,e1+e4we3+c\,e2we3$$

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

$$y := -e2+3\,e1-e4$$

```
> s:=time():LC(p,q);time()-s;
```

$$2\,B_{2,2}\,B_{1,3}\,Id-2\,B_{2,3}\,B_{1,2}\,Id+6\,B_{2,1}\,B_{1,2}\,e3-6\,B_{2,1}\,B_{1,3}\,e2-6\,B_{2,2}\,B_{1,1}\,e3+6\,B_{2,2}\,B_{1,3}\,e1+6\,B_{2,3}\,B_{1,1}\,e2-6\,B_{2,3}\,B_{1,2}\,e1$$

$$0.016$$

```
> LC(2*Id,e1);
```

$$2\, e1$$

```
> s:=time():LC(x,u);time()-s;
```

$$2\, B_{1,2}\, e3 - 2\, B_{1,3}\, e2 - B_{1,1}\, Id + B_{1,3}\, Id + a\, B_{1,2}\, e3 - a\, B_{1,3}\, e2 - 4\, B_{2,2}\, e3 + 4\, B_{2,3}\, e2 + 2\, B_{2,1}\, Id - 2\, B_{2,3}\, Id - 2\, a\, B_{2,2}\, e3 + 2\, a\, B_{2,3}\, e2 + 2\, B_{3,2}\, e3 - 2\, B_{3,3}\, e2 - B_{3,1}\, Id + B_{3,3}\, Id + a\, B_{3,2}\, e3 - a\, B_{3,3}\, e2$$

$$0.015$$

Example 2: Let's verify now these properties of the left contraction: (here _| denotes the left contraction)

Property 1: A vector contracting a vector gives the dot product of the two vectors only when the bilinear form is symmetric [the right-hand side of (1) below is often identified with the "dot" product on a vector space].

x _| y = (1/2)*(x &c y + y &c x) (1)

```
> L1:=LC(x,y); #left hand side of (1)
```

$$L1 := -B_{1,2}\, Id + 3\, B_{1,1}\, Id - B_{1,4}\, Id + 2\, B_{2,2}\, Id - 6\, B_{2,1}\, Id + 2\, B_{2,4}\, Id - B_{3,2}\, Id + 3\, B_{3,1}\, Id - B_{3,4}\, Id$$

```
> L2:=(x &c y + y &c x)/2; #right hand side of (1)
```

$$L2 := -\frac{1}{2}\left(B_{3,2} + B_{1,2} - 2\, B_{2,2} - 3\, B_{3,1} - 3\, B_{1,1} + 6\, B_{2,1} - 2\, B_{2,4} + B_{1,4} + B_{3,4}\right) Id - \frac{1}{2}\left(B_{4,1} + B_{2,1} - 3\, B_{1,1} - 2\, B_{4,2} - 2\, B_{2,2} + 6\, B_{1,2} - 3\, B_{1,3} + B_{2,3} + B_{4,3}\right) Id$$

```
> out:=clicollect(L1-L2);
```

$$out := \left(\frac{1}{2} B_{4,3} - B_{4,2} + \frac{5}{2} B_{1,2} + \frac{1}{2} B_{2,3} - \frac{1}{2} B_{3,2} - \frac{1}{2} B_{1,4} - \frac{1}{2} B_{3,4} + \frac{1}{2} B_{4,1} - \frac{5}{2} B_{2,1} + \frac{3}{2} B_{3,1} - \frac{3}{2} B_{1,3} + B_{2,4}\right) Id$$

Notice that the above difference between L1 and L2 is zero if B is symmetric, that is, when B[1,3] = B[3,1], B[2,4] = B[4,2], B[4,3] = B[3,4], B[2,3] = B[3,2], B[1,2] = B[2,1], B[4,1] = B[1,4, the above expression equals zero.]

For example, define the following symmetric matrix:

```
> B:=matrix(3,3,[1,a,b,a,1,c,b,c,1]);
```

$$B := \begin{bmatrix} 1 & a & b \\ a & 1 & c \\ b & c & 1 \end{bmatrix}$$

```
> x:=x1*e1+x2*e2+x3*e3:y:=y1*e1+y2*e2+y3*e3:
> L1:=clicollect(LC(x,y)); #left hand side of (1)
```

$$L1 := (x1\,y1 + x1\,y2\,a + x1\,y3\,b + x2\,y1\,a + x2\,y2 + x2\,y3\,c + x3\,y1\,b + x3\,y2\,c + x3\,y3)\,Id$$

```
> L2:=clicollect((x &c y + y &c x)/2); #right hand side of (1)
```

$$L2 := (x1\,y1 + x1\,y2\,a + x1\,y3\,b + x2\,y1\,a + x2\,y2 + x2\,y3\,c + x3\,y1\,b + x3\,y2\,c + x3\,y3)\,Id$$

```
> L1-L2;
```

$$0$$

However, it is easily seen that if the bilinear form has an antisymmetric part, then formula (1) is no longer valid:

```
> A:=matrix(3,3,[0,A1,A2,
                 -A1,0,A3,
                 -A2,-A3,0]);
```

$$A := \begin{bmatrix} 0 & A1 & A2 \\ -A1 & 0 & A3 \\ -A2 & -A3 & 0 \end{bmatrix}$$

```
> B:=evalm(B+A);
```

$$B := \begin{bmatrix} 1 & a + A1 & b + A2 \\ a - A1 & 1 & c + A3 \\ b - A2 & c - A3 & 1 \end{bmatrix}$$

```
> L1:=clicollect(LC(x,y)); #left hand side of (1)
```

$$L1 := (x1\,y1 + x1\,y2\,a + x1\,y2\,A1 + x1\,y3\,b + x1\,y3\,A2 + x2\,y1\,a - x2\,y1\,A1 + x2\,y2 + x2\,y3\,c + x2\,y3\,A3 + x3\,y1\,b - x3\,y1\,A2 + x3\,y2\,c - x3\,y2\,A3 + x3\,y3)\,Id$$

```
> L2:=clicollect((x &c y + y &c x)/2); #right hand side of (1)
```

$$L2 := (x1\,y1 + x1\,y2\,a + x1\,y3\,b + x2\,y1\,a + x2\,y2 + x2\,y3\,c + x3\,y1\,b + x3\,y2\,c + x3\,y3)\,Id$$

```
> clicollect(L1-L2);
```

$$(x1\,y2\,A1 + x1\,y3\,A2 - x2\,y1\,A1 + x2\,y3\,A3 - x3\,y1\,A2 - x3\,y2\,A3)\,Id$$

```
> B:='B':dim_V:=9: #let's erase B and assign the largest allowed
  value to dim_V
```

Property 2: Left contraction by a vector is a derivation of the Grassmann algebra for any bilinear form B:

x _| (u &w v) = (x _| u) &w v + gradeinv(u) &w (x _| v) (2)

```
> L1:=clicollect(LC(x,u &w v)); #left hand side of (2)
```

$$L1 := (6\,x3\,B_{3,2} + 6\,x2\,B_{2,2} + x2\,B_{2,4} + x3\,c\,B_{3,2} + 3\,x1\,a\,B_{1,2} + 3\,x2\,a\,B_{2,2} + x1\,B_{1,4} + 6\,x1\,B_{1,2} + 3\,x3\,a\,B_{3,2} + x3\,B_{3,4} + x1\,c\,B_{1,2} + x2\,c\,B_{2,2})\,e1we3 - (6\,x3\,B_{3,1} + 3\,x2\,a\,B_{2,1} + x1\,c\,B_{1,1} + 3\,x3\,a\,B_{3,1} + 3\,x1\,a\,B_{1,1} + x2\,c\,B_{2,1} + x3\,c\,B_{3,1} + 6\,x1\,B_{1,1} + 6\,x2\,B_{2,1})\,e2we3 - (x1\,c\,B_{1,3} + 6\,x3\,B_{3,3} + x2\,c\,B_{2,3} + 3\,x2\,a\,B_{2,3} + 3\,x3\,a\,B_{3,3} + 6\,x1\,B_{1,3} + 3\,x1\,a\,B_{1,3} + x3\,c\,B_{3,3} + 6\,x2\,B_{2,3})\,e1we2$$

$$- (3\, x2\, B_{2,3} + 2\, x3\, B_{3,2} + 3\, x3\, B_{3,3} + 2\, x1\, B_{1,2} + 3\, x1\, B_{1,3} + 2\, x2\, B_{2,2})\, e1$$
$$+ 2\, (x3\, B_{3,1} - x1\, B_{1,3} + x2\, B_{2,1} - x3\, B_{3,3} + x1\, B_{1,1} - x2\, B_{2,3})\, e2$$
$$+ (3\, x2\, B_{2,1} + 2\, x1\, B_{1,2} + 3\, x1\, B_{1,1} + 2\, x2\, B_{2,2} + 2\, x3\, B_{3,2} + 3\, x3\, B_{3,1})\, e3$$
$$+ (x3\, B_{3,1} + x2\, B_{2,1} + x1\, B_{1,1})\, e3we4 - (x1\, B_{1,3} + x3\, B_{3,3} + x2\, B_{2,3})\, e1we4$$

```
> L2:=clicollect(LC(x,u) &w v + gradeinv(u) &w LC(x,v)); #right
  hand side of (2)
```

$$L2 := (6\, x3\, B_{3,2} + 6\, x2\, B_{2,2} + x2\, B_{2,4} + x3\, c\, B_{3,2} + 3\, x1\, a\, B_{1,2} + 3\, x2\, a\, B_{2,2} + x1\, B_{1,4}$$
$$+ 6\, x1\, B_{1,2} + 3\, x3\, a\, B_{3,2} + x3\, B_{3,4} + x1\, c\, B_{1,2} + x2\, c\, B_{2,2})\, e1we3 - (6\, x3\, B_{3,1} + 3\, x2\, a\, B_{2,1}$$
$$+ x1\, c\, B_{1,1} + 3\, x3\, a\, B_{3,1} + 3\, x1\, a\, B_{1,1} + x2\, c\, B_{2,1} + x3\, c\, B_{3,1} + 6\, x1\, B_{1,1} + 6\, x2\, B_{2,1})\, e2we3$$
$$- (x1\, c\, B_{1,3} + 6\, x3\, B_{3,3} + x2\, c\, B_{2,3} + 3\, x2\, a\, B_{2,3} + 3\, x3\, a\, B_{3,3} + 6\, x1\, B_{1,3} + 3\, x1\, a\, B_{1,3}$$
$$+ x3\, c\, B_{3,3} + 6\, x2\, B_{2,3})\, e1we2$$
$$- (3\, x2\, B_{2,3} + 2\, x3\, B_{3,2} + 3\, x3\, B_{3,3} + 2\, x1\, B_{1,2} + 3\, x1\, B_{1,3} + 2\, x2\, B_{2,2})\, e1$$
$$+ 2\, (x3\, B_{3,1} - x1\, B_{1,3} + x2\, B_{2,1} - x3\, B_{3,3} + x1\, B_{1,1} - x2\, B_{2,3})\, e2$$
$$+ (3\, x2\, B_{2,1} + 2\, x1\, B_{1,2} + 3\, x1\, B_{1,1} + 2\, x2\, B_{2,2} + 2\, x3\, B_{3,2} + 3\, x3\, B_{3,1})\, e3$$
$$+ (x3\, B_{3,1} + x2\, B_{2,1} + x1\, B_{1,1})\, e3we4 - (x1\, B_{1,3} + x3\, B_{3,3} + x2\, B_{2,3})\, e1we4$$

```
> L1-L2;
```

$$0$$

Notice that property (2) is valid for ANY bilinear form B, that is, B does not need to be symmetric.

Property 3: Left contraction is left distributive with respect to the Grassmann (wedge) product for any bilinear form B:

(u &w v) _| q = u _| (v _| q) (3)

```
> L1:=clicollect(LC(u &w v, q)): #left hand side of (3)
> L2:=LC(u,LC(v,q)): #right hand side of (3)
> simplify(L1-L2);
```

$$0$$

Notice that property (3) is valid for ANY bilinear form B.

The left contraction by a vector is also a derivation in the Clifford algebra Cl(B):

Property 4: Left contraction by a vector is also a derivation in Cl(B) for any bilinear form B.

x _| (u &c v) = (x _| u) &c v + gradeinv(u) &c (x _| v) (4)

```
> L1:=LC(x,u &c v): #left hand side of (4)
```

```
> L2:=LC(x,u) &c v + gradeinv(u) &c LC(x,v): #rhs of (4)
> simplify(reorder(L1-L2));
```

$$0$$

Notice that property (4) is valid for ANY bilinear form B.

Example 3: In Cl(3) the following formula is valid:

bu = b _| u + (1/2)*(b &c u - u &c b) + b &w u (5)

where b is an arbitrary bivector and u is any element in Cl(3). Let's verify formula (5) now:

```
> B:=linalg[diag](1,1,1):
> clibas:=cbasis(3);
```

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

```
> eval(makealiases(3));
```

$$e12, e21, e13, e31, e23, e32, e123, e132, e213, e231, e312, e321$$

```
> b:=b12*e12+b13*e13+b23*e23; #an arbitrary bivector in Cl(3)
```

$$b := b12\, e12 + b13\, e13 + b23\, e23$$

```
> u:=u0+u1*e1+u2*e2+u3*e3+u12*e12+u13*e13+u23*e23+u123*e123; #any
  u in Cl(3)
```

$$u := u0 + u1\, e1 + u2\, e2 + u3\, e3 + u12\, e12 + u13\, e13 + u23\, e23 + u123\, e123$$

```
> L1:=b &c u; #left hand side of (5)
```

$$\begin{aligned} L1 := & -(b12\, u12 + b13\, u13 + b23\, u23)\, Id + (b13\, u0 - b23\, u12 + b12\, u23)\, e13 \\ & - (-b13\, u12 - b23\, u0 + b12\, u13)\, e23 + (b23\, u13 + b12\, u0 - b13\, u23)\, e12 \\ & + (b13\, u3 + b12\, u2 - b23\, u123)\, e1 - (b12\, u1 - b13\, u123 - b23\, u3)\, e2 \\ & - (b23\, u2 + b13\, u1 + b12\, u123)\, e3 + (b23\, u1 - b13\, u2 + b12\, u3)\, e123 \end{aligned}$$

```
> L2:=clicollect(LC(b,u) + (1/2)*(b &c u - u &c b) + b &w u);
Warning, since B has been (re-)assigned, value of dim_V has been reduced by 'we
dge' to 3
```

$$\begin{aligned} L2 := & -(b12\, u12 + b13\, u13 + b23\, u23)\, Id + (b13\, u0 - b23\, u12 + b12\, u23)\, e13 \\ & - (-b13\, u12 - b23\, u0 + b12\, u13)\, e23 + (b23\, u13 + b12\, u0 - b13\, u23)\, e12 \\ & + (b13\, u3 + b12\, u2 - b23\, u123)\, e1 - (b12\, u1 - b13\, u123 - b23\, u3)\, e2 \\ & - (b23\, u2 + b13\, u1 + b12\, u123)\, e3 + (b23\, u1 - b13\, u2 + b12\, u3)\, e123 \end{aligned}$$

```
> L1-L2;
```

$$0$$

Example 4: LCbig and LC can now accept input expressed in terms of the Clifford basis or both, the Clifford and the Grassmann bases. If elements of the Clifford basis are detected, then output is given in terms of the Clifford basis.

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

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

```
> printlevel:=1:
> LC(e1 &C e3,e2 &C e3);
```

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

```
> LC(e1 &C e3,e2 &C e3);
```

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

```
> LCbig(e1 &C e3,e2 &C e3);
```

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

LC and LCbig can accept third optional parameter of type 'name' which they use in place of B. However, it has to match the index used in `&C`:

```
> B:=matrix(3,3,[ 1, 1, 1,
                 -1, 1, 1,
                 -1,-1, 1]):
> LC(e1 &C e3,e2 &C e3,K); ## <<<--- Error due to different indices
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}
> LC(&C(e1,e3),`&C`[M](e2,e3),K);## <<<--- Error due to different indices
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, M}
```

Enclose &C in left quotes when using indices as in `&C`[K]:

```
> LC(`&C`[K](e1,e3),`&C`[K](e2,e3),K); ## No error
```

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

```
> LC(e1 &C e3,e2 &C e3,B);
```

$$-2\,Id + \&C_B(e2, e3)$$

```
> LC(e1 &C e3,e2 &C e3); #Default is B
```

$$-2\,Id + \&C_B(e2, e3)$$

```
> LC(`&C`[-B](e1,e3),`&C`[-B](e2,e3),-B);
```

$$-2\,Id - \&C_{-B}(e2, e3)$$

```
>
```

However, when used without that parameter, computations are performed with B:

```
> LC(e1 &C e3,e2 &C e3);
```

$$-2\,Id + \&C_B(e2, e3)$$

```
> B:='B':dim_V:=9:
> LC(p1,p2);
```

$$p1\,p2$$

```
> out1:=clicollect(LC(p1,p2)):out2:=clicollect(LCbig(p1,p2)):
> out1-out2;
```

$$0$$

```
> LC(2,3);LC(2*Id,3);LC(2*Id,3*Id);LC(2,p1);LC(2*Id,p1);LCbig(2,3
  );
  LCbig(2*Id,3);LCbig(2*Id,3*Id);LCbig(2,p1);LCbig(2*Id,p1);
```

$$6$$
$$6\ Id$$
$$6\ Id$$
$$2\ p1$$
$$2\ p1\ Id$$
$$6$$
$$6\ Id$$
$$6\ Id$$
$$2\ p1$$
$$2\ p1\ Id$$

```
> LC(2,3,M);LC(2*Id,3,M);LC(2*Id,3*Id,M);LC(2,p1,B);LC(2*Id,p1,B)
  ;
  LCbig(2,3,B);LCbig(2*Id,3,B);LCbig(2*Id,3*Id,B);LCbig(2,p1,B);L
  Cbig(2*Id,p1,B);
```

$$6$$
$$6\ Id$$
$$6\ Id$$
$$2\ p1$$
$$2\ p1\ Id$$
$$6$$
$$6\ Id$$
$$6\ Id$$
$$2\ p1$$
$$2\ p1\ Id$$

Example 5: Using the optional parameter of 'type/name` allows one to use a different form than the default bilinear form 'B'. For example, one could use, for example, the symmetric or the antisymmetric part of B in LC.

```
> restart:with(Clifford):with(linalg):with(Cliplus):
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
> B:=matrix(3,3,[d1,a,b,-a,d2,c,-b,-c,d3]);
```

$$B := \begin{bmatrix} d1 & a & b \\ -a & d2 & c \\ -b & -c & d3 \end{bmatrix}$$

```
> g,A:=evalm(1/2*(B+transpose(B))),evalm(1/2*(B-transpose(B)));
```

$$g, A := \begin{bmatrix} d1 & 0 & 0 \\ 0 & d2 & 0 \\ 0 & 0 & d3 \end{bmatrix}, \begin{bmatrix} 0 & a & b \\ -a & 0 & c \\ -b & -c & 0 \end{bmatrix}$$

```
> clicollect(LC(e1we2+e3-2*e1we2we3,e1we2+e3-2*e1we2we3,g));
  clicollect(LCbig(e1we2+e3-2*e1we2we3,e1we2+e3-2*e1we2we3,g));
```

$$-(d2\ d1 - d3 + 4\ d3\ d2\ d1)\ Id - 2\ d3\ e1we2 + 2\ d2\ d1\ e3$$

$$-(d2\ d1 - d3 + 4\ d3\ d2\ d1)\ Id - 2\ d3\ e1we2 + 2\ d2\ d1\ e3$$

```
> clicollect(LC(e1we2+e3-2*e1we2we3,e1we2+e3-2*e1we2we3,A));
  clicollect(LCbig(e1we2+e3-2*e1we2we3,e1we2+e3-2*e1we2we3,A));
```

$$-a^2\ Id - 2\ c\ e1we3 + 2\ b\ e2we3 + 2\ a^2\ e3 + c\ (1 + 2\ a)\ e1 - b\ (1 + 2\ a)\ e2$$

$$-a^2\ Id - 2\ c\ e1we3 + 2\ b\ e2we3 + 2\ a^2\ e3 + c\ (1 + 2\ a)\ e1 - b\ (1 + 2\ a)\ e2$$

```
>
```

See Also: Clifford:-RC, Clifford:-RCQ, Clifford:-LCQ, Clifford:-cmul, Clifford:-`type/clipolynom`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-LCQ - left contraction in Cl(Q), the Clifford algebra of a quadratic form Q

Calling Sequence:

LCQ(u,v);
LCQ(u,v,lname);

Parameters:

u, v - expressions of the type 'cliscalar', 'clibasmon', 'climon', or 'clipolynom'
lname - (optional) argument of type name, symbol, matrix, array, or
`&*`(numeric,{name,symbol,array,matrix})

Description:

- Procedure 'LCQ' defines a *left contraction* in Cl(Q) between a multivector u acting from the left on a multivector v.
- When used with the third optional argument, e.g., K or -K, it computes left contraction with respect to the diagonal entries of K or -K, respectively. Otherwise, it computes with respect to the diagonal entries of B.
- This procedure is used to define Clifford multiplication 'cmulQ' in Cl(Q). See cmulQ for more help.
- This procedure is obsolete in version 6 of CLIFFORD as the procedure Clifford:-LC can accept a third argument of type name, symbol, matrix, or array, that is, it can compute the left contraction LC(u,v,K) of v by u from the left with respect to the explicitly specified third argument.

Examples:

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

Example 1: Contraction with respect to an unassigned name:

```
> LCQ(e1+2*e2,3*e1we3+b*e2we3);     # contraction in Cl(B)
  LCQ(e1+2*e2,3*e1we3+b*e2we3,B);   # contraction in Cl(B)
  LCQ(e1+2*e2,3*e1we3+b*e2we3,-B);  # contraction in Cl(-B)
  LCQ(e1+2*e2,3*e1we3+b*e2we3,K);  # contraction in Cl(K)
  LCQ(e1+2*e2,3*e1we3+b*e2we3,-K);  # contraction in Cl(-K)
```

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

$$3\,B_{1,1}\,e3 + 2\,b\,B_{2,2}\,e3$$

$$3\,B_{1,1}\,e3 + 2\,b\,B_{2,2}\,e3$$

$$-3\,B_{1,1}\,e3 - 2\,b\,B_{2,2}\,e3$$

$$3\,K_{1,1}\,e3 + 2\,b\,K_{2,2}\,e3$$

$$-3\,K_{1,1}\,e3 - 2\,b\,K_{2,2}\,e3$$

```
> LCQ(e1+4,3*Id+2*eiwej);
```

$$2\,B_{1,i}\,ej - 2\,B_{1,j}\,ei + 12\,Id + 8\,eiwej$$

```
> LCQ(es,eiwej+ekwel);
```

$$B_{s,i}\, ej - B_{s,j}\, ei + B_{s,k}\, el - B_{s,l}\, ek$$

```
> LCQ(e2+3*e1,2*Pi*Id);
```

$$0$$

```
>
```

Example 2: Contraction with respect to an assigned name:

```
> B,K:=matrix(3,3,[a,b,c,d,m,f,g,h,k]),matrix(3,3,[a,0,0,0,m,0,0,
  0,k]);
```

$$B, K := \begin{bmatrix} a & b & c \\ d & m & f \\ g & h & k \end{bmatrix}, \begin{bmatrix} a & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & k \end{bmatrix}$$

```
> LCQ(e1+2*e2,3*e1we3+b*e2we3);     # contraction in Cl(B)
  LCQ(e1+2*e2,3*e1we3+b*e2we3,B);   # contraction in Cl(B)
  LCQ(e1+2*e2,3*e1we3+b*e2we3,-B);  # contraction in Cl(-B)
  LCQ(e1+2*e2,3*e1we3+b*e2we3,K);   # contraction in Cl(K)
  LCQ(e1+2*e2,3*e1we3+b*e2we3,-K);  # contraction in Cl(-K)
```

$$3\, a\, e3 + 2\, b\, m\, e3$$

$$3\, a\, e3 + 2\, b\, m\, e3$$

$$-3\, a\, e3 - 2\, b\, m\, e3$$

$$3\, a\, e3 + 2\, b\, m\, e3$$

$$-3\, a\, e3 - 2\, b\, m\, e3$$

```
>
```

See Also: Clifford:-RC, Clifford:-RCQ, Clifford:-LC, Clifford:-cmulQ, Clifford:-`type/clipolynom`, Clifford:-`type/climon`, Clifford:-`type/clibasmon`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-makealiases - define aliases for basis Clifford monomials

Calling Sequence:

makealiases(p);
makealiases(p,s);

Parameters:

p - a positive integer larger than 1
s - (optional) symbol 'ordered'

Description:

- Procedure 'makealiases' allows the user to alias basis monomials in a Clifford algebra Cl(V), e.g., to alias e1we2 as e12, or e2we1 as e21. The procedure accepts a positive integer p>1 where p denotes the dimension of the vector space V. A practical limitation on p is of course the amount of memory Maple will allocate to store these aliases since every basis monomial, not necessarily written in the standard order, will be aliased. This procedure is intended to be used when p < 5 although it can be used also when p < 10. Remember that to unalias e12 one needs to either restart Maple or simply assign e12:='e12'.
- As a memory saving feature, 'ordered' may be entered as a second parameter. If the second parameter is used, aliases are created only for monomials with ordered indices, for example, e12 will be an alias for e1we2. It is advisable to use this extra parameter for p values larger than 5.
- The procedure returns a list of aliases to be defined so they can be seen by the user. In order to finish the definition process, use 'eval' as shown below.
- Once these new aliases have been defined, they may be used in the user's input.
- See also procedure Cliplus:-makeclialiases in the supplementary package 'Cliplus' which creates aliases for Clifford products rather than for the wedge products. For example, element e1 &c e2 (or cmul(e1,e2)) will be aliased as e12.

Examples:

```
> restart:with(Clifford):
> makealiases(3);
```

$$\text{alias}(e12 = e1we2, e21 = e2we1, e13 = e1we3, e31 = e3we1, e23 = e2we3, e32 = e3we2, e123 = e1we2we3, e132 = e1we3we2, e213 = e2we1we3, e231 = e2we3we1, e312 = e3we1we2, e321 = e3we2we1)$$

```
> eval(%);
```

$$e12, e21, e13, e31, e23, e32, e123, e132, e213, e231, e312, e321$$

```
> cmulQ(1+2*e12,-2*e123+3);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$-2\, e123 + 4\, B_{2,2}\, B_{1,1}\, e3 + 3\, Id + 6\, e12$$

```
> restart:with(Clifford):
> makealiases(3,'ordered');
```

$$\text{alias}(e12 = e1we2, e13 = e1we3, e23 = e2we3, e123 = e1we2we3)$$

```
> eval(%);
```

$$e12, e13, e23, e123$$

```
> cmulQ(1+2*e12,-2*e123+3);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$-2\, e123 + 4\, B_{2,2}\, B_{1,1}\, e3 + 3\, Id + 6\, e12$$

```
>
```

See Also: Cliplus:-makeclialiases

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-makeclibasmon - creates a Grassmann monomial with the given indices

Calling Sequence:

makeclibasmons(L);

Parameters:

L - list of indices

Description:

- Procedure 'makeclibasmon' creates a basis Grassmann monomial that is, an element of `type/clibasmon` .
- The desired indices are entered as a list L. The list can be empty, in which case procedure returns the identity element 'Id' in Ext(V) and Cl(V,B). If the list is not empty, it is expected to contain distinct positive integers between 1 and 9 inclusive, or distinct one-character strings for symbolic indices. To achieve speed, this procedure does not check whether indices are one-character strings or whether the numeric indices are positive integers as indicated. It has a remember table.
- If the indices are not as described, procedure cliparse which parses user's input for errors, will return an error message (see below).
- Procedure reorder can be used to reorder indices of a basis Grassmann monomial. It also computes the sign of the permutation that reorders the indices and puts it in front of the result.
- Procedure extract is a reverse procedure to makeclibasmon in that it extracts indices from a basis Grassmann monomial.
- The basis elements can be aliased, e.g., 'e1we2' can be aliased as 'e12' with the procedure makealiases.

Examples:

```
> restart:with(Clifford):
> makeclibasmon([]);
```

$$Id$$

```
> makeclibasmon([2,1,4,5]);
```

$$e2we1we4we5$$

```
> cliparse(%);
```

$$true$$

```
> makeclibasmon([1,4,2,j,i,3]);
```

$$e1we4we2wejweiwe3$$

```
> reorder(%);
```

$$-e1we2we3we4weiwej$$

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

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

```
> map(extract,L);
```

$$[[\ ], [1], [2], [3], [1, 2], [1, 3], [2, 3], [1, 2, 3]]$$

```
> map(makeclibasmon,%);
```

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

```
>
```

See Also: Clifford:-extract, Clifford:-`type/clibasmon`, Clifford:-reorder, Clifford:-wedge, Clifford:-makealiases

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-matKrepr - find a matrix representation of Cl(Q) over a field K in a minimal ideal S

Calling Sequence:

matKrepr();
matKrepr(p);
matKrepr(p,s);
matKrepr([p,q]);
matKrepr([p,q],s);

Parameters:

p - (optional) Clifford polynomial whose matrix representation over a field K is to be found
s - (optional) symbol 'left' or 'right' (or left, right)
[p,q] - (optional) signature of some form Q

Description:

- Procedure 'matKrepr' uses matrices of basis 1-vectors computed for a left (or right) spinor representation of Cl(Q) to find a matrix representing any Clifford polynomial p from Cl(Q). It can also display matrices of 1-vectors for a specified signature which are stored in the six data files described below.
- The matrix representation of p is given in a left minimal ideal of Cl(Q) if no optional symbol is entered or if 'left' is entered. If the symbol 'right' is entered, the representation of p is in a right minimal ideal.
- The minimal ideals, in which these matrices are found, have been generated by the primitive idempotents stored by the procedure 'clidata' under clidata([p,q])[4] for the given signature (p,q). See clidata for more help.
- The procedure checks if the bilinear form B has been defined as a diagonal matrix. Then, it loads automatically an appropriate file depending on the signature of B. If B has not been defined, an error message is returned.
- When used without an argument, the procedure displays matrices of the basis 1-vectors for the current B (provided it has been defined) in a left minimal ideal generated by the idempotent stored under clidata()[4].
- User may view matrices of 1-vectors which are stored in the data files by entering a desired signature [p,q] as the first argument. If no second argument is entered, or if 'left' is entered as the second argument, matrices representing 1-vectors in a minimal left ideal will be displayed. If the second argument is 'right', matrices representing 1-vectors in a minimal right ideal will be displayed.
- Depending on the signature (p,q) of the quadratic form Q, the matrices of 1-vectors, which have been computed with the procedure 'spinorKrepr' (see spinorKrepr for more help) for most signatures in both left and right minimal ideals, are:

- real if (p - q) mod 8 is 0, 1, 2;
- complex if (p - q) mod 8 is 3 or 7;
- quaternionic if (p - q) mod 8 is 4, 5, or 6.

- The matrices of 1-vectors in dimensions from 2 to 8 are stored in tables which have been saved as Maple .m files named as follows:

- for minimal left ideals:

  real matrices are stored in a table 'matrealL' which has been saved in a file 'matrealL.m',

  complex matrices are stored in a table 'matcompL' which has been saved in a file 'matcompL.m',

  quaternionic matrices are stored in a table 'matquatL' which has been saved in a file 'matquatL.m'.

- for minimal right ideals:

  real matrices are stored in a table 'matrealR' which has been saved in a file 'matrealR.m',

  complex matrices are stored in a table 'matcompR' which has been saved in a file 'matcompR.m',

  quaternionic matrices are stored in a table 'matquatR' which has been saved in a file 'matquatR.m'.

- Note that it is not necessary to load any of these files manually since 'matKrepr' can do it automatically. One reason why one would load manually any of these files would be to see its indices and/or matrices of 1-vectors.
- For more information on Maple tables and the read command try read, table, indices, entries.
- Matrices representing Clifford polynomials in minimal ideals are generally computed with 'matKrepr' much faster than with 'spinorKrepr' because the former uses already stored matrices of 1-vectors and the procedure 'rmulm' (see rmulm for more help) to compute matrices representing other basis monomials and polynomials. Note however that when standard bases are entered into 'spinorKrepr', it invokes 'matKrepr' for speedy computation.
- This procedure is linear. It checks the signature of the currently defined bilinear form B and retrieves matrices for that signature from one of the files. Then, it computes with 'rmulm' the matrix that represents the polynomial p.

Comments:

- In the examples below it has been assumed that the data files 'matrealL.m', 'matrealR.m', 'matcompL.m', 'matcompR.m', 'matquatL.m', and 'matquatR.m' have been installed in the Clifford library given in libname[1].
- Set global variable _prolevel to true to speed up computations. You can find more about _prolevel in Clifford:-setup.

Examples:

```
> restart:with(Clifford):eval(makealiases(6)):
  _default_Clifford_product;
```

```
useproduct(cmulNUM);
_default_Clifford_product;
```

Clifford:-cmulRS

```
Warning, cmul will use cmulNUM; for help see pages ?cmul, ?Clifford:-intro, or
?cmulNUM
```

Clifford:-cmulNUM

Example 1: Let's see the indices of 'matrealL' and signatures for which real matrices have been computed in S=Cl(Q)f:

```
> indices(matrealL); #display indices of 'matrealL'
```

[[9, 0]], [[4, 2]], [[2, 1]], [[4, 3]], [[5, 3]], [[0, 8]], [[1, 8]], [[3, 3]], [[0, 7]], [[0, 6]], [[1, 7]], [[3, 2]], [[3, 1]], [[1, 1]], [[4, 4]], [[8, 0]], [[5, 4]], [[2, 2]], [[2, 0]]

Show Dirac gamma matrices representing basis 1-vectors in the signature (3,1):

```
> matrealL[[3,1]];
```

$$\left[e1=\begin{bmatrix}1&0&0&0\\0&-1&0&0\\0&0&-1&0\\0&0&0&1\end{bmatrix}, e2=\begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}, e3=\begin{bmatrix}0&0&1&0\\0&0&0&-1\\1&0&0&0\\0&-1&0&0\end{bmatrix}, e4=\begin{bmatrix}0&0&-1&0\\0&0&0&1\\1&0&0&0\\0&-1&0&0\end{bmatrix}\right]$$

Show matrices representing basis 1-vectors in the signature (2,0):

```
> matrealL[[2,0]];
```

$$\left[e1=\begin{bmatrix}1&0\\0&-1\end{bmatrix}, e2=\begin{bmatrix}0&1\\1&0\end{bmatrix}\right]$$

```
>
```

Example 2: Let's see the indices of 'matcompL' and signatures for which complex matrices have been computed in S=Cl(Q)f.

```
> indices(matcompL); #display indices of 'matcompL'
```

[[3, 0]], [[2, 3]], [[4, 1]], [[1, 6]], [[2, 7]], [[0, 9]], [[8, 1]], [[5, 2]], [[7, 0]], [[0, 5]], [[3, 4]], [[6, 3]], [[4, 5]], [[1, 2]]

Show Pauli matrices representing basis 1-vectors in the signature (3,0) over the field K spanned by [Id,e2we3]:

```
> matcompL[[3,0]];
```

$$\left[e1=\begin{bmatrix}1&0\\0&-1\end{bmatrix}, e2=\begin{bmatrix}0&1\\1&0\end{bmatrix}, e3=\begin{bmatrix}0&-e23\\e23&0\end{bmatrix}\right]$$

Show gamma matrices representing basis 1-vectors in the signature (2,3) over the field K spanned by [Id,e3]:

```
> matcompL[[2,3]];
```

$$\left[e1=\begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}, e2=\begin{bmatrix}0&0&1&0\\0&0&0&-1\\1&0&0&0\\0&-1&0&0\end{bmatrix}, e3=\begin{bmatrix}e3&0&0&0\\0&-e3&0&0\\0&0&-e3&0\\0&0&0&e3\end{bmatrix}, e4=\begin{bmatrix}0&-1&0&0\\1&0&0&0\\0&0&0&-1\\0&0&1&0\end{bmatrix},\right.$$

$$e5 = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}\Bigg]$$

```
>
```

Example 3: Let's see the indices of 'matquatL' and signatures for which quaternionic matrices have been computed in S=Cl(Q)f.

```
> indices(matquatL); #display indices of 'matquatL'
```

[[6, 0]], [[0, 4]], [[2, 6]], [[6, 2]], [[7, 1]], [[7, 2]], [[2, 4]], [[2, 5]], [[6, 1]], [[5, 1]], [[1, 3]], [[0, 3]], [[1, 4]], [[3, 6]], [[3, 5]], [[0, 2]], [[1, 5]], [[4, 0]], [[5, 0]]

Show matrices representing basis 1-vectors in the signature (1,3) over the field K spanned by [Id,e2,e3,e2we3]:

```
> matquatL[[1,3]];
```

$$\left[e1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, e2 = \begin{bmatrix} e2 & 0 \\ 0 & -e2 \end{bmatrix}, e3 = \begin{bmatrix} e3 & 0 \\ 0 & -e3 \end{bmatrix}, e4 = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\right]$$

Show matrices representing basis 1-vectors in the signature (5,1) over the field K spanned by [Id,e2we3,e2we4,e3we4]:

```
> matquatL[[5,1]];
```

$$\left[e1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, e2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, e3 = \begin{bmatrix} 0 & -e23 & 0 & 0 \\ e23 & 0 & 0 & 0 \\ 0 & 0 & 0 & -e23 \\ 0 & 0 & e23 & 0 \end{bmatrix},\right.$$

$$\left. e4 = \begin{bmatrix} 0 & -e24 & 0 & 0 \\ e24 & 0 & 0 & 0 \\ 0 & 0 & 0 & -e24 \\ 0 & 0 & e24 & 0 \end{bmatrix}, e5 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}, e6 = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}\right]$$

```
>
```

Example 4: To see matrices of 1-vectors for the currently defined B in a left minimal ideal, enter:

```
> restart:bench:=time():with(Clifford):
  B:=linalg[diag](1,1,1,1):  #defining B of signature (4,0)
> matKrepr(); #these matrices are
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$\left[e1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, e2 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, e3 = \begin{bmatrix} 0 & -e2we3 \\ e2we3 & 0 \end{bmatrix}, e4 = \begin{bmatrix} 0 & -e2we4 \\ e2we4 & 0 \end{bmatrix}\right]$$

```
> matKrepr(e1,'right'); #to see the matrix of e1 in a right ideal
```

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

```
> matKrepr(e2,'right'); #to see the matrix of e2 in a right ideal
```

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

```
> matKrepr(e3,'right'); #to see the matrix of e3 in a right ideal
```

$$\begin{bmatrix} 0 & e2we3 \\ -e2we3 & 0 \end{bmatrix}$$

```
> matKrepr(e4,'right'); #to see the matrix of e4 in a right ideal
```

$$\begin{bmatrix} 0 & e2we4 \\ -e2we4 & 0 \end{bmatrix}$$

```
> ucoeffs:=[u||(0..(2^(linalg[coldim](B))-1))]:
> u:=add(ucoeffs[k]*cbasis(linalg[coldim](B))[k],k=1..nops(ucoeff
  s)):
> matKrepr(u,'left');
```

$$[u0 + u1 + u8\ e2we3 + u9\ e2we4 + u10\ e3we4 + u11\ e2we3 + u12\ e2we4 + u13\ e3we4\ ,$$
$$u2 - u3\ e2we3 - u4\ e2we4 + u5 - u6\ e2we3 - u7\ e2we4 + u14\ e3we4 + u15\ e3we4]$$
$$[u2 + u3\ e2we3 + u4\ e2we4 - u5 - u6\ e2we3 - u7\ e2we4 + u14\ e3we4 - u15\ e3we4\ ,$$
$$u0 - u1 - u8\ e2we3 - u9\ e2we4 + u10\ e3we4 + u11\ e2we3 + u12\ e2we4 - u13\ e3we4]$$

```
> map(clicollect,%);
```

$$[(u0 + u1)\ Id + (u8 + u11)\ e2we3 + (u9 + u12)\ e2we4 + (u10 + u13)\ e3we4\ ,$$
$$(u2 + u5)\ Id - (u3 + u6)\ e2we3 - (u4 + u7)\ e2we4 + (u14 + u15)\ e3we4]$$
$$[(u2 - u5)\ Id + (u3 - u6)\ e2we3 + (u4 - u7)\ e2we4 + (u14 - u15)\ e3we4\ ,$$
$$(u0 - u1)\ Id - (u8 - u11)\ e2we3 - (u9 - u12)\ e2we4 + (u10 - u13)\ e3we4]$$

```
>
```

Example 5: Let's find a matrix representing some Clifford polynomial in the Clifford algebra Cl(1,3) isomorphic to H(2):

```
> B:=linalg[diag](1,-1,-1,-1): #defining B of signature (1,3)
> p:=a+2*e1-e3+e2we3+e4we1;
```

$$p := a + 2\ e1 - e3 + e2we3 + e4we1$$

```
> matKrepr(p); #finding the matrix of p in a left minimal ideal
```

$$\begin{bmatrix} a - e3 + e2we3 - 1 & 2 \\ 2 & a + e3 + e2we3 + 1 \end{bmatrix}$$

```
> matKrepr(p,'left'); #finding the matrix of p in a left minimal
  ideal
```

$$\begin{bmatrix} a - e3 + e2we3 - 1 & 2 \\ 2 & a + e3 + e2we3 + 1 \end{bmatrix}$$

```
> matKrepr(p,'right'); #finding the matrix of p in a right
  minimal ideal
```

$$\begin{bmatrix} a - e3 + e2we3 - 1 & 2 \\ 2 & a + e3 + e2we3 + 1 \end{bmatrix}$$

```
>
```

Example 6: Let's find a matrix representing some Clifford polynomial in the Clifford algebra Cl(2,3) isomorphic to C(4):

```
> B:=linalg[diag](1,1,-1,-1,-1):  #defining B of signature (2,3)
```

```
> p:=2*e1we2-e2we3we4+1;
```

$$p := 2\,e1we2 - e2we3we4 + 1$$

```
> matKrepr(p,'right'); #in a right minimal ideal
```

$$\begin{bmatrix} 1 & 0 & 0 & -2-e3 \\ 0 & 1 & 2+e3 & 0 \\ 0 & -2+e3 & 1 & 0 \\ 2-e3 & 0 & 0 & 1 \end{bmatrix}$$

```
> matKrepr(p,'left'); #in a left minimal ideal
```

$$\begin{bmatrix} 1 & 0 & 0 & -2-e3 \\ 0 & 1 & 2+e3 & 0 \\ 0 & -2+e3 & 1 & 0 \\ 2-e3 & 0 & 0 & 1 \end{bmatrix}$$

```
>
```

Example 7: One can also see matrices representing basis 1-vectors in a left or right minimal ideal S in the Clifford algebra Cl(Q) where the quadratic form has a specific signature (p,q). Just enter the signature in a form of a list as the first argument, and 'left' or 'right' as the second argument. If no matrices have been computed for that signature, an error message is returned:

```
> matKrepr([1,3]); #matrices of 1-vectors of Cl(1,3)
```

$$\left[e1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, e2 = \begin{bmatrix} e2 & 0 \\ 0 & -e2 \end{bmatrix}, e3 = \begin{bmatrix} e3 & 0 \\ 0 & -e3 \end{bmatrix}, e4 = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\right]$$

```
> matKrepr([1,3],'left');#matrices of 1-vectors of Cl(1,3) in a
  left ideal
```

$$\left[e1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, e2 = \begin{bmatrix} e2 & 0 \\ 0 & -e2 \end{bmatrix}, e3 = \begin{bmatrix} e3 & 0 \\ 0 & -e3 \end{bmatrix}, e4 = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\right]$$

```
> matKrepr([1,3],'right');#matrices of 1-vectors of Cl(1,3) in a
  right ideal
```

$$\left[e1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, e2 = \begin{bmatrix} e2 & 0 \\ 0 & -e2 \end{bmatrix}, e3 = \begin{bmatrix} e3 & 0 \\ 0 & -e3 \end{bmatrix}, e4 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\right]$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 1.592000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

Example 8: Semisimple Clifford algebras.

```
> restart:bench:=time():with(Clifford):
> all_sigs(1..9,'quat','semisimple');
```

$$[[0,3],[1,4],[2,5],[3,6],[5,0],[6,1],[7,2]]$$

```
> p:=1:q:=4:
> clidata([p,q]);
```

$$\left[quaternionic, 2, semisimple, 'cmulQ'\left(\frac{Id}{2} + \frac{e2we3we4}{2}, \frac{Id}{2} + \frac{e1we5}{2}\right),\right.$$

$$[Id, e1, e2, e3, e4, e1we2, e1we3, e1we4], [Id, e2, e3, e2we3], [Id, e1]\Big]$$

```
> B:=linalg[diag](1$p,-1$q);
```

$$B := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

```
> matKrepr();
```

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

$$\Big[e1 = \begin{bmatrix} [0,0] & [1,1] \\ [1,1] & [0,0] \end{bmatrix}, e2 = \begin{bmatrix} [e2,e2] & [0,0] \\ [0,0] & [-e2,-e2] \end{bmatrix}, e3 = \begin{bmatrix} [e3,e3] & [0,0] \\ [0,0] & [-e3,-e3] \end{bmatrix},$$
$$e4 = \begin{bmatrix} [-e2we3, e2we3] & [0,0] \\ [0,0] & [e2we3, -e2we3] \end{bmatrix}, e5 = \begin{bmatrix} [0,0] & [-1,-1] \\ [1,1] & [0,0] \end{bmatrix}\Big]$$

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

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

```
> for g in clibasis do M[g]:=matKrepr(g) od;
```

$$M_{Id} := \begin{bmatrix} [1,1] & [0,0] \\ [0,0] & [1,1] \end{bmatrix}$$

$$M_{e1} := \begin{bmatrix} [0,0] & [1,1] \\ [1,1] & [0,0] \end{bmatrix}$$

$$M_{e2} := \begin{bmatrix} [e2,e2] & [0,0] \\ [0,0] & [-e2,-e2] \end{bmatrix}$$

$$M_{e3} := \begin{bmatrix} [e3,e3] & [0,0] \\ [0,0] & [-e3,-e3] \end{bmatrix}$$

$$M_{e1we2} := \begin{bmatrix} [0,0] & [-e2,-e2] \\ [e2,e2] & [0,0] \end{bmatrix}$$

$$M_{e1we3} := \begin{bmatrix} [0,0] & [-e3,-e3] \\ [e3,e3] & [0,0] \end{bmatrix}$$

$$M_{e2we3} := \begin{bmatrix} [e2we3, e2we3] & [0,0] \\ [0,0] & [e2we3, e2we3] \end{bmatrix}$$

$$M_{e1we2we3} := \begin{bmatrix} [0,0] & [e2we3, e2we3] \\ [e2we3, e2we3] & [0,0] \end{bmatrix}$$

```
> matKrepr(1+2*e2-2*e2we3);
```

$$\begin{bmatrix} [1+2\,e2-2\,e2we3, 1+2\,e2-2\,e2we3] & [0,0] \\ [0,0] & [1-2\,e2-2\,e2we3, 1-2\,e2-2\,e2we3] \end{bmatrix}$$

```
> ucoeffs:=[u||(0..(2^(linalg[coldim](B))-1))]:
> u:=add(ucoeffs[k]*cbasis(linalg[coldim](B))[k],k=1..nops(ucoeff
  s)):
> evalm(B);
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

```
> matKrepr(u,'left');
```

$$[[-u29\ e2 + u11\ e3 + u10\ e2we3 + u2\ e2 + u0 + u3\ e3 - u31 - u13\ e2 - u20\ e3 + u27\ e2we3 + u9 + u21\ e2we3 + u28\ e3 - u18\ e2 + u22 - u4\ e2we3, u29\ e2 - u11\ e3 + u10\ e2we3 + u2\ e2 + u0 + u3\ e3 + u31 + u13\ e2 - u20\ e3 + u27\ e2we3 + u9 - u21\ e2we3 - u28\ e3 - u18\ e2 - u22 + u4\ e2we3], [-u14\ e3 - u19\ e2 + u1 + u15\ e2we3 - u26 - u30 + u25\ e2 - u7\ e3 - u5 - u6\ e2 + u8\ e2we3 + u16\ e2we3 - u12\ e2 - u24\ e3 + u17\ e3 - u23\ e2we3, -u14\ e3 + u19\ e2 + u1 - u15\ e2we3 + u26 + u30 - u25\ e2 - u7\ e3 - u5 - u6\ e2 - u8\ e2we3 + u16\ e2we3 - u12\ e2 + u24\ e3 - u17\ e3 - u23\ e2we3]]$$

$$[[-u14\ e3 - u19\ e2 + u1 + u15\ e2we3 + u26 - u30 - u25\ e2 + u7\ e3 + u5 + u6\ e2 - u8\ e2we3 + u16\ e2we3 - u12\ e2 + u24\ e3 + u17\ e3 + u23\ e2we3, -u14\ e3 + u19\ e2 + u1 - u15\ e2we3 - u26 + u30 + u25\ e2 + u7\ e3 + u5 + u6\ e2 + u8\ e2we3 + u16\ e2we3 - u12\ e2 - u24\ e3 - u17\ e3 + u23\ e2we3], [u29\ e2 + u11\ e3 + u10\ e2we3 - u2\ e2 + u0 - u3\ e3 - u31 - u13\ e2 - u20\ e3 - u27\ e2we3 - u9 + u21\ e2we3 - u28\ e3 - u18\ e2 - u22 + u4\ e2we3, -u29\ e2 - u11\ e3 + u10\ e2we3 - u2\ e2 + u0 - u3\ e3 + u31 + u13\ e2 - u20\ e3 - u27\ e2we3 - u9 - u21\ e2we3 + u28\ e3 - u18\ e2 + u22 - u4\ e2we3]]$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 2.671000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

Comments:

- The above examples may be redone for right minimal ideals if option 'right' is used as in Example 6.

See Also:

Clifford:-adfmatrix, Clifford:-mdfmatrix, Clifford:-spinorKrepr, Clifford:-rmulm, Clifford:-spinorKbasis, Clifford:-Kfield, Clifford:-Bsignature, Clifford:-`type/primitiveidemp`, Clifford:-`type/clipolynom`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-maxgrade - find the maximum grade in a Clifford polynomial

Calling Sequence:

maxgrade(p);

Parameters:

p - expression of the type 'cliscalar' or 'clipolynom'

Description:

- This useful procedure finds the greatest grade in the given Clifford polynomial. See `type/cliscalar` and `type/clipolynom` for more help on the argument types.
- The greatest index is returned as a numeral.
- It returns 0 for a Clifford scalar, i.e., an element of the type 'cliscalar'.

Examples:

```
> restart:with(Clifford):
> maxgrade(3*Pi*e2we1we3+4-e2we1);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$3$$

```
> maxgrade(a*B[i,j]*ejwei+3*ekwei);
```

$$2$$

```
> maxgrade(4*e1wej+ekwe3+2*e2we1+e1we2we3we4);
```

$$4$$

```
> maxgrade(4*alpha*Pi);
```

$$0$$

```
> maxgrade(4*Id+8);
```

$$0$$

```
> maxgrade(eiwej);
```

$$2$$

```
>
```

See Also: Clifford:-maxindex

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-maxindex - find the maximum index in a Clifford polynomial, list, or set

Calling Sequence:

maxindex(p);

Parameters:

p - expression of the type 'cliscalar', 'clipolynom', 'list', or 'set'

Description:

- This procedure finds the greatest index in the given Clifford polynomial or in the given list or set of Clifford polynomials. See `type/cliscalar` and `type/clipolynom` for more help on the argument types.
- The greatest index is returned as a numeral.
- It returns 0 for a Clifford scalar, i.e., an element of the type 'cliscalar'.
- This procedure is linear. It is useful, for example, when determining the smallest Clifford algebra in which the given polynomial might have an inverse.
- If input contains a symbolic index, then an error message is returned.

Examples:

```
> restart:with(Clifford):
> maxindex(3*Pi*e2we1we3+4-e2we1);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$3$$

```
> maxindex(a*B[i,j]*ejwei+3*ekwei); ###<<<-Intended error message
Error, (in Clifford:-maxindex) cannot determine maximum index because input
contains symbolic index or indices
> maxindex(4*e1wej+ekwe3+2*e2we1);  ###<<<-Intended error message
Error, (in Clifford:-maxindex) cannot determine maximum index because input
contains symbolic index or indices
> maxindex([e1we2,e4,e3we5]);
```

$$5$$

```
> maxindex({e1we2+e7,e4,e3we6});
```

$$7$$

```
> maxindex([e1we2,e2we3]);
```

$$3$$

```
>
```

See Also: Clifford:-maxgrade, Clifford:-extract

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-adfmatrix, Clifford:-mdfmatrix - add or multiply matrices over double field

Calling Sequence:

adfmatrix(m1,m2) - add two matrices m1 and m2 of `type/dfmatrix` and of the same dimension n x n,
mdfmatrix(m1,m2) - multiply two matrices m1 and m2 of `type/dfmatrix` and of the same dimension n x n,

Parameters:

m1,m2 - matrices of `type/dfmatrix` of the same size

Description:

- A matrix M is of type 'ddfmatrix' if it is a square matrix whose entries are two element lists. These matrices arise when faithful matrix representations of semi-simple Clifford algebras are calculated. Use all_sigs to display signatures of semisimple Clifford algebras in dimensions 1 through 9.
- Faithful spinor representations of semisimple Clifford algebras have been precomputed. They are stored in a library file and can be retrieved with the procedure clidata.

Examples:

```
> restart:bench:=time():
  with(Clifford):with(linalg):
  _default_Clifford_product;
  #useproduct(cmulNUM);
  _default_Clifford_product;
```

$$Clifford{:}\text{-}cmulRS$$

$$Clifford{:}\text{-}cmulRS$$

Let's display signatures of semi-simple Clifford algebras in dimensions 1 through 9 which are isomorphic to rings of matrices over double real field R+R:

```
> all_sigs(1..9,'real','semisimple');
```

$$[[0, 7], [1, 0], [1, 8], [2, 1], [3, 2], [4, 3], [5, 4], [9, 0]]$$

Let's display signatures of semi-simple Clifford algebras in dimensions 1 through 9 which are isomorphic to rings of matrices over double quaternionic field H+H:

```
> all_sigs(1..9,'quat','semisimple');
```

$$[[0, 3], [1, 4], [2, 5], [3, 6], [5, 0], [6, 1], [7, 2]]$$

There are no semisimple Clifford algebras that would be isomorphic to rings of matrices of double complex field C+C:

```
> all_sigs(1..9,'complex','semisimple');
```

$$[\ ]$$

More information about Clifford algebra Cl(Q) of the quadratic form of signature (2,1) can be

found using procedure clidata as follows:

```
> clidata([2,1]);
```

$$\left[real, 2, semisimple, 'cmulQ'\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e2we3}{2}\right), [Id, e2], [Id], [Id, e2]\right]$$

Example 1: Let's view matrices m[i], i=1..3, representing 1-vectors {e1,e2,e3} in Cl(2,1) = Mat(2,2,R+R). These matrices are of type `type/dfmatrix` and have been precomputed. They can be displayed with the procedure matKrepr:

```
> pq:=[2,1];
  B:=diag(1$pq[1],-1$pq[2]);
```

$$pq := [2, 1]$$

$$B := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

```
> L:=matKrepr();
```

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

$$L := \left[e1 = \begin{bmatrix} [1, -1] & [0, 0] \\ [0, 0] & [-1, 1] \end{bmatrix}, e2 = \begin{bmatrix} [0, 0] & [1, 1] \\ [1, 1] & [0, 0] \end{bmatrix}, e3 = \begin{bmatrix} [0, 0] & [-1, -1] \\ [1, 1] & [0, 0] \end{bmatrix}\right]$$

Let's assign these matrices to m[i], i=1..3:

```
> for i from 1 to nops(L) do m[i]:=rhs(L[i]) od;
```

$$m_1 := \begin{bmatrix} [1, -1] & [0, 0] \\ [0, 0] & [-1, 1] \end{bmatrix}$$

$$m_2 := \begin{bmatrix} [0, 0] & [1, 1] \\ [1, 1] & [0, 0] \end{bmatrix}$$

$$m_3 := \begin{bmatrix} [0, 0] & [-1, -1] \\ [1, 1] & [0, 0] \end{bmatrix}$$

Observe that matrices m[1], m[2], and m[3] satisfy appropriate relations:

(A) Squares of m[1], m[2], and m[3] are:

```
> mdfmatrix(m[1],m[1]),mdfmatrix(m[2],m[2]),mdfmatrix(m[3],m[3]);
```

$$\begin{bmatrix} [1, 1] & [0, 0] \\ [0, 0] & [1, 1] \end{bmatrix}, \begin{bmatrix} [1, 1] & [0, 0] \\ [0, 0] & [1, 1] \end{bmatrix}, \begin{bmatrix} [-1, -1] & [0, 0] \\ [0, 0] & [-1, -1] \end{bmatrix}$$

(B) Pair-wise, matrices m[1], m[2], and m[3] anticommute:

```
> adfmatrix(mdfmatrix(m[1],m[2]),mdfmatrix(m[2],m[1]));
```

$$\begin{bmatrix} [0, 0] & [0, 0] \\ [0, 0] & [0, 0] \end{bmatrix}$$

```
> adfmatrix(mdfmatrix(m[1],m[3]),mdfmatrix(m[3],m[1]));
```

$$\begin{bmatrix} [0, 0] & [0, 0] \\ [0, 0] & [0, 0] \end{bmatrix}$$

```
> adfmatrix(mdfmatrix(m[2],m[3]),mdfmatrix(m[3],m[2]));
```

$$\begin{bmatrix} [0,0] & [0,0] \\ [0,0] & [0,0] \end{bmatrix}$$

Thus, we can find a unique matrix representing each basis Grassmann monomial in Cl(Q) as follows. Let's define a homomorphism f from Cl(Q) to Mat(2,2,R+R) as a Maple function:

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

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

```
> f:=proc() end:
  f(Id):=mdfmatrix(m[1],m[1]):
  f(e1):=evalm(m[1]):
  f(e2):=evalm(m[2]):
  f(e3):=evalm(m[3]):
  f(e1we2):=mdfmatrix(m[1],m[2]):
  f(e1we3):=mdfmatrix(m[1],m[3]):
  f(e2we3):=mdfmatrix(m[2],m[3]):
  f(e1we2we3):=mdfmatrix(mdfmatrix(m[1],m[2]),m[3]):
```

Then, we can apply f to the entries of clibas:

```
> for x in clibas do
  x, " is mapped by f to ", f(x);
  od;
```

$$Id, \text{" is mapped by f to "}, \begin{bmatrix} [1,1] & [0,0] \\ [0,0] & [1,1] \end{bmatrix}$$

$$e1, \text{" is mapped by f to "}, \begin{bmatrix} [1,-1] & [0,0] \\ [0,0] & [-1,1] \end{bmatrix}$$

$$e2, \text{" is mapped by f to "}, \begin{bmatrix} [0,0] & [1,1] \\ [1,1] & [0,0] \end{bmatrix}$$

$$e3, \text{" is mapped by f to "}, \begin{bmatrix} [0,0] & [-1,-1] \\ [1,1] & [0,0] \end{bmatrix}$$

$$e1we2, \text{" is mapped by f to "}, \begin{bmatrix} [0,0] & [1,-1] \\ [-1,1] & [0,0] \end{bmatrix}$$

$$e1we3, \text{" is mapped by f to "}, \begin{bmatrix} [0,0] & [-1,1] \\ [-1,1] & [0,0] \end{bmatrix}$$

$$e2we3, \text{" is mapped by f to "}, \begin{bmatrix} [1,1] & [0,0] \\ [0,0] & [-1,-1] \end{bmatrix}$$

$$e1we2we3, \text{" is mapped by f to "}, \begin{bmatrix} [1,-1] & [0,0] \\ [0,0] & [1,-1] \end{bmatrix}$$

```
>
```

Example 2: Let's view matrices m[i], i=1..3, representing 1-vectors {e1,e2,e3} in Cl(0,3) = Mat(1,1,H+H). These matrices are of type `type/dfmatrix` and have been precomputed. They can be displayed with the procedure matKrepr:

```
> pq:=[0,3];
  B:=diag(1$pq[1],-1$pq[2]);
```

$$pq := [0, 3]$$

$$B := \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

```
> L:=matKrepr();
```

$$L := [e1 = [[e1, e1]], e2 = [[e2, e2]], e3 = [[-e1we2, e1we2]]]$$

Let's assign these matrices to m[i], i=1..3:

```
> for i from 1 to nops(L) do m[i]:=rhs(L[i]) od;
```

$$m_1 := [[e1, e1]]$$

$$m_2 := [[e2, e2]]$$

$$m_3 := [[-e1we2, e1we2]]$$

```
>
```

Observe that matrices m[1], m[2], and m[3] satisfy appropriate relations:

(A) Squares of m[1], m[2], and m[3] are:

```
> mdfmatrix(m[1],m[1]),mdfmatrix(m[2],m[2]),mdfmatrix(m[3],m[3]);
```

$$[[-1, -1]], [[-1, -1]], [[-1, -1]]$$

(B) Pair-wise, matrices m[1], m[2], and m[3] anticommute:

```
> adfmatrix(mdfmatrix(m[1],m[2]),mdfmatrix(m[2],m[1]));
```

$$[[0, 0]]$$

```
> adfmatrix(mdfmatrix(m[1],m[3]),mdfmatrix(m[3],m[1]));
```

$$[[0, 0]]$$

```
> adfmatrix(mdfmatrix(m[2],m[3]),mdfmatrix(m[3],m[2]));
```

$$[[0, 0]]$$

Thus, we can find a unique matrix representing each basis Grassmann monomial in Cl(Q) as follows. Let's define a homomorphism h from Cl(Q) to Mat(1,1,H+H) as a Maple function:

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

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

```
> h:=proc() end:
  h(Id):=mdfmatrix(m[1],m[1]):
  h(e1):=evalm(m[1]):
  h(e2):=evalm(m[2]):
  h(e3):=evalm(m[3]):
  h(e1we2):=mdfmatrix(m[1],m[2]):
  h(e1we3):=mdfmatrix(m[1],m[3]):
  h(e2we3):=mdfmatrix(m[2],m[3]):
  h(e1we2we3):=mdfmatrix(mdfmatrix(m[1],m[2]),m[3]):
```

Then, we can apply f to the entries of clibas:

```
> for x in clibas do
  x, " is mapped by h to ", h(x);
```

```
od;
```

$$Id, \text{" is mapped by h to "}, [[-1, -1]]$$
$$e1, \text{" is mapped by h to "}, [[e1, e1]]$$
$$e2, \text{" is mapped by h to "}, [[e2, e2]]$$
$$e3, \text{" is mapped by h to "}, [[-e1we2, e1we2]]$$
$$e1we2, \text{" is mapped by h to "}, [[e1we2, e1we2]]$$
$$e1we3, \text{" is mapped by h to "}, [[e2, -e2]]$$
$$e2we3, \text{" is mapped by h to "}, [[-e1, e1]]$$
$$e1we2we3, \text{" is mapped by h to "}, [[1, -1]]$$

It is easy to see from the above display, that in order for f to be an isomorphism, there is a need to represent each element in the Grassmann basis by a pair of matrices, or, equivalently, by a matrix in a double fields. For example, let's split into a pair of matrices from Mat(1,1,H) double-field matrices representing e3 and e1we2 elements:

```
> ddfmatrix(h(e3));ddfmatrix(h(e1we2));
```

$$[[-e1we2], [e1we2]]$$
$$[[e1we2], [e1we2]]$$

Notice, that the second matrix [e1we2] is identical in both pairs, or, equivalently, element e1we2 is present in both matrices

```
> h(e3),h(e1we2);
```

$$[[-e1we2, e1we2]], [[e1we2, e1we2]]$$

Thus, the assignment e3 --> e1we2, e1we2 --> e1we2 would not be an isomorphism. There is a need for another element to distinguish these two images by adding a second element. That is, e3 --> [-e1we2, e1we2] and e1we2 --> [e1we2,e1we2].

It may be worth recalling that these pairs of matrices are simply representations of the given element u from Cl(Q) in two left (or right) minimal (spinor) ideals S1 and S2 generated by the primitive idempotents f and gradeinv(f) respectively where 'gradeinv' is the grade involution in Cl(Q). We can find f and generators for S1:=Cl(Q)f from 'clidata':

```
> cdata:=clidata();
```

$$cdata := \left[quaternionic, 1, semisimple, \frac{Id}{2} + \frac{e1we2we3}{2}, [Id, e1, e2, e3], [Id, e1, e2, e1we2], [Id]\right]$$

```
> f:=cdata[4];field:=cdata[6];gens:=cdata[7];
```

$$f := \frac{Id}{2} + \frac{e1we2we3}{2}$$
$$field := [Id, e1, e2, e1we2]$$
$$gens := [Id]$$

Notice that f is an idempotent:

```
> cmul(f,f)-f;
```

$$0$$

and that f is a primitive idempotent:

```
> type(f,primitiveidemp);
```

$$true$$

Notice also that elements in the list 'field' generate a subalgebra in Cl(Q) isomorphic with the ring of quaternions. Let's display the multiplication table of these elements:

```
> M:=matrix(4,4,(i,j)->cmul(field[i],field[j]));
```

$$M := \begin{bmatrix} Id & e1 & e2 & e1we2 \\ e1 & -Id & e1we2 & -e2 \\ e2 & -e1we2 & -Id & e1 \\ e1we2 & e2 & -e1 & -Id \end{bmatrix}$$

Thus, the basis in S1=Cl(Q)f over the K = gen(field) = H is just f, that is, S1=Cl(Q)f=<f>. Recall also that S1 = Cl(Q)f is a *right* K module. Thus, we have

```
> cmul(e3,f)=cmul(f,-e1we2); #representation in S1=Cl(Q)f
```

$$-\frac{e1we2}{2}+\frac{e3}{2}=-\frac{e1we2}{2}+\frac{e3}{2}$$

which means that element '-e1we2' from K represents e3 in S1. Likewise, in S2=Cl(Q)gradeinv(f). Recall also that S1 = Cl(Q)gradeinv(f) is also a *right* K module.

```
> fg:=gradeinv(f);
```

$$fg := \frac{Id}{2}-\frac{e1we2we3}{2}$$

```
> cmul(e3,fg)=cmul(fg,e1we2); #representation in S2=Cl(Q)fg
  f(e3);
```

$$\frac{e1we2}{2}+\frac{e3}{2}=\frac{e1we2}{2}+\frac{e3}{2}$$

$$\frac{1}{2}\,\mathrm{Id}(e3)+\frac{1}{2}\,\mathrm{e1we2we3}(e3)$$

Thus, the pair (-e1we2,e1we2) represents e3 in S1+S2 (here `+` denotes the direct sum), which is the same as matrix m[3] above with an entry in H+H. Let's compute the pair representing e1we2:

```
> cmul(e1we2,f)=cmul(f,e1we2); #representation in S1=Cl(Q)f
```

$$\frac{e1we2}{2}-\frac{e3}{2}=\frac{e1we2}{2}-\frac{e3}{2}$$

```
> cmul(e1we2,fg)=cmul(fg,e1we2); #representation in S2=Cl(Q)fg
```

$$\frac{e1we2}{2}+\frac{e3}{2}=\frac{e1we2}{2}+\frac{e3}{2}$$

Thus, the pair (e1we2,e1we2) represents e1we2 in S1+S2. Let's verify in a similar manner that matrices representing e1, e2, e1we3, e2we3, and e1we2we3 shown above are correct:

```
> cmul(e1,f)=cmul(f,e1);cmul(e1,fg)=cmul(fg,e1);h(e1);
```

$$-\frac{e2we3}{2}+\frac{e1}{2}=-\frac{e2we3}{2}+\frac{e1}{2}$$

$$\frac{e2we3}{2}+\frac{e1}{2}=\frac{e2we3}{2}+\frac{e1}{2}$$

$$[[e1, e1]]$$

Thus, the pair (e1,e1) represents e1 in S1+S2, which is the same as matrix m[1] above with an entry in H+H.

```
> cmul(e2,f)=cmul(f,e2);cmul(e2,fg)=cmul(fg,e2);h(e2);
```

$$\frac{e1we3}{2}+\frac{e2}{2}=\frac{e1we3}{2}+\frac{e2}{2}$$

$$-\frac{e1we3}{2}+\frac{e2}{2}=-\frac{e1we3}{2}+\frac{e2}{2}$$

$$[[e2, e2]]$$

Thus, the pair (e2,e2) represents e2 in S1+S2, which is the same as matrix m[2] above with an entry in H+H.

```
> cmul(e1we3,f)=cmul(f,e2);cmul(e1we3,fg)=cmul(fg,-e2);h(e1we3);
```

$$\frac{e1we3}{2}+\frac{e2}{2}=\frac{e1we3}{2}+\frac{e2}{2}$$

$$\frac{e1we3}{2}-\frac{e2}{2}=\frac{e1we3}{2}-\frac{e2}{2}$$

$$[[e2, -e2]]$$

Thus, the pair (e2,-e2) represents e1we3 in S1+S2.

```
> cmul(e2we3,f)=cmul(f,-e1);cmul(e2we3,fg)=cmul(fg,e1);h(e2we3);
```

$$\frac{e2we3}{2}-\frac{e1}{2}=\frac{e2we3}{2}-\frac{e1}{2}$$

$$\frac{e2we3}{2}+\frac{e1}{2}=\frac{e2we3}{2}+\frac{e1}{2}$$

$$[[-e1, e1]]$$

Thus, the pair (-e1,e1) represents e2we3 in S1+S2.

```
> cmul(e1we2we3,f)=cmul(f,1);cmul(e1we2we3,fg)=cmul(fg,-1);h(e1we
  2we3);
```

$$\frac{Id}{2}+\frac{e1we2we3}{2}=\frac{Id}{2}+\frac{e1we2we3}{2}$$

$$-\frac{Id}{2}+\frac{e1we2we3}{2}=-\frac{Id}{2}+\frac{e1we2we3}{2}$$

$$[[1, -1]]$$

Thus, the pair (1,1) represents e1we2we3 in S1+S2.

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 1.327000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
```

>

For more information how to find spinor representations in higher dimensions, see help pages for Kfield, gradeinv, marKrepr, minimalideal, spinorKbasis, spinorKrepr. See also cbasis, clidata, `type/primitiveidemp`, `type/idempotent`.

See Also: Clifford:-`type/dfmatrix`, Clifford:-ddfmatrix, Clifford:-cdfmatrix

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-minimalideal - find a real basis in a minimal left or right ideal of a Clifford algebra

Calling Sequence:

minimalideal(list,f,s);

Parameters:

list - a list of Clifford basis monomials created, for example, by the procedure 'cbasis'
f - a primitive idempotent in Cl(B) which is to generate a minimal ideal
s - symbol 'left' or 'right' depending whether the ideal is to be left or right

Description:

- Procedure 'minimalideal' finds a real basis in a left S=Cl(Q)f or right S=fCl(Q) minimal ideal in the algebra Cl(Q) where f is a primitive idempotent in Cl(Q).
- The first argument in the procedure is a list of basis monomials in Cl(Q), e.g., a Clifford basis generated by the procedure 'cbasis' (see cbasis for more help).
- If desired, the list entered as the first argument may be first sorted by grade using option 'bygrade' in the Maple's procedure 'sort'. The output from 'cbasis' is already sorted that way. See bygrade for more information on this sorting option.
- The second argument in the procedure is a primitive idempotent f. Note that the procedure checks if the polynomial f entered is a primitive idempotent. See `type/primitiveidemp` for more help.
- Depending whether the third argument is 'left' or 'right', a basis for the left S=Cl(Q)f or right S=fCl(Q) minimal ideal S in Cl(Q) is computed.
- In the given Clifford algebra, there are many different primitive idempotents. It is known that if the signature of Q is (p,q) then a primitive idempotent f is a product of N = q - RHnumber(q-p) commuting projection operators where 'RHnumber' is the Radon-Hurwitz function (see RHnumber for more help). The projection operators may be defined in terms of N commuting basis monomials (see `type/clibasmon` for more help) which may be found with the procedure commutingelements.
- For each orthogonal Clifford algebra Cl(Q) in dimensions 1 through 9 there is a data table stored under the procedure clidata. The table is labeled by the signature (p,q) of Q entered as a list [p,q]. The simplest way to access the data for the Clifford algebra over the signature [p,q] is to enter the command

```
> clidata([p,q]);
```

- To see data for the Clifford algebra of the current form B enter

```
> clidata();
```

  or use 'clidata' without any arguments provided that the bilinear form B has been defined as a diagonal matrix. For more help on 'clidata' see clidata.

- Computation of a basis in S can be dramatically speeded up by using pre-defined primitive idempotents stored under clidata()[4] for each signature. These primitive idempotents are the same as those pre-defined in Pertti Lounesto's CLICAL.
- If the primitive idempotent f is the same as clidata()[4] and if the global variable _shortcut_in_minimalideal is set to 'true', then the procedure uses Clifford basis monomials stored under clidata()[5] in a form of a list to compute expanded basis elements in S. Depending whether the third argument is 'left' or 'right', these basis monomials simply multiply f from the left or right.
- To display all currently assigned values of environmental variables in 'CLIFFORD', use procedure CLIFFORD_ENV.
- If f is different than clidata()[4], complete computations are performed and they usually take longer.
- To speed up computations, set global variable _prolevel to true. To find out more, see Clifford:-setup. You can also use a cmulNUM instead of cmulRS for faster computations when B has been assigned a numeric matrix. Selection of the Clifford product procedure is done using useproduct.
- The procedure returns a list consisting of three elements:

- the first element is an ordered list of expanded basis elements for S,
- the second element is the ordered list clidata()[5],
- the third element is the string 's' initially entered as the third argument (to remind the user which ideal was used: left or right).
- There is a one-to-one relationship between the elements of the two ordered lists.

Examples:

```
> restart:with(Clifford):bench:=time():
  _prolevel:=true:
  _shortcut_in_minimalideal:=false:
  '_default_Clifford_product'=_default_Clifford_product;
  useproduct(cmulNUM);
```

$$\_default\_Clifford\_product = Clifford{:}\text{-}cmulRS$$

```
Warning, cmul will use cmulNUM; for help see pages ?cmul, ?Clifford:-intro, or
?cmulNUM
```

To shorten output, we will use aliases. Since below we will not exceed dimension 6, we can define aliases first:

```
> eval(makealiases(6,'ordered')):
```

Thus, for example,

```
> e1 &c e2;
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$e12 + B_{1,2}\, Id$$

where e12 = e1e2. When the bilinear form B is diagonal, then e12 = e1e2 = e1we2, etc.

Example 1: Clifford algebra Cl(1,1) is isomorphic to R(2).

```
> dim:=2:B:=linalg[diag](1,-1): #define the bilinear form B for Cl(1,1)
  clibasis:=cbasis(dim): #compute a Clifford basis for Cl(B)
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data := \left[real, 2, simple, \frac{Id}{2} + \frac{e12}{2}, [Id, e1], [Id], [Id, e1]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own here
> left_sbasis:=minimalideal(clibasis,f,'left'); #compute a real basis in Cl(B)f
```

$$left\_sbasis := \left[\left[\frac{Id}{2} + \frac{e12}{2}, \frac{e1}{2} + \frac{e2}{2}\right], [Id, e1], left\right]$$

```
> right_sbasis:=minimalideal(clibasis,f,'right'); #compute a real basis in fCl(B)
```

$$right\_sbasis := \left[\left[\frac{Id}{2} + \frac{e12}{2}, \frac{e1}{2} - \frac{e2}{2}\right], [Id, e1], right\right]$$

```
>
```

Example 2: Clifford algebra Cl(2,0) is also isomorphic to R(2)

```
> dim:=2:B:=linalg[diag](1,1): #define the bilinear form B for Cl(2,0)
  clibasis:=cbasis(dim): #compute a Clifford basis for Cl(B)
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data := \left[real, 2, simple, \frac{Id}{2} + \frac{e1}{2}, [Id, e2], [Id], [Id, e2]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own here
> left_sbasis:=minimalideal(clibasis,f,'left'); #compute a real basis in Cl(B)f
```

$$left\_sbasis := \left[\left[\frac{Id}{2} + \frac{e1}{2}, \frac{e2}{2} - \frac{e12}{2}\right], [Id, e2], left\right]$$

```
> right_sbasis:=minimalideal(clibasis,f,'right'); #compute a real basis in fCl(B)
```

$$right\_sbasis := \left[\left[\frac{Id}{2} + \frac{e1}{2}, \frac{e2}{2} + \frac{e12}{2}\right], [Id, e2], right\right]$$

```
>
```

Example 3: Clifford algebra Cl(2,2) is isomorphic to R(4)

```
> dim:=4:B:=linalg[diag](1,1,-1,-1): #define the bilinear form B for Cl(2,2)
```

```
clibasis:=cbasis(dim): #compute a Clifford basis for Cl(B)
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data := \left[real, 4, simple, \text{'Clifford:-cmulQ'}\left(\frac{Id}{2}+\frac{e13}{2}, \frac{Id}{2}+\frac{e24}{2}\right), [Id, e1, e2, e12], [Id], [Id, e1, e2, e12]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own here
> left_sbasis:=minimalideal(clibasis,f,'left'); #compute a real basis in Cl(B)f
```

$$left\_sbasis := \left[\left[\frac{Id}{4}+\frac{e13}{4}+\frac{e24}{4}-\frac{e1234}{4}, \frac{e1}{4}+\frac{e3}{4}+\frac{e124}{4}-\frac{e234}{4}, \frac{e2}{4}-\frac{e123}{4}+\frac{e4}{4}+\frac{e134}{4}, \frac{e12}{4}-\frac{e23}{4}+\frac{e14}{4}+\frac{e34}{4}\right], [Id, e1, e2, e12], left\right]$$

```
> right_sbasis:=minimalideal(clibasis,f,'right'); #compute a real basis in fCl(B)
```

$$right\_sbasis := \left[\left[\frac{Id}{4}+\frac{e13}{4}+\frac{e24}{4}-\frac{e1234}{4}, \frac{e1}{4}-\frac{e3}{4}+\frac{e124}{4}+\frac{e234}{4}, \frac{e2}{4}-\frac{e123}{4}-\frac{e4}{4}-\frac{e134}{4}, \frac{e12}{4}+\frac{e23}{4}-\frac{e14}{4}+\frac{e34}{4}\right], [Id, e1, e2, e12], right\right]$$

```
>
```

Example 4: Now we compute real bases in the left and right minimal ideals of Cl(3,0) = C(2).

```
> B:=linalg[diag](1,1,1):
> data:=clidata(); #display data about Cl(3,0)
```

$$data := \left[complex, 2, simple, \frac{Id}{2}+\frac{e1}{2}, [Id, e2, e3, e23], [Id, e23], [Id, e2]\right]$$

```
> f:=data[4];  #this is a pre-stored primitive idempotent
```

$$f := \frac{Id}{2}+\frac{e1}{2}$$

```
> clibasis:=cbasis(3);  #a standard basis for Cl(Q)
```

$$clibasis := [Id, e1, e2, e3, e12, e13, e23, e123]$$

```
> left_sbasis:=minimalideal(clibasis,f,'left'); #basis and generators for Cl(Q)f
```

$$left\_sbasis := \left[\left[\frac{Id}{2}+\frac{e1}{2}, \frac{e2}{2}-\frac{e12}{2}, \frac{e3}{2}-\frac{e13}{2}, \frac{e23}{2}+\frac{e123}{2}\right], [Id, e2, e3, e23], left\right]$$

```
> right_sbasis:=minimalideal(clibasis,f,'right'); #basis and generators for fCl(Q)
```

$$right\_sbasis := \left[\left[\frac{Id}{2}+\frac{e1}{2}, \frac{e2}{2}+\frac{e12}{2}, \frac{e3}{2}+\frac{e13}{2}, \frac{e23}{2}+\frac{e123}{2}\right], [Id, e2, e3, e23], right\right]$$

```
> f:=(1/2)*(Id+e3); #a different primitive idempotent is chosen now
```

$$f := \frac{Id}{2} + \frac{e3}{2}$$

```
> f &c f, type(f,primitiveidemp);
```

$$\frac{Id}{2} + \frac{e3}{2}, true$$

```
> left_sbasis:=minimalideal(clibasis,f,'left'); #basis and generators for Cl(Q)f
```

$$left\_sbasis := \left[\left[\frac{Id}{2} + \frac{e3}{2}, \frac{e2}{2} + \frac{e23}{2}, \frac{e1}{2} + \frac{e13}{2}, \frac{e12}{2} + \frac{e123}{2}\right], [Id, e2, e1, e12], left\right]$$

```
> right_sbasis:=minimalideal(clibasis,f,'right'); #basis and generators for fCl(Q)
```

$$right\_sbasis := \left[\left[\frac{Id}{2} + \frac{e3}{2}, \frac{e2}{2} - \frac{e23}{2}, \frac{e1}{2} - \frac{e13}{2}, \frac{e12}{2} + \frac{e123}{2}\right], [Id, e2, e1, e12], right\right]$$

```
>
```

Example 5: Clifford algebra Cl(1,3) is isomorphic to H(2).

```
> dim:=4:B:=linalg[diag](1,-1,-1,-1):#define form B for Cl(1,3)
clibasis:=cbasis(dim): #compute a Clifford basis for Cl(B)
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data := \left[quaternionic, 2, simple, \frac{Id}{2} + \frac{e14}{2}, [Id, e1, e2, e3, e12, e13, e23, e123], [Id, e2, e3, e23], [Id, e1]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own here
> left_sbasis:=minimalideal(clibasis,f,'left'); #compute a real basis in Cl(B)f
```

$$left\_sbasis := \left[\left[\frac{Id}{2} + \frac{e14}{2}, \frac{e1}{2} + \frac{e4}{2}, \frac{e2}{2} - \frac{e124}{2}, \frac{e3}{2} - \frac{e134}{2}, \frac{e12}{2} - \frac{e24}{2}, \frac{e13}{2} - \frac{e34}{2}, \frac{e23}{2} + \frac{e1234}{2}, \frac{e123}{2} + \frac{e234}{2}\right], [Id, e1, e2, e3, e12, e13, e23, e123], left\right]$$

```
> right_sbasis:=minimalideal(clibasis,f,'right'); #compute a real basis in fCl(B)
```

$$right\_sbasis := \left[\left[\frac{Id}{2} + \frac{e14}{2}, \frac{e1}{2} - \frac{e4}{2}, \frac{e2}{2} - \frac{e124}{2}, \frac{e3}{2} - \frac{e134}{2}, \frac{e12}{2} + \frac{e24}{2}, \frac{e13}{2} + \frac{e34}{2}, \frac{e23}{2} + \frac{e1234}{2}, \frac{e123}{2} - \frac{e234}{2}\right], [Id, e1, e2, e3, e12, e13, e23, e123], right\right]$$

```
>
```

Here is a K_basis returned for S=Cl(B)f. Since for the current signature (1,3) we have that K = H and Cl(1,3)=H(2), the output from 'spinorKbasis' shown below has two basis vectors and their generators modulo f:

```
> SBgens:=data[5]:FBgens:=data[6]:
> K_basis:=spinorKbasis(SBgens,f,FBgens,'left');
```

$$K\_basis := \left[\left[\frac{Id}{2}+\frac{e14}{2}, \frac{e1}{2}+\frac{e4}{2}\right], [Id, e1], left\right]$$

```
>
```

Example 6: Clifford algebra Cl(3,1) is isomorphic to R(4).

```
> dim:=4:B:=linalg[diag](1,1,1,-1):#define form B for Cl(3,1)
  clibasis:=cbasis(dim): #compute a Clifford basis for Cl(B)
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data := \left[real, 4, simple, 'Clifford{:}\text{-}cmulQ'\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e34}{2}\right), [Id, e2, e3, e23], [Id], [Id, e2, e3, e23]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own
  here
> left_sbasis:=minimalideal(clibasis,f,'left'); #compute a real
  basis in Cl(B)f
```

$$left\_sbasis := \left[\left[\frac{Id}{4}+\frac{e1}{4}+\frac{e34}{4}+\frac{e134}{4}, \frac{e2}{4}-\frac{e12}{4}+\frac{e234}{4}-\frac{e1234}{4}, \frac{e3}{4}-\frac{e13}{4}+\frac{e4}{4}-\frac{e14}{4}, \frac{e23}{4}+\frac{e123}{4}+\frac{e24}{4}+\frac{e124}{4}\right], [Id, e2, e3, e23], left\right]$$

```
> right_sbasis:=minimalideal(clibasis,f,'right'); #compute a real
  basis in fCl(B)
```

$$right\_sbasis := \left[\left[\frac{Id}{4}+\frac{e1}{4}+\frac{e34}{4}+\frac{e134}{4}, \frac{e2}{4}+\frac{e12}{4}+\frac{e234}{4}+\frac{e1234}{4}, \frac{e3}{4}+\frac{e13}{4}-\frac{e4}{4}-\frac{e14}{4}, \frac{e23}{4}+\frac{e123}{4}-\frac{e24}{4}-\frac{e124}{4}\right], [Id, e2, e3, e23], right\right]$$

```
>
```

Example 7: Clifford algebra Cl(3,3) is isomorphic to R(8).

```
> dim:=6:B:=linalg[diag](1,1,1,-1,-1,-1):#define form B for
  Cl(3,3)
  clibasis:=cbasis(dim): #compute a Clifford basis for Cl(B)
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data := \left[real, 8, simple, 'Clifford{:}\text{-}cmulQ'\left(\frac{Id}{2}+\frac{e14}{2}, \frac{Id}{2}+\frac{e25}{2}, \frac{Id}{2}+\frac{e36}{2}\right),\right.$$

$$[Id, e1, e2, e3, e12, e13, e23, e123], [Id], [Id, e1, e2, e3, e12, e13, e23, e123]]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own here
> left_sbasis:=minimalideal(clibasis,f,'left'); #compute a real basis in Cl(B)f
```

$$left\_sbasis := \left[\left[\frac{Id}{8}+\frac{e14}{8}+\frac{e25}{8}-\frac{e1245}{8}+\frac{e36}{8}-\frac{e1346}{8}-\frac{e2356}{8}-\frac{e123456}{8},\right.\right.$$
$$\frac{e1}{8}+\frac{e4}{8}+\frac{e125}{8}-\frac{e245}{8}+\frac{e136}{8}-\frac{e346}{8}-\frac{e12356}{8}-\frac{e23456}{8},$$
$$\frac{e2}{8}-\frac{e124}{8}+\frac{e5}{8}+\frac{e145}{8}+\frac{e236}{8}+\frac{e12346}{8}-\frac{e356}{8}+\frac{e13456}{8},$$
$$\frac{e3}{8}-\frac{e134}{8}-\frac{e235}{8}-\frac{e12345}{8}+\frac{e6}{8}+\frac{e146}{8}+\frac{e256}{8}-\frac{e12456}{8},$$
$$\frac{e12}{8}-\frac{e24}{8}+\frac{e15}{8}+\frac{e45}{8}+\frac{e1236}{8}+\frac{e2346}{8}-\frac{e1356}{8}+\frac{e3456}{8},$$
$$\frac{e13}{8}-\frac{e34}{8}-\frac{e1235}{8}-\frac{e2345}{8}+\frac{e16}{8}+\frac{e46}{8}+\frac{e1256}{8}-\frac{e2456}{8},$$
$$\frac{e23}{8}+\frac{e1234}{8}-\frac{e35}{8}+\frac{e1345}{8}+\frac{e26}{8}-\frac{e1246}{8}+\frac{e56}{8}+\frac{e1456}{8},$$
$$\left.\frac{e123}{8}+\frac{e234}{8}-\frac{e135}{8}+\frac{e345}{8}+\frac{e126}{8}-\frac{e246}{8}+\frac{e156}{8}+\frac{e456}{8}\right],$$
$$[Id, e1, e2, e3, e12, e13, e23, e123], left\Big]$$

```
>
```

Example 8: Clifford algebra Cl(4,2) is isomorphic to R(8).

```
> dim:=6:B:=linalg[diag](1,1,1,1,-1,-1):#define form B for Cl(3,3)
  clibasis:=cbasis(dim): #compute a Clifford basis for Cl(B)
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data := \left[real, 8, simple, 'Clifford{:}\text{-}cmulQ'\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e35}{2}, \frac{Id}{2}+\frac{e46}{2}\right),\right.$$
$$\left.[Id, e2, e3, e4, e23, e24, e34, e234], [Id], [Id, e2, e3, e4, e23, e24, e34, e234]\right]$$

```
> f:=data[4];#assign pre-stored idempotent to f or use your own here
```

$$f := Clifford{:}\text{-}cmulQ\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e35}{2}, \frac{Id}{2}+\frac{e46}{2}\right)$$

```
> left_sbasis:=minimalideal(clibasis,f,'left'); #compute a real basis in Cl(B)f
```

$$left\_sbasis := \left[\left[\frac{Id}{8}+\frac{e1}{8}+\frac{e35}{8}+\frac{e135}{8}+\frac{e46}{8}+\frac{e146}{8}-\frac{e3456}{8}-\frac{e13456}{8},\right.\right.$$
$$\frac{e2}{8}-\frac{e12}{8}+\frac{e235}{8}-\frac{e1235}{8}+\frac{e246}{8}-\frac{e1246}{8}-\frac{e23456}{8}+\frac{e123456}{8},$$
$$\frac{e3}{8}-\frac{e13}{8}+\frac{e5}{8}-\frac{e15}{8}+\frac{e346}{8}-\frac{e1346}{8}-\frac{e456}{8}+\frac{e1456}{8},$$
$$\frac{e4}{8}-\frac{e14}{8}-\frac{e345}{8}+\frac{e1345}{8}+\frac{e6}{8}-\frac{e16}{8}+\frac{e356}{8}-\frac{e1356}{8},$$
$$\frac{e23}{8}+\frac{e123}{8}+\frac{e25}{8}+\frac{e125}{8}+\frac{e2346}{8}+\frac{e12346}{8}-\frac{e2456}{8}-\frac{e12456}{8},$$
$$\frac{e24}{8}+\frac{e124}{8}-\frac{e2345}{8}-\frac{e12345}{8}+\frac{e26}{8}+\frac{e126}{8}+\frac{e2356}{8}+\frac{e12356}{8},$$
$$\frac{e34}{8}+\frac{e134}{8}-\frac{e45}{8}-\frac{e145}{8}+\frac{e36}{8}+\frac{e136}{8}+\frac{e56}{8}+\frac{e156}{8},$$
$$\left.\frac{e234}{8}-\frac{e1234}{8}-\frac{e245}{8}+\frac{e1245}{8}+\frac{e236}{8}-\frac{e1236}{8}+\frac{e256}{8}-\frac{e1256}{8}\right],$$
$$\left.[Id, e2, e3, e4, e23, e24, e34, e234], left\right]$$

```
>
```

Example 9: Testing an error message when f is not primitive:

```
> _prolevel:=false:
  dim:=4:B:=linalg[diag](1,1,1,-1):#define form B for Cl(3,1)
  clibasis:=cbasis(dim): #compute a Clifford basis for Cl(B)
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data := \left[real, 4, simple, 'Clifford{:}\text{-}cmulQ'\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e34}{2}\right), [Id, e2, e3, e23], [Id],\right.$$
$$\left.[Id, e2, e3, e23]\right]$$

Thus, a primitive idempotent in Cl(3,1) must have two commuting simple factors. Therefore, f1 defined below is *not a primitive* idempotent, although it is an idempotent:

```
> f1:=(1/2)*(1+e1);
```

$$f1 := \frac{1}{2}+\frac{e1}{2}$$

```
> f1 &c f1, type(f1,primitiveidemp);
```

$$\frac{Id}{2}+\frac{e1}{2}, false$$

Attempting to find a real basis in S=Cl(3,1)f1 will result in an error message:

```
> sbasis:=minimalideal(clibasis,f1,'left');
Error, (in Clifford:-minimalideal) second argument must be a primitive
idempotent
```

Therefore, let's use data[4]:

```
> f:=eval(data[4]); f &c f, type(f,primitiveidemp);
```

$$f := \frac{Id}{4} + \frac{e1}{4} + \frac{e34}{4} + \frac{e134}{4}$$

$$\frac{Id}{4} + \frac{e34}{4} + \frac{e1}{4} + \frac{e134}{4}, true$$

```
> sbasis:=minimalideal(clibasis,f,'left');
```

$$sbasis := \left[\left[\frac{Id}{4} + \frac{e34}{4} + \frac{e1}{4} + \frac{e134}{4}, \frac{e2}{4} + \frac{e234}{4} - \frac{e12}{4} - \frac{e1234}{4}, \frac{e3}{4} + \frac{e4}{4} - \frac{e13}{4} - \frac{e14}{4}, \frac{e23}{4} + \frac{e24}{4} + \frac{e123}{4} + \frac{e124}{4}\right], [Id, e2, e3, e23], left\right]$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 4.968000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

Comments:

- Using 'minimalideal' as shown above is the first step towards the computation of the spinor representation of Cl(Q) in a minimal ideal S over the field K. The next step is to use the procedure 'Kfield' to find a basis for the field K (see Kfield).
- The above examples may be redone for right minimal ideals with the same generators for the field K (replace 'left' with 'right'). This is because K is isomorphic to the intersection of the left and right minimal ideals Cl(B)f and fCl(B).
- Examples 1 -- 7 are continued in the help page for 'Kfield'. Enter Kfield to see them.

See Also:

Clifford:-`type/fieldelement`, Clifford:-squaremodf, Clifford:-cbasis, Clifford:-bygrade, Clifford:-RHnumber, Clifford:-Bsignature, Clifford:-`type/primitiveidemp`, Clifford:-commutingelements, Clifford:-spinorKrepr, Clifford:-spinorKbasis, Clifford:-matKrepr, Clifford:-Kfield, Clifford:-`type/clipolynom`, Clifford:-clidata

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-`convert/mlist` - convert matrix entries into a list

Calling Sequence:

convert(m, mlist);

Parameters:

m - matrix defined via the command linalg[matrix]

Description:

- This useful conversion procedure used by, for example, procedure 'rmulm' (see rmulm for more help), converts entries of a matrix into a list. Essentially it converts rows of a matrix into lists which are subsequently appended to each other.

Examples:

```
> restart:with(Clifford):
> m:=linalg[matrix](2,2,[1,2,3,4]); # a numerical matrix
```

$$m := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
> convert(m,mlist);
```

$$[1, 2, 3, 4]$$

```
> m:=linalg[matrix](2,2,[Id,e1+e2,e3,e2-e4]); #a Clifford matrix
```

$$m := \begin{bmatrix} Id & e1 + e2 \\ e3 & e2 - e4 \end{bmatrix}$$

```
> convert(m,mlist);
```

$$[Id, e1 + e2, e3, e2 - e4]$$

```
>
```

See Also: Clifford:-`type/climatrix`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-`type/nilpotent` - define a new type: a nilpotent element in a Clifford algebra

Calling Sequence:

type(u, nilpotent);
type([u,K],nilpotent);

Parameters:

u - an algebraic expression of type 'cliscalar' or 'clipolynom'
lname - (optional) argument entered as a second element in a list of type name, symbol, matrix, array, or `&*`(numeric,{name,symbol,array,matrix})

Description:

- Definition of type 'nilpotent'. When the first argument is just u, the procedure verifies whether it is a nilpotent element in the given Clifford algebra Cl(B). See `type/cliscalar` and `type/clipolynom` for more information about these types.
- When the first argument is a list, e.g., [u,K] or [u,-K], then the procedure checks if u is idempotent in Cl(K) or Cl(-K), respectively.
- Recall that a non-zero element u of Cl(B) is *nilpotent* if u^k = cmul(u,u,...,u) = 0 for some power k.
- If the element u happens to be an idempotent, or if some power of that element equals the element itself, or if the non-zero element u is of type 'cliscalar', then the procedure returns 'false'. Otherwise, the procedure checks if any power of its argument up to and including the dimension of the space is zero.
- It is known that if u is a nilpotent element of Cl(B) that its nilpotency index is less than or equal to the dimension of the space it is acting on. Hence, if u is a nilpotent element of Cl(B) then its nilpotency index is less than of equal to dim Cl(B).

Examples:

```
> restart:with(Clifford):B:=linalg[diag](1,1,1):
> p:=(1/2)*(e1+e1we3);
```

$$p := \frac{e1}{2} + \frac{e1we3}{2}$$

```
> type(p,nilpotent); #p is nilpotent in Cl(B)
  type([p,B],nilpotent); #p is nilpotent in Cl(B)
  type([p,-B],nilpotent); #p is not nilpotent in Cl(-B)
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$true$$

$$true$$

$$false$$

```
> cmul(p,p);
```

$$0$$

```
> type(2*e2we1,nilpotent);
```

false

```
> type((1/2)*(Id+e1),nilpotent); #this is in fact an idempotent
```

false

```
> type((1/2)*(Id+e1),idempotent);
```

true

```
> type(3*cos(alpha)*Id,nilpotent);
```

false

```
> B:='B':  #unassigning B
> type((1/2)*(e1+e1we3),nilpotent); #testing an error message
Error, (in type/nilpotent) matrix must be assigned to B
>
```

See Also: Clifford:-`type/idempotent`, Clifford:-`type/clipolynom`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-`type/oddelement` - define a new type: an odd element in Cl(B)

Calling Sequence:

type(p, oddelement);

Parameters:

p - an algebraic expression of type 'clipolynom'

Description:

- A *odd* element in Cl(B) is a Clifford polynomial which contains only odd grades, or which changes sign under grade involution in Cl(B) (see gradeinv for more help).
- The procedure returns 'true' or 'false' depending whether the input p is or is not of the type 'oddelement'.

Examples:

```
> restart:with(Clifford):
> type(2+2*e1+e1we2,oddelement);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$false$$

```
> p1:=2*e2+e1-e1we2we3;p2:=e2-e3;
```

$$p1 := 2\,e2 + e1 - e1we2we3$$

$$p2 := e2 - e3$$

```
> type(p1,oddelement),type(p2,oddelement);
```

$$true, true$$

```
> type(cmul(p1,p2),evenelement);
```

$$true$$

```
> p3:=1+b*e2we3;type(p3,evenelement);
```

$$p3 := 1 + b\,e2we3$$

$$true$$

```
> type(cmul(p3,p3),evenelement);
```

$$true$$

```
> type(cmul(p1,p3),oddelement);
```

$$true$$

```
>
```

See Also: Clifford:-`type/clipolynom`, Clifford:-gradeinv, Clifford:-`type/evenelement`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-ord - find location of the subscripts in a basis monomial

Calling Sequence:

ord(m);

Parameters:

m - expression of the type 'cliscalar' or 'climon'

Description:

- The procedure 'ord' returns an ordered list of positions in a monomial where vector indices are found.
- For example, nops(ord(e1we2)) can then be used to find the order (grade) of the monomial.
- For consistency we have ord(Id) = ord(numeric) = ord(numeric*Id) = ord(cliscalar)=[] where 'cliscalar' is any object of the type 'cliscalar', i.e., the returned list is empty. See cliscalar, clibasmon, and climon for more information on the types.
- This procedure is for internal use.

Examples:

```
> restart:with(Clifford):
> ord(3*Pi*e1we2we3);
```

$$[2, 5, 8]$$

```
> ord(3*Id);
```

$$[\ ]$$

```
> ord(4*Pi);
```

$$[\ ]$$

```
> ord(e1wej);
```

$$[2, 5]$$

```
>
```

See Also: Clifford:-reorder, Clifford:-clicollect

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-permsign - compute sign of a permutation

Calling Sequence:

permsign(L);

Parameters:

L - list of objects than can be sorted

Description:

- Procedure 'permsign' computes the sign of a permutation needed to sort a list of elements entered in a list L. It returns +1 when the permutation is even and -1 when the permutation is odd.
- It is used in procedure reorder that reorder indices of a basis Grassmann monomial. It returns the sign of the permutation needed to reorder the indices.

Examples:

```
> restart:with(Clifford):
> permsign([1,2,3]);
```

$$1$$

```
> permsign([3,2]);
```

$$-1$$

```
> permsign([1,2,9,k,a,j,i]);
```

$$1$$

```
> permsign([]);
```

$$1$$

```
> makeclibasmon([]);
```

$$Id$$

```
> makeclibasmon([1,4,2,j,i,3]);
```

$$e1we4we2wejweiwe3$$

```
> reorder(%);
```

$$-e1we2we3we4weiwej$$

```
> permsign([1,4,2,j,i,3]);
```

$$-1$$

```
>
```

See Also: Clifford:-extract, Clifford:-`type/clibasmon`, Clifford:-reorder, Clifford:-wedge, Clifford:-makeclibasmon

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-`type/primitiveidemp` - define a new type: a primitive idempotent in Cl(B)

Calling Sequence:

type(e,primitiveidemp);

Parameters:

e - element of Cl(B) of type 'idempotent'

Description:

- This procedure checks if its argument e is a primitive idempotent in Cl(B) for the currently defined bilinear form B. See `type/idempotent` for more help.
- Recall that an idempotent e is *primitive* if it cannot be written as a sum e' + e" of mutually annihilating idempotents, i.e., cmul(e',e') = e', cmul(e",e") = e", cmul(e',e") = cmul(e",e') = 0 where 'cmul' denotes Clifford multiplication in Cl(B). See cmul for more help.
- In a canonical basis for Cl(Q) there are sets of k = q - RHnumber(q-p) basis monomials which will commute and have square 1 mod Id (these sets need not be mutually disjoint). Here [p,q] is the signature of the quadratic form Q and 'RHnumber' is the Radon-Hurwitz function (see RHnumber and commutingelements for more help).
- Each such set of k commuting basis monomials generates a group of order 2^k.
- Suppose that [f1,f2,...,fk] is a list of k commuting basis monomials in a canonical basis for Cl(Q). Then, each of these defines a *simple* idempotent (1/2)*(1+fi), i=1..k, in Cl(Q). A product of these k simple idempotents is then a primitive idempotent f in Cl(Q).
- Primitive idempotents generate minimal left and right ideals, e.g., Cl(Q)f and fCl(Q) in Cl(Q) which are then defined as spinor spaces. See minimalideal, spinorKbasis, spinorKrepr, and Kfield for more help on spinor representations of Cl(Q) in such ideals.
- For this procedure to work, the bilinear form B must be defined first. It returns 'true' or 'false' depending whether its argument is or is not primitive.
- See also procedure factoridempotent which can be used to factor an idempotent into a product of simple idempotents.

Examples:

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

Example 1: Defining some primitive idempotents in Cl(3,1):

```
> clibasis:=cbasis(4):B:=linalg[diag](1,1,1,-1):
> L1:=commutingelements(clibasis);#find a set of commuting
  elements
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$L1 := [e1, e2we4]$$

```
> L2:=commutingelements(remove(member,clibasis,L1));#find another
```

```
set
```

$$L2 := [e2, e1we4]$$

```
> L3:=commutingelements(remove(member,clibasis,[op(L1),op(L2)]));
#find another set
```

$$L3 := [e3, e1we3we4]$$

We may now define three different primitive idempotents using the above commuting sets:

```
> f1:=cmul((1/2)*(1+e1),(1/2)*(1+e2we4));
```

$$f1 := \frac{Id}{4} + \frac{e1we2we4}{4} + \frac{e1}{4} + \frac{e2we4}{4}$$

```
> type(f1,primitiveidemp); #verify that f1 is primitive
```

$$true$$

```
> factoridempotent(f1); #factor f1 and write it as a Clifford
product
```

$$Clifford{:}\text{-}cmulQ\left(\frac{Id}{2} + \frac{e1}{2}, \frac{Id}{2} + \frac{e2we4}{2}\right)$$

```
> f2:=cmul((1/2)*(1+e2),(1/2)*(1+e1we4));
```

$$f2 := \frac{Id}{4} - \frac{e1we2we4}{4} + \frac{e2}{4} + \frac{e1we4}{4}$$

```
> type(f2,primitiveidemp); #verify that f2 is primitive
```

$$true$$

```
> factoridempotent(f2); #factor f2 and write it as a Clifford
product
```

$$Clifford{:}\text{-}cmulQ\left(\frac{Id}{2} + \frac{e2}{2}, \frac{Id}{2} + \frac{e1we4}{2}\right)$$

```
> f3:=cmul((1/2)*(1+e3),(1/2)*(1+e1we3we4));
```

$$f3 := \frac{Id}{4} + \frac{e1we3we4}{4} + \frac{e3}{4} - \frac{e1we4}{4}$$

```
> type(f3,primitiveidemp); #verify that f3 is primitive
```

$$true$$

```
> f33:=factoridempotent(f3); #factor f3 and write it as a
Clifford product
```

$$f33 := Clifford{:}\text{-}cmulQ\left(\frac{Id}{2} + \frac{e3}{2}, \frac{Id}{2} - \frac{e1we4}{2}\right)$$

NOTE: although the above factorization of f3 does have different factors than f3, this is another factorization of f3 as seen next in the following verification:

```
> evalb(f33=f3);
```

$$true$$

```
>
```

Example 2: Some non-primitive idempotents in Cl(3,1):

```
> f4:=(1/2)*(1+e1); #a non-primitive idempotent since it has one factor only
```

$$f4 := \frac{1}{2} + \frac{e1}{2}$$

```
> type(f4,idempotent);
```

$$true$$

```
> type(f4,primitiveidemp);
```

$$false$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 1.049000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM with Win XP Professional
>
```

See Also: Clifford:-Kfield, Clifford:-spinorKbasis, Clifford:-spinorKrepr, Clifford:-minimalideal, Clifford:-cmulQ, Clifford:-cmul, Clifford:-commutingelements, Clifford:-factoridempotent, Clifford:-RHnumber, Clifford:-`type/idempotent`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-pseudodet - compute a pseudodeterminant of a 2 x 2 Clifford matrix

Calling Sequence:

pseudodet(m1);

Parameters:

m1 - expression of the type 'climatrix' or 'matrix'

Description:

- Procedure 'pseudodet' computes a *pseudodeterminant* of a 2 x 2 matrix with entries in a Clifford algebra Cl(B).
- Recall that if M is a 2 x 2 matrix [a b] then its pseudodeterminant is defined as:
  [c d]

  pseudodet(M) = a &c reversion(d) - b &c reversion(c)
- If a pseudodeterminant of a 2 x 2 matrix is 1 or -1, and if certain additional conditions are met by the entries of M, then M is a *Vahlen matrix.* Vahlen matrices form a *Vahlen group* under matrix multiplication. The Vahlen group of R^n is connected and it is a two-fold covering group of the sense-preserving Mobius group of R^n. See isVahlenmatrix for more information on Vahlen matrices.

Examples:

```
> restart:with(Clifford):B:=linalg[diag](1,1,1,-1):
> M1:=linalg[matrix](2,2,[Id,2*e1+4,-e2+e1we2,-e3]);
```

$$M1 := \begin{bmatrix} Id & 2\,e1+4 \\ -e2+e1we2 & -e3 \end{bmatrix}$$

```
> p1:=pseudodet(M1);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$p1 := -e3 + 6\,e2 + 6\,e1we2$$

```
> M2:=linalg[matrix](2,2,[e1+e2,e4-3,-e2,2*e1]);
```

$$M2 := \begin{bmatrix} e1+e2 & e4-3 \\ -e2 & 2\,e1 \end{bmatrix}$$

```
> p2:=pseudodet(M2);
```

$$p2 := 2\,Id - 2\,e1we2 - e2we4 - 3\,e2$$

```
> M3:=linalg[matrix](2,2,[1,e1we2+e1we4,e3,e4we3]);
```

$$M3 := \begin{bmatrix} 1 & e1we2+e1we4 \\ e3 & e4we3 \end{bmatrix}$$

```
> pseudodet(M3);
```

$$e3we4 - e1we2we3 + e1we3we4$$

[>

See Also: Clifford:-reversion, Clifford:-isVahlenmatrix, Clifford:-isproduct, Clifford:-cmul, Clifford:-`type/climatrix`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-`type/purequatbasis` - define a new type: a basis for pure quaternions in Cl(B)

Calling Sequence:

type(L, purequatbasis);

Parameters:

L - a list of three elements of type 'climon', 'clibasmon' or 'clipolynom'

Description:

- This procedure checks if the three elements in the list L form a basis for *pure quaternions* in Cl(B). By definition, Clifford polynomials p1, p2, and p2 in Cl(B) will form such basis if the following relations are satisfied:

  cmul(p1,p1) = -Id, cmul(p2,p2) = -Id, cmul(p3,p3) = -Id,

  cmul(p1,p2) = -cmul(p2,p1), cmul(p1,p3) = -cmul(p3,p1), cmul(p2,p3) = -cmul(p3,p2),

  or if the elements anti-commute and square to -1 in Cl(B). See cmul, `type/clipolynom`, `type/climon`, and `type/clibasmon` for more help.
- The procedure returns 'true' or 'false' depending whether the above relations are or are not satisfied.
- If the list L is of type 'purequatbasis', then by adding the identity element 'Id' (or just 1) to the list we obtain a basis for a subalgebra K of Cl(B) which is isomorphic with a division ring H of quaternions.
- The so called *split-quaternions* have a basis satisfying a different set of relations and are not considered here.
- This procedure is useful in identifying subsets of the canonical basis for Cl(B) which may provide a basis for pure quaternions. It is used by the procedure 'Kfield' which finds a suitable basis for the field K of a spinor representation of Cl(B). See Kfield and spinorKrepr for more information.
- For this procedure to work, the list L must have exactly three arguments.

Examples:

```
> restart:with(Clifford):
> B:=linalg[diag](1,-1,-1,-1):
> data:=clidata();
```

$$data := \left[quaternionic, 2, simple, \frac{Id}{2} + \frac{e1we4}{2}, [Id, e1, e2, e3, e1we2, e1we3, e2we3, e1we2we3], [Id, e2, e3, e2we3], [Id, e1] \right]$$

```
> type([e2,e3,e2we3],purequatbasis);
```

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

$$true$$

```
> clibasis:=cbasis(4);
```

$$clibasis := [Id, e1, e2, e3, e4, e1we2, e1we3, e1we4, e2we3, e2we4, e3we4, e1we2we3, e1we2we4, e1we3we4, e2we3we4, e1we2we3we4]$$

```
> nsq:=[]:
  for g in clibasis do if cmul(g,g)=-Id then nsq:=[op(nsq),g] fi
  od;
> nsq; #all elements in 'nsq' have square -1
```

$$[e2, e3, e4, e2we3, e2we4, e3we4, e1we2we3, e1we2we4, e1we3we4, e1we2we3we4]$$

Thus, there are several triples of anticommuting basis monomials in 'nsq' which form pure quaternion bases other than [e2,e3,e2we3]:

```
> type([e2,e4,e2we4],purequatbasis);
```

$$true$$

```
> type([e2we3,e2we4,e3we4],purequatbasis);
```

$$true$$

```
> type([e1we2we3,e1we2we4,e3we4],purequatbasis);
```

$$true$$

```
> type([e2,e1we2we3,e3],purequatbasis);
```

$$false$$

```
>
```

See Also: Clifford:-spinorKrepr, Clifford:-`type/climon`, Clifford:-`type/clibasmon`, Clifford:-Kfield, Clifford:-squaremodf

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-q_conjug - quaternionic conjugation in the division ring H of quaternions

Calling Sequence:

q_conjug(p);

Parameters:

p - an algebraic expression of type 'quaternion'

Description:

- Quaternionic conjugation in the division ring of quaternions H is implemented via the procedure 'q_conjug'. Recall that complex conjugation in complexified Clifford algebras is implemented with the procedure c_conjug while the Clifford conjugation in Cl(B) is implemented via the procedure conjugation.
- Recall that quaternions (as opposed to generalized quaternions of `type/genquaternion` are defined as the even elements in Cl(3), or as the para-bivectors in Cl(3). Thus, a quaternionic basis is [Id, e1we2,e1we3,e2we3].
- Any element that belongs to the vector space H spanned by [Id, e1we2,e1we3,e2we3] in Cl(3) is now of
  type *quaternion*.
- It is possible to display or enter expressions in terms of the basis {1, qi, qj, qk} where

  qi = e3we2 = -e2we3, qj = e1we3, qk = e2we1 = -e1we2

  (see P. Lounesto, "Clifford Algebras and Spinors", page 49, for more information on quaternions defined that way). Apply qdisplay to any quaternion to see it displayid in terms of {1,qi,qj,qk}.
- For the generalized quaternionic conjugation see the help page `type/genquaternion` .
- The default assignment of the quaternionic basis is done in Clifford:-setup by storing it under global environmental variable _quatbasis. To see it, just type _quatbasis at the Maple prompt. To display all environmental variables used in 'CLIFFORD', use procedure CLIFFORD_ENV.
- For the quaternionic norm, quaternionic inverse, and quaternionic multiplication see respectively qnorm, qinv, and qmul. For multiplication of quaternionic matrices see rmulm with option `&q`.

Examples:

```
> restart:with(Clifford):
> q1:=2+2*e1we2+e1we3+e2we3;q2:=-2*e2we3+3+e1we2;
```

$$q1 := 2 + 2\, e1we2 + e1we3 + e2we3$$

$$q2 := -2\, e2we3 + 3 + e1we2$$

```
> type(q1,quaternion); ###<<<- Intended error message
Error, (in type/quaternion) bilinear form B has not been assigned yet. It must
```

be defined as the identity 3 x 3 matrix.

```
> B:=linalg[diag](1,1,1):
> type(q1,quaternion),type(q2,quaternion);
```

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

$$true, true$$

This is how Maple will display quaternions if you use 'qdisplay':

```
> _quatbasis;
```

$$[[Id, e3we2, e1we3, e2we1], \{Maple\ has\ assigned\ qi:=-e2we3,\ qj:=e1we3,\ qk:=-e1we2\}]$$

```
> q1:=qdisplay(q1);q2:=qdisplay(q2);
```

$$q1 := 2 - 2\,qk + qj - qi$$

$$q2 := 3 - qk + 2\,qi$$

```
> type(q1,quaternion);type(q2,quaternion);
```

$$true$$

$$true$$

Quaternionic conjugation reverses the sign of the pure-quaternion part of any quaternion:

```
> purequatpart:=qdisplay(vectorpart(q1,2));
```

$$purequatpart := -2\,qk + qj - qi$$

```
> q1;q_conjug(q1);
```

$$2 + 2\,e1we2 + e1we3 + e2we3$$

$$2 + 2\,qk - qj + qi$$

It is known that the quaternionic norm of q is defined as the square root of the product of q and its quaternionic conjugate. See qnorm for more help.

```
> sqrt(qmul(q1,q_conjug(q1))); #long way
```

$$\sqrt{10}$$

```
> qnorm(q1); #short way
```

$$\sqrt{10}$$

Notice also the following well-known property of the quaternionic conjugation:

```
> q_conjug(q1 &q q2) = q_conjug(q2) &q q_conjug(q1);
```

$$6 + 10\,qk + 2\,qj = 6 + 10\,qk + 2\,qj$$

The standard assignment of elements e1we2, e1we3, and e2we3 to the quaternionic units can be seen by displaying the contents of the global variable _quatbasis:

```
> _quatbasis;
```

$$[[Id, e3we2, e1we3, e2we1], \{Maple\ has\ assigned\ qi:=-e2we3,\ qj:=e1we3,\ qk:=-e1we2\}]$$

```
>
```

See Also: Clifford:-`type/quaternion`, Clifford:-`type/genquaternion`, Clifford:-qdisplay, Clifford:-rot3d, Clifford:-qmul, Clifford:-qnorm, Clifford:-qinv

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-qdisplay - display quaternions in the basis {1,qi,qj,qk}

Calling Sequence:

qdisplay(p);

Parameters:

p - an algebraic expression of type 'quaternion' or 'matrix'

Description:

- Elements of the division ring of quaternions H are defined as the even elements in Cl(3), or as the para-bivectors in Cl(3). Thus, a quaternionic basis is [Id, e1we2,e1we3,e2we3] and it is available as the first component of a global variable 'quatbasis' defined at the initialization time. Type _quatbasis or _quatbasis[2] at the Maple prompt to see it, or use procedure CLIFFORD_ENV to display all environmental variables and their values currently used in 'CLIFFORD'.
- Any element that belongs to the vector space H spanned by [Id, e1we2,e1we3,e2we3] in Cl(3) is now of type *quaternion*. See `type/quaternion` for more help.
- It is possible to display or enter expressions in terms of the basis {1, qi, qj, qk} where

  qi = e3we2 = -e2we3, qj = e1we3, qk = e2we1 = -e1we2

  (see P. Lounesto, "Clifford Algebras and Spinors", page 49, for more information on quaternions defined that way). Apply 'qdisplay' to any quaternion to see it displayid in terms of {1,qi,qj,qk}.
- Notice that the definition of the pure quaternionic basis {qi, qj, qk} in terms of the exterior products is based on the following relation between the exterior product and the cross product in Cl(3):

  wedge(a,b) = a &w b = (a x b)e123

  a x b = -cmul(wedge(a, b), e123) = -(a &w b) &c e123

  where e123 = e1we2we3 is the unit pseudoscalar or volume element. Since e123 &c 123 = -1, the inverse of e123 is -e123 and we have:

  e1 &c (-e123) = qi, e1 = qi &c e123

  e2 &c (-e123) = qj, e2 = qj &c e123

  e3 &c (-e123) = qk, e3 = qk &c e123
- For generalized quaternions see `type/genquaternion`.
- For the quaternionic conjugation, quaternionic norm, quaternionic inverse, and quaternionic multiplication see respectively q_conjug, qnorm, qinv, and qmul. For multiplication of

quaternionic matrices see rmulm with option `&q`.

- One may apply 'qdisplay' to matrices as shown below.

Examples:

```
> restart:with(Clifford):
> q1:=2+2*e1we2+e1we3+e2we3;q2:=-2*e2we3+3+e1we2;
```

$$q1 := 2 + 2\ e1we2 + e1we3 + e2we3$$

$$q2 := -2\ e2we3 + 3 + e1we2$$

```
> type(q1,quaternion); ###<<<- Intended error message
Error, (in type/quaternion) bilinear form B has not been assigned yet. It must
be defined as the identity 3 x 3 matrix.
> B:=linalg[diag](1,1,1):
> type(q1,quaternion),type(q2,quaternion);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$true, true$$

This is how Maple will display quaternions if you use 'qdisplay':

```
> _quatbasis;
```

$$[[Id, e3we2, e1we3, e2we1], \{Maple\ has\ assigned\ qi:=-e2we3,\ qj:=e1we3,\ qk:=-e1we2\}]$$

```
> q1:=qdisplay(q1);q2:=qdisplay(q2);
```

$$q1 := 2 - 2\ qk + qj - qi$$

$$q2 := 3 - qk + 2\ qi$$

```
> type(q1,quaternion);type(q2,quaternion);
```

$$true$$

$$true$$

One can enter quaternions in terms of the basis {1,qi,qj,qk} as well:

```
> q3:=2-2*'qi'+3*'qj'-'qk';
```

$$q3 := 2 - 2\ qi + 3\ qj - qk$$

```
> qdisplay(q2);
```

$$3 - qk + 2\ qi$$

Quaternion multiplication is done with 'qmul' or '&q':

```
> qmul(q1,q3);
  qmul(q1,q3,q1,q3);
```

$$-3 - 7\ qk + 11\ qj - qi$$

$$-162 + 42\ qk - 66\ qj + 6\ qi$$

```
> (q1 &q q3) &q q3;
```

$$-48 + 8\ qk + 26\ qj + 14\ qi$$

Let's find the multiplication table for H:

```
> Q:=[1,'qi','qj','qk'];
```

$$Q := [1, qi, qj, qk]$$

```
> A := array(1..4,1..4):
  for i from 1 to 4 do for j from 1 to 4 do
  A[i,j]:=qmul(Q[i],Q[j]) od od;
> print(A);
```

$$\begin{bmatrix} 1 & qi & qj & qk \\ qi & -1 & qk & -qj \\ qj & -qk & -1 & qi \\ qk & qj & -qi & -1 \end{bmatrix}$$

Matrices with quaternionic entries also may be entered and manipulated with. Here is an example:

```
> M:=linalg[matrix](2,2,[2+qi+2*qj,-2*qi-qj,3*qk,2-qi]);
```

$$M := \begin{bmatrix} 2 - e2we3 + 2\, e1we3 & 2\, e2we3 - e1we3 \\ -3\, e1we2 & 2 + e2we3 \end{bmatrix}$$

To see the {1,qi,qj,qk} basis used in M, apply 'qdisplay' to it:

```
> qdisplay(M);
```

$$\begin{bmatrix} 2 + qi + 2\, qj & -2\, qi - qj \\ 3\, qk & 2 - qi \end{bmatrix}$$

```
> M:=linalg[matrix](2,2,[1,2,3,4]);
```

$$M := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
> qdisplay(M);
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
>
```

See Also: Clifford:-`type/quaternion`, Clifford:-`type/genquaternion`, Clifford:-rot3d, Clifford:-qmul, Clifford:-q_conjug, Clifford:-qnorm, Clifford:-qinv

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-qinv - find the inverse of a quaternion

Calling Sequence:

qinv(p);

Parameters:

p - an algebraic expression of type 'quaternion' or 'cliscalar'

Description:

- Elements of the division ring of quaternions H are defined as the even elements in Cl(3), or as the para-bivectors in Cl(3). Thus, a quaternionic basis is [Id, e1we2,e1we3,e2we3] and it is available as the first component of a global variable 'quatbasis' defined at the initialization time. Type _quatbasis or _quatbasis[2] at the Maple prompt to see it.
- Since H is a division ring, any non-zero quaternion has a unique inverse in H.
- Any element that belongs to the vector space H spanned by [Id, e1we2,e1we3,e2we3] in Cl(3) is now of type *quaternion*. See `type/quaternion` for more help.
- It is possible to display or enter expressions in terms of the basis {1, qi, qj, qk} where

  qi = e3we2 = -e2we3, qj = e1we3, qk = e2we1 = -e1we2

  (see P. Lounesto, "Clifford Algebras and Spinors", page 49, for more information on quaternions defined that way). Apply 'qdisplay' to any quaternion to see it displayid in terms of {1,qi,qj,qk}.
- For generalized quaternions see `type/genquaternion`.
- For the quaternionic conjugation, quaternionic norm, and quaternionic multiplication see respectively q_conjug, qnorm, and qmul. For multiplication of quaternionic matrices see rmulm with option `&q`.
- To display environmental variables, including _quatbasis, used by 'CLIFFORD', use procedure CLIFFORD_ENV.

Examples:

```
> restart:with(Clifford):
> q1:=2+2*e1we2+e1we3+e2we3;q2:=-2*e2we3+3+e1we2;
```

$$q1 := 2 + 2\ e1we2 + e1we3 + e2we3$$

$$q2 := -2\ e2we3 + 3 + e1we2$$

```
> type(q1,quaternion); # <<< - Expected error message
Error, (in type/quaternion) bilinear form B has not been assigned yet. It must
be defined as the identity 3 x 3 matrix.
> B:=linalg[diag](1,1,1):
> type(q1,quaternion),type(q2,quaternion);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
```

```
clude &C and &C[K]. Type ?cliprod for help.
```

$$true, true$$

This is how Maple will display quaternions if you use 'qdisplay':

```
> _quatbasis;_quatbasis[2];
```

$$[[Id, e3we2, e1we3, e2we1], \{Maple\ has\ assigned\ qi:=-e2we3,\ qj:=e1we3,\ qk:=-e1we2\}]$$

$$\{Maple\ has\ assigned\ qi:=-e2we3,\ qj:=e1we3,\ qk:=-e1we2\}$$

```
> q1:=qdisplay(q1);q2:=qdisplay(q2);
```

$$q1 := 2 - 2\,qk + qj - qi$$

$$q2 := 3 - qk + 2\,qi$$

```
> type(q1,quaternion);type(q2,quaternion);
```

$$true$$

$$true$$

Quaternionic inverse is then computed as follows:

```
> q1i:=qinv(q1);
```

$$q1i := \frac{1}{5} + \frac{qk}{5} - \frac{qj}{10} + \frac{qi}{10}$$

```
> qmul(q1i,q1),qmul(q1,q1i);
```

$$1, 1$$

```
> q2:=2*Id-2*e1we2+3*e2we3-4*e1we3;
```

$$q2 := 2\,Id - 2\,e1we2 + 3\,e2we3 - 4\,e1we3$$

```
> q2:=qdisplay(q2);
```

$$q2 := 2 + 2\,qk - 4\,qj - 3\,qi$$

```
> q2i:=qinv(q2);
```

$$q2i := \frac{2}{33} - \frac{2\,qk}{33} + \frac{4\,qj}{33} + \frac{qi}{11}$$

```
> qmul(q2i,q2),qmul(q2,q2i);
```

$$1, 1$$

Let's find the multiplication table for H:

```
> Q:=[1,'qi','qj','qk'];
```

$$Q := [1, qi, qj, qk]$$

```
> A := array(1..4,1..4):
  for i from 1 to 4 do for j from 1 to 4 do
  A[i,j]:=qmul(Q[i],Q[j]) od od;
> print(A);
```

$$\begin{bmatrix} 1 & qi & qj & qk \\ qi & -1 & qk & -qj \\ qj & -qk & -1 & qi \\ qk & qj & -qi & -1 \end{bmatrix}$$

Notice the following well-known property of the inverse operator:

```
> qinv(q1 &q q2) - qinv(q2) &q qinv(q1);
```

0

```
> 
```

See Also: Clifford:-`type/quaternion`, Clifford:-`type/genquaternion`, Clifford:-rot3d, Clifford:-qmul, Clifford:-q_conjug, Clifford:-qnorm

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-qmul - quaternionic product and its infix form '&q'

Calling Sequence:

qmul(q1,q2,...,qn);

q1 &q q2 &q ... &q qn;

Parameters:

q1, q2, ..., qn - expressions of the type 'cliscalar' or 'quaternion'

Description:

- Procedure 'qmul' and its infix form '&q' is a special version of the Clifford product 'cmul' applied to the even elements in Cl(3) of `type/quaternion`.
- Via the procedure 'rmulm', this multiplication can also be applied to matrices with quaternionic entries. See rmulm for more help.
- This procedure is multilinear. The infix form may be used as follows: q1 &q e2 &q q3 where the latter equals qmul(q1,q2,q3), etc..
- For the quaternionic conjugation, quaternionic norm, quaternionic inverse, and quaternionic display see respectively q_conjug, qnorm, qinv, and qdisplay. For multiplication of quaternionic matrices see rmulm with the option `&q`.
- Quaternions are defined as even elements in Cl(3). A quaternionic basis {1,qi,qj,qk} is defined as follows:

$$qi = e3we2 = -e2we3, \quad qj = e1we3, \quad qk = e2we1 = -e1we2$$

- To see quaternions displayed in this basis use qdisplay.

Examples:

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

$$Clifford{:}\text{-}cmulRS$$

```
> q1:=2+2*qi;q2:=-2*qj+3+qk;q3:=2+3*qk;
```

$$q1 := 2 - 2\ e2we3$$

$$q2 := -2\ e1we3 + 3 - e1we2$$

$$q3 := 2 - 3\ e1we2$$

To see quaternions displayed in terms of the basis {1,qi,qj,qk} use 'qdisplay' or enter 'qi', 'qj', 'qk':

```
> q1:=2+2*'qi';q2:=-2*'qj'+3+'qk';q3:=2+3*'qk';
```

$$q1 := 2 + 2\ qi$$

$$q2 := -2\ qj + 3 + qk$$

$$q3 := 2 + 3\ qk$$

```
> type(q1,quaternion);
```

```
Error, (in type/quaternion) bilinear form B has not been assigned yet. It must
be defined as the identity 3 x 3 matrix.
```

```
> B:=linalg[diag](1,1,1):
> type(q1,quaternion),type(q2,quaternion),type(q3,quaternion);
```

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

$$true, true, true$$

This is how Maple will display quaternions if you use 'qdisplay':

```
> q1:=qdisplay(q1);q2:=qdisplay(q2);q3:=qdisplay(q3);
```

$$q1 := 2 + 2\,qi$$

$$q2 := -2\,qj + 3 + qk$$

$$q3 := 2 + 3\,qk$$

Quaternion multiplication is done with 'qmul' or '&q':

```
> qmul(q1,q2),q1 &q q2;
```

$$6 - 2\,qk - 6\,qj + 6\,qi,\ 6 - 2\,qk - 6\,qj + 6\,qi$$

```
> qmul(q1,q2,q3),q1 &q q2 &q q3;
```

$$18 + 14\,qk - 30\,qj - 6\,qi,\ 18 + 14\,qk - 30\,qj - 6\,qi$$

```
> _default_Clifford_product;
```

$$Clifford{:}\text{-}cmulNUM$$

Quaternionic inverse is found with the procedure qinv as follows:

```
> q1i:=qinv(q1);q2i:=qinv(q2);q3i:=qinv(q3);
  q1i &q q1,q2i &q q2,q3i &q q3;
```

$$q1i := \frac{1}{4} - \frac{qi}{4}$$

$$q2i := \frac{3}{14} - \frac{qk}{14} + \frac{qj}{7}$$

$$q3i := \frac{2}{13} - \frac{3\,qk}{13}$$

$$1, 1, 1$$

> Let's find the multiplication table for H:

```
> Q:=[1,'qi','qj','qk'];
```

$$Q := [1, qi, qj, qk]$$

```
> A := array(1..4,1..4):
  for i from 1 to 4 do for j from 1 to 4 do
  A[i,j]:=qmul(Q[i],Q[j]) od od;
> print(A);
```

$$\begin{bmatrix} 1 & qi & qj & qk \\ qi & -1 & qk & -qj \\ qj & -qk & -1 & qi \\ qk & qj & -qi & -1 \end{bmatrix}$$

Matrices with quaternionic entries also may be entered and manipulated with. Here is an example:

```
> M:=linalg[matrix](2,2,[2+qi+2*qj,-2*qi-qj,3*qk,2-qi]);
```

$$M := \begin{bmatrix} 2 - e2we3 + 2\, e1we3 & 2\, e2we3 - e1we3 \\ -3\, e1we2 & 2 + e2we3 \end{bmatrix}$$

```
> M1:=qdisplay(M);
```

$$M1 := \begin{bmatrix} 2 + qi + 2\, qj & -2\, qi - qj \\ 3\, qk & 2 - qi \end{bmatrix}$$

```
> M2:=q_conjug(M1);
```

$$M2 := \begin{bmatrix} 2 - 2\, qj - qi & qj + 2\, qi \\ -3\, qk & 2 + qi \end{bmatrix}$$

```
> M1 &qm M2;
```

$$\begin{bmatrix} 9 - 6\, qj + 3\, qi & -2 - 2\, qk \\ -6\, qj + 6\, qi & 5 + 6\, qj - 3\, qi \end{bmatrix}$$

```
> q1 &qm M2;
```

$$\begin{bmatrix} 6 - 4\, qk - 4\, qj + 2\, qi & -4 + 2\, qk + 2\, qj + 4\, qi \\ -6\, qk + 6\, qj & 2 + 6\, qi \end{bmatrix}$$

```
> M1 &qm q2;
```

$$\begin{bmatrix} 10 + qj + 5\, qi & -2 + 4\, qk - qj - 7\, qi \\ -3 + 9\, qk + 6\, qi & 6 + 4\, qk - 3\, qj - 3\, qi \end{bmatrix}$$

```
>
```

See Also: Clifford:-cmul, Clifford:-rmulm, Clifford:-`type/clipolynom`, Clifford:-`type/climon`, Clifford:-`type/clibasmon`, Clifford:-`type/genquaternion`, Clifford:-qdisplay, Clifford:-rot3d, Clifford:-q_conjug, Clifford:-qnorm, Clifford:-qinv

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-qnorm - find the quaternionic norm of a quaternion

Calling Sequence:

qnorm(p);

Parameters:

p - an algebraic expression of type 'quaternion' or 'cliscalar'

Description:

- Elements of the division ring of quaternions H are defined as the even elements in Cl(3), or as the para-bivectors in Cl(3). Thus, a quaternionic basis is [Id, e1we2,e1we3,e2we3] and it is available as the first component of a global variable '_quatbasis' defined at the initialization time. Type _quatbasis or _quatbasis[2] at the Maple prompt to see it, or use procedure CLIFFORD_ENV to display values of all environmental variables in 'CLIFFORD'..
- Since H is a division ring, any non-zero quaternion has a unique inverse in H.
- The *quaternionic norm* in H is defined as a product of a quaternion q and its quaternionic conjugate qc. See q_conjug for more help on quaternionic conjugation.
- Any element that belongs to the vector space H spanned by [Id, e1we2,e1we3,e2we3] in Cl(3) is now of type *quaternion*. See `type/quaternion` for more help.
- It is possible to display or enter expressions in terms of the basis {1, qi, qj, qk} where

  qi = e3we2 = -e2we3, qj = e1we3, qk = e2we1 = -e1we2

  (see P. Lounesto, "Clifford Algebras and Spinors", page 49, for more information on quaternions defined that way). Apply displayid to any quaternion to see it displayed in terms of {1,qi,qj,qk}.
- For generalized quaternions see `type/genquaternion`.
- For the quaternionic conjugation, quaternionic inverse, and quaternionic multiplication see respectively q_conjug, qinv, and qmul. For multiplication of quaternionic matrices see rmulm with the option `&q`.

Examples:

```
> restart:with(Clifford):
> q1:=2+2*e1we2+e1we3+e2we3;q2:=-2*e2we3+3+e1we2;
```

$$q1 := 2 + 2\ e1we2 + e1we3 + e2we3$$

$$q2 := -2\ e2we3 + 3 + e1we2$$

```
> type(q1,quaternion);
Error, (in type/quaternion) bilinear form B has not been assigned yet. It must
be defined as the identity 3 x 3 matrix.
> B:=linalg[diag](1,1,1):
> type(q1,quaternion),type(q2,quaternion);
```

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

$$true, true$$

This is how Maple will display quaternions if you use 'qdisplay':

```
> _quatbasis;
```

$$[[Id, e3we2, e1we3, e2we1], \{Maple\ has\ assigned\ qi:=-e2we3,\ qj:=e1we3,\ qk:=-e1we2\}]$$

```
> q1:=qdisplay(q1);q2:=qdisplay(q2);
```

$$q1 := 2 - 2\,qk + qj - qi$$

$$q2 := 3 - qk + 2\,qi$$

```
> type(q1,quaternion),type(q2,quaternion);
```

$$true, true$$

```
> qnorm(q1),qnorm(q2);
```

$$\sqrt{10}, \sqrt{14}$$

```
> qnorm(q1 &q q2)=qnorm(q1)*qnorm(q2);
```

$$2\sqrt{35} = \sqrt{10}\,\sqrt{14}$$

The above equality shows that 'qnorm' is an algebra homomorphism from H to R.
Let's find the multiplication table for H:

```
> Q:=[1,'qi','qj','qk'];
```

$$Q := [1, qi, qj, qk]$$

```
> A := array(1..4,1..4):
  for i from 1 to 4 do for j from 1 to 4 do
  A[i,j]:=qmul(Q[i],Q[j]) od od;
> print(A);
```

$$\begin{bmatrix} 1 & qi & qj & qk \\ qi & -1 & qk & -qj \\ qj & -qk & -1 & qi \\ qk & qj & -qi & -1 \end{bmatrix}$$

```
>
```

See also: Clifford:-cmul, Clifford:-rmulm, Clifford:-`type/clipolymom`, Clifford:-`type/climon`, Clifford:-`type/clibasmon`, Clifford:-`type/genquaternion`, Clifford:-`type/quaternion`, Clifford:-qdisplay, Clifford:-qmul

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-`type/quaternion` - define a new type: a quaternionic element in Cl(B)

Calling Sequence:

type(p, quaternion);

Parameters:

p - an algebraic expression of type 'clipolynom'

Description:

- Elements of the division ring of quaternions H are defined as the even elements in Cl(3), or as the para-bivectors in Cl(3). Thus, a quaternionic basis is [Id, e1we2,e1we3,e2we3] and it is available as the first component of a global variable 'quatbasis' defined at the initialization time. Type _quatbasis or _quatbasis[2] at the Maple prompt to see it.
- Any element that belongs to the vector space H spanned by [Id, e1we2,e1we3,e2we3] in Cl(3) is now of type *quaternion*.
- It is possible to display or enter expressions in terms of the basis {1, qi, qj, qk} where

  qi = e3we2 = -e2we3, qj = e1we3, qk = e2we1 = -e1we2

  (see P. Lounesto, "Clifford Algebras and Spinors", page 49, for more information on quaternions defined that way). Apply qdisplay to any quaternion to see it displayid in terms of {1,qi,qj,qk}.
- Notice that the definition of the pure quaternionic basis {qi, qj, qk} in terms of the exterior products is based on the following relation between the exterior product and the cross product in Cl(3):

  wedge(a,b) = a &w b = (a x b)e123

  a x b = -cmul(wedge(a, b), e123) = -(a &w b) &c e123

  where e123 = e1we2we3 is the unit pseudoscalar or volume element. Since e123 &c 123 = -1, the inverse of e123 is -e123 and we have:

  e1 &c (-e123) = qi, e1 = qi &c e123

  e2 &c (-e123) = qj, e2 = qj &c e123

  e3 &c (-e123) = qk, e3 = qk &c e123
- For generalized quaternions see `type/genquaternion`.
- For the quaternionic conjugation, quaternionic norm, quaternionic inverse, and quaternionic multiplication see respectively q_conjug, qnorm, qinv, and qmul. For multiplication of quaternionic matrices see rmulm with the option `&q`.

Examples:

```
> restart:with(Clifford):
> q1:=2+2*e1we2+e1we3+e2we3;q2:=-2*e2we3+3+e1we2;
```

$$q1 := 2 + 2\ e1we2 + e1we3 + e2we3$$

$$q2 := -2\ e2we3 + 3 + e1we2$$

```
> type(q1,quaternion); #testing an error message
Error, (in type/quaternion) bilinear form B has not been assigned yet. It must
be defined as the identity 3 x 3 matrix.
> B:=matrix(2,2,[1,0,0,1]);
```

$$B := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

```
> type(q1,quaternion); #testing another error message
Error, (in type/quaternion) identity 3 x 3 matrix must be assigned to B
> B:=linalg[diag](1,1,1):
> type(q1,quaternion),type(q2,quaternion);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$true, true$$

This is how Maple will display quaternions if you use 'qdisplay':

```
> _quatbasis;_quatbasis[2];
```

$$[[Id, e3we2, e1we3, e2we1], \{Maple\ has\ assigned\ qi:=-e2we3,\ qj:=e1we3,\ qk:=-e1we2\}]$$

$$\{Maple\ has\ assigned\ qi:=-e2we3,\ qj:=e1we3,\ qk:=-e1we2\}$$

```
> q1:=qdisplay(q1);q2:=qdisplay(q2);
```

$$q1 := 2 - 2\ qk + qj - qi$$

$$q2 := 3 - qk + 2\ qi$$

```
> type(q1,quaternion);type(q2,quaternion);
```

$$true$$

$$true$$

One can enter quaternions in terms of the basis {1,qi,qj,qk} as well:

```
> q3:=2-2*'qi'+3*'qj'-'qk';
```

$$q3 := 2 - 2\ qi + 3\ qj - qk$$

```
> type(q3,quaternion);
```

$$true$$

Quaternion multiplication is done with 'qmul' or '&q':

```
> qmul(q1,q3);
```

$$-3 - 7\ qk + 11\ qj - qi$$

```
> q1 &q q3;
```

$$-3 - 7\ qk + 11\ qj - qi$$

Let's find the multiplication table for H:

```
> Q:=[1,'qi','qj','qk'];
```

$$Q := [1, qi, qj, qk]$$

```
> A := array(1..4,1..4):
  for i from 1 to 4 do for j from 1 to 4 do
  A[i,j]:=qmul(Q[i],Q[j]) od od;
> print(A);
```

$$\begin{bmatrix} 1 & qi & qj & qk \\ qi & -1 & qk & -qj \\ qj & -qk & -1 & qi \\ qk & qj & -qi & -1 \end{bmatrix}$$

Matrices with quaternionic entries also may be entered and manipulated with. Here is an example:

```
> M:=linalg[matrix](2,2,[2+qi+2*qj,-2*qi-qj,3*qk,2-qi]);
```

$$M := \begin{bmatrix} 2 - e2we3 + 2\, e1we3 & 2\, e2we3 - e1we3 \\ -3\, e1we2 & 2 + e2we3 \end{bmatrix}$$

To see the {1,qi,qj,qk} basis used in M, apply 'qdisplay' to it:

```
> qdisplay(M);
```

$$\begin{bmatrix} 2 + qi + 2\, qj & -2\, qi - qj \\ 3\, qk & 2 - qi \end{bmatrix}$$

```
>
```

See Also: Clifford:-`type/genquaternion`, Clifford:-qdisplay, Clifford:-rot3d, Clifford:-qmul, Clifford:-q_conjug, Clifford:-qnorm, Clifford:-qinv

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-RC - right contraction in Cl(B)

Calling Sequence:

RC(u,v);
RC(u,v,name);

Parameters:

u, v - expressions of the type 'cliscalar', 'clibasmon', 'climon', or 'clipolynom'
name - (optional) parameter of type 'name', 'symbol, 'array', or 'matrix', or `&*`(numeric,{name,symbol,matrix,array})

Description:

- Procedure 'RC' defines a *right contraction* in Cl(B), when no third argument is used or when name=B, between a multivector v that acts from the right on a multivector u. Otherwise, contraction is computed in Cl(K) if name=K.
- Both inputs to this procedure are elements in the Clifford algebra Cl(B) of a totally arbitrary bilinear form B. However, it is possible to use this procedure with a third optional parameter of `type/name`, `type/symbol`, `type/array`, or `type/matrix` in which case indices are appended to this parameter. See example below.
- This procedure can be used to define Clifford multiplication 'cmul' in Cl(B). Normally, we use the left contraction LC (see cmul for more help) but the right contract RC can be used instead. See one of the examples below.
- See procedure CLIFFORD_ENV to display values of environmental variables in 'CLIFFORD' such as dim_V.
- To collect complicated expressions, use clicollect.
- Procedure 'RC' can also be applied to expressions containing terms of `type/cliprod` upon loading the supplementary package Cliplus (see below). Notice that unevaluated Clifford products must be entered this way: `&C`[K](e1,e2) when the optional index is used. When no index is used, enter &C(e1,e2).

Examples:

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

$$9$$

Procedure 'RC' gives the right-contraction u |_ v in the Clifford algebra Cl(B) of any element u by any element v entered as a second parameter:

```
> RC(e1,e2),RC(e2,e3),RC(e3,e2),RC(e2,e3);
```

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

```
> RC(2*e1we2+e4+3*e3we4,2*e1-e3+e4);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$4 B_{2,1}\, e1 - 4 B_{1,1}\, e2 - 2 B_{2,3}\, e1 + 2 B_{1,3}\, e2 + 2 B_{2,4}\, e1 - 2 B_{1,4}\, e2 + 2 B_{4,1}\, Id - B_{4,3}\, Id + B_{4,4}\, Id + 6 B_{4,1}\, e3 - 6 B_{3,1}\, e4 - 3 B_{4,3}\, e3 + 3 B_{3,3}\, e4 + 3 B_{4,4}\, e3 - 3 B_{3,4}\, e4$$

```
> clicollect(RC(e1we2we3we4+e3we4-e2-3,2+3*e1-e3+e2we3we4));
```

$$(-6 + B_{2,3} - 3 B_{2,1})\, Id + (B_{2,2} B_{4,3} B_{3,4} + B_{4,2} B_{3,3} B_{2,4} - B_{2,2} B_{3,3} B_{4,4} - B_{4,2} B_{2,3} B_{3,4} + B_{3,2} B_{2,3} B_{4,4} - B_{3,2} B_{4,3} B_{2,4})\, e1 - (-B_{3,2} B_{4,3} B_{1,4} + B_{1,2} B_{4,3} B_{3,4} + 2 - B_{4,2} B_{1,3} B_{3,4} + B_{3,2} B_{1,3} B_{4,4} - B_{1,2} B_{3,3} B_{4,4} + B_{4,2} B_{3,3} B_{1,4})\, e2 - (B_{1,2} B_{2,3} B_{4,4} - B_{4,2} B_{2,3} B_{1,4} - 3 B_{4,1} - B_{1,2} B_{4,3} B_{2,4} - B_{2,2} B_{1,3} B_{4,4} + B_{4,2} B_{1,3} B_{2,4} + B_{4,3} + B_{2,2} B_{4,3} B_{1,4})\, e3 + 2\, e3we4 + (-3 B_{3,1} - B_{1,2} B_{3,3} B_{2,4} + B_{2,2} B_{3,3} B_{1,4} + B_{3,2} B_{1,3} B_{2,4} - B_{3,2} B_{2,3} B_{1,4} + B_{1,2} B_{2,3} B_{3,4} + B_{3,3} - B_{2,2} B_{1,3} B_{3,4})\, e4 + 2\, e1we2we3we4 - (-B_{1,3} + 3 B_{1,1})\, e2we3we4 + (3 B_{4,1} - B_{4,3})\, e1we2we3 - (3 B_{3,1} - B_{3,3})\, e1we2we4 - (B_{2,3} - 3 B_{2,1})\, e1we3we4$$

Note these special cases:

```
> RC(0,2);
```

$$0$$

```
> p:=a*Id+2*e1we3;
```

$$p := a\, Id + 2\, e1we3$$

```
> RC(p,3);RC(p,3*Id);
```

$$3\, a\, Id + 6\, e1we3$$

$$3\, a\, Id + 6\, e1we3$$

```
> RC(2,3);
```

$$6\, Id$$

```
> RC(2*Id,3);
```

$$6\, Id$$

```
> RC(2*Id,3*Id);
```

$$6\, Id$$

Normally, we use the left contraction in LC to define Clifford product &c between a 1-vector x and a multivector u as follows:

x &c u = x &w u + x _| u (1)

where _| denotes the left contraction LC of u by x acting from the left with respect to the bilinear form B in Cl(B). Then, we extend (1) by linearity, the postulated associativity of the Clifford product &c, and the properties of the left contraction to any two multivectors.

It is equally possible to define Clifford product in Cl(B) with the help of the right contraction RC in a similar manner as above, namely:

u &c x = u &w x + u |_ x (2)

where |_ denotes the right contraction RC of u by x acting from the right with respect to the bilinear form B in Cl(B). Then, we extend (2) by linearity, the postulated associativity of the Clifford product &c, and the properties of the right contraction to any two multivectors.

Let's check:

```
> u:=1+e2+3*e1we4;x:=e1-2*e3;
```

$$u := 1 + e2 + 3\ e1we4$$

$$x := e1 - 2\ e3$$

```
> cmul(u,x) = wedge(u,x) + RC(u,x);
```

$$-(-B_{2,1} + 2\ B_{2,3})\ Id - 2\ e2we3 - e1we2 + (1 - 6\ B_{4,3} + 3\ B_{4,1})\ e1 - 2\ e3 - 3\ (B_{1,1} - 2\ B_{1,3})\ e4 + 6\ e1we3we4 = e1 - e1we2 + B_{2,1}\ Id + 3\ B_{4,1}\ e1 - 3\ B_{1,1}\ e4 - 2\ e3 - 2\ e2we3 - 2\ B_{2,3}\ Id + 6\ e1we3we4 - 6\ B_{4,3}\ e1 + 6\ B_{1,3}\ e4$$

Example 1: Simple computations:

```
> p:=2*e1we2;q:=2+e2+e2we3+3*e1we2we3;
  u:=2*e2we3-e1+e3+a*e2we3;v:=-2*e2-3*e1+e4we3+c*e2we3;
  x:=e1-2*e2+e3;
  y:=-e2+3*e1-e4;
```

$$p := 2\ e1we2$$

$$q := 2 + e2 + e2we3 + 3\ e1we2we3$$

$$u := 2\ e2we3 - e1 + e3 + a\ e2we3$$

$$v := -2\ e2 - 3\ e1 + e4we3 + c\ e2we3$$

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

$$y := -e2 + 3\ e1 - e4$$

```
> s:=time():RC(q,p);time()-s;
```

$$2\ B_{3,1}\ B_{2,2}\ Id - 2\ B_{2,1}\ B_{3,2}\ Id + 6\ B_{3,1}\ B_{2,2}\ e1 - 6\ B_{3,1}\ B_{1,2}\ e2 - 6\ B_{2,1}\ B_{3,2}\ e1 + 6\ B_{2,1}\ B_{1,2}\ e3 + 6\ B_{1,1}\ B_{3,2}\ e2 - 6\ B_{1,1}\ B_{2,2}\ e3$$

$$0.016$$

```
> RC(2*Id,e1);RC(e1,2*Id);
```

$$0$$

$$2\ e1$$

```
> s:=time():RC(u,x);time()-s;
```

$$2\ B_{3,1}\ e2 - 2\ B_{2,1}\ e3 - 4\ B_{3,2}\ e2 + 4\ B_{2,2}\ e3 + 2\ B_{3,3}\ e2 - 2\ B_{2,3}\ e3 - B_{1,1}\ Id + 2\ B_{1,2}\ Id - B_{1,3}\ Id + B_{3,1}\ Id - 2\ B_{3,2}\ Id + B_{3,3}\ Id + a\ B_{3,1}\ e2 - a\ B_{2,1}\ e3 - 2\ a\ B_{3,2}\ e2 + 2\ a\ B_{2,2}\ e3 + a\ B_{3,3}\ e2 - a\ B_{2,3}\ e3$$

$$0.016$$

Example 2: Let's verify now these properties of the right contraction: (here |_ denotes the right contraction)

Property 1: A vector contracting a vector gives the dot product of the two vectors only when the bilinear form is symmetric [the right-hand side of (1) below is often identified with the "dot" product on a vector space].

x |_ y = (1/2)*(x &c y + y &c x) (1)

```
> L1:=RC(x,y); #left hand side of (1)
```

$$L1 := -B_{1,2}\,Id + 3\,B_{1,1}\,Id - B_{1,4}\,Id + 2\,B_{2,2}\,Id - 6\,B_{2,1}\,Id + 2\,B_{2,4}\,Id - B_{3,2}\,Id + 3\,B_{3,1}\,Id - B_{3,4}\,Id$$

```
> L2:=(x &c y + y &c x)/2; #right hand side of (1)
```

$$L2 := -\frac{1}{2}(B_{3,2} + B_{1,2} - 2\,B_{2,2} - 3\,B_{3,1} - 3\,B_{1,1} + 6\,B_{2,1} - 2\,B_{2,4} + B_{1,4} + B_{3,4})\,Id - \frac{1}{2}(B_{4,1} + B_{2,1} - 3\,B_{1,1} - 2\,B_{4,2} - 2\,B_{2,2} + 6\,B_{1,2} - 3\,B_{1,3} + B_{2,3} + B_{4,3})\,Id$$

```
> out:=clicollect(L1-L2);
```

$$out := \left(\frac{1}{2}B_{4,3} + \frac{5}{2}B_{1,2} - \frac{1}{2}B_{3,2} - \frac{5}{2}B_{2,1} + B_{2,4} - \frac{1}{2}B_{3,4} - B_{4,2} + \frac{1}{2}B_{4,1} + \frac{1}{2}B_{2,3} - \frac{1}{2}B_{1,4} - \frac{3}{2}B_{1,3} + \frac{3}{2}B_{3,1}\right)Id$$

Notice that the above difference between L1 and L2 is zero if B is symmetric, that is, when B[1,3] = B[3,1], B[2,4] = B[4,2], B[4,3] = B[3,4], B[2,3] = B[3,2], B[1,2] = B[2,1], B[4,1] = B[1,4, the above expression equals zero.]

For example, define the following symmetric matrix:

```
> B:=matrix(3,3,[1,a,b,
                 a,1,c,
                 b,c,1]);
```

$$B := \begin{bmatrix} 1 & a & b \\ a & 1 & c \\ b & c & 1 \end{bmatrix}$$

```
> x:=x1*e1+x2*e2+x3*e3:y:=y1*e1+y2*e2+y3*e3:
> L1:=clicollect(RC(x,y)); #left hand side of (1)
```

$$L1 := (x1\,y1 + x1\,y2\,a + x1\,y3\,b + x2\,y1\,a + x2\,y2 + x2\,y3\,c + x3\,y1\,b + x3\,y2\,c + x3\,y3)\,Id$$

```
> L2:=clicollect((x &c y + y &c x)/2); #right hand side of (1)
```

$$L2 := (x1\,y1 + x1\,y2\,a + x1\,y3\,b + x2\,y1\,a + x2\,y2 + x2\,y3\,c + x3\,y1\,b + x3\,y2\,c + x3\,y3)\,Id$$

```
> L1-L2;
```

$$0$$

However, it is easily seen that if the bilinear form has an antisymmetric part, then formula (1) is no longer valid:

```
> A:=matrix(3,3,[0,A1,A2,
                 -A1,0,A3,
                 -A2,-A3,0]);
```

$$A := \begin{bmatrix} 0 & A1 & A2 \\ -A1 & 0 & A3 \\ -A2 & -A3 & 0 \end{bmatrix}$$

```
> B:=evalm(B+A);
```

$$B := \begin{bmatrix} 1 & a + A1 & b + A2 \\ a - A1 & 1 & c + A3 \\ b - A2 & c - A3 & 1 \end{bmatrix}$$

```
> L1:=clicollect(RC(x,y)); #left hand side of (1)
```

$$L1 := (x1\, y1 + x1\, y2\, a + x1\, y2\, A1 + x1\, y3\, b + x1\, y3\, A2 + x2\, y1\, a - x2\, y1\, A1 + x2\, y2 + x2\, y3\, c + x2\, y3\, A3 + x3\, y1\, b - x3\, y1\, A2 + x3\, y2\, c - x3\, y2\, A3 + x3\, y3)\, Id$$

```
> L2:=clicollect((x &c y + y &c x)/2); #right hand side of (1)
```

$$L2 := (x1\, y1 + x1\, y2\, a + x1\, y3\, b + x2\, y1\, a + x2\, y2 + x2\, y3\, c + x3\, y1\, b + x3\, y2\, c + x3\, y3)\, Id$$

```
> clicollect(L1-L2);
```

$$(x1\, y2\, A1 + x1\, y3\, A2 - x2\, y1\, A1 + x2\, y3\, A3 - x3\, y1\, A2 - x3\, y2\, A3)\, Id$$

```
> B:='B':dim_V:=9: #let's erase B and assign the largest allowed value to dim_V
```

Property 2: Right contraction by a vector is a derivation of the Grassmann algebra for any bilinear form B:

(u &w v) |_ x = (u |_ x) &w gradeinv(v) + u &w (v |_ x) (2)

```
> L1:=clicollect(RC(u &w v,x)); #left hand side of (2)
```

$$\begin{aligned} L1 := {} & (3\, a\, x2\, B_{2,2} + x1\, B_{4,1} + 6\, x1\, B_{2,1} + 3\, a\, x3\, B_{2,3} + x3\, B_{4,3} + c\, x1\, B_{2,1} + 6\, x3\, B_{2,3} \\ & + 6\, x2\, B_{2,2} + c\, x2\, B_{2,2} + 3\, a\, x1\, B_{2,1} + c\, x3\, B_{2,3} + x2\, B_{4,2})\, e1we3 - (3\, a\, x2\, B_{1,2} + 6\, x3\, B_{1,3} \\ & + c\, x3\, B_{1,3} + 6\, x1\, B_{1,1} + 3\, a\, x3\, B_{1,3} + 3\, a\, x1\, B_{1,1} + c\, x2\, B_{1,2} + 6\, x2\, B_{1,2} + c\, x1\, B_{1,1})\, e2we3 \\ & - (c\, x3\, B_{3,3} + 6\, x1\, B_{3,1} + 3\, a\, x3\, B_{3,3} + c\, x2\, B_{3,2} + c\, x1\, B_{3,1} + 6\, x3\, B_{3,3} + 3\, a\, x2\, B_{3,2} \\ & + 3\, a\, x1\, B_{3,1} + 6\, x2\, B_{3,2})\, e1we2 \\ & + (2\, x2\, B_{2,2} + 3\, x3\, B_{3,3} + 2\, x1\, B_{2,1} + 3\, x2\, B_{3,2} + 2\, x3\, B_{2,3} + 3\, x1\, B_{3,1})\, e1 \\ & - 2\, (x1\, B_{1,1} - x3\, B_{3,3} - x1\, B_{3,1} + x2\, B_{1,2} - x2\, B_{3,2} + x3\, B_{1,3})\, e2 \\ & - (3\, x1\, B_{1,1} + 3\, x3\, B_{1,3} + 2\, x3\, B_{2,3} + 3\, x2\, B_{1,2} + 2\, x2\, B_{2,2} + 2\, x1\, B_{2,1})\, e3 \\ & + (x1\, B_{1,1} + x3\, B_{1,3} + x2\, B_{1,2})\, e3we4 - (x2\, B_{3,2} + x3\, B_{3,3} + x1\, B_{3,1})\, e1we4 \end{aligned}$$

```
> L2:=clicollect(RC(u,x) &w gradeinv(v) + u &w RC(v,x)); #right hand side of (2)
```

$$\begin{aligned} L2 := {} & (3\, a\, x2\, B_{2,2} + x1\, B_{4,1} + 6\, x1\, B_{2,1} + 3\, a\, x3\, B_{2,3} + x3\, B_{4,3} + c\, x1\, B_{2,1} + 6\, x3\, B_{2,3} \\ & + 6\, x2\, B_{2,2} + c\, x2\, B_{2,2} + 3\, a\, x1\, B_{2,1} + c\, x3\, B_{2,3} + x2\, B_{4,2})\, e1we3 - (3\, a\, x2\, B_{1,2} + 6\, x3\, B_{1,3} \end{aligned}$$

$$+ c\,x3\,B_{1,3} + 6\,x1\,B_{1,1} + 3\,a\,x3\,B_{1,3} + 3\,a\,x1\,B_{1,1} + c\,x2\,B_{1,2} + 6\,x2\,B_{1,2} + c\,x1\,B_{1,1})\,e2we3$$
$$- (c\,x3\,B_{3,3} + 6\,x1\,B_{3,1} + 3\,a\,x3\,B_{3,3} + c\,x2\,B_{3,2} + c\,x1\,B_{3,1} + 6\,x3\,B_{3,3} + 3\,a\,x2\,B_{3,2}$$
$$+ 3\,a\,x1\,B_{3,1} + 6\,x2\,B_{3,2})\,e1we2$$
$$+ (2\,x2\,B_{2,2} + 3\,x3\,B_{3,3} + 2\,x1\,B_{2,1} + 3\,x2\,B_{3,2} + 2\,x3\,B_{2,3} + 3\,x1\,B_{3,1})\,e1$$
$$- 2\,(x1\,B_{1,1} - x3\,B_{3,3} - x1\,B_{3,1} + x2\,B_{1,2} - x2\,B_{3,2} + x3\,B_{1,3})\,e2$$
$$- (3\,x1\,B_{1,1} + 3\,x3\,B_{1,3} + 2\,x3\,B_{2,3} + 3\,x2\,B_{1,2} + 2\,x2\,B_{2,2} + 2\,x1\,B_{2,1})\,e3$$
$$+ (x1\,B_{1,1} + x3\,B_{1,3} + x2\,B_{1,2})\,e3we4 - (x2\,B_{3,2} + x3\,B_{3,3} + x1\,B_{3,1})\,e1we4$$

```
> L1-L2;
```

$$0$$

Notice that property (2) is valid for ANY bilinear form B, that is, B does not need to be symmetric.

Property 3: Right contraction is right distributive with respect to the Grassmann (wedge) product for any bilinear form B:

q |_ (u &w v) = q |_ (u |_ v) (3)

```
> L1:=clicollect(RC(q,u &w v)): #left hand side of (3)
> L2:=RC(RC(q,u),v): #right hand side of (3)
> simplify(L1-L2);
```

$$0$$

Notice that property (3) is valid for ANY bilinear form B.

The right contraction by a vector is also a derivation in the Clifford algebra Cl(B):

Property 4: Right contraction by a vector is also a derivation in Cl(B) for any bilinear form B.

(u &c v) |_ x = (u |_ x) &c gradeinv(v) + u &c (v |_ x) (4)

```
> L1:=RC(u &c v,x): #left hand side of (4)
> L2:=RC(u,x) &c gradeinv(v) + u &c RC(v,x): #rhs of (4)
> simplify(reorder(L1-L2));
```

$$0$$

Notice that property (4) is valid for ANY bilinear form B.

Example 3: In Cl(3) the following formula is valid:

ub = u |_ b - (1/2)*(b &c u - u &c b) + u &w b (5)

where b is an arbitrary bivector and u is any element in Cl(3). Let's verify formula (5) now:

```
> B:=linalg[diag](1,1,1):
> clibas:=cbasis(3);
```

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

```
> eval(makealiases(3));
```

$$e12, e21, e13, e31, e23, e32, e123, e132, e213, e231, e312, e321$$

```
> b:=b12*e12+b13*e13+b23*e23; #an arbitrary bivector in Cl(3)
```

$$b := b12\, e12 + b13\, e13 + b23\, e23$$

```
> u:=u0+u1*e1+u2*e2+u3*e3+u12*e12+u13*e13+u23*e23+u123*e123; #any
  u in Cl(3)
```

$$u := u0 + u1\, e1 + u2\, e2 + u3\, e3 + u12\, e12 + u13\, e13 + u23\, e23 + u123\, e123$$

```
> L1:=u &c b; #left hand side of (5)
```

$$\begin{aligned} L1 := & -(u13\, b13 + u23\, b23 + u12\, b12)\, Id - (u23\, b12 - u12\, b23 - u0\, b13)\, e13 \\ & + (u13\, b12 + u0\, b23 - u12\, b13)\, e23 + (u23\, b13 + u0\, b12 - u13\, b23)\, e12 \\ & - (u2\, b12 + u123\, b23 + u3\, b13)\, e1 + (u1\, b12 + u123\, b13 - u3\, b23)\, e2 \\ & - (u123\, b12 - u2\, b23 - u1\, b13)\, e3 + (u3\, b12 + u1\, b23 - u2\, b13)\, e123 \end{aligned}$$

```
> L2:=clicollect(RC(u,b) - (1/2)*(b &c u - u &c b) + u &w b);
Warning, since B has been (re-)assigned, value of dim_V has been reduced by 'we
dge' to 3
```

$$\begin{aligned} L2 := & -(u13\, b13 + u23\, b23 + u12\, b12)\, Id - (u23\, b12 - u12\, b23 - u0\, b13)\, e13 \\ & + (u13\, b12 + u0\, b23 - u12\, b13)\, e23 + (u23\, b13 + u0\, b12 - u13\, b23)\, e12 \\ & - (u2\, b12 + u123\, b23 + u3\, b13)\, e1 + (u1\, b12 + u123\, b13 - u3\, b23)\, e2 \\ & - (u123\, b12 - u2\, b23 - u1\, b13)\, e3 + (u3\, b12 + u1\, b23 - u2\, b13)\, e123 \end{aligned}$$

```
> L1-L2;
```

$$0$$

Example 4: RCbig and RC can now accept input expressed in terms of the Clifford basis or both, the Clifford and the Grassmann bases. If elements of the Clifford basis are detected, then output is given in terms of the Clifford basis.

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

$$[LCbig, RCbig, clibasis, clieval, cliexpand, climul, clirev, dottedcbasis, dwedge, makeclialiases]$$

```
> p1:=2+3*(e1 &C e2) - &C(e1,e2,e3)+e2we3;
```

$$p1 := 2 + 3\,(e1\ \&C\ e2) - \&C(e1, e2, e3) + e2we3$$

```
> p2:=&C(e1,e2,e3,e4)+2*(e2 &C e3);
```

$$p2 := \&C(e1, e2, e3, e4) + 2\,(e2\ \&C\ e3)$$

```
> RC(p1,p2);
```

$$\begin{aligned} & -B_{2,3}^{\ 2}\, B_{1,4}\, Id - B_{1,2}\, B_{3,4}\, \&C_B(e1, e2, e3) + 3\, B_{2,3}\, B_{1,4}\, \&C_B(e1, e2) + 2\, B_{1,2}\, B_{2,3}\, e3 \\ & - 2\, B_{3,2}\, B_{2,3}\, e1 + B_{2,3}\, B_{1,4}\, \&C_B(e2, e3) - 2\, B_{1,3}\, B_{2,4}\, Id + 3\, B_{1,2}\, B_{3,4}\, \&C_B(e1, e2) \end{aligned}$$

$$- B_{2,3} B_{1,4} \&C_B(e1, e2, e3) - 2 B_{1,2} B_{3,3} e2 - 3 B_{1,3} B_{2,4} \&C_B(e1, e2) + 6 B_{2,2} B_{1,3} Id$$
$$- 6 B_{2,3} B_{1,2} Id + 2 B_{2,3} B_{1,4} Id + 2 B_{1,2} B_{3,4} Id + 3 B_{3,4} B_{2,1} B_{1,2} Id - 3 B_{3,4} B_{1,1} B_{2,2} Id$$
$$+ 3 B_{2,3} B_{2,1} B_{1,4} Id + B_{1,2} B_{3,4} \&C_B(e2, e3) + 4 B_{2,3} Id + B_{2,4} B_{2,1} B_{3,3} Id$$
$$- 3 B_{2,4} B_{2,1} B_{1,3} Id - B_{1,3} B_{3,2} B_{2,4} Id + B_{1,3} B_{2,2} B_{3,4} Id + B_{1,2} B_{3,3} B_{2,4} Id$$
$$- 2 B_{1,2} B_{2,3} B_{3,4} Id + B_{1,3} B_{2,4} \&C_B(e1, e2, e3) + 6 B_{2,3} \&C_B(e1, e2) - B_{2,4} B_{3,1} B_{1,3} e2$$
$$- B_{2,4} B_{2,1} B_{3,3} e1 + B_{2,4} B_{2,1} B_{1,3} e3 + B_{2,4} B_{1,1} B_{3,3} e2 - B_{1,4} B_{3,2} B_{2,3} e1$$
$$+ B_{1,4} B_{2,2} B_{3,3} e1 + 2 B_{3,2} B_{1,3} e2 + 2 B_{2,2} B_{3,3} e1 - 2 B_{2,2} B_{1,3} e3 - B_{3,4} B_{3,1} B_{2,2} e1$$
$$+ B_{3,4} B_{3,1} B_{1,2} e2 + B_{3,4} B_{2,1} B_{3,2} e1 - B_{3,4} B_{2,1} B_{1,2} e3 - B_{3,4} B_{1,1} B_{3,2} e2$$
$$+ B_{3,4} B_{1,1} B_{2,2} e3 + 2 B_{2,3} \&C_B(e2, e3) - 2 B_{2,3}{}^2 Id + 2 B_{3,2} B_{2,3} Id - 2 B_{2,2} B_{3,3} Id$$
$$+ B_{2,3} B_{3,1} B_{1,4} e2 + B_{2,3} B_{2,1} B_{3,4} e1 - B_{2,3} B_{2,1} B_{1,4} e3 - B_{2,3} B_{1,1} B_{3,4} e2$$
$$+ B_{1,3} B_{3,2} B_{2,4} e1 - B_{1,3} B_{2,2} B_{3,4} e1 + B_{1,4} B_{3,2} B_{2,3} Id - B_{1,4} B_{2,2} B_{3,3} Id$$
$$+ B_{3,4} B_{3,1} B_{2,2} Id - B_{3,4} B_{2,1} B_{3,2} Id - B_{2,3} B_{2,1} B_{3,4} Id - B_{1,3} B_{2,4} \&C_B(e2, e3)$$
$$- B_{1,2} B_{3,3} B_{2,4} e1 + B_{1,2} B_{2,3} B_{3,4} e1 + B_{1,3} B_{2,4} B_{2,3} Id - 2 B_{2,3} \&C_B(e1, e2, e3)$$

```
> RC(e1 &C e3,e2 &C e3);
  LC(e1 &C e3,e2 &C e3);
```

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

```
> RCbig(e1 &C e3,e2 &C e3);
  LCbig(e1 &C e3,e2 &C e3);
```

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

RC and RCbig can accept third optional parameter of type 'name' which they use in place of B. However, it has to match the index in `&C`:

```
> B:=matrix(3,3,[1,1,1,
                 -1,1,1,
                 -1,-1,1]):
```

```
> RC(e1 &C e3,e2 &C e3,K); ## <<<--- Error due to different
  indices
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}
```

```
> RC(e1 &C e3,e2 &C e3,B);
```

$$-2 Id + \&C_B(e1, e3)$$

```
> RC(&C[K](e1,e3),&C[K](e2,e3),K); ## <<<--- Error message
  because index K is ignored
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}
```

When using index, enclose `&C` symbol in left quotes:

```
> RC(`&C`[K](e1,e3),`&C`[K](e2,e3),K); ## No error
```

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

```
> RC(`&C`[-K](e1,e3),`&C`[-K](e2,e3),-K);
```

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

However, when used without that parameter, computations are performed with B:

```
> RC(e1 &C e3,e2 &C e3);
```

$$-2\,Id + \&C_B(e1, e3)$$

```
> B:='B':dim_V:=9:
> RC(p1,p2);
```

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

```
> out1:=clicollect(RC(p1,p2)):out2:=clicollect(RCbig(p1,p2)):
> out1-out2;
```

$$0$$

```
> RC(2,3);RC(2*Id,3);RC(2*Id,3*Id);RC(2,p1);RC(2*Id,p1);
  RCbig(2,3);RCbig(2*Id,3);RCbig(2*Id,3*Id);RCbig(2,p1);RCbig(2*I
  d,p1);
```

$$6\,Id$$

$$6\,Id$$

$$6\,Id$$

$$4\,Id + 6\,B_{1,2}\,Id$$
$$4\,Id + 6\,B_{1,2}\,Id$$
$$6\,Id$$
$$6\,Id$$
$$6\,Id$$
$$4\,Id + 6\,B_{1,2}\,Id$$
$$4\,Id + 6\,B_{1,2}\,Id$$

```
> RC(2,3,B);RC(2*Id,3,B);RC(2*Id,3*Id,B);RC(2,p1,B);RC(2*Id,p1,B)
;
RCbig(2,3,B);RCbig(2*Id,3,B);RCbig(2*Id,3*Id,B);RCbig(2,p1,B);R
Cbig(2*Id,p1,B);
```

$$6\,Id$$
$$6\,Id$$
$$6\,Id$$
$$4\,Id + 6\,B_{1,2}\,Id$$
$$4\,Id + 6\,B_{1,2}\,Id$$
$$6\,Id$$
$$6\,Id$$
$$6\,Id$$
$$4\,Id + 6\,B_{1,2}\,Id$$
$$4\,Id + 6\,B_{1,2}\,Id$$

Example 5: Using the optional parameter of 'type/name` allows one to use a different form than the default bilinear form 'B'. For example, one could use, for example, the symmetric or the antisymmetric part of B in RC.

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

```
> B:=matrix(3,3,[d1,a,b,
                 -a,d2,c,
                 -b,-c,d3]);
```

$$B := \begin{bmatrix} d1 & a & b \\ -a & d2 & c \\ -b & -c & d3 \end{bmatrix}$$

```
> g,A:=evalm(1/2*(B+transpose(B))),evalm(1/2*(B-transpose(B)));
```

$$g, A := \begin{bmatrix} d1 & 0 & 0 \\ 0 & d2 & 0 \\ 0 & 0 & d3 \end{bmatrix}, \begin{bmatrix} 0 & a & b \\ -a & 0 & c \\ -b & -c & 0 \end{bmatrix}$$

```
> clicollect(RC(e1we2+e3-2*e1we2we3,e1we2+e3-2*e1we2we3,g));
clicollect(RCbig(e1we2+e3-2*e1we2we3,e1we2+e3-2*e1we2we3,g));
```

$$-(d1\ d2 - d3 + 4\ d1\ d2\ d3)\ Id - 2\ d3\ e1we2 + 2\ d1\ d2\ e3$$

$$-(d1\ d2 - d3 + 4\ d1\ d2\ d3)\ Id - 2\ d3\ e1we2 + 2\ d1\ d2\ e3$$

```
> clicollect(RC(e1we2+e3-2*e1we2we3,e1we2+e3-2*e1we2we3,A));
  clicollect(RCbig(e1we2+e3-2*e1we2we3,e1we2+e3-2*e1we2we3,A));
```

$$-a^2\ Id + 2\ c\ e1we3 - 2\ b\ e2we3 + 2\ a^2\ e3 + c\ (2\ a + 1)\ e1 - b\ (2\ a + 1)\ e2$$

$$-a^2\ Id + 2\ c\ e1we3 - 2\ b\ e2we3 + 2\ a^2\ e3 + c\ (2\ a + 1)\ e1 - b\ (2\ a + 1)\ e2$$

```
>
```

See Also: Clifford:-LC, Clifford:-LCQ, Clifford:-RCQ, Clifford:-cmul, Clifford:-`type/clipolynom`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-RCQ - right contraction in Cl(Q), the Clifford algebra of a quadratic form Q

Calling Sequence:

RCQ(u,v);
RCQ(u,v,lname);

Parameters:

u, v - expressions of the type 'cliscalar', 'clibasmon', 'climon', or 'clipolynom'
lname - (optional) argument of type name, symbol, matrix, array, or
`&*`(numeric,{name,symbol,array,matrix})

Description:

- Procedure 'RCQ' defines a *right contraction* in Cl(Q) between a multivector v acting from the right on a multivector u.
- When used with the third optional argument, e.g., K or -K, it computes right contraction with respect to the diagonal entries of K or -K, respectively. Otherwise, it computes with respect to the diagonal entries of B
- This procedure is obsolete in version 6 of CLIFFORD as the procedure Clifford:-RC can accept a third argument of type name, symbol, matrix, or array, that is, it can compute the right contraction RC(u,v,K) of u by v from the right with respect to the explicitly specified third argument.

Examples:

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

Example 1: Right contraction with respect to an unassigned name:

```
> RCQ(3*e1we3+b*e2we3,e1+2*e2);    # right contraction in Cl(B)
  RCQ(3*e1we3+b*e2we3,e1+2*e2,B); # right contraction in Cl(B)
  RCQ(3*e1we3+b*e2we3,e1+2*e2,-B);# right contraction in Cl(-B)
  RCQ(3*e1we3+b*e2we3,e1+2*e2,K); # right contraction in Cl(K)
  RCQ(3*e1we3+b*e2we3,e1+2*e2,-K);# right contraction in Cl(-K)
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$-3\, B_{1,1}\, e3 - 2\, b\, B_{2,2}\, e3$$

$$-3\, B_{1,1}\, e3 - 2\, b\, B_{2,2}\, e3$$

$$3\, B_{1,1}\, e3 + 2\, b\, B_{2,2}\, e3$$

$$-3\, K_{1,1}\, e3 - 2\, b\, K_{2,2}\, e3$$

$$3\, K_{1,1}\, e3 + 2\, b\, K_{2,2}\, e3$$

```
> RCQ(3*Id+2*eiwej,e1+4);
```

$$12\, Id + 2\, B_{j,1}\, ei - 2\, B_{i,1}\, ej + 8\, eiwej$$

```
> RCQ(eiwej+ekwel,es);
```

$$B_{j,s}\, ei - B_{i,s}\, ej + B_{l,s}\, ek - B_{k,s}\, el$$

```
> RCQ(e2+3*e1,2*Pi*Id);
```

$$2\,\pi\,e2 + 6\,\pi\,e1$$

Example 2: Right contraction with respect to an assigned name:

```
> B,K:=matrix(3,3,[a,b,c,d,m,f,g,h,k]),matrix(3,3,[a,0,0,0,m,0,0,
  0,k]);
```

$$B, K := \begin{bmatrix} a & b & c \\ d & m & f \\ g & h & k \end{bmatrix}, \begin{bmatrix} a & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & k \end{bmatrix}$$

```
> RCQ(3*e1we3+b*e2we3,e1+2*e2);     # contraction in Cl(B)
  RCQ(3*e1we3+b*e2we3,e1+2*e2,B);   # contraction in Cl(B)
  RCQ(3*e1we3+b*e2we3,e1+2*e2,-B);  # contraction in Cl(-B)
  RCQ(3*e1we3+b*e2we3,e1+2*e2,K);   # contraction in Cl(K)
  RCQ(3*e1we3+b*e2we3,e1+2*e2,-K);  # contraction in Cl(-K)
```

$$-3\,a\,e3 - 2\,b\,m\,e3$$

$$-3\,a\,e3 - 2\,b\,m\,e3$$

$$3\,a\,e3 + 2\,b\,m\,e3$$

$$-3\,a\,e3 - 2\,b\,m\,e3$$

$$3\,a\,e3 + 2\,b\,m\,e3$$

```
>
```

See Also: Clifford:-RC, Clifford:-LCQ, Clifford:-LC, Clifford:-cmulQ, Clifford:-`type/clipolynom`, Clifford:-`type/climon`, Clifford:-`type/clibasmon`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-rd_clibasmon, Clifford:-rd_climon, Clifford:-rd_clipolynom - generate random Grassmann basis monomial, monomial, and polynomial with integer coefficients

Calling Sequence:

rd_clibasmon();
rd_clibasmon(a1);
rd_clibasmon(a1,a2);
rd_climon();
rd_climon(a1);
rd_climon(a1,a2,a3);
rd_clipolynom();
rd_clipolynom(a1);
rd_clipolynom(a1,a2);
rd_clipolynom(a1,a2,a3);
rd_clipolynom(a1,a2,a3,a4);

Parameters:

a1 - non negative integer (maximum index)
a2 - non negative integer <= a1 (maximum grade)
a3 - positive integer (maximum absolute value of integer coefficients
a4 - positive integer (maximum number of monomial terms)

Description:

- These three procedures are for testing purposes. They generate random Grassmann basis monomial, monomial and polynomial with random integer coefficients of types `type/clibasmon`, `type/climon`, and `type/clipolynom` respectively.
- When used without any parameters, default values are used as follows:

  - maximum index <= 9;
  - maximum grade <= 4;
  - maximum absolute value of coefficients <= 12;
  - maximum number of monomial terms <= 5.
- When used with one parameter, the parameter will be used as the maximum value of any index set.
- When used with two parameters, the first parameter will be the maximum value of any index set while the second parameter will be the maximum grade;
- When used with three parameters, the first parameter will be the maximum value of any index set, the second parameter will be the maximum grade, and the third parameter will be the maximum absolute value of any coefficient;

- When used with four parameters, the first parameter will be the maximum value of any index set, the second parameter will be the maximum grade, the third parameter will be the maximum absolute value of any coefficient, and the fourth parameter will be the maximum number of monomial terms in a polynomial.

Examples:

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

Example 1: Generating random Grassmann basis monomials:

```
> L:={}: for k from 1 to 30 do L:={op(L),rd_clibasmon()} end do:L;
```

$\{Id, e1we3, e4we6we7we8, e1we3we4we9, e2we6we7we9, e5, e3, e1, e6we9, e3we6, e2we3we4we7, e1we2we5we7, e2we3we4we6, e3we6we7we9, e7, e8, e4we6we8, e3we6we8we9, e2we7, e2we5, e4we8, e3we4we7we9, e1we2we3, e2we5we9, e9\}$

```
> L:={}: for k from 1 to 30 do L:={op(L),rd_clibasmon(3)} end do:L;
```

$\{Id, e1we3, e1we2, e3, e2, e1we2we3\}$

```
> L:={}: for k from 1 to 30 do L:={op(L),rd_clibasmon(5,3)} end do:L;
```

$\{e1we3we4, e2we3we4, e1we2we5, e1we2we3, e1we3we5, e1we4we5, e2we3we5, e3we4we5, e1we2we4, e2we4we5\}$

Example 2: Generating random Grassmann monomials:

```
> L:={}: for k from 1 to 30 do L:={op(L),rd_climon()} end do:L;
```

$\{Id, -Id, e1we3we5, 10\, e4we6we7we8, -6\, Id, -2\, e1we2we3we6, 4\, Id, -e4we5we8, -10\, Id, -4\, e1we4we5we6, -7\, e2we4we6we9, -9\, Id, -4\, Id, 6\, e1we6we8we9, 2\, Id, -e2, -5\, e1we3we6we9, 2\, e3we4we8we9, -e1we3we7we9, e2we3we9, -2\, e2we6, -6\, e2we8, -7\, e1we3we9, 5\, e1we4we5we7, e5we7we8, e2we4we7we8, -5\, e3, -e6we7we8\}$

```
> L:={}: for k from 1 to 30 do L:={op(L),rd_climon(5)} end do:L;
```

$\{Id, e1we2we3we4we5, 9\, e2, e4, 4\, e1we4, e1, e2, e1we2we4we5, -6\, e1we3we4, -2\, Id, 12\, e3, -6\, Id, 4\, Id, -3\, Id, 4\, e1we3we4, -5\, e1we2we3we4we5, -4\, Id, -e2, 2\, e2, 6\, e1we4, e1we2we3we5, 4\, e1we4we5, 9\, e1we2we3we4we5, -2\, e4, 2\, e1we3we4we5, 5\, e1, 3\, e1we5, -5\, e1we2we3we5\}$

```
> L:={}: for k from 1 to 30 do L:={op(L),rd_climon(5,3)} end do:L;
```

$\{e1we2we5, 5\, e1we3we5, -4\, e3we4we5, 2\, e1we3we5, e1we2we3, e3we4we5, 8\, e1we2we5, 2\, e1we2we3, -3\, e1we2we5, 7\, e1we3we4, -4\, e1we2we4, 9\, e1we4we5, -e3we4we5, 9\, e1we2we4, -2\, e2we4we5, -e2we3we4, -2\, e1we2we3, -5\, e1we2we4, 3\, e1we2we5, -8\, e2we4we5, -7\, e1we2we3, -2\, e2we3we4, 5\, e1we2we5, 8\, e1we3we4, 2\, e2we3we4, 3\, e1we4we5\}$

Example 3: Generating random Grassmann monomials:

```
> L:={}: for k from 1 to 20 do L:={op(L),rd_clipolynom()} end do:L;
```

$\{0, -4\,Id + e3we4we9 - 4\,e2we6 - 4\,e1we6we7 - 4\,e1we6 - 4\,e6we7, Id, -Id,$
$-3\,e8 + 3\,e1 - 3\,e5, -7\,e1we4we8we9 + 4\,e1we2we3we6 + 4\,e2we5,$
$Id + e1we2we9 + e4we8we9, 2\,Id - 4\,e1we3we5we6 - 4\,e1we3we6we9,$
$-3\,Id - 3\,e1we5we8 + 2\,e4we9 + 2\,e1we4we7 + 2\,e1we2we9, 3\,e8 + 3\,Id + 3\,e2 - 4\,e9 - 4\,e3,$
$-3\,e1we2we5, 8\,Id + 4\,e3 + 4\,e4 - 3\,e5 + 4\,e6,$
$-4\,Id - 4\,e3we6 + 9\,e4we8 + 9\,e2we3we8 - 4\,e6, 11\,e8we9 - 10\,e6, 2\,Id,$
$-12\,Id + 2\,e3we4we6 - 12\,e3we8 - 12\,e1we5we7, 4\,Id,$
$7\,Id + 7\,e1we5we7 - 3\,e2we4we6 - 3\,e4we5we7, 6\,Id - e5 + 6\,e3 - e6 - e4 + 6\,e2, 2\,e9 - Id\}$

```
> L:={}: for k from 1 to 20 do L:={op(L),rd_clipolynom(4)} end do:L;
```

$\{Id, -2\,Id + 2\,e4, e3we4 - e2we4 - e4, -e1we3we4 + 11\,e3we4 + 11\,e4,$
$6\,e1we4 - 3\,e2we3 - 3\,e1we2we3, -e3we4 + 4\,e4, 5\,Id, -5\,Id - 5\,e2, 12\,Id,$
$9\,Id + 9\,e2 - 4\,e1 - 4\,e3, -3\,Id - 3\,e1 + e4 + e2, -6\,e1we2we4 + e1we3,$
$-2\,Id + 2\,e3 + 2\,e2 - 2\,e1, -e3 + 7\,Id - e1 + 7\,e2 + 7\,e4, -4\,e1we2 - 4\,e2we3 - 4\,e4 + 4\,e1,$
$-3\,e4 + 5\,Id - 3\,e2 + 5\,e3we4, 4\,e1we3 - 2\,e2 - 2\,e3we4 + 4\,e2we3, 8\,Id,$
$-6\,Id + 5\,e2we3we4 + 5\,e2we3 - 6\,e1we2we3we4 - 6\,e3 + 5\,e2we4, 3\,Id\}$

```
> L:={}: for k from 1 to 20 do L:={op(L),rd_clipolynom(5,2)} end do:L;
```

$\{2\,Id, Id, -4\,Id, e2 + e4 - e1 + e3, -3\,Id - 3\,e2 - 3\,e1 - 3\,e5, -2\,Id, -2\,e5, 5\,e2we4,$
$-2\,Id - e5 + e3 - e1 - e4, 4\,e1 + 4\,e5 - 4\,e4 + 4\,e3, -2\,Id + 3\,e3, -4\,e1 + e5 + e2 + e3 + e4,$
$2\,Id - 3\,e4 + 2\,e3 - 3\,e2, Id + e4 + e5 - 2\,e2 + e1, 8\,Id - 9\,e2 - 9\,e4 - 9\,e5,$
$8\,Id - 3\,e1 + 4\,e5 - 3\,e3 - 3\,e2, e2 + e2we3 - e4we5 + e3we4, Id + e2,$
$-6\,e5 + 12\,e4 + 12\,Id - 6\,e3 - 6\,e2\}$

```
> L:={}: for k from 1 to 20 do L:={op(L),rd_clipolynom(5,3,10)} end do:L;
```

$\{7\,Id - 3\,e1 - 3\,e5, Id, -4\,Id + 7\,e4 - 4\,e2we3, 5\,Id + 5\,e5 - 8\,e2 - 8\,e1 + 5\,e1we5,$
$6\,e2 - 9\,e1 + 6\,e4, -5\,e3we4 - 5\,Id, -9\,e2 - 9\,e3 - 9\,e1 + e5, -3\,e2we3we5,$
$3\,Id + 3\,e3 - 6\,e2we4 + 3\,e3we5 - 6\,e3we4 - 6\,e2we5, -9\,e1 - 9\,e2, -9\,Id, 2\,Id + 2\,e4we5,$
$5\,e4 - 9\,e2 - 9\,e1 + 5\,Id, 6\,Id, 8\,Id, -4\,Id - 2\,e4 - 2\,e1 + 2\,e2, -2\,e1, 3\,Id\}$

```
> L:={}: for k from 1 to 10 do L:={op(L),rd_clipolynom(5,3,10,5)} end do:L;
```

$\{7\,Id - e2we5 - e3we4 + 7\,e3 + 7\,e2we3we5 - e4,$
$-Id + 3\,e2we3we5 + 3\,e4 + 3\,e1we4 - e1we2we4 - e5,$
$-6\,e2we4 - 6\,e1we3we5 - 6\,e3we4we5 + 6\,e2 - 6\,e1we3we4,$
$5\,e1we5 - 9\,e1we3we5 + 5\,e1we2we4 - 9\,e2we4 - 9\,e5,$
$3\,Id - 3\,e2 - 3\,e2we4 - 3\,e2we3 - 3\,e1we4we5,$

$$-10\, Id + 4\, e1we5 + 4\, e2we3we4 - 10\, e2we3 - 10\, e1we3we4 + 4\, e1we2we4,$$
$$-5\, Id + 6\, e2we5 + 6\, e5 - 5\, e1we2we4 - 5\, e4we5 + 6\, e3we4,$$
$$-Id - 2\, e3we4we5 - 2\, e2we4 - 2\, e1we4we5 - 2\, e1,$$
$$3\, e3we4 - 4\, e1we2we4 - 4\, e1we2we3 - 4\, e4 - 4\, e1we4,$$
$$Id - 3\, e1we2we3 + 4\, e1we2we5 - 3\, e2we3 + 4\, e2we3we5\}$$

```
>
```

See Also: Clifford:-`type/clipolynom`, Clifford:-`type/climon`, Clifford:-`type/clibasmon`, Clifford:-cmul

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-rd_clibasmon, Clifford:-rd_climon, Clifford:-rd_clipolynom - generate random Grassmann basis monomial, monomial, and polynomial with integer coefficients

Calling Sequence:

rd_clibasmon();
rd_clibasmon(a1);
rd_clibasmon(a1,a2);
rd_climon();
rd_climon(a1);
rd_climon(a1,a2,a3);
rd_clipolynom();
rd_clipolynom(a1);
rd_clipolynom(a1,a2);
rd_clipolynom(a1,a2,a3);
rd_clipolynom(a1,a2,a3,a4);

Parameters:

a1 - non negative integer (maximum index)
a2 - non negative integer <= a1 (maximum grade)
a3 - positive integer (maximum absolute value of integer coefficients
a4 - positive integer (maximum number of monomial terms)

Description:

- These three procedures are for testing purposes. They generate random Grassmann basis monomial, monomial and polynomial with random integer coefficients of types `type/clibasmon`, `type/climon`, and `type/clipolynom` respectively.
- When used without any parameters, default values are used as follows:

  - maximum index <= 9;
  - maximum grade <= 4;
  - maximum absolute value of coefficients <= 12;
  - maximum number of monomial terms <= 5.
- When used with one parameter, the parameter will be used as the maximum value of any index set.
- When used with two parameters, the first parameter will be the maximum value of any index set while the second parameter will be the maximum grade;
- When used with three parameters, the first parameter will be the maximum value of any index set, the second parameter will be the maximum grade, and the third parameter will be the maximum absolute value of any coefficient;

- When used with four parameters, the first parameter will be the maximum value of any index set, the second parameter will be the maximum grade, the third parameter will be the maximum absolute value of any coefficient, and the fourth parameter will be the maximum number of monomial terms in a polynomial.

Examples:

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

Example 1: Generating random Grassmann basis monomials:

```
> L:={}: for k from 1 to 30 do L:={op(L),rd_clibasmon()} end do:L;
```

$\{Id, e1we3, e4we6we7we8, e1we3we4we9, e2we6we7we9, e5, e3, e1, e6we9, e3we6, e2we3we4we7, e1we2we5we7, e2we3we4we6, e3we6we7we9, e7, e8, e4we6we8, e3we6we8we9, e2we7, e2we5, e4we8, e3we4we7we9, e1we2we3, e2we5we9, e9\}$

```
> L:={}: for k from 1 to 30 do L:={op(L),rd_clibasmon(3)} end do:L;
```

$\{Id, e1we3, e1we2, e3, e2, e1we2we3\}$

```
> L:={}: for k from 1 to 30 do L:={op(L),rd_clibasmon(5,3)} end do:L;
```

$\{e1we3we4, e2we3we4, e1we2we5, e1we2we3, e1we3we5, e1we4we5, e2we3we5, e3we4we5, e1we2we4, e2we4we5\}$

Example 2: Generating random Grassmann monomials:

```
> L:={}: for k from 1 to 30 do L:={op(L),rd_climon()} end do:L;
```

$\{Id, -Id, e1we3we5, 10\, e4we6we7we8, -6\, Id, -2\, e1we2we3we6, 4\, Id, -e4we5we8, -10\, Id, -4\, e1we4we5we6, -7\, e2we4we6we9, -9\, Id, -4\, Id, 6\, e1we6we8we9, 2\, Id, -e2, -5\, e1we3we6we9, 2\, e3we4we8we9, -e1we3we7we9, e2we3we9, -2\, e2we6, -6\, e2we8, -7\, e1we3we9, 5\, e1we4we5we7, e5we7we8, e2we4we7we8, -5\, e3, -e6we7we8\}$

```
> L:={}: for k from 1 to 30 do L:={op(L),rd_climon(5)} end do:L;
```

$\{Id, e1we2we3we4we5, 9\, e2, e4, 4\, e1we4, e1, e2, e1we2we4we5, -6\, e1we3we4, -2\, Id, 12\, e3, -6\, Id, 4\, Id, -3\, Id, 4\, e1we3we4, -5\, e1we2we3we4we5, -4\, Id, -e2, 2\, e2, 6\, e1we4, e1we2we3we5, 4\, e1we4we5, 9\, e1we2we3we4we5, -2\, e4, 2\, e1we3we4we5, 5\, e1, 3\, e1we5, -5\, e1we2we3we5\}$

```
> L:={}: for k from 1 to 30 do L:={op(L),rd_climon(5,3)} end do:L;
```

$\{e1we2we5, 5\, e1we3we5, -4\, e3we4we5, 2\, e1we3we5, e1we2we3, e3we4we5, 8\, e1we2we5, 2\, e1we2we3, -3\, e1we2we5, 7\, e1we3we4, -4\, e1we2we4, 9\, e1we4we5, -e3we4we5, 9\, e1we2we4, -2\, e2we4we5, -e2we3we4, -2\, e1we2we3, -5\, e1we2we4, 3\, e1we2we5, -8\, e2we4we5, -7\, e1we2we3, -2\, e2we3we4, 5\, e1we2we5, 8\, e1we3we4, 2\, e2we3we4, 3\, e1we4we5\}$

Example 3: Generating random Grassmann monomials:

```
> L:={}: for k from 1 to 20 do L:={op(L),rd_clipolynom()} end do:L;
```

$\{0, -4\,Id + e3we4we9 - 4\,e2we6 - 4\,e1we6we7 - 4\,e1we6 - 4\,e6we7, Id, -Id,$
$-3\,e8 + 3\,e1 - 3\,e5, -7\,e1we4we8we9 + 4\,e1we2we3we6 + 4\,e2we5,$
$Id + e1we2we9 + e4we8we9, 2\,Id - 4\,e1we3we5we6 - 4\,e1we3we6we9,$
$-3\,Id - 3\,e1we5we8 + 2\,e4we9 + 2\,e1we4we7 + 2\,e1we2we9, 3\,e8 + 3\,Id + 3\,e2 - 4\,e9 - 4\,e3,$
$-3\,e1we2we5, 8\,Id + 4\,e3 + 4\,e4 - 3\,e5 + 4\,e6,$
$-4\,Id - 4\,e3we6 + 9\,e4we8 + 9\,e2we3we8 - 4\,e6, 11\,e8we9 - 10\,e6, 2\,Id,$
$-12\,Id + 2\,e3we4we6 - 12\,e3we8 - 12\,e1we5we7, 4\,Id,$
$7\,Id + 7\,e1we5we7 - 3\,e2we4we6 - 3\,e4we5we7, 6\,Id - e5 + 6\,e3 - e6 - e4 + 6\,e2, 2\,e9 - Id\}$

```
> L:={}: for k from 1 to 20 do L:={op(L),rd_clipolynom(4)} end do:L;
```

$\{Id, -2\,Id + 2\,e4, e3we4 - e2we4 - e4, -e1we3we4 + 11\,e3we4 + 11\,e4,$
$6\,e1we4 - 3\,e2we3 - 3\,e1we2we3, -e3we4 + 4\,e4, 5\,Id, -5\,Id - 5\,e2, 12\,Id,$
$9\,Id + 9\,e2 - 4\,e1 - 4\,e3, -3\,Id - 3\,e1 + e4 + e2, -6\,e1we2we4 + e1we3,$
$-2\,Id + 2\,e3 + 2\,e2 - 2\,e1, -e3 + 7\,Id - e1 + 7\,e2 + 7\,e4, -4\,e1we2 - 4\,e2we3 - 4\,e4 + 4\,e1,$
$-3\,e4 + 5\,Id - 3\,e2 + 5\,e3we4, 4\,e1we3 - 2\,e2 - 2\,e3we4 + 4\,e2we3, 8\,Id,$
$-6\,Id + 5\,e2we3we4 + 5\,e2we3 - 6\,e1we2we3we4 - 6\,e3 + 5\,e2we4, 3\,Id\}$

```
> L:={}: for k from 1 to 20 do L:={op(L),rd_clipolynom(5,2)} end do:L;
```

$\{2\,Id, Id, -4\,Id, e2 + e4 - e1 + e3, -3\,Id - 3\,e2 - 3\,e1 - 3\,e5, -2\,Id, -2\,e5, 5\,e2we4,$
$-2\,Id - e5 + e3 - e1 - e4, 4\,e1 + 4\,e5 - 4\,e4 + 4\,e3, -2\,Id + 3\,e3, -4\,e1 + e5 + e2 + e3 + e4,$
$2\,Id - 3\,e4 + 2\,e3 - 3\,e2, Id + e4 + e5 - 2\,e2 + e1, 8\,Id - 9\,e2 - 9\,e4 - 9\,e5,$
$8\,Id - 3\,e1 + 4\,e5 - 3\,e3 - 3\,e2, e2 + e2we3 - e4we5 + e3we4, Id + e2,$
$-6\,e5 + 12\,e4 + 12\,Id - 6\,e3 - 6\,e2\}$

```
> L:={}: for k from 1 to 20 do L:={op(L),rd_clipolynom(5,3,10)} end do:L;
```

$\{7\,Id - 3\,e1 - 3\,e5, Id, -4\,Id + 7\,e4 - 4\,e2we3, 5\,Id + 5\,e5 - 8\,e2 - 8\,e1 + 5\,e1we5,$
$6\,e2 - 9\,e1 + 6\,e4, -5\,e3we4 - 5\,Id, -9\,e2 - 9\,e3 - 9\,e1 + e5, -3\,e2we3we5,$
$3\,Id + 3\,e3 - 6\,e2we4 + 3\,e3we5 - 6\,e3we4 - 6\,e2we5, -9\,e1 - 9\,e2, -9\,Id, 2\,Id + 2\,e4we5,$
$5\,e4 - 9\,e2 - 9\,e1 + 5\,Id, 6\,Id, 8\,Id, -4\,Id - 2\,e4 - 2\,e1 + 2\,e2, -2\,e1, 3\,Id\}$

```
> L:={}: for k from 1 to 10 do L:={op(L),rd_clipolynom(5,3,10,5)} end do:L;
```

$\{7\,Id - e2we5 - e3we4 + 7\,e3 + 7\,e2we3we5 - e4,$
$-Id + 3\,e2we3we5 + 3\,e4 + 3\,e1we4 - e1we2we4 - e5,$
$-6\,e2we4 - 6\,e1we3we5 - 6\,e3we4we5 + 6\,e2 - 6\,e1we3we4,$
$5\,e1we5 - 9\,e1we3we5 + 5\,e1we2we4 - 9\,e2we4 - 9\,e5,$
$3\,Id - 3\,e2 - 3\,e2we4 - 3\,e2we3 - 3\,e1we4we5,$

$$-10\, Id + 4\, e1we5 + 4\, e2we3we4 - 10\, e2we3 - 10\, e1we3we4 + 4\, e1we2we4,$$
$$-5\, Id + 6\, e2we5 + 6\, e5 - 5\, e1we2we4 - 5\, e4we5 + 6\, e3we4,$$
$$-Id - 2\, e3we4we5 - 2\, e2we4 - 2\, e1we4we5 - 2\, e1,$$
$$3\, e3we4 - 4\, e1we2we4 - 4\, e1we2we3 - 4\, e4 - 4\, e1we4,$$
$$Id - 3\, e1we2we3 + 4\, e1we2we5 - 3\, e2we3 + 4\, e2we3we5\}$$

```
>
```

See Also: Clifford:-`type/clipolynom`, Clifford:-`type/climon`, Clifford:-`type/clibasmon`, Clifford:-cmul

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-rd_clibasmon, Clifford:-rd_climon, Clifford:-rd_clipolynom - generate random Grassmann basis monomial, monomial, and polynomial with integer coefficients

Calling Sequence:

rd_clibasmon();
rd_clibasmon(a1);
rd_clibasmon(a1,a2);
rd_climon();
rd_climon(a1);
rd_climon(a1,a2,a3);
rd_clipolynom();
rd_clipolynom(a1);
rd_clipolynom(a1,a2);
rd_clipolynom(a1,a2,a3);
rd_clipolynom(a1,a2,a3,a4);

Parameters:

a1 - non negative integer (maximum index)
a2 - non negative integer <= a1 (maximum grade)
a3 - positive integer (maximum absolute value of integer coefficients
a4 - positive integer (maximum number of monomial terms)

Description:

- These three procedures are for testing purposes. They generate random Grassmann basis monomial, monomial and polynomial with random integer coefficients of types `type/clibasmon`, `type/climon`, and `type/clipolynom` respectively.
- When used without any parameters, default values are used as follows:

  - maximum index <= 9;
  - maximum grade <= 4;
  - maximum absolute value of coefficients <= 12;
  - maximum number of monomial terms <= 5.
- When used with one parameter, the parameter will be used as the maximum value of any index set.
- When used with two parameters, the first parameter will be the maximum value of any index set while the second parameter will be the maximum grade;
- When used with three parameters, the first parameter will be the maximum value of any index set, the second parameter will be the maximum grade, and the third parameter will be the maximum absolute value of any coefficient;

- When used with four parameters, the first parameter will be the maximum value of any index set, the second parameter will be the maximum grade, the third parameter will be the maximum absolute value of any coefficient, and the fourth parameter will be the maximum number of monomial terms in a polynomial.

Examples:

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

Example 1: Generating random Grassmann basis monomials:

```
> L:={}: for k from 1 to 30 do L:={op(L),rd_clibasmon()} end do:L;
```

$\{Id, e1we3, e4we6we7we8, e1we3we4we9, e2we6we7we9, e5, e3, e1, e6we9, e3we6, e2we3we4we7, e1we2we5we7, e2we3we4we6, e3we6we7we9, e7, e8, e4we6we8, e3we6we8we9, e2we7, e2we5, e4we8, e3we4we7we9, e1we2we3, e2we5we9, e9\}$

```
> L:={}: for k from 1 to 30 do L:={op(L),rd_clibasmon(3)} end do:L;
```

$\{Id, e1we3, e1we2, e3, e2, e1we2we3\}$

```
> L:={}: for k from 1 to 30 do L:={op(L),rd_clibasmon(5,3)} end do:L;
```

$\{e1we3we4, e2we3we4, e1we2we5, e1we2we3, e1we3we5, e1we4we5, e2we3we5, e3we4we5, e1we2we4, e2we4we5\}$

Example 2: Generating random Grassmann monomials:

```
> L:={}: for k from 1 to 30 do L:={op(L),rd_climon()} end do:L;
```

$\{Id, -Id, e1we3we5, 10\, e4we6we7we8, -6\, Id, -2\, e1we2we3we6, 4\, Id, -e4we5we8, -10\, Id, -4\, e1we4we5we6, -7\, e2we4we6we9, -9\, Id, -4\, Id, 6\, e1we6we8we9, 2\, Id, -e2, -5\, e1we3we6we9, 2\, e3we4we8we9, -e1we3we7we9, e2we3we9, -2\, e2we6, -6\, e2we8, -7\, e1we3we9, 5\, e1we4we5we7, e5we7we8, e2we4we7we8, -5\, e3, -e6we7we8\}$

```
> L:={}: for k from 1 to 30 do L:={op(L),rd_climon(5)} end do:L;
```

$\{Id, e1we2we3we4we5, 9\, e2, e4, 4\, e1we4, e1, e2, e1we2we4we5, -6\, e1we3we4, -2\, Id, 12\, e3, -6\, Id, 4\, Id, -3\, Id, 4\, e1we3we4, -5\, e1we2we3we4we5, -4\, Id, -e2, 2\, e2, 6\, e1we4, e1we2we3we5, 4\, e1we4we5, 9\, e1we2we3we4we5, -2\, e4, 2\, e1we3we4we5, 5\, e1, 3\, e1we5, -5\, e1we2we3we5\}$

```
> L:={}: for k from 1 to 30 do L:={op(L),rd_climon(5,3)} end do:L;
```

$\{e1we2we5, 5\, e1we3we5, -4\, e3we4we5, 2\, e1we3we5, e1we2we3, e3we4we5, 8\, e1we2we5, 2\, e1we2we3, -3\, e1we2we5, 7\, e1we3we4, -4\, e1we2we4, 9\, e1we4we5, -e3we4we5, 9\, e1we2we4, -2\, e2we4we5, -e2we3we4, -2\, e1we2we3, -5\, e1we2we4, 3\, e1we2we5, -8\, e2we4we5, -7\, e1we2we3, -2\, e2we3we4, 5\, e1we2we5, 8\, e1we3we4, 2\, e2we3we4, 3\, e1we4we5\}$

Example 3: Generating random Grassmann monomials:

```
> L:={}: for k from 1 to 20 do L:={op(L),rd_clipolynom()} end do:L;
```

$\{0, -4\,Id + e3we4we9 - 4\,e2we6 - 4\,e1we6we7 - 4\,e1we6 - 4\,e6we7, Id, -Id,$
$-3\,e8 + 3\,e1 - 3\,e5, -7\,e1we4we8we9 + 4\,e1we2we3we6 + 4\,e2we5,$
$Id + e1we2we9 + e4we8we9, 2\,Id - 4\,e1we3we5we6 - 4\,e1we3we6we9,$
$-3\,Id - 3\,e1we5we8 + 2\,e4we9 + 2\,e1we4we7 + 2\,e1we2we9, 3\,e8 + 3\,Id + 3\,e2 - 4\,e9 - 4\,e3,$
$-3\,e1we2we5, 8\,Id + 4\,e3 + 4\,e4 - 3\,e5 + 4\,e6,$
$-4\,Id - 4\,e3we6 + 9\,e4we8 + 9\,e2we3we8 - 4\,e6, 11\,e8we9 - 10\,e6, 2\,Id,$
$-12\,Id + 2\,e3we4we6 - 12\,e3we8 - 12\,e1we5we7, 4\,Id,$
$7\,Id + 7\,e1we5we7 - 3\,e2we4we6 - 3\,e4we5we7, 6\,Id - e5 + 6\,e3 - e6 - e4 + 6\,e2, 2\,e9 - Id\}$

```
> L:={}: for k from 1 to 20 do L:={op(L),rd_clipolynom(4)} end do:L;
```

$\{Id, -2\,Id + 2\,e4, e3we4 - e2we4 - e4, -e1we3we4 + 11\,e3we4 + 11\,e4,$
$6\,e1we4 - 3\,e2we3 - 3\,e1we2we3, -e3we4 + 4\,e4, 5\,Id, -5\,Id - 5\,e2, 12\,Id,$
$9\,Id + 9\,e2 - 4\,e1 - 4\,e3, -3\,Id - 3\,e1 + e4 + e2, -6\,e1we2we4 + e1we3,$
$-2\,Id + 2\,e3 + 2\,e2 - 2\,e1, -e3 + 7\,Id - e1 + 7\,e2 + 7\,e4, -4\,e1we2 - 4\,e2we3 - 4\,e4 + 4\,e1,$
$-3\,e4 + 5\,Id - 3\,e2 + 5\,e3we4, 4\,e1we3 - 2\,e2 - 2\,e3we4 + 4\,e2we3, 8\,Id,$
$-6\,Id + 5\,e2we3we4 + 5\,e2we3 - 6\,e1we2we3we4 - 6\,e3 + 5\,e2we4, 3\,Id\}$

```
> L:={}: for k from 1 to 20 do L:={op(L),rd_clipolynom(5,2)} end do:L;
```

$\{2\,Id, Id, -4\,Id, e2 + e4 - e1 + e3, -3\,Id - 3\,e2 - 3\,e1 - 3\,e5, -2\,Id, -2\,e5, 5\,e2we4,$
$-2\,Id - e5 + e3 - e1 - e4, 4\,e1 + 4\,e5 - 4\,e4 + 4\,e3, -2\,Id + 3\,e3, -4\,e1 + e5 + e2 + e3 + e4,$
$2\,Id - 3\,e4 + 2\,e3 - 3\,e2, Id + e4 + e5 - 2\,e2 + e1, 8\,Id - 9\,e2 - 9\,e4 - 9\,e5,$
$8\,Id - 3\,e1 + 4\,e5 - 3\,e3 - 3\,e2, e2 + e2we3 - e4we5 + e3we4, Id + e2,$
$-6\,e5 + 12\,e4 + 12\,Id - 6\,e3 - 6\,e2\}$

```
> L:={}: for k from 1 to 20 do L:={op(L),rd_clipolynom(5,3,10)} end do:L;
```

$\{7\,Id - 3\,e1 - 3\,e5, Id, -4\,Id + 7\,e4 - 4\,e2we3, 5\,Id + 5\,e5 - 8\,e2 - 8\,e1 + 5\,e1we5,$
$6\,e2 - 9\,e1 + 6\,e4, -5\,e3we4 - 5\,Id, -9\,e2 - 9\,e3 - 9\,e1 + e5, -3\,e2we3we5,$
$3\,Id + 3\,e3 - 6\,e2we4 + 3\,e3we5 - 6\,e3we4 - 6\,e2we5, -9\,e1 - 9\,e2, -9\,Id, 2\,Id + 2\,e4we5,$
$5\,e4 - 9\,e2 - 9\,e1 + 5\,Id, 6\,Id, 8\,Id, -4\,Id - 2\,e4 - 2\,e1 + 2\,e2, -2\,e1, 3\,Id\}$

```
> L:={}: for k from 1 to 10 do L:={op(L),rd_clipolynom(5,3,10,5)} end do:L;
```

$\{7\,Id - e2we5 - e3we4 + 7\,e3 + 7\,e2we3we5 - e4,$
$-Id + 3\,e2we3we5 + 3\,e4 + 3\,e1we4 - e1we2we4 - e5,$
$-6\,e2we4 - 6\,e1we3we5 - 6\,e3we4we5 + 6\,e2 - 6\,e1we3we4,$
$5\,e1we5 - 9\,e1we3we5 + 5\,e1we2we4 - 9\,e2we4 - 9\,e5,$
$3\,Id - 3\,e2 - 3\,e2we4 - 3\,e2we3 - 3\,e1we4we5,$

$$-10\, Id + 4\, e1we5 + 4\, e2we3we4 - 10\, e2we3 - 10\, e1we3we4 + 4\, e1we2we4,$$
$$-5\, Id + 6\, e2we5 + 6\, e5 - 5\, e1we2we4 - 5\, e4we5 + 6\, e3we4,$$
$$-Id - 2\, e3we4we5 - 2\, e2we4 - 2\, e1we4we5 - 2\, e1,$$
$$3\, e3we4 - 4\, e1we2we4 - 4\, e1we2we3 - 4\, e4 - 4\, e1we4,$$
$$Id - 3\, e1we2we3 + 4\, e1we2we5 - 3\, e2we3 + 4\, e2we3we5\}$$

```
>
```

See Also: Clifford:-`type/clipolynom`, Clifford:-`type/climon`, Clifford:-`type/clibasmon`, Clifford:-cmul

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-reorder - reorder basis monomials in a Clifford polynomial

Calling Sequence:

reorder(p);

Parameters:

p - expression of the type 'cliscalar', 'clibasmon', 'climon', 'clipolynom', or 'matrix'

Description:

- Procedure 'reorder' reorders Clifford monomials in the given Clifford polynomial using standard ordering and calculates sign of each permutation.
  See `type/cliscalar`, `type/clibasmon`, `type/climon`, and, `type/clipolynom` for more help on the types.
- If indices in a monomial are both numerals and letters, then letters are placed at the end while numerals are placed at the front of the sorted list.
- If all indices in a monomial are letters, e.g., i, j, k, then reordering is done according to the lexicographic order.
- If all indices in a monomial are numerical strings, then reordering is done according to the '<' order.
- This procedure is linear.
- Procedure 'reorder' may also be applied to matrices with entries in a Clifford algebra in which case it is then applied to each matrix entry.

Examples:

```
> restart:with(Clifford):
> reorder(3*Pi*e2we1we3+4-e2we1);
```

$$-3\,\pi\, e1we2we3 + 4 + e1we2$$

```
> reorder(a*B[i,j]*ejwei+3*ekwei);
```

$$-a\, B_{i,j}\, eiwej - 3\, eiwek$$

```
> reorder(4*e1wej+ekwe3+2*e2we1+3*ejweiwe4we3wekwe1);
```

$$4\, e1wej - e3wek - 2\, e1we2 - 3\, e1we3we4weiwejwek$$

```
> M:=linalg[matrix](2,2,[2*e2we3we1,e9we8,e4we6we3we1,2*e4we3]);
```

$$M := \begin{bmatrix} 2\, e2we3we1 & e9we8 \\ e4we6we3we1 & 2\, e4we3 \end{bmatrix}$$

```
> reorder(M);
```

$$\begin{bmatrix} 2\, e1we2we3 & -e8we9 \\ -e1we3we4we6 & -2\, e3we4 \end{bmatrix}$$

```
>
```

See Also: Clifford:-extract

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-reversion - reversion anti-automorphism in a Clifford algebra Cl(B) or Cl(K)

Calling Sequence:

reversion(p);
reversion(p,K);

Parameters:

p - expression of the type 'cliscalar' or 'clipolynom'
K - (optional) argument of type name, symbol, matrix, or array, or
`&*`(numeric,{name,symbol,array,matrix})

Description:

- Procedure 'reversion' calculates reversion in the Clifford algebra Cl(K). When no optional argument is used, it operates in Cl(B), that is, the default value of K is B. It is linear in its argument and it is always a Clifford algebra anti-automorphism. See `type/cliscalar` and `type/clipolynom` for more help on the types.
- When the antisymmetric part of B is not zero, 'reversion' does not preserve the multilinear structure of the algebra because it mixes grades, i.e., it does not preserve the gradation of the exterior algebra.
- Procedure of 'reversion' is an automorphism of Cl(B) of order 2.
- This procedure may also be applied to matrices with entries in a Clifford algebra in which case it is applied to each matrix entry.

Examples:

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

Example 1: Reversing with respect to the default bilinear form B in Cl(B):

```
> reversion((2+alpha)*Id);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$2\,Id + \alpha\,Id$$

```
> reversion(e1we2); #note the off-diagonal entries of B
```

$$B_{2,1}\,Id - B_{1,2}\,Id - e1we2$$

```
> reversion(e2we1); #note the off-diagonal entries of B
```

$$B_{1,2}\,Id - B_{2,1}\,Id + e1we2$$

```
> reversion(e1we2we3):clicollect(%);
```

$$-(B_{2,3} - B_{3,2})\,e1 + (-B_{3,1} + B_{1,3})\,e2 - e1we2we3 - (-B_{2,1} + B_{1,2})\,e3$$

```
>
```

The same but reversion is now performed with respect to some other form K and -K:

```
> reversion((2+alpha)*Id,K);
  reversion((2+alpha)*Id,-K);
```

$$2\,Id + \alpha\,Id$$

$$2\,Id + \alpha\,Id$$

```
> reversion(e1we2,K); #note the off-diagonal entries of K
  reversion(e1we2,-K); #note the off-diagonal entries of K
```

$$K_{2,1}\,Id - K_{1,2}\,Id - e1we2$$

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

```
> reversion(e2we1,K); #note the off-diagonal entries of K
  reversion(e2we1,-K); #note the off-diagonal entries of K
```

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

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

```
> reversion(e1we2we3,K):clicollect(%);
  reversion(e1we2we3,-K):clicollect(%);
```

$$(-K_{2,3} + K_{3,2})\,e1 - (K_{3,1} - K_{1,3})\,e2 - e1we2we3 - (-K_{2,1} + K_{1,2})\,e3$$

$$-(-K_{2,3} + K_{3,2})\,e1 + (K_{3,1} - K_{1,3})\,e2 - e1we2we3 + (-K_{2,1} + K_{1,2})\,e3$$

```
> u:=4+Pi*e1we4+2*e1we2we3:v:=a+3*e2we4+e3:
```

Notice that when the bilinear form B is not symmetric, 'reversion' does not preserve the gradation in Cl(B):

```
> revv:=clicollect(reversion(v));
```

$$revv := (a + 3\,B_{4,2} - 3\,B_{2,4})\,Id + e3 - 3\,e2we4$$

```
> revu:=clicollect(reversion(u));
```

$$revu := -(-4 - \pi\,B_{4,1} + \pi\,B_{1,4})\,Id - 2\,(B_{2,3} - B_{3,2})\,e1 + 2\,(-B_{3,1} + B_{1,3})\,e2 - 2\,e1we2we3 - 2\,(-B_{2,1} + B_{1,2})\,e3 - \pi\,e1we4$$

However, 'reversion' still is an anti-automorphism in Cl(B):

```
> uv:=cmul(u,v); #compute product of u and v in Cl(B)
```

$$uv := 6\,B_{3,2}\,e1we2we4 + (3\,\pi\,B_{4,2}\,B_{1,4} - 3\,\pi\,B_{4,4}\,B_{1,2} + 4\,a)\,Id - 2\,B_{2,3}\,e1we3 + 2\,B_{1,3}\,e2we3 + (2\,B_{3,3} - 3\,\pi\,B_{4,4})\,e1we2 - (-6\,B_{3,2}\,B_{2,4} + 6\,B_{3,4}\,B_{2,2} - \pi\,B_{4,3})\,e1 + 6\,(-B_{3,2}\,B_{1,4} + B_{3,4}\,B_{1,2})\,e2 - (\pi + 6\,B_{2,2})\,e1we3we4 + 6\,B_{1,2}\,e2we3we4 + 2\,(-3\,B_{2,4} + a)\,e1we2we3 - 2\,(3\,B_{2,4}\,B_{1,2} - 2 - 3\,B_{2,2}\,B_{1,4})\,e3 + \pi\,(3\,B_{4,2} + a)\,e1we4 - 3\,(-4 + \pi\,B_{1,4})\,e2we4 - \pi\,B_{1,3}\,e4$$

```
> evalb(expand(reversion(uv)-cmul(revv,revu))=0);
```

$$true$$

The same computation but in Cl(K):

```
> revv:=clicollect(reversion(v,K));
  revu:=clicollect(reversion(u,K));
  uv:=cmul[K](u,v); #compute product of u and v in Cl(K)
  evalb(expand(reversion(uv,K)-cmul[K](revv,revu))=0);
```

$$revv := (a + 3\,K_{4,2} - 3\,K_{2,4})\,Id + e3 - 3\,e2we4$$

$$revu := (4 + \pi K_{4,1} - \pi K_{1,4})\, Id + 2\,(-K_{2,3} + K_{3,2})\, e1 - 2\,(K_{3,1} - K_{1,3})\, e2 - 2\, e1we2we3 - 2\,(-K_{2,1} + K_{1,2})\, e3 - \pi\, e1we4$$

$$uv := 6\, K_{3,2}\, e1we2we4 + (4\,a + 3\,\pi\, K_{4,2}\, K_{1,4} - 3\,\pi\, K_{4,4}\, K_{1,2})\, Id - 2\, K_{2,3}\, e1we3 + 2\, K_{1,3}\, e2we3 + (2\, K_{3,3} - 3\,\pi\, K_{4,4})\, e1we2 + (\pi\, K_{4,3} + 6\, K_{3,2}\, K_{2,4} - 6\, K_{3,4}\, K_{2,2})\, e1 - 6\,(K_{3,2}\, K_{1,4} - K_{3,4}\, K_{1,2})\, e2 - (\pi + 6\, K_{2,2})\, e1we3we4 + 6\, K_{1,2}\, e2we3we4 + 2\,(a - 3\, K_{2,4})\, e1we2we3 + 2\,(2 + 3\, K_{2,2}\, K_{1,4} - 3\, K_{2,4}\, K_{1,2})\, e3 + \pi\,(3\, K_{4,2} + a)\, e1we4 - 3\,(-4 + \pi\, K_{1,4})\, e2we4 - \pi\, K_{1,3}\, e4$$

$$true$$

Reversion when matrix has been assigned to a name:

```
> B,K:=matrix(2,2,[a,b,c,d]),matrix(2,2,[a,0,0,d]);
```

$$B, K := \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \begin{bmatrix} a & 0 \\ 0 & d \end{bmatrix}$$

```
> u:=4+Pi*e1we2+2*e1+e2we1;
```

$$u := 4 + \pi\, e1we2 + 2\, e1 + e2we1$$

```
> reversion(u,B);
```

$$4\, Id + \pi\, c\, Id - \pi\, b\, Id + b\, Id - c\, Id - \pi\, e1we2 + e1we2 + 2\, e1$$

```
> reversion(u,-B);
```

$$4\, Id - \pi\, c\, Id + \pi\, b\, Id - b\, Id + c\, Id - \pi\, e1we2 + e1we2 + 2\, e1$$

```
> reversion(u,2*B);
```

$$4\, Id + 2\,\pi\, c\, Id - 2\,\pi\, b\, Id + 2\, b\, Id - 2\, c\, Id - \pi\, e1we2 + e1we2 + 2\, e1$$

```
> reversion(u,-3*K);
```

$$4\, Id - \pi\, e1we2 + e1we2 + 2\, e1$$

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

$$4\, Id - \pi\, e1we2 + e1we2 + 2\, e1$$

Example 2: When the bilinear form B is diagonal, the reversion does preserve the gradation in Cl(Q) and it is also an anti-automorphism in Cl(Q):

```
> B:=linalg[diag](1,1,-1):
> u:=4+Pi*e1we3+2*e1we2we3;revu:=reversion(u);
  v:=a*e1+3*e2we3+e3;revv:=reversion(v);
```

$$u := 4 + \pi\, e1we3 + 2\, e1we2we3$$

$$revu := 4\, Id - \pi\, e1we3 - 2\, e1we2we3$$

$$v := a\, e1 + 3\, e2we3 + e3$$

$$revv := -3\, e2we3 + a\, e1 + e3$$

```
> clicollect(reversion(cmulQ(u,v)));
```

$$-2\,(a + 6)\, e2we3 - (3\,\pi - 2)\, e1we2 + (6 - \pi + 4\,a)\, e1 - (\pi\, a - 4)\, e3$$

```
> cmulQ(revv,revu);
```

$$-2\,(a + 6)\, e2we3 - (3\,\pi - 2)\, e1we2 + (6 - \pi + 4\,a)\, e1 - (\pi\, a - 4)\, e3$$

```
> M:=linalg[matrix](2,2,[3+e2we3,2-3*e1we2we3,2*Id,e1we2-e3we2]);
```

$$M := \begin{bmatrix} 3 + e2we3 & 2 - 3\, e1we2we3 \\ 2\, Id & e1we2 - e3we2 \end{bmatrix}$$

```
> reversion(M);
```

$$\begin{bmatrix} 3\, Id - e2we3 & 2\, Id + 3\, e1we2we3 \\ 2\, Id & -e2we3 - e1we2 \end{bmatrix}$$

```
>
```

Example 3: Let's see some examples how we can compute reversion in Cl(B) with a specific matrix assigned.

```
> K:='K':g:='g':B:='B':F:='F':
  reversion(e1we2);
  reversion(e1we2,X);
  reversion(e1we2,-X);
  reversion(e1we2,B);
```

$$B_{2,1}\, Id - B_{1,2}\, Id - e1we2$$

$$X_{2,1}\, Id - X_{1,2}\, Id - e1we2$$

$$-X_{2,1}\, Id + X_{1,2}\, Id - e1we2$$

$$B_{2,1}\, Id - B_{1,2}\, Id - e1we2$$

```
> reversion(2*Pi*e1we2);
  reversion(2*Pi*e1we2,X);
  reversion(2*Pi*e1we2,B);
```

$$2\, \pi\, B_{2,1}\, Id - 2\, \pi\, B_{1,2}\, Id - 2\, \pi\, e1we2$$

$$2\, \pi\, X_{2,1}\, Id - 2\, \pi\, X_{1,2}\, Id - 2\, \pi\, e1we2$$

$$2\, \pi\, B_{2,1}\, Id - 2\, \pi\, B_{1,2}\, Id - 2\, \pi\, e1we2$$

```
> reversion(2*e1we2+3-2*e4we5-eiwej);
  reversion(2*e1we2+3-2*e4we5-eiwej,B);
  reversion(2*e1we2+3-2*e4we5-eiwej,K);
```

$$-B_{j,i}\, Id + 2\, B_{2,1}\, Id - 2\, B_{1,2}\, Id + 3\, Id - 2\, B_{5,4}\, Id + 2\, B_{4,5}\, Id + B_{i,j}\, Id - 2\, e1we2 + 2\, e4we5 + eiwej$$

$$-B_{j,i}\, Id + 2\, B_{2,1}\, Id - 2\, B_{1,2}\, Id + 3\, Id - 2\, B_{5,4}\, Id + 2\, B_{4,5}\, Id + B_{i,j}\, Id - 2\, e1we2 + 2\, e4we5 + eiwej$$

$$-K_{j,i}\, Id + 2\, K_{2,1}\, Id - 2\, K_{1,2}\, Id + 3\, Id - 2\, K_{5,4}\, Id + 2\, K_{4,5}\, Id + K_{i,j}\, Id - 2\, e1we2 + 2\, e4we5 + eiwej$$

Now, let's assign a specific matrix to B:

```
> dim_V:=3:
  matF:=linalg[matrix](dim_V,dim_V,(i,j)->if i=j then RETURN(0)
  elif i<j then RETURN(F[i,j]) else RETURN(-F[j,i]) fi);
  matg:=linalg[matrix](dim_V,dim_V,(i,j)-> if i<j then
  RETURN(g[i,j]) else RETURN(g[j,i]) fi);
```

```
matB:=evalm(matg+matF);
```

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

$$matg := \begin{bmatrix} g_{1,1} & g_{1,2} & g_{1,3} \\ g_{1,2} & g_{2,2} & g_{2,3} \\ g_{1,3} & g_{2,3} & g_{3,3} \end{bmatrix}$$

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

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

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

```
> for x in cbas do
print(`x = `,x);
'reversion(x)'=reversion(x);
'reversion(x,-matB)'=reversion(x,-matB);
'reversion(x,-matF)'=reversion(x,-matF);
'reversion(x,-matg)'=reversion(x,-matg);
print(`***************`);
od;
```

$$x = , Id$$
$$Cliplus{:}\text{-}clirev(x) = Id$$
$$Cliplus{:}\text{-}clirev(x, -matB) = Id$$
$$Cliplus{:}\text{-}clirev(x, -matF) = Id$$
$$Cliplus{:}\text{-}clirev(x, -matg) = Id$$

****

$$x = , e1$$
$$Cliplus{:}\text{-}clirev(x) = e1$$
$$Cliplus{:}\text{-}clirev(x, -matB) = e1$$
$$Cliplus{:}\text{-}clirev(x, -matF) = e1$$
$$Cliplus{:}\text{-}clirev(x, -matg) = e1$$

****

$$x = , e2$$
$$Cliplus{:}\text{-}clirev(x) = e2$$
$$Cliplus{:}\text{-}clirev(x, -matB) = e2$$
$$Cliplus{:}\text{-}clirev(x, -matF) = e2$$
$$Cliplus{:}\text{-}clirev(x, -matg) = e2$$

****

$$x = , e3$$

$$Cliplus{:}\text{-}clirev(x) = e3$$

$$Cliplus{:}\text{-}clirev(x, -matB) = e3$$

$$Cliplus{:}\text{-}clirev(x, -matF) = e3$$

$$Cliplus{:}\text{-}clirev(x, -matg) = e3$$

****

$$x = , e1we2$$

$$Cliplus{:}\text{-}clirev(x) = B_{2,1}\, Id - B_{1,2}\, Id - e1we2$$

$$Cliplus{:}\text{-}clirev(x, -matB) = 2\, F_{1,2}\, Id - e1we2$$

$$Cliplus{:}\text{-}clirev(x, -matF) = 2\, F_{1,2}\, Id - e1we2$$

$$Cliplus{:}\text{-}clirev(x, -matg) = -e1we2$$

****

$$x = , e1we3$$

$$Cliplus{:}\text{-}clirev(x) = B_{3,1}\, Id - B_{1,3}\, Id - e1we3$$

$$Cliplus{:}\text{-}clirev(x, -matB) = 2\, F_{1,3}\, Id - e1we3$$

$$Cliplus{:}\text{-}clirev(x, -matF) = 2\, F_{1,3}\, Id - e1we3$$

$$Cliplus{:}\text{-}clirev(x, -matg) = -e1we3$$

****

$$x = , e2we3$$

$$Cliplus{:}\text{-}clirev(x) = B_{3,2}\, Id - B_{2,3}\, Id - e2we3$$

$$Cliplus{:}\text{-}clirev(x, -matB) = 2\, F_{2,3}\, Id - e2we3$$

$$Cliplus{:}\text{-}clirev(x, -matF) = 2\, F_{2,3}\, Id - e2we3$$

$$Cliplus{:}\text{-}clirev(x, -matg) = -e2we3$$

****

$$x = , e1we2we3$$

$$Cliplus{:}\text{-}clirev(x) = -B_{2,3}\, e1 + B_{3,2}\, e1 - B_{3,1}\, e2 + B_{1,3}\, e2 - e1we2we3 + B_{2,1}\, e3 - B_{1,2}\, e3$$

$$Cliplus{:}\text{-}clirev(x, -matB) = 2\, F_{2,3}\, e1 - 2\, F_{1,3}\, e2 - e1we2we3 + 2\, F_{1,2}\, e3$$

$$Cliplus{:}\text{-}clirev(x, -matF) = 2\, F_{2,3}\, e1 - 2\, F_{1,3}\, e2 - e1we2we3 + 2\, F_{1,2}\, e3$$

$$Cliplus{:}\text{-}clirev(x, -matg) = -e1we2we3$$

****

```
> M:=matrix(2,2,[2*e1we2+2*e4-eiwej+4*Id,e3-4*e5,2-4*e1we2,1+2*Pi
*e3we2]);
```

$$M := \begin{bmatrix} 2\, e1we2 + 2\, e4 - eiwej + 4\, Id & e3 - 4\, e5 \\ 2 - 4\, e1we2 & 1 + 2\, \pi\, e3we2 \end{bmatrix}$$

```
> reversion(M);
```

$$\begin{bmatrix} 2\,B_{2,1}\,Id - 2\,B_{1,2}\,Id + 4\,Id - B_{j,i}\,Id + B_{i,j}\,Id - 2\,e1we2 + eiwej + 2\,e4\,, -4\,e5 + e3 \\ 2\,Id - 4\,B_{2,1}\,Id + 4\,B_{1,2}\,Id + 4\,e1we2\,, \quad Id + 2\,\pi\,B_{2,3}\,Id - 2\,\pi\,B_{3,2}\,Id + 2\,\pi\,e2we3 \end{bmatrix}$$

```
> reversion(M,X);
```

$$\begin{bmatrix} 2\,X_{2,1}\,Id - 2\,X_{1,2}\,Id + 4\,Id - X_{j,i}\,Id + X_{i,j}\,Id - 2\,e1we2 + eiwej + 2\,e4\,, -4\,e5 + e3 \\ 2\,Id - 4\,X_{2,1}\,Id + 4\,X_{1,2}\,Id + 4\,e1we2\,, \quad Id + 2\,\pi\,X_{2,3}\,Id - 2\,\pi\,X_{3,2}\,Id + 2\,\pi\,e2we3 \end{bmatrix}$$

```
> reversion(M,-X);
```

$$\begin{bmatrix} -2\,X_{2,1}\,Id + 2\,X_{1,2}\,Id + 4\,Id + X_{j,i}\,Id - X_{i,j}\,Id - 2\,e1we2 + eiwej + 2\,e4\,, -4\,e5 + e3 \\ 2\,Id + 4\,X_{2,1}\,Id - 4\,X_{1,2}\,Id + 4\,e1we2\,, \quad Id - 2\,\pi\,X_{2,3}\,Id + 2\,\pi\,X_{3,2}\,Id + 2\,\pi\,e2we3 \end{bmatrix}$$

```
> reversion(M,g);
```

$$\begin{bmatrix} 2\,g_{2,1}\,Id - 2\,g_{1,2}\,Id + 4\,Id - g_{j,i}\,Id + g_{i,j}\,Id - 2\,e1we2 + eiwej + 2\,e4\,, -4\,e5 + e3 \\ 2\,Id - 4\,g_{2,1}\,Id + 4\,g_{1,2}\,Id + 4\,e1we2\,, \quad Id + 2\,\pi\,g_{2,3}\,Id - 2\,\pi\,g_{3,2}\,Id + 2\,\pi\,e2we3 \end{bmatrix}$$

Note that since the M[1,1] entry in the matrix M contained a Grassmann basis monomial eiwej with unassigned indices i and j, the result also contains these indices as in matg[i,j].

```
>
```

Example 4: The reversion procedure in Cl(B) can be extended and applied to Clifford polynomials expressed in terms of the Clifford basis monomials, or, expressions of `type/cliprod`. To extend 'reversion' to such terms, user needs to load in package 'Cliplus':

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

$$[LCbig, RCbig, clibasis, clieval, cliexpand, climul, clirev, dottedcbasis, dwedge, makeclialiases]$$

Procedure 'clirev' in that package extends 'reversion'. That is, macro(reversion=clirev) is defined when 'Cliplus' is loaded in. One can use direct procedure 'clirev' or one can continue using the name 'reversion':

```
> reversion(e1 &C e2),clirev(e1 &C e2);
```

$$e2\ \&C\ e1,\ e2\ \&C\ e1$$

```
> reversion(`&C`[B](e1,e2),B),clirev(`&C`[B](e1,e2),B);
```

$$e2\ \&C\ e1,\ e2\ \&C\ e1$$

```
> reversion(`&C`[K](e1,e2),K),clirev(`&C`[K](e1,e2),K);
```

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

An error message will appear if appropriate name is not used in 'reversion' or 'clirev', or when mixed indices are used:

```
> reversion(`&C`[K](e1,e2),B);    ###<<<<--Intended error message
Error, (in Cliplus:-clirev) optional (or default B) parameter in clirev differs
from indices encountered in its cliprod arguments. Found these names as indices
of &C: {B, K}
```

```
> reversion(`&C`[K](e1,e2),K);
```

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

[>

See Also: Clifford:-conjugation, Clifford:-gradeinv, Clifford:-`type/clipolynom`, Clifford:-`type/climon`, Clifford:-`type/clibasmon`, Clifford:-`type/cliscalar`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-RHnumber - Radon-Hurwitz function

Calling Sequence:

RHnumber(p);

Parameters:

p - an integer

Description:

- Procedure 'RHnumber' gives the Radon-Hurwitz number for any integer p. This function is defined via the following table of values:

  RHnumber(0)=0, RHnumber(1)=1, RHnumber(2)=2, RHnumber(3)=2,
  RHnumber(4)=3, RHnumber(5)=3, RHnumber(6)=3, RHnumber(7)=3,

  and a recursive relation RHnumber(k+8) = RHnumber(k) + 4.
- It is known that in a standard Clifford basis of the Clifford algebra Cl(B) there are k different commuting basis monomials with square 1 which generate a group of order 2^k, where k = q - RHnumber(q-p) and (p,q) is the signature of B.
- These commuting basis elements are used to define primitive idempotents which in turn generate minimal ideals in Cl(B). They can be found by the procedure 'commutingelements'. See commutingelements for more information.
- Each such primitive idempotent is a product of k idempotents.
- The signature of B is returned by the procedure 'Bsignature'. See Bsignature for more information.

Examples:

```
> restart:with(Clifford):
> RHnumber(4);
```

$$3$$

```
> RHnumber(12);
```

$$7$$

```
> B:=linalg[diag](1,1,-1):
> Bsignature(B);
```

$$[2, 1]$$

```
> p:=2:q:=1:k:=q-RHnumber(q-p); #there are 2 factors in f for
  (2,1)
```

$$k := 2$$

```
> clidata()[4]; #display f and its factors
```

$$'cmulQ'\left(\frac{Id}{2}+\frac{e1}{2},\frac{Id}{2}+\frac{e2we3}{2}\right)$$

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

$$\frac{Id}{4}+\frac{e1}{4}+\frac{e2we3}{4}+\frac{e1we2we3}{4}$$

```
>
```

See Also: Clifford:-Bsignature, Clifford:-`type/primitiveidemp`, Clifford:-cbasis, Clifford:-clidata, Clifford:-minimalideal

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-rmulm - find a product of two Clifford matrices or polynomials

Calling Sequence:

rmulm(a1,a2,s);
rmulm(a1,a2,s,name);

Parameters:

a1 - expression of type 'cliscalar', 'clibasmon', 'climon', 'clipolynom', 'matrix', 'climatrix', 'dfmatrix', or 'list(matrix)'
a2 - expression of type 'cliscalar', 'clibasmon', 'climon', 'clipolynom', 'matrix', 'climatrix', 'dfmatrix', or 'list(matrix)'
s - a symbol of type 'name', 'symbol', 'function', or 'procedure'
name - (optional) argument of type 'name', 'symbol', 'matrix', 'array', or `&*`(numeric,{name,symbol,matrix,array})

Description:

- Procedure 'rmulm' extends Clifford product, wedge/exterior product, quaternionic product, and some generic, assumed to be non-commutative, multiplication to matrices of the type 'climatrix' (see `type/climatrix` for more help).
- The first two arguments may be of type 'cliscalar', 'clibasmon', 'climon', 'clipolynom', 'matrix', 'climatrix', 'dfmatrix', or 'list(matrix)'.
- The third argument must be one of the following names: 'cmul', 'cmulQ', 'wedge', 'qmul', 'omul', `&r`, or `&*`. Here `&r` denotes some undefined non-commutative product that user may define, while `&*` stands for the standard matrix product used by Maple.
- When the fourth optional argument is not used and s = 'cmul', 'cmulQ', or `&r`, Clifford multiplication in Cl(B) is applied to the matrix entries. When the fourth optional argument of type 'name', 'symbol', 'matrix', 'array', or `&*`(numeric,{name,symbol,matrix,array}) is used, for example, K or -K, and s = cmul, cmulQ, or `&r`, Clifford multiplication in Cl(K) or Cl(-K) is applied to the matrix entries. If s = 'wedge', 'qmul', 'omul', or `&*`, the fourth argument is ignored.
- When the third argument is `&*`, the procedure expects the first two entries to be matrices with numeric or commuting symbolic entries. Thus, 'rmulm' with the option `&*` extends Maple's built-in matrix product `&*`.
- If one of the first two arguments is a Clifford polynomial of type 'clipolynom' while the other argument is of type 'matrix', then the polynomial simply multiplies every entry of the matrix according to the third argument.
- If both arguments a1 and a2 are Clifford polynomials, their product is then found according to the third argument.
- The following are the infix forms of 'rmulm' depending on the third argument:

- when s = 'cmul' then instead of rmulm(a1,a1,s) one may enter a1 &cm a2,

- when s = 'cmulQ' then instead of rmulm(a1,a2,s) one may enter a1 &cQm a2,
- when s = 'qmul' then instead of rmulm(a1,a2,s) one may enter a1 &qm a2,
- when s = 'omul' then instead of rmulm(a1,a2,s) one may enter a1 &om a2,
- when s = `&*` then instead of rmulm(a1,a2,`&*`) one may enter a1 &* a2,
- when s = 'wedge' then instead of rmulm(a1,a2,s) one may enter a1 &wm a2,
- when s = `&r` then instead of rmulm(a1,a2,`&r`) one may enter a1 &rm a2.

- The following are the infix forms of 'rmulm' depending on the third argument and fourth argument:

- when s = 'cmul' and name = 'K' then instead of rmulm(a1,a1,s,name) one may enter &cm[K](a1,a2),
- when s = 'cmulQ' and name = 'K' then instead of rmulm(a1,a1,s,name) one may enter &cQm[K](a1,a2),
- when s = 'qmul', then the fourth argument in rmulm(a1,a1,s,name) is ignored, so enter &qm(a1,a2) or a1 &qm a2,
- when s = `&*`, then the fourth argument in rmulm(a1,a1,s,name) is ignored, so enter &*(a1,a2) or a1 &* a2,
- when s = 'wedge', then the fourth argument in rmulm(a1,a1,s,name) is ignored, so enter &wm(a1,a2) or a1 &wm a2,
- when s = `&r` and name = 'K' then instead of rmulm(a1,a1,s,name) one may enter &rm[K](a1,a2).

- When the first two arguments are lists L1 and L2 of matrices (lists must be of the same length and all matrices must be of the same size), the result is a list whose entries are products rmulm(L1[i],L2[i],s,name),i=1..nops(L1). See example below.
- When the first two arguments are matrices of type 'dfmatrix', they are multiplied according to the third and possibly the fourth arguments. See example below.
- This procedure is used by 'matKrepr' (see matKrepr) to compute matrix representations in minimal ideals (see minimalideal).
- Since this procedure must be able to distinguish between different input types, when _prolevel is set to 'true' by the user, this procedure internally resets _prolevel to false and when it is done computing, it sets _prolevel again to 'true'.

Examples:

```
> restart:bench:=time():with(Clifford):
  _prolevel:=true:eval(makealiases(6,'ordered')):
> M1:=linalg[matrix](2,2,[1,2*e1+4,-e2+e1we2,-e3]);
```

$$M1 := \begin{bmatrix} 1 & 2\,e1 + 4 \\ -e2 + e12 & -e3 \end{bmatrix}$$

```
> M2:=linalg[matrix](2,2,[e1+e2,e4-3,-e2,2*e1]);
```

$$M2 := \begin{bmatrix} e1 + e2 & e4 - 3 \\ -e2 & 2\,e1 \end{bmatrix}$$

```
> p1:=2+e1-2*e3;p2:=e1-2*e3;
```

$$p1 := 2 + e1 - 2\,e3$$

$$p2 := e1 - 2\,e3$$

```
> type(M1,climatrix),type(M2,climatrix);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$true, true$$

```
> type(p1,clipolynom),type(p2,clipolynom);
```

$$true, true$$

Example 1: Let's apply various products to the matrix entries of M1 and M2:

```
> rmulm(M1,M2,cmul); #the Clifford product in Cl(B)
  &cm(M1,M2); #the ampersand form of Clifford product in Cl(B)
```

$$[e1 - 3\,e2 - 2\,B_{1,2} - 2\,e12\,, -3 + e4 + 4\,B_{1,1} + 8\,e1]$$

$$[-B_{2,1} - B_{2,2} + e12 + e1\,B_{2,1} + e1\,B_{2,2} - e2\,B_{1,1} - e2\,B_{1,2} + B_{3,2} - e23\,,$$
$$-B_{2,4} - 3\,e12 - e24 + e124 + B_{2,4}\,e1 + 3\,e2 - e2\,B_{1,4} - 2\,B_{3,1} + 2\,e13]$$

$$[e1 - 3\,e2 - 2\,B_{1,2} - 2\,e12\,, -3 + e4 + 4\,B_{1,1} + 8\,e1]$$

$$[-B_{2,1} - B_{2,2} + e12 + e1\,B_{2,1} + e1\,B_{2,2} - e2\,B_{1,1} - e2\,B_{1,2} + B_{3,2} - e23\,,$$
$$-B_{2,4} - 3\,e12 - e24 + e124 + B_{2,4}\,e1 + 3\,e2 - e2\,B_{1,4} - 2\,B_{3,1} + 2\,e13]$$

```
> rmulm(M1,M2,'cmulQ'); #the Clifford product in Cl(Q)
  &cQm(M1,M2); #the ampersand form of the Clifford product in
  Cl(Q)
```

$$[e1 - 3\,e2 - 2\,B_{1,2} - 2\,e12\,, -3 + e4 + 4\,B_{1,1} + 8\,e1]$$

$$[-B_{2,1} - B_{2,2} + e12 + e1\,B_{2,1} + e1\,B_{2,2} - e2\,B_{1,1} - e2\,B_{1,2} + B_{3,2} - e23\,,$$
$$-B_{2,4} - 3\,e12 - e24 + e124 + B_{2,4}\,e1 + 3\,e2 - e2\,B_{1,4} - 2\,B_{3,1} + 2\,e13]$$

$$\begin{bmatrix} e1 - 3\,e2 - 2\,e12 & -3 + e4 + 4\,B_{1,1} + 8\,e1 \\ e12 - e2\,B_{1,1} - B_{2,2} + e1\,B_{2,2} - e23 & 3\,e2 - 3\,e12 - e24 + e124 + 2\,e13 \end{bmatrix}$$

```
> rmulm(M1,M2,'wedge'); #the wedge/exterior product
  &wm(M1,M2); #the ampersand form of the wedge/exterior product
```

$$\begin{bmatrix} e1 - 3\,e2 - 2\,e12 & -3 + e4 + 8\,e1 \\ e12 - e23 & 3\,e2 - 3\,e12 - e24 + e124 + 2\,e13 \end{bmatrix}$$

$$\begin{bmatrix} e1 - 3\,e2 - 2\,e12 & -3 + e4 + 8\,e1 \\ e12 - e23 & 3\,e2 - 3\,e12 - e24 + e124 + 2\,e13 \end{bmatrix}$$

```
> rmulm(M1,M2,`&r`); #an undefined non-commutative product in
```

```
Cl(B)
&rm(M1,M2); #the ampersand form of an undefined non-commutative
product in Cl(B)
```

$$\begin{bmatrix} \&r_B(1, e1+e2) + \&r_B(2\,e1+4, -e2) & \&r_B(1, e4-3) + \&r_B(2\,e1+4, 2\,e1) \\ \&r_B(-e2+e12, e1+e2) + \&r_B(-e3, -e2) & \&r_B(-e2+e12, e4-3) + \&r_B(-e3, 2\,e1) \end{bmatrix}$$

$$\begin{bmatrix} \&r_B(1, e1+e2) + \&r_B(2\,e1+4, -e2) & \&r_B(1, e4-3) + \&r_B(2\,e1+4, 2\,e1) \\ \&r_B(-e2+e12, e1+e2) + \&r_B(-e3, -e2) & \&r_B(-e2+e12, e4-3) + \&r_B(-e3, 2\,e1) \end{bmatrix}$$

```
> B:=linalg[diag](1$3):
> q1:=2+2*e1we2-3*e2we3;q2:=-3-2*e2we3-3*e1we3;
```

$$q1 := 2 + 2\,e12 - 3\,e23$$

$$q2 := -3 - 2\,e23 - 3\,e13$$

```
> q1:=qdisplay(q1);q2:=qdisplay(q2);
```

$$q1 := 2 - 2\,qk + 3\,qi$$

$$q2 := -3 - 3\,qj + 2\,qi$$

```
> type(q1,quaternion),type(q2,quaternion);
```

$$true, true$$

```
> q3:=q_conjug(q1);q4:=q_conjug(q2);
```

$$q3 := 2 + 2\,qk - 3\,qi$$

$$q4 := -3 + 3\,qj - 2\,qi$$

```
> M:=qdisplay(matrix(2,2,[q1,q2,q3,q4]));#matrix with
quaternionic entries
```

$$M := \begin{bmatrix} 2 - 2\,qk + 3\,qi & -3 - 3\,qj + 2\,qi \\ 2 + 2\,qk - 3\,qi & -3 + 3\,qj - 2\,qi \end{bmatrix}$$

```
> Mc:=map(q_conjug,M);#mapping conjugation to the entries of M
```

$$Mc := \begin{bmatrix} 2 + 2\,qk - 3\,qi & -3 + 3\,qj - 2\,qi \\ 2 - 2\,qk + 3\,qi & -3 - 3\,qj + 2\,qi \end{bmatrix}$$

```
> M &qm Mc;#multiplying out two quaternionic matrices
```

$$\begin{bmatrix} 5 + 15\,qk - 2\,qj + qi & 15\,qk + 28\,qj - 19\,qi - 4 \\ -9 + 5\,qk - 31\,qi + 2\,qj & 10 - 15\,qk + 2\,qj - qi \end{bmatrix}$$

```
> q1 &qm M;#product of a quaternion and a matrix
```

$$\begin{bmatrix} -9 - 8\,qk + 12\,qi & -12 - 3\,qk - 10\,qj - 11\,qi \\ 17 & 15\,qk + 10\,qj - 7\,qi \end{bmatrix}$$

```
> M &qm q1;#product of a matrix and a quaternion
```

$$\begin{bmatrix} -9 - 8\,qk + 12\,qi & -12 + 15\,qk - 2\,qj + qi \\ 17 & -3\,qk + 2\,qj - 19\,qi \end{bmatrix}$$

```
> q1 &qm q2;#product of two quaternions
```

$$-12 + 3\,e12 - 10\,e13 + 11\,e23$$

```
>
```

Example 2: Let's see more of the infix notation and how 'rmulm' works with Clifford

polynomials:

```
> B:=linalg[diag](1,1,-1,-1):
> M2 &cm M2; #Clifford product &c applied to matrix entries
```

$$\begin{bmatrix} 2+e24+3\,e2 & -e14+e24-9\,e1-3\,e2 \\ -e12-1 & -e24+3\,e2+4 \end{bmatrix}$$

```
> rmulm(M2,M2,'cmul',B); #Clifford product cmul applied to matrix entries
```

$$\begin{bmatrix} 2+e24+3\,e2 & -e14+e24-9\,e1-3\,e2 \\ -e12-1 & -e24+3\,e2+4 \end{bmatrix}$$

```
> rmulm(M2,M2,'cmul',-B); #Clifford product cmul applied to matrix entries
```

$$\begin{bmatrix} -2+e24+3\,e2 & -e14+e24-9\,e1-3\,e2 \\ 1-e12 & -e24+3\,e2-4 \end{bmatrix}$$

```
> M2 &wm M2; #wedge product &w applied to matrix entries
Warning, since B has been (re-)assigned, value of dim_V has been reduced by 'wedge' to 4
```

$$\begin{bmatrix} e24+3\,e2 & -e14+e24-9\,e1-3\,e2 \\ -e12 & -e24+3\,e2 \end{bmatrix}$$

```
> evalm(M1);
```

$$\begin{bmatrix} 1 & 2\,e1+4 \\ -e2+e12 & -e3 \end{bmatrix}$$

```
> M1 &cQm p2; #result is a matrix
```

$$\begin{bmatrix} e1-2\,e3 & 2\,Id+4\,e1-4\,e13-8\,e3 \\ -e2+e12+2\,e23-2\,e123 & e13-2\,Id \end{bmatrix}$$

```
> p2 &cQm M1; #result is a matrix
```

$$\begin{bmatrix} e1-2\,e3 & 2\,Id+4\,e13+4\,e1-8\,e3 \\ -e12-2\,e23+e2-2\,e123 & -e13-2\,Id \end{bmatrix}$$

```
> M1 &rm M2; #some generic undefined product applied to matrix entries
```

$$\begin{bmatrix} \&r_B(1, e1+e2)+\&r_B(2\,e1+4, -e2) & \&r_B(1, e4-3)+\&r_B(2\,e1+4, 2\,e1) \\ \&r_B(-e2+e12, e1+e2)+\&r_B(-e3, -e2) & \&r_B(-e2+e12, e4-3)+\&r_B(-e3, 2\,e1) \end{bmatrix}$$

```
>
```

Example 3: Let's apply various products to the matrix entries of M1 and M2 using an optional fourth argument:

```
> rmulm(M1,M2,'cmul',K); #the Clifford product in Cl(K)
  rmulm(M1,M2,'cmul',-K); #the Clifford product in Cl(-K)
```

$$[e1-3\,e2-2\,K_{1,2}-2\,e12\,, -3+e4+4\,K_{1,1}+8\,e1]$$

$$[-K_{2,1}-K_{2,2}+e12+e1\,K_{2,1}+e1\,K_{2,2}-e2\,K_{1,1}-e2\,K_{1,2}+K_{3,2}-e23\,,$$
$$-K_{2,4}-3\,e12-e24+e124+K_{2,4}\,e1+3\,e2-K_{1,4}\,e2-2\,K_{3,1}+2\,e13]$$

$$[e1 - 3\,e2 + 2\,K_{1,2} - 2\,e12\,,\ -3 + e4 - 4\,K_{1,1} + 8\,e1]$$

$$[K_{2,1} + K_{2,2} + e12 - e1\,K_{2,1} - e1\,K_{2,2} + e2\,K_{1,1} + e2\,K_{1,2} - K_{3,2} - e23\,,$$
$$K_{2,4} - 3\,e12 - e24 + e124 - K_{2,4}\,e1 + 3\,e2 + K_{1,4}\,e2 + 2\,K_{3,1} + 2\,e13]$$

```
> rmulm(M1,M2,'cmulQ',K); #the Clifford product in Cl(K)
(diagonal entries in K used only)
rmulm(M1,M2,'cmulQ',-K); #the Clifford product in Cl(K)
(diagonal entries in K used only)
```

$$[e1 - 3\,e2 - 2\,K_{1,2} - 2\,e12\,,\ -3 + e4 + 4\,K_{1,1} + 8\,e1]$$

$$[-K_{2,1} - K_{2,2} + e12 + e1\,K_{2,1} + e1\,K_{2,2} - e2\,K_{1,1} - e2\,K_{1,2} + K_{3,2} - e23\,,$$
$$-K_{2,4} - 3\,e12 - e24 + e124 + K_{2,4}\,e1 + 3\,e2 - K_{1,4}\,e2 - 2\,K_{3,1} + 2\,e13]$$

$$[e1 - 3\,e2 + 2\,K_{1,2} - 2\,e12\,,\ -3 + e4 - 4\,K_{1,1} + 8\,e1]$$

$$[K_{2,1} + K_{2,2} + e12 - e1\,K_{2,1} - e1\,K_{2,2} + e2\,K_{1,1} + e2\,K_{1,2} - K_{3,2} - e23\,,$$
$$K_{2,4} - 3\,e12 - e24 + e124 - K_{2,4}\,e1 + 3\,e2 + K_{1,4}\,e2 + 2\,K_{3,1} + 2\,e13]$$

```
> rmulm(M1,M2,'wedge',K); #the wedge/exterior product, so K is
ignored
rmulm(M1,M2,'wedge',-K); #the wedge/exterior product, so K is
ignored
```

$$\begin{bmatrix} e1 - 3\,e2 - 2\,e12 & -3 + e4 + 8\,e1 \\ e12 - e23 & 3\,e2 - 3\,e12 - e24 + e124 + 2\,e13 \end{bmatrix}$$

$$\begin{bmatrix} e1 - 3\,e2 - 2\,e12 & -3 + e4 + 8\,e1 \\ e12 - e23 & 3\,e2 - 3\,e12 - e24 + e124 + 2\,e13 \end{bmatrix}$$

```
> rmulm(M1,M2,`&r`); #an undefined non-commutative product in
Cl(B)
```

$$\begin{bmatrix} \&r_B(1, e1 + e2) + \&r_B(2\,e1 + 4, -e2) & \&r_B(1, e4 - 3) + \&r_B(2\,e1 + 4, 2\,e1) \\ \&r_B(-e2 + e12, e1 + e2) + \&r_B(-e3, -e2) & \&r_B(-e2 + e12, e4 - 3) + \&r_B(-e3, 2\,e1) \end{bmatrix}$$

```
> rmulm(M1,M2,`&r`,K); #an undefined non-commutative product in
Cl(K)
```

$$\begin{bmatrix} \&r_K(1, e1 + e2) + \&r_K(2\,e1 + 4, -e2) & \&r_K(1, e4 - 3) + \&r_K(2\,e1 + 4, 2\,e1) \\ \&r_K(-e2 + e12, e1 + e2) + \&r_K(-e3, -e2) & \&r_K(-e2 + e12, e4 - 3) + \&r_K(-e3, 2\,e1) \end{bmatrix}$$

```
> rmulm(M1,M2,`&r`,-K); #an undefined non-commutative product in
Cl(-K)
```

$$\begin{bmatrix} \&r_{-K}(1, e1 + e2) + \&r_{-K}(2\,e1 + 4, -e2) & \&r_{-K}(1, e4 - 3) + \&r_{-K}(2\,e1 + 4, 2\,e1) \\ \&r_{-K}(-e2 + e12, e1 + e2) + \&r_{-K}(-e3, -e2) & \&r_{-K}(-e2 + e12, e4 - 3) + \&r_{-K}(-e3, 2\,e1) \end{bmatrix}$$

The same results can be obtained using the ampersand forms:

```
> &cm[K](M1,M2); #the Clifford product in Cl(K)
```

$$[e1 - 3\,e2 - 2\,e12 - 2\,K_{1,2}\,,\ e4 - 3 + 4\,K_{1,1} + 8\,e1]$$

$$[-K_{2,1} - K_{2,2} + e12 + K_{2,1}\,e1 + K_{2,2}\,e1 - K_{1,1}\,e2 - K_{1,2}\,e2 - e23 + K_{3,2}\,,$$

$$-K_{2,4} - 3\,e12 - e24 + e124 + K_{2,4}\,e1 - K_{1,4}\,e2 + 3\,e2 + 2\,e13 - 2\,K_{3,1}]$$

```
> &cQm[K](M1,M2); #the Clifford product in Cl(K) (diagonal entries in K used only)
```

$$\begin{bmatrix} e1 - 3\,e2 - 2\,e12 & e4 - 3 + 4\,K_{1,1} + 8\,e1 \\ e12 - K_{1,1}\,e2 - K_{2,2} + K_{2,2}\,e1 - e23 & -e24 + e124 + 3\,e2 - 3\,e12 + 2\,e13 \end{bmatrix}$$

```
> &wm(M1,M2); #the wedge/exterior product, so K is not used
```

$$\begin{bmatrix} e1 - 3\,e2 - 2\,e12 & e4 - 3 + 8\,e1 \\ e12 - e23 & -e24 + e124 + 3\,e2 - 3\,e12 + 2\,e13 \end{bmatrix}$$

```
> &rm[K](M1,M2); #the ampersand form of an undefined non-commutative product in Cl(B)
```

$$\begin{bmatrix} \&r_K(1, e1 + e2) + \&r_K(2\,e1 + 4, -e2) & \&r_K(1, e4 - 3) + \&r_K(2\,e1 + 4, 2\,e1) \\ \&r_K(-e2 + e12, e1 + e2) + \&r_K(-e3, -e2) & \&r_K(-e2 + e12, e4 - 3) + \&r_K(-e3, 2\,e1) \end{bmatrix}$$

Example 4: Let's see more products when other types of inputs are used.

```
> L:=[evalm(M1),evalm(M2)]; #list of two matrices of type list(matrix)
```

$$L := \left[\begin{bmatrix} 1 & 2\,e1 + 4 \\ -e2 + e12 & -e3 \end{bmatrix}, \begin{bmatrix} e1 + e2 & e4 - 3 \\ -e2 & 2\,e1 \end{bmatrix}\right]$$

When the first two arguments are lists L1 and L2 of matrices (lists must be of the same length and all matrices must be of the same size), the result is a list whose entries are products rmulm(L1[i],L2[i],s,name),i=1..nops(L1). For example,

```
> rmulm(L,L,cmul);
```

$$\left[\begin{bmatrix} 1 + 2\,e12 - 2\,e2 & 4 + 2\,e1 - 2\,e13 - 4\,e3 \\ e12 - e2 - e23 - e123 & 6\,e12 - 6\,e2 - 1 \end{bmatrix}, \begin{bmatrix} 2 + e24 + 3\,e2 & -e14 + e24 - 9\,e1 - 3\,e2 \\ -1 - e12 & -e24 + 3\,e2 + 4 \end{bmatrix}\right]$$

When studying spinor representations of semi-simple Clifford algebras, one has to deal with matrices over double fields. In CLIFFORD, these matrices are of type dfmatrix. To multiply two such matrices, one can use procedure mdfmatrix, however rmulm is more versatile since it allows for selecting which product is to be applied to the entries of these matrices. Furthermore, the fourth argument in rmulm allows for more flexibility. Procedures cdfmatrix and ddfmatrix create and decompose, respectively, such matrices from a list or into a list. For example,

```
> M:=cdfmatrix(M1,M2);
```

$$M := \begin{bmatrix} [1, e1 + e2] & [2\,e1 + 4, e4 - 3] \\ [e12 - e2, -e2] & [-e3, 2\,e1] \end{bmatrix}$$

```
> rmulm(M,M,cmul);
```

$$\begin{bmatrix} [1 + 2\,e12 - 2\,e2, 2 + e24 + 3\,e2] & [4 + 2\,e1 - 2\,e13 - 4\,e3, -e14 + e24 - 9\,e1 - 3\,e2] \\ [e12 - e2 - e23 - e123, -1 - e12] & [6\,e12 - 6\,e2 - 1, -e24 + 3\,e2 + 4] \end{bmatrix}$$

```
> ddfmatrix(%);
```

$$\left[\begin{bmatrix} 1 + 2\,e12 - 2\,e2 & 4 + 2\,e1 - 2\,e13 - 4\,e3 \\ e12 - e2 - e23 - e123 & 6\,e12 - 6\,e2 - 1 \end{bmatrix},\right.$$

$$\begin{bmatrix} 2 + e24 + 3\,e2 & -e14 + e24 - 9\,e1 - 3\,e2 \\ -1 - e12 & -e24 + 3\,e2 + 4 \end{bmatrix}]$$

which is the same result as above.

```
> rmulm(M,M,wedge);
```

$$\begin{bmatrix} [1 + 2\,e12 - 4\,e2, e24 + 3\,e2] & [4 + 2\,e1 - 2\,e13 - 4\,e3, -e14 + e24 - 9\,e1 - 3\,e2] \\ [e12 - e2 - e23 - e123, -e12] & [6\,e12 - 4\,e2, -e24 + 3\,e2] \end{bmatrix}$$

```
>
```

Example 5: Numeric and symbolic matrices with entries in a Clifford algebra:

```
> B:='B': #the bilinear form B is now undefined
> M1 &cm M1; #Clifford product &c applied to matrix entries
```

$$[1 - 2\,B_{1,2} + 2\,e12 - 2\,B_{1,2}\,e1 + 2\,B_{1,1}\,e2 - 4\,e2\,, 4 + 2\,e1 - 4\,e3 - 2\,e13 - 2\,B_{1,3}]$$

$$[e12 - e2 - e123 - B_{3,1}\,e2 + B_{3,2}\,e1 - e23 + B_{3,2}\,,$$

$$-2\,B_{2,1} + 6\,e12 + 2\,B_{2,1}\,e1 - 4\,e2 - 2\,B_{1,1}\,e2 + B_{3,3}]$$

```
> M1 &cQm M1; #Clifford product &cQ applied to matrix entries
```

$$\begin{bmatrix} 1 + 2\,e12 + 2\,B_{1,1}\,e2 - 4\,e2 & 4 + 2\,e1 - 2\,e13 - 4\,e3 \\ e12 - e2 - e23 - e123 & 6\,e12 - 4\,e2 - 2\,B_{1,1}\,e2 + B_{3,3} \end{bmatrix}$$

```
> M1 &wm M1; #wedge product &w applied to matrix entries
```

$$\begin{bmatrix} 1 + 2\,e12 - 4\,e2 & 4 + 2\,e1 - 2\,e13 - 4\,e3 \\ e12 - e2 - e23 - e123 & 6\,e12 - 4\,e2 \end{bmatrix}$$

```
> m1:=linalg[matrix](2,2,[1,2,3,4]); #just a numeric matrix
```

$$m1 := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
> m1 &cm m1;map(scalarpart,%);
```

$$\begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$$

$$\begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$$

```
> m1 &cQm m1;map(scalarpart,%);
```

$$\begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$$

$$\begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$$

```
> m1 &wm m1;map(scalarpart,%);
```

$$\begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$$

$$\begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$$

```
> m1 &* m1;
```

$$m1\ \&*\ m1$$

```
> evalm(%);
```

$$\begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$$

Example 6: Octonionic matrices can be multiplied as follows:

```
> B:='B':with(Octonion):
> o1:=2-e3+e7;o2:=e4+e5-3*e7;o3:=1-2*e7;o4:=e3-e4+e7;
```

$$o1 := 2 - e3 + e7$$

$$o2 := e4 + e5 - 3\,e7$$

$$o3 := 1 - 2\,e7$$

$$o4 := e3 - e4 + e7$$

```
> M:=matrix(2,2,[o1,o2,o3,o4]);
```

$$M := \begin{bmatrix} 2 - e3 + e7 & e4 + e5 - 3\,e7 \\ 1 - 2\,e7 & e3 - e4 + e7 \end{bmatrix}$$

```
> rmulm(M,M,'omul');
```

$$\begin{bmatrix} -4 - 4\,e3 + e7 - e4 + 3\,e5 & 2\,e4 - 2\,e6 + 5\,e5 - 2\,e2 - 5\,e7 + 6\,e1 + 7 \\ 6 - 2\,e7 - 4\,e1 - e4 - 2\,e5 & 3\,e4 - e5 - 3\,e7 - 9 \end{bmatrix}$$

```
> M &om M;
```

$$\begin{bmatrix} -4 - 4\,e3 + e7 - e4 + 3\,e5 & 2\,e4 - 2\,e6 + 5\,e5 - 2\,e2 - 5\,e7 + 6\,e1 + 7 \\ 6 - 2\,e7 - 4\,e1 - e4 - 2\,e5 & 3\,e4 - e5 - 3\,e7 - 9 \end{bmatrix}$$

```
> M1:=map(o_conjug,M);
```

$$M1 := \begin{bmatrix} 2 + e3 - e7 & -e4 - e5 + 3\,e7 \\ 1 + 2\,e7 & -e3 + e4 - e7 \end{bmatrix}$$

```
> rmulm(M,M1,'omul');
```

$$\begin{bmatrix} 12 + 3\,e4 - e5 - 3\,e7 & -2\,e4 + 2\,e6 - 5\,e5 + 2\,e2 + 5\,e7 - 6\,e1 - 7 \\ -4\,e7 + 2\,e3 + 4\,e1 - e4 + 2\,e5 - 2 & -3\,e4 + e5 + 3\,e7 + 9 \end{bmatrix}$$

```
> M &om M1;
```

$$\begin{bmatrix} 12 + 3\,e4 - e5 - 3\,e7 & -2\,e4 + 2\,e6 - 5\,e5 + 2\,e2 + 5\,e7 - 6\,e1 - 7 \\ -4\,e7 + 2\,e3 + 4\,e1 - e4 + 2\,e5 - 2 & -3\,e4 + e5 + 3\,e7 + 9 \end{bmatrix}$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 2.796000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

See Also: Clifford:-minimalideal, Clifford:-qmul, Clifford:-wedge, Clifford:-cmulQ, Clifford:-cmul, Clifford:-`type/quaternion`, Clifford:-matKrepr, Clifford:-spinorKrepr, Clifford:-`type/clipolynom`, Clifford:-`type/climatrix`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-rot3d - rotation in R^3 done with quaternions

Calling Sequence:

rot3(p, q);

Parameters:

p - an algebraic expression of type 'clipolynom'
q - an algebaric expression of type 'quaternion'

Description:

- Procedure 'rot3d' rotates a vector in a 3-dimensional Euclidean space V using the quaternion multiplication. Any vector v is transformed according to the following law:

  v -> q &c v &c qinv(q)

  where q is a quaternion given in the basis {1, qi, qj, qk}.

- The first entry should be a vector (or any element of the Clifford algebra Cl(3) while the second element is a quaternion (see `type/quaternion` for more help). Elements qi, qj, and qk are defined at the time of initialization and denote the pure-quaternion basis elements. Type >_quatbasis at the Maple prompt to see how these elements are defined in terms of the even basis elements in Cl(3).

- It is assumed that the user has defined a bilinear form B as the 3 x 3 identify matrix.

Examples:

```
> restart:bench:=time():with(Clifford):_prolevel:=true:eval(makea
liases(3));
B:=linalg[diag](1,1,1):_quatbasis;
```

$$e12, e21, e13, e31, e23, e32, e123, e132, e213, e231, e312, e321$$

$$[[Id, e32, e13, e21], \{Maple\ has\ assigned\ qi:=-e2we3,\ qj:=e1we3,\ qk:=-e1we2\}]$$

Example 1: Rotations in coordinate planes.
Let's define first unit quaternions reponsible for the rotations in the coordinate planes. These are counter-clockwise rotations when looking down the rotation axis.

```
> q12:=cos(alpha/2)+sin(alpha/2)*'qk';#rotation in the xy-plane
```

$$q12 := \cos\left(\frac{\alpha}{2}\right) + \sin\left(\frac{\alpha}{2}\right) qk$$

```
> q12i:=qinv(q12); #inverse of q12
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$q12i := \cos\left(\frac{\alpha}{2}\right) - \sin\left(\frac{\alpha}{2}\right) qk$$

```
> q13:=cos(beta/2)+sin(beta/2)*'qj';#rotation in the xz-plane
```

$$q13 := \cos\left(\frac{\beta}{2}\right) + \sin\left(\frac{\beta}{2}\right) qj$$

```
> q13i:=qinv(q13); #inverse of q13
```

$$q13i := \cos\left(\frac{\beta}{2}\right) - \sin\left(\frac{\beta}{2}\right) qj$$

```
> q23:=cos(gamma/2)+sin(gamma/2)*'qi';#rotation in the yz-plane
```

$$q23 := \cos\left(\frac{\gamma}{2}\right) + \sin\left(\frac{\gamma}{2}\right) qi$$

```
> q23i:=qinv(q23); #inverse of q23
```

$$q23i := \cos\left(\frac{\gamma}{2}\right) - \sin\left(\frac{\gamma}{2}\right) qi$$

```
> qnorm(q12),qnorm(q13),qnorm(q23); # these are unit quaternions
```

$$1, 1, 1$$

```
> type(q12,quaternion),type(q13,quaternion),type(q23,quaternion);
```

$$true, true, true$$

```
> q12 &q q12i,q13 &q q13i,q23 &q q23i;
```

$$1, 1, 1$$

Notice that to rotate by an angle n*alpha it is enough to find the n-th Clifford power of the appropriate quaternion:

```
> q12 &q q12; #rotation by the angle 2*alpha
```

$$\cos(\alpha) + \sin(\alpha)\, qk$$

```
> q12 &q q12 &q q12; #rotation by the angle 3*alpha
```

$$\cos\left(\frac{3\,\alpha}{2}\right) + \sin\left(\frac{3\,\alpha}{2}\right) qk$$

```
> q12 &q q12 &q q12 &q q12; #rotation by the angle 4*alpha
```

$$\cos(2\,\alpha) + \sin(2\,\alpha)\, qk$$

```
> q12 &q q12 &q q12 &q q12 &q q12; #rotation by the angle 5*alpha
```

$$\cos\left(\frac{5\,\alpha}{2}\right) + \sin\left(\frac{5\,\alpha}{2}\right) qk$$

Example 2: Let's see how basis vectors {e1,e2,e3} are rotated by the quaternions q12, q13, and q23 in the coordinate planes xy- , xz-, and yz- respectively.

```
> e11:=rot3d(e1,q12); #rotation of e1 by the quaternion q12
```

$$e11 := \cos(\alpha)\, e1 + \sin(\alpha)\, e2$$

```
> e22:=rot3d(e2,q12); #rotation of e2 by the quaternion q12
```

$$e22 := -\sin(\alpha)\, e1 + \cos(\alpha)\, e2$$

```
> e33:=rot3d(e3,q12); #rotation of e3 by the quaternion q12
```

$$e33 := e3$$

We can also compose rotations. For example, let's follow rotation q12 with the rotation q13:

```
> e111:=rot3d(e1,q13 &q q12); #rotation of e1 by the quaternion
```

```
q13 &q q12
```

$$e111 := e1\cos(\beta)\cos(\alpha) + \sin(\alpha)\,e2 - e3\sin(\beta)\cos(\alpha)$$

```
> e222:=rot3d(e2,q13 &q q12); #rotation of e2 by the quaternion
q13 &q q12
```

$$e222 := -e1\sin(\alpha)\cos(\beta) + \cos(\alpha)\,e2 + e3\sin(\beta)\sin(\alpha)$$

```
> e333:=rot3d(e3,q13 &q q12); #rotation of e3 by the quaternion
q13 &q q12
```

$$e333 := \sin(\beta)\,e1 + \cos(\beta)\,e3$$

It is not difficult to see that the sets {e11,e22,e33} and {e111,e222,e333} are orthonormal. Let's verify that assertion for the latter set:

```
> A:=array(1..3,1..3,[]):
for i from 1 to 3 do for j from 1 to 3 do
A[i,j]:=scalarpart((e||i||i||i &c e||j||j||j + e||j||j||j &c
e||i||i||i)/2) end do end do;
> print(A);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

We can check orientation of the set {e111,e222,e333} by checking that the wedge products e111 &w e222, e222 &w e333, and e333 &w e111 are dual to the vectors e333, e111, and e222 respectively. Remembering that e1we2 is dual to e3, e2we3 is dual to e1, and e3we1 is dual to e2, we have the following dual pars:

```
> e111 &w e222, e333;
Warning, since B has been (re-)assigned, value of dim_V has been reduced by 'we
dge' to 3
```

$$\cos(\beta)\,e12 + \sin(\beta)\,e23,\ \sin(\beta)\,e1 + \cos(\beta)\,e3$$

```
> e222 &w e333, e111;
```

$$-\cos(\alpha)\sin(\beta)\,e12 - \sin(\alpha)\,e13 + \cos(\alpha)\cos(\beta)\,e23,$$
$$e1\cos(\beta)\cos(\alpha) + \sin(\alpha)\,e2 - e3\sin(\beta)\cos(\alpha)$$

```
> e333 &w e111, e222;
```

$$\sin(\beta)\sin(\alpha)\,e12 - \cos(\beta)\sin(\alpha)\,e23 - \cos(\alpha)\,e13,$$
$$-e1\sin(\alpha)\cos(\beta) + \cos(\alpha)\,e2 + e3\sin(\beta)\sin(\alpha)$$

Another way to check that the sets {e1,e2,e3}, {e11,e22,e33}, and {e111,e222,e333} have the same orientation is to verify that their wedge poducts are all equal:

```
> wedge(e1,e2,e3),wedge(e11,e22,e33),wedge(e111,e222,e333);
```

$$e123,\ e123,\ e123$$

Example 3: Let's see now how these basis rotations in the coordinate planes act on an arbitrary vector.

```
> v:=a*e1+b*e2+c*e3;
```

$$v := a\,e1 + b\,e2 + c\,e3$$

The length of v can be found as follows:

```
> vlength:=sqrt(scalarpart(v &cQ v));
```

$$vlength := \sqrt{a^2 + b^2 + c^2}$$

Certainly, rotations do not change length. For example, let's rotate v by q13 &q q12:

```
> v123:=rot3d(v,q13 &q q12); #rotation q12 followed by q13
```

$$v123 := (-b \sin(\alpha) \cos(\beta) + c \sin(\beta) + a \cos(\beta) \cos(\alpha))\, e1 + e2\, (b \cos(\alpha) + a \sin(\alpha)) - (-b \sin(\beta) \sin(\alpha) + a \sin(\beta) \cos(\alpha) - c \cos(\beta))\, e3$$

```
> vlength2:=sqrt(scalarpart(v123 &cQ v123));
```

$$vlength2 := \sqrt{a^2 + b^2 + c^2}$$

Thus, the length of v123 is the same as the length of v. However, rotations do not commute. We will show that by applying quaternion q12 &q q13 to v and comparing with q123:

```
> v132:=rot3d(v,q12 &q q13); #rotation q13 followed by q12
```

$$v132 := (a \cos(\beta) \cos(\alpha) + c \sin(\beta) \cos(\alpha) - b \sin(\alpha))\, e1 + (b \cos(\alpha) + c \sin(\beta) \sin(\alpha) + a \cos(\beta) \sin(\alpha))\, e2 - e3\, (a \sin(\beta) - c \cos(\beta))$$

```
> clicollect(v123-v132);
```

$$-(b \sin(\alpha) \cos(\beta) - c \sin(\beta) - b \sin(\alpha) + c \sin(\beta) \cos(\alpha))\, e1 - e2 \sin(\alpha)\, (c \sin(\beta) + \cos(\beta)\, a - a) - e3 \sin(\beta)\, (-b \sin(\alpha) + a \cos(\alpha) - a)$$

As it can be seen, v123 <> v132.

Example 4: Counter-clockwise rotation by an angle 'alpha' around the given vector 'axis'. Suppose we want to rotate vector 'v' around some given unit vector 'axis' by the angle 'alpha'. This rotation will be counter-clockwise when looking down the axis towards to origin (0,0,0) of the coordinate system.

```
> _quatbasis;
```

$$[[Id, e32, e13, e21], \{Maple\ has\ assigned\ qi:=-e2we3,\ qj:=e1we3,\ qk:=-e1we2\}]$$

```
> axis:=(a1*e1+a2*e2+a3*e3);alias(N=sqrt(scalarpart(axis &c
  axis)));
```

$$axis := a1\ e1 + a2\ e2 + a3\ e3$$

$$e12, e21, e13, e31, e23, e32, e123, e132, e213, e231, e312, e321, N$$

```
> axis:=axis/N;#we normalize axis
```

$$axis := \frac{a1\ e1 + a2\ e2 + a3\ e3}{\sqrt{a1^2 + a2^2 + a3^2}}$$

```
> qaxis:=subs({e3=-e1we2,e1=-e2we3,e2=e1we3},axis);#define dual
  quaternion
```

$$qaxis := \frac{-a1\ e23 + a2\ e13 - a3\ e12}{\sqrt{a1^2 + a2^2 + a3^2}}$$

Notice that to find the dual quaternion 'qaxis' we could have also multiplied the axis vector 'axis' by the unit pseudoscalar e1we2we3 (unit volume element) on the right as follows:

```
> qaxis:= axis &c (-e1we2we3);
```

$$qaxis := \frac{a2\ e13}{\sqrt{a1^2+a2^2+a3^2}} - \frac{a1\ e23}{\sqrt{a1^2+a2^2+a3^2}} - \frac{a3\ e12}{\sqrt{a1^2+a2^2+a3^2}}$$

```
> qaxis:=qdisplay(qaxis);
```

$$qaxis := \frac{a3\ qk}{\sqrt{a1^2+a2^2+a3^2}} + \frac{a2\ qj}{\sqrt{a1^2+a2^2+a3^2}} + \frac{a1\ qi}{\sqrt{a1^2+a2^2+a3^2}}$$

```
> qrot:=cos(alpha/2)+sin(alpha/2)*qaxis;
```

$$qrot := \cos\left(\frac{\alpha}{2}\right) + \sin\left(\frac{\alpha}{2}\right)\left(\frac{a2\ e13}{\sqrt{a1^2+a2^2+a3^2}} - \frac{a1\ e23}{\sqrt{a1^2+a2^2+a3^2}} - \frac{a3\ e12}{\sqrt{a1^2+a2^2+a3^2}}\right)$$

Let's define various rotation quaternions:

```
> q100:=subs({a1=1,a2=0,a3=0},qrot);#rotation about the axis
  (1,0,0)
```

$$q100 := \cos\left(\frac{\alpha}{2}\right) - \sin\left(\frac{\alpha}{2}\right) e23$$

```
> q010:=subs({a1=0,a2=1,a3=0},qrot);#rotation about the axis
  (0,1,0)
```

$$q010 := \cos\left(\frac{\alpha}{2}\right) + \sin\left(\frac{\alpha}{2}\right) e13$$

```
> q001:=subs({a1=0,a2=0,a3=1},qrot);#rotation about the axis
  (0,0,1)
```

$$q001 := \cos\left(\frac{\alpha}{2}\right) - \sin\left(\frac{\alpha}{2}\right) e12$$

```
> q101:=subs({a1=1,a2=0,a3=1},qrot);#rotation about the axis
  (1,0,1)
```

$$q101 := \cos\left(\frac{\alpha}{2}\right) + \sin\left(\frac{\alpha}{2}\right)\left(-\frac{\sqrt{2}\ e23}{2} - \frac{\sqrt{2}\ e12}{2}\right)$$

```
> q011:=subs({a1=0,a2=1,a3=1},qrot);#rotation about the axis
  (0,1,1)
```

$$q011 := \cos\left(\frac{\alpha}{2}\right) + \sin\left(\frac{\alpha}{2}\right)\left(\frac{\sqrt{2}\ e13}{2} - \frac{\sqrt{2}\ e12}{2}\right)$$

```
> q110:=subs({a1=1,a2=1,a3=0},qrot);#rotation about the axis
  (1,1,0)
```

$$q110 := \cos\left(\frac{\alpha}{2}\right) + \sin\left(\frac{\alpha}{2}\right)\left(\frac{\sqrt{2}\ e13}{2} - \frac{\sqrt{2}\ e23}{2}\right)$$

```
> q111:=subs({a1=1,a2=1,a3=1},qrot);#rotation about the axis
  (1,1,1)
```

$$q111 := \cos\left(\frac{\alpha}{2}\right) + \sin\left(\frac{\alpha}{2}\right)\left(\frac{\sqrt{3}\ e13}{3} - \frac{\sqrt{3}\ e23}{3} - \frac{\sqrt{3}\ e12}{3}\right)$$

Let's try to rotate first a basis vector e1 around various axes listed above by some angle alpha.

After we find a general formula for the components of the rotated e1, we will verify it for various angles.

```
> v1:='v1':v2:='v2':v3:='v3':v:=e1;
```

$$v := e1$$

```
> vnew:=rot3d(v,q100); #the result of the rotation around the
  (100) axis
```

$$vnew := e1$$

```
> eval(subs(alpha=Pi/2,vnew));
```

$$e1$$

```
> vnew:=rot3d(v,q010); #the result of the rotation around the
  (010) axis
```

$$vnew := \cos(\alpha)\, e1 - \sin(\alpha)\, e3$$

```
> eval(subs(alpha=Pi/2,vnew));
```

$$-e3$$

```
> vnew:=rot3d(v,q001); #the result of the rotation around the
  (001) axis
```

$$vnew := \cos(\alpha)\, e1 + \sin(\alpha)\, e2$$

```
> eval(subs(alpha=Pi/2,vnew));
```

$$e2$$

```
> vnew:=rot3d(v,q101); #the result of the rotation around the
  (101) axis
```

$$vnew := \frac{1}{2}(\cos(\alpha)+1)\, e1 + \frac{1}{2}\sqrt{2}\,\sin(\alpha)\, e2 - \frac{1}{2}(-1+\cos(\alpha))\, e3$$

```
> eval(subs(alpha=Pi/2,vnew));
```

$$\frac{e1}{2} + \frac{\sqrt{2}\, e2}{2} + \frac{e3}{2}$$

```
> vnew:=rot3d(v,q011); #the result of the rotation around the
  (011) axis
```

$$vnew := \cos(\alpha)\, e1 + \frac{1}{2}\sqrt{2}\,\sin(\alpha)\, e2 - \frac{1}{2}\sin(\alpha)\sqrt{2}\, e3$$

```
> eval(subs(alpha=Pi/2,vnew));
```

$$\frac{\sqrt{2}\, e2}{2} - \frac{\sqrt{2}\, e3}{2}$$

```
> vnew:=rot3d(v,q110); #the result of the rotation around the
  (110) axis
```

$$vnew := \frac{1}{2}(\cos(\alpha)+1)\, e1 - \frac{1}{2}(-1+\cos(\alpha))\, e2 - \frac{1}{2}\sin(\alpha)\sqrt{2}\, e3$$

```
> eval(subs(alpha=Pi/2,vnew));
```

$$\frac{e1}{2}+\frac{e2}{2}-\frac{\sqrt{2}\,e3}{2}$$

```
> vnew:=rot3d(v,q111); #the result of the rotation around the
  (111) axis
```

$$vnew := \frac{1}{3}\,e1\,(2\cos(\alpha)+1)+\frac{1}{3}\,(\sqrt{3}\,\sin(\alpha)+1-\cos(\alpha))\,e2-\frac{1}{3}\,(-1+\cos(\alpha)+\sqrt{3}\,\sin(\alpha))\,e3$$

```
> eval(subs(alpha=Pi/2,vnew));
```

$$\frac{e1}{3}+\frac{(\sqrt{3}+1)\,e2}{3}-\frac{(-1+\sqrt{3})\,e3}{3}$$

```
> eval(subs(alpha=Pi,vnew));
```

$$\frac{2\,e2}{3}+\frac{2\,e3}{3}-\frac{e1}{3}$$

Example 5: Finally, we derive a general formula for a rotation of an arbitrary vector 'v' around an arbitrary axis called 'axis' and by an arbitrary angle 'alpha'. Recall that the general quaternion defined in terms of the 'axis' components was defined in Example 3 above as 'qrot'.

```
> alias(N=N):
  v1:='v1':v2:='v2':v3:='v3':v:=v1*e1+v2*e2+v3*e3; #an arbitrary
  vector
  'qrot'=qrot;
```

$$v := v1\,e1+v2\,e2+v3\,e3$$

$$qrot=\cos\left(\frac{\alpha}{2}\right)+\sin\left(\frac{\alpha}{2}\right)\left(\frac{a2\,e13}{\sqrt{a1^2+a2^2+a3^2}}-\frac{a1\,e23}{\sqrt{a1^2+a2^2+a3^2}}-\frac{a3\,e12}{\sqrt{a1^2+a2^2+a3^2}}\right)$$

```
> qnew:=rot3d(v,qrot);
```

$$\begin{aligned}
qnew := {} & e1\,(-a3^3\,v2\,\sin(\alpha)+a2\,v3\,a3^2\,\sin(\alpha)+a2^3\,v3\,\sin(\alpha)+a2\,v3\,a1^2\,\sin(\alpha)\\
& -a3\,v2\,a1^2\,\sin(\alpha)-a3\,v2\,a2^2\,\sin(\alpha)+a2^2\,v1\,\sqrt{a1^2+a2^2+a3^2}\,\cos(\alpha)\\
& +a3^2\,v1\,\sqrt{a1^2+a2^2+a3^2}\,\cos(\alpha)+a1^2\,v1\,\sqrt{a1^2+a2^2+a3^2}\\
& -a2\,a1\,v2\,\sqrt{a1^2+a2^2+a3^2}\,\cos(\alpha)+a2\,a1\,v2\,\sqrt{a1^2+a2^2+a3^2}\\
& -a1\,a3\,v3\,\sqrt{a1^2+a2^2+a3^2}\,\cos(\alpha)+a1\,a3\,v3\,\sqrt{a1^2+a2^2+a3^2})\Big/(a1^2+a2^2+a3^2)^{(3/2)}\\
& -e2\,(a1^3\,v3\,\sin(\alpha)+a1\,v3\,a2^2\,\sin(\alpha)-a3\,v1\,a2^2\,\sin(\alpha)-a3^3\,v1\,\sin(\alpha)\\
& +a1\,v3\,a3^2\,\sin(\alpha)-a3\,v1\,a1^2\,\sin(\alpha)-a1^2\,v2\,\sqrt{a1^2+a2^2+a3^2}\,\cos(\alpha)\\
& -a3^2\,v2\,\sqrt{a1^2+a2^2+a3^2}\,\cos(\alpha)+a2\,a3\,v3\,\sqrt{a1^2+a2^2+a3^2}\,\cos(\alpha)\\
& -a2\,a3\,v3\,\sqrt{a1^2+a2^2+a3^2}+a2\,a1\,v1\,\sqrt{a1^2+a2^2+a3^2}\,\cos(\alpha)\\
& -a2\,a1\,v1\,\sqrt{a1^2+a2^2+a3^2}-a2^2\,v2\,\sqrt{a1^2+a2^2+a3^2})\Big/(a1^2+a2^2+a3^2)^{(3/2)}+e3\,(\\
& a1^3\,v2\,\sin(\alpha)+a1\,v2\,a2^2\,\sin(\alpha)-a2^3\,v1\,\sin(\alpha)-a2\,v1\,a3^2\,\sin(\alpha)+a1\,v2\,a3^2\,\sin(\alpha)
\end{aligned}$$

$$- a2\, v1\, a1^2 \sin(\alpha) + a1^2\, v3 \sqrt{a1^2 + a2^2 + a3^2} \cos(\alpha) + a2^2\, v3 \sqrt{a1^2 + a2^2 + a3^2} \cos(\alpha) + a3^2\, v3 \sqrt{a1^2 + a2^2 + a3^2} - a2\, a3\, v2 \sqrt{a1^2 + a2^2 + a3^2} \cos(\alpha) + a2\, a3\, v2 \sqrt{a1^2 + a2^2 + a3^2} - a1\, a3\, v1 \sqrt{a1^2 + a2^2 + a3^2} \cos(\alpha) + a1\, a3\, v1 \sqrt{a1^2 + a2^2 + a3^2}\,) \Big/ (a1^2 + a2^2 + a3^2)^{(3/2)}$$

For example, let's rotate v=e1+2*e2+3*e3 around the axis (1,2,3) by any angle alpha. Certainly, since the vector v is on the axis of rotation, it should not change:

```
> subs({v1=1,v2=2,v3=3,a1=1,a2=2,a3=3},qnew);
```

$$e1 + 2\, e2 + 3\, e3$$

Finally, let's rotate v=e1-2*e2+4*e3 around the axis (2,-3,4) by an angle alpha=Pi/4.

```
> eval(subs({v1=1,v2=-2,v3=4,a1=2,a2=-3,a3=4,alpha=Pi/4},qnew));
```

$$\frac{e1 \sqrt{29} \left(-58 \sqrt{2} - \frac{19 \sqrt{29} \sqrt{2}}{2} + 48 \sqrt{29}\right)}{841} - \frac{e2\, (58 \sqrt{2} - 7 \sqrt{29} \sqrt{2} + 72 \sqrt{29})\, \sqrt{29}}{841} + \frac{e3 \sqrt{29} \left(-\frac{29 \sqrt{2}}{2} + 10 \sqrt{29} \sqrt{2} + 96 \sqrt{29}\right)}{841}$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 7.267000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

See Also: Clifford:-wedge, Clifford:-`type/quaternion`, Clifford:-qdisplay, Clifford:-qmul, Clifford:-q_conjug, Clifford:-qnorm, Clifford:-qinv

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-scalarpart - find the scalar or the 0-vector part in a Clifford polynomial

Calling Sequence:

scalarpart(p);

Parameters:

p - expression of the type 'cliscalar' or 'clipolynom'

Description:

- Procedure 'scalarpart' finds and displays the scalar or the 0-vector part of the given Clifford polynomial p. See `type/cliscalar` and `type/clipolynom` for more help.
- When p does not contains the scalar part (or term of grade 0), the procedure returns 0. See gradeinv for more help on grades.
- This procedure is linear.

Examples:

```
> restart:with(Clifford):_prolevel:=true:
> scalarpart(3*Pi+e2we1we3+4-e2we1);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$3\,\pi + 4$$

```
> scalarpart(a*B[i,j]+3*ekwei);
```

$$a\,B_{i,j}$$

```
> scalarpart(4*e1wej+ekwe3+2*e2we1+e1we2we3we4);
```

$$0$$

```
> scalarpart(cmul(ei,ej)+cmul(ej,ei));
```

$$B_{i,j} + B_{j,i}$$

```
> scalarpart(cmul(e1,e2)+cmul(e2,e1));
```

$$B_{1,2} + B_{2,1}$$

```
>
```

See Also: Clifford:-vectorpart

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-setup - the initialization procedure for the package 'CLIFFORD'

Calling Sequence:

none

Parameters:

none

Description:

- Procedure 'setup' is the initialization procedure for the 'CLIFFORD' package. It is executed automatically when the package is loaded with the command >with(Clifford);.
- The most important global variable defined at the initialization time is _default_Clifford_product which takes on the default value of 'cmulRS'. See cmul, cmulRS, cmulNUM and CLIFFORD_ENV for more help.
- At the time of loading, the following are defined:

- `&c` or 'cmul' for Clifford multiplication in Cl(B) (see `&c` for more help),
- `&cQ` or 'cmulQ' for Clifford multiplication in Cl(Q) (see `&cQ` for more help),
- `&w` or 'wedge' for wedge/exterior multiplication in Cl(B) or in the Grassmann algebra (see `&w` for more help),
- `&q` or 'qmul' for quaternionic multiplication in Cl(B) (see `&q` for more help),
- `&cm` for multiplication of matrices with entries in Cl(B) when the Clifford product 'cmul' is applied to the matrix entries (see `&cm` for more help),
- `&cQm` for multiplication of matrices with entries in Cl(Q) when the Clifford product 'cmulQ' is applied to the matrix entries (see `&cQm` for more help),
- `&wm` for multiplication of matrices with entries in a Grassmann algebra (see `&wm` for more help),
- `&rm` for some generic non-commutative user-defined multiplication of matrices with entries in Cl(B) (see rmulm for more help),
- 'quatbasis' which stores definitions of the quaternionic basis {qi, qj, qk} in terms of the even basis monomials in Cl(3). To see the basis type >quatbasis; at the Maple prompt (see `type/quaternion` for more help).

- Also, at the time of loading the package, all procedure names and constants like 'quatbasis' and {qi, qj, qk} are becoming protected Maple names.
- This procedure need not be used in any way by the user.
- To see this procedure or any other procedure in the package, follow the example below.
- At the time of loading, global variable _prolevel is set to 'false'. See Clifford:-cliparse and

Clifford:-`type/clipolynom` for more information.

- 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.
- All needed types such as `type/clibasmon`, `type/clipolynom`, etc. are defined at the initialization time.

Examples:

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

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


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


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



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

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

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

```
> dim_V:=5:_prolevel:=true:_warnings_flag:=false: #changing some
  values
> CLIFFORD_ENV();
```

```
`>>> Global variables defined in Clifford:-setup are now available and have the
se values: <<<`
`************* Start *************`
dim_V = 5
_default_Clifford_product = Clifford:-cmulNUM
_prolevel = true
_shortcut_in_minimalideal = true
_shortcut_in_Kfield = true
_shortcut_in_spinorKbasis = true
_shortcut_in_spinorKrepr = true
_warnings_flag = false
_scalartypes = {constant, RootOf, `^`, complex, indexed, numeric, function, rat
ional, mathfunc}
_quatbasis = [[Id, e3we2, e1we3, e2we1], {`Maple has assigned qi:=-e2we3, qj:=e
1we3, qk:=-e1we2`}]
`************* End *************`



`>>> Global variables defined in Cliplus:-setup are now available and have thes
e values:  <<<`
`************* Start *************`
macro(Clifford:-cmul = climul)
macro(Clifford:-cmulQ = climul)
macro(`&c` = climul)
macro(`&cQ` = climul)
macro(Clifford:-reversion = clirev)
macro(Clifford:-LC = LCbig)
macro(Clifford:-RC = RCbig)
`************* End *************`



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

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

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

>

When additional packages are loaded, CLIFFORD_ENV displays their environmental variables and macros, if used. For example,

```
> with(Cliplus);
```

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

```
> CLIFFORD_ENV();
```

```
`>>> Global variables defined in Clifford:-setup are now available and have the
se values: <<<`
`************* Start *************`
dim_V = 5
_default_Clifford_product = Clifford:-cmulNUM
_prolevel = true
_shortcut_in_minimalideal = true
_shortcut_in_Kfield = true
_shortcut_in_spinorKbasis = true
_shortcut_in_spinorKrepr = true
_warnings_flag = false
_scalartypes = {constant, RootOf, `^`, complex, indexed, numeric, function, rat
ional, mathfunc}
_quatbasis = [[Id, e3we2, e1we3, e2we1], {`Maple has assigned qi:=-e2we3, qj:=e
1we3, qk:=-e1we2`}]
`************* End *************`



`>>> Global variables defined in Cliplus:-setup are now available and have thes
e values:  <<<`
`************* Start *************`
macro(Clifford:-cmul = climul)
macro(Clifford:-cmulQ = climul)
macro(`&c` = climul)
macro(`&cQ` = climul)
macro(Clifford:-reversion = clirev)
macro(Clifford:-LC = LCbig)
macro(Clifford:-RC = RCbig)
`************* End *************`
```

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



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


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

Other packages whose environmental variables may be displayed with CLIFFORD_ENV are: GTP, Octonion, Bigebra.

>

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, Clifford:-`type/quaternion`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-sexp - Clifford exponential in the Clifford algebra Cl(B)

Calling Sequence:

sexp(p,n);
sexp(p,n,K);

Parameters:

p - an expression of type 'numeric', 'cliscalar' or 'clipolynom'
n - non-negative integer
K - (optional) argument of type name, symbol, array, or matrix, or
`&*`(numeric,{name,symbol,array,matrix})

Description:

- Procedure 'sexp' computes Clifford exponential of a Clifford polynomial p in Cl(B) up to the order specified by the second argument n which is expected to be a nonnegative integer.
- It can now accept optional argument of type name, symbol, matrix, or array, or `&*`(numeric,{name,symbol,array,matrix}).
- Computations are performed modulo the minimal polynomial of p. The minimal polynomial of p is computed with the procedure climinpoly.
- If n = 0 then the procedure returns 1 when p is of type 'numeric' or 'cliscalar' (see `type/cliscalar`) and 'Id' when p is of type 'clipolynom' (see `type/clipolynom`).
- This procedure is much faster than two similar procedures cexp and cexpQ when n is larger than the degree d of the minimal polynomial of p. When n is smaller than or equal to d, procedures 'cexp' and 'cexpQ' are faster since then no minimal polynomial is computed.
- It is not necessary that the form Q (or B) be defined.
- Use 'wexp' to compute exterior exponential in Cl(B). See wexp for more help.

Examples:

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

Example 1:

```
> p:=(e1+e3)*t;
```

$$p := (e1 + e3)\, t$$

```
> climinpoly(p);
  climinpoly[K](p);
  climinpoly[-K](p);
```

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

$$-t^2 B_{1,1} - t^2 B_{3,1} - t^2 B_{1,3} - t^2 B_{3,3} + x^2$$

$$-t^2 K_{1,1} - t^2 K_{3,1} - t^2 K_{1,3} - t^2 K_{3,3} + x^2$$

$$t^2 K_{1,1} + t^2 K_{3,1} + t^2 K_{1,3} + t^2 K_{3,3} + x^2$$

```
> degree(%,'x');
```

$$2$$

Since the degree of the minimal polynomial of p is 4, we will use procedure 'cexp' only when n <= 4; otherwise we will use the faster procedure 'sexp':

```
> s:=time():
  p1:=sexp(p,3); #note the use of off-diagonal B entries
  time()-s;
```

$$p1 := \frac{1}{2}(2 + t^2 B_{3,3} + t^2 B_{1,1} + t^2 B_{3,1} + t^2 B_{1,3})\, Id + \frac{1}{6} t\,(6 + t^2 B_{1,1} + t^2 B_{3,1} + t^2 B_{1,3} + t^2 B_{3,3})\, e1 + \frac{1}{6} t\,(6 + t^2 B_{1,1} + t^2 B_{3,1} + t^2 B_{1,3} + t^2 B_{3,3})\, e3$$

$$0.125$$

```
> s:=time():
  p2:=cexp(p,3);
  time()-s;
```

$$p2 := Id + t\, e1 + t\, e3 + \frac{1}{2}(t^2 B_{1,1} + t^2 B_{3,1} + t^2 B_{1,3} + t^2 B_{3,3})\, Id + \frac{1}{6} t^3 (B_{1,1} + B_{3,1} + B_{1,3} + B_{3,3})\, e1 + \frac{1}{6} t^3 (B_{1,1} + B_{3,1} + B_{1,3} + B_{3,3})\, e3$$

$$0.125$$

The same computation as above but with respect to a new form K:

```
> s:=time():
  p1:=sexp(p,3,K); #note the use of off-diagonal B entries
  time()-s;
```

$$p1 := \frac{1}{2}(2 + t^2 K_{3,3} + t^2 K_{1,1} + t^2 K_{3,1} + t^2 K_{1,3})\, Id + \frac{1}{6} t\,(6 + t^2 K_{1,1} + t^2 K_{3,1} + t^2 K_{1,3} + t^2 K_{3,3})\, e1 + \frac{1}{6} t\,(6 + t^2 K_{1,1} + t^2 K_{3,1} + t^2 K_{1,3} + t^2 K_{3,3})\, e3$$

$$0.093$$

```
> s:=time():
  p2:=cexp(p,3,K);
  time()-s;
```

$$p2 := Id + t\,e1 + t\,e3 + \frac{1}{2}\left(t^2 K_{1,1} + t^2 K_{3,1} + t^2 K_{1,3} + t^2 K_{3,3}\right) Id + \frac{1}{6} t^3 \left(K_{1,1} + K_{3,1} + K_{1,3} + K_{3,3}\right) e1 + \frac{1}{6} t^3 \left(K_{1,1} + K_{3,1} + K_{1,3} + K_{3,3}\right) e3$$

$$0.125$$

```
> simplify(p1-p2);
```

$$0$$

```
>
```

Example 2: Let's see some additional examples in Cl(2,2). Since B will be assigned a numeric matrix, we will use cmulNUM for Clifford internal multiplication instead of cmulRS. The change will be made using procedure useproduct.

```
> useproduct(cmulNUM);
Warning, cmul will use cmulNUM; for help see pages ?cmul, ?Clifford:-intro, or
?cmulNUM
```

```
> B:=linalg[diag](1,1,-1,-1):
  p:=e1we2+e1we3+e4: #define B and p
> climinpoly(p); #finding the minimal polynomial of p
```

$$x^4 + 2x^2 + 1$$

```
> sexp(p,0);
```

$$Id$$

```
> sexp(p,1);
```

$$Id + e1we2 + e1we3 + e4$$

```
> sexp(p,2);
```

$$\frac{Id}{2} + e1we2 + e1we3 + e4 + e1we2we4 + e1we3we4$$

```
> sexp(p,3);
```

$$\frac{Id}{2} + \frac{e1we2}{2} + \frac{e1we3}{2} + \frac{5\,e4}{6} + e1we2we4 + e1we3we4$$

```
> sexp(p,4);
```

$$\frac{13\,Id}{24} + \frac{e1we2}{2} + \frac{e1we3}{2} + \frac{5\,e4}{6} + \frac{5\,e1we2we4}{6} + \frac{5\,e1we3we4}{6}$$

```
> sexp(p,5);
```

$$\frac{13\,Id}{24} + \frac{13\,e1we2}{24} + \frac{13\,e1we3}{24} + \frac{101\,e4}{120} + \frac{5\,e1we2we4}{6} + \frac{5\,e1we3we4}{6}$$

Additional examples:

```
> sexp((e1+e2we3)*t,15);
```

$$\frac{t^2\left(182\,t^{10} + 1891890\,t^4 + 6006\,t^8 + 14189175\,t^2 + 135135\,t^6 + 4\,t^{12} + 42567525\right) e1we2we3}{42567525}$$

$$+ ($$

$$4\,t^{14} + 42567525 + 6006\,t^{10} + 14189175\,t^{4} + 182\,t^{12} + 1891890\,t^{6} + 135135\,t^{8} + 42567525\,t^{2})\,Id / 42567525 + t\,(638512875 + 8\,t^{14} + 425675250\,t^{2} + 85135050\,t^{4} + 8108100\,t^{6} + 450450\,t^{8} + 16380\,t^{10} + 420\,t^{12})\,e2we3 / 638512875 + t\,(638512875 + 8\,t^{14} + 425675250\,t^{2} + 85135050\,t^{4} + 8108100\,t^{6} + 450450\,t^{8} + 16380\,t^{10} + 420\,t^{12})\,e1 / 638512875$$

```
> sexp(2*alpha,0);
```

$$1$$

```
> sexp(2*alpha,7);
```

$$1 + 2\,\alpha + 2\,\alpha^{2} + \frac{4}{3}\,\alpha^{3} + \frac{2}{3}\,\alpha^{4} + \frac{4}{15}\,\alpha^{5} + \frac{4}{45}\,\alpha^{6} + \frac{8}{315}\,\alpha^{7}$$

```
> sexp(2*alpha*Id,10);
```

$$\left(1 + 2\,\alpha + 2\,\alpha^{2} + \frac{4}{3}\,\alpha^{3} + \frac{2}{3}\,\alpha^{4} + \frac{4}{15}\,\alpha^{5} + \frac{4}{45}\,\alpha^{6} + \frac{8}{315}\,\alpha^{7} + \frac{2}{315}\,\alpha^{8} + \frac{4}{2835}\,\alpha^{9} + \frac{4}{14175}\,\alpha^{10}\right) Id$$

```
> sexp(.6,20);
```

$$1.822118801$$

```
>
```

See Also: Clifford:-climinpoly, Clifford:-cexp, Clifford:-wexp, Clifford:-cexpQ

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-specify_constants - define new constants

Calling Sequence:

specify_constants(str1,str2,..,strn)

Parameters:

str1, str2, ..., strn - names of new constants entered as strings

Description:

- Procedure 'specify_constants' allows user to specify any new symbolic constants, e.g., a, b, c, B, etc., which
  are to be known to Maple. The originally known constants are stored in a global, non-protected variable 'constants' and must be saved separately, if needed. This procedure is needed when sorting or collecting multivariate Clifford polynomials containing expressions like 'aa*eiwej' in which 'aa' is intended to be a constant and 'eiwej' is intended to be a Clifford basis monomial with indices i and j.
- Constants of length one are automatically assumed to be Maple constants.

Examples:

```
> restart:with(Clifford):
> specify_constants(a,aa,a1);
Maple now knows the following constant(s): infinity, Catalan, FAIL, false, gamm
a, true, Pi, a, a1, aa
> cmulQ(1+a*e1we2,-aa*e1we2we3+3);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$-aa\ e1we2we3 + a\ aa\ B_{2,2}\ B_{1,1}\ e3 + 3\ Id + 3\ a\ e1we2$$

```
>
```

See Also: Clifford:-cliparse, Clifford:-clicollect, Clifford:-clisort

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-spinorKbasis - find a spinor basis over a field K in a minimal ideal S of Cl(Q)

Calling Sequence:

spinorKbasis(list1,f,list2,str);

Parameters:

list1 - a list of elements of the type 'clibasmon' which generate a real basis for S
f - a primitive idempotent in Cl(Q) used to generate minimal ideal S
list2 - a list of elements of the type 'clibasmon' which generate the field K
str - a string 'left' or 'right' depending whether S is a left or right minimal ideal

Description:

- Procedure 'spinorKbasis' finds a spinor basis in a left minimal ideal S=Cl(B)f or a right minimal ideal S=fCl(B) over a field K. The field K is isomorphic to the reals, complexes, or quaternions depending whether (p-q) mod 8 is 0, 1, 2, or 3, 7, or 4, 5, 6, respectively (here [p,q] is the signature of B).
- The first argument in 'spinorKbasis' is an ordered list containing generators of type 'clibasmon' (see `type/clibasmon` for more help) of a real basis in S (it doesn't matter whether the generators were computed for a left or right ideal S). These generators are found by the procedure 'minimalideal' (see minimalideal for more help) and are returned by it as a second list. They are also stored under clidata()[5] if f is the same as the one stored under clidata()[4] (see clidata for more help).
- The second argument is the primitive idempotent f used to generate S (see `type/primitiveidemp` for more help on primitive idempotents). One possible choice for f is stored under clidata()[4].
- The third argument is an ordered list of generators that generate the field K for the given signature and f; these generators are returned by the procedure 'Kfield' as the second list (see Kfield for more help). If f equals clidata()[4], these generators are stored under clidata()[6].
- The fourth argument is either 'left' or 'right' depending whether we deal with the left minimal ideal Cl(B)f or the right minimal ideal Cl(B)f.
- If the first three arguments in the input match respectively clidata()[5], clidata()[4], and clidata()[6] in that order, i.e., list1=clidata()[5], f=clidata()[4], and list2=clidata()[6], then the procedure returns previously computed generators of S over K. These generators are stored as clidata()[7]. They are then used to compute an expanded K-basis for S=Cl(B)f or S=fCl(B) depending whether the fourth argument was 'left' or 'right'.
- If any of the arguments differs from the ones stored under clidata(), computations are performed and they may take some time.
- The procedure 'spinorKbasis' returns a list of three elements:

- the first list is an ordered list of Clifford polynomials which provide a K-basis for S,

- the second list is an ordered list of generators over f which give the elements in the first list. There is a one-to-one correspondence between the elements of the two lists.
- the third element in the output is either 'left' or 'right' and it matches the fourth argument in the input to the procedure. That element is to remind the user that the basis returned is for the left or right ideal respectively.

- Note: the following examples are a continuation of the examples from the help page for 'Kfield'. See Kfield for more help.
- For more on environmental variables such as _shortcut_in_spinorKbasis, _prolevel, etc. see procedure CLIFFORD_ENV.

Examples:

```
> restart:bench:=time():with(Clifford):
  _prolevel:=false:
  _shortcut_in_spinorKbasis:=false;
```

$$\_shortcut\_in\_spinorKbasis := false$$

To shorten output, we will use aliases. Since below we will not exceed dimension 6, we can define aliases first:

```
> eval(makealiases(6,'ordered')):
```

Thus, for example,

```
> e1 &c e2;
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$e12 + B_{1,2}\, Id$$

where e12 = e1e2. When the bilinear form B is diagonal, then e12 = e1e2 = e1we2, etc.

Example 1: Clifford algebra Cl(1,1) is isomorphic to R(2)

```
> dim:=2:B:=linalg[diag](1,-1): #define the bilinear form B for
  Cl(1,1)
  clibasis:=cbasis(dim): #compute a Clifford basis for Cl(B)
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data := \left[real, 2, simple, \frac{Id}{2} + \frac{e12}{2}, [Id, e1], [Id], [Id, e1]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own
  here
> sbasis:=minimalideal(clibasis,f,'left'):#compute a real basis
  in Cl(B)f
> Kbasis:=Kfield(sbasis,f); #compute a basis for the field K
```

$$Kbasis := \left[\left[\frac{Id}{2} + \frac{e12}{2}\right], [Id]\right]$$

```
> SBgens:=sbasis[2]; #generators for a real basis in S
```

$$SBgens := [Id, e1]$$

```
> FBgens:=Kbasis[2]; #generators for K
```

$$FBgens := [Id]$$

Here is a K_basis returned for S=Cl(B)f. Since for the current signature (1,1) we have that K = R and Cl(1,1) = R(2), the output from 'spinorKbasis' shown below has two basis vectors and their generators, and is the same as the one from minimalideal.

```
> K_basis:=spinorKbasis(SBgens,f,FBgens,'left');
```

$$K\_basis := \left[\left[\frac{Id}{2}+\frac{e12}{2}, \frac{e1}{2}+\frac{e2}{2}\right], [Id, e1], left\right]$$

Compare with 'sbasis' which gives a real basis for S:

```
> sbasis;
```

$$\left[\left[\frac{Id}{2}+\frac{e12}{2}, \frac{e1}{2}+\frac{e2}{2}\right], [Id, e1], left\right]$$

Example 2: Clifford algebra Cl(2,0) is isomorphic to R(2).

```
> dim:=2:B:=linalg[diag](1,1):#define the bilinear form B for
  Cl(2,0)
  clibasis:=cbasis(dim): #compute a Clifford basis for Cl(B)
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data := \left[real, 2, simple, \frac{Id}{2}+\frac{e1}{2}, [Id, e2], [Id], [Id, e2]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own
  here
> sbasis:=minimalideal(clibasis,f,'left'):#compute a real basis
  in Cl(B)f
> Kbasis:=Kfield(sbasis,f); #compute a basis for the field K
```

$$Kbasis := \left[\left[\frac{Id}{2}+\frac{e1}{2}\right], [Id]\right]$$

```
> SBgens:=sbasis[2];#generators for a real basis in S
```

$$SBgens := [Id, e2]$$

```
> FBgens:=Kbasis[2]; #generators for K
```

$$FBgens := [Id]$$

Here is a K_basis returned for S=Cl(B)f. Since for the current signature (2,0) we have that K = R and Cl(2,0)=R(2), the output from 'spinorKbasis' shown below has two basis vectors and their generators modulo f:

```
> K_basis:=spinorKbasis(SBgens,f,FBgens,'left');
```

$$K\_basis := \left[\left[\frac{Id}{2}+\frac{e1}{2}, \frac{e2}{2}-\frac{e12}{2}\right], [Id, e2], left\right]$$

Example 3: Clifford algebra Cl(2,2) is isomorphic to R(4).

```
> dim:=4:B:=linalg[diag](1,1,-1,-1):#define the bilinear form B
  for Cl(2,2)
  clibasis:=cbasis(dim): #compute a Clifford basis for Cl(B)
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data := \left[real, 4, simple, 'Clifford{:}\text{-}cmulQ'\left(\frac{Id}{2}+\frac{e13}{2}, \frac{Id}{2}+\frac{e24}{2}\right), [Id, e1, e2, e12], [Id], [Id, e1, e2, e12]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own here
> sbasis:=minimalideal(clibasis,f,'left'):#compute a real basis in Cl(B)f
> Kbasis:=Kfield(sbasis,f); #compute a basis for the field K
```

$$Kbasis := \left[\left[\frac{Id}{4}+\frac{e13}{4}+\frac{e24}{4}-\frac{e1234}{4}\right], [Id]\right]$$

```
> SBgens:=sbasis[2];#generators for a real basis in S
```

$$SBgens := [Id, e1, e2, e12]$$

```
> FBgens:=Kbasis[2]; #generators for K
```

$$FBgens := [Id]$$

Here is a K_basis returned for S=Cl(B)f. Since for the current signature (2,2) we have that K = R and Cl(2,2)=R(4), the output from 'spinorKbasis' shown below has four basis vectors and their generators modulo f:

```
> K_basis:=spinorKbasis(SBgens,f,FBgens,'left');
```

$$K\_basis := \left[\left[\frac{Id}{4}+\frac{e13}{4}+\frac{e24}{4}-\frac{e1234}{4}, \frac{e1}{4}+\frac{e3}{4}+\frac{e124}{4}-\frac{e234}{4}, \frac{e2}{4}-\frac{e123}{4}+\frac{e4}{4}+\frac{e134}{4}, \frac{e12}{4}-\frac{e23}{4}+\frac{e14}{4}+\frac{e34}{4}\right], [Id, e1, e2, e12], left\right]$$

Example 4: Clifford algebra Cl(3,0) is isomorphic to C(2).

```
> dim:=3:B:=linalg[diag](1,1,1):#define the bilinear form B for Cl(3,0)
  clibasis:=cbasis(dim): #compute a Clifford basis for Cl(B)
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data := \left[complex, 2, simple, \frac{Id}{2}+\frac{e1}{2}, [Id, e2, e3, e23], [Id, e23], [Id, e2]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own here
> sbasis:=minimalideal(clibasis,f,'left'):#compute a real basis in Cl(B)f
> Kbasis:=Kfield(sbasis,f); #compute a basis for the field K
```

$$Kbasis := \left[\left[\frac{Id}{2}+\frac{e1}{2}, \frac{e23}{2}+\frac{e123}{2}\right], [Id, e23]\right]$$

```
> SBgens:=sbasis[2];#generators for a real basis in S
```

$$SBgens := [Id, e2, e3, e23]$$

```
> FBgens:=Kbasis[2]; #generators for K
```

$$FBgens := [Id, e23]$$

Here is a K_basis returned for S=Cl(B)f. Since for the current signature (3,0) we have that K = C and Cl(3,0)=C(2), the output from 'spinorKbasis' shown below has two basis vectors and their generators modulo f:

```
> K_basis:=spinorKbasis(SBgens,f,FBgens,'left');
```

$$K\_basis := \left[\left[\frac{Id}{2}+\frac{e1}{2}, \frac{e2}{2}-\frac{e12}{2}\right], [Id, e2], left\right]$$

Example 5: Clifford algebra Cl(1,3) is isomorphic to H(2).

```
> dim:=4:B:=linalg[diag](1,-1,-1,-1):#define form B for Cl(1,3)
  clibasis:=cbasis(dim): #compute a Clifford basis for Cl(B)
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data := \left[quaternionic, 2, simple, \frac{Id}{2}+\frac{e14}{2}, [Id, e1, e2, e3, e12, e13, e23, e123], [Id, e2, e3, e23], [Id, e1]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own
  here
> sbasis:=minimalideal(clibasis,f,'left'):#compute a real basis
  in Cl(B)f
> Kbasis:=Kfield(sbasis,f); #compute a basis for the field K
```

$$Kbasis := \left[\left[\frac{Id}{2}+\frac{e14}{2}, \frac{e2}{2}-\frac{e124}{2}, \frac{e3}{2}-\frac{e134}{2}, \frac{e23}{2}+\frac{e1234}{2}\right], [Id, e2, e3, e23]\right]$$

```
> SBgens:=sbasis[2];#generators for a real basis in S
```

$$SBgens := [Id, e1, e2, e3, e12, e13, e23, e123]$$

```
> FBgens:=Kbasis[2]; #generators for K
```

$$FBgens := [Id, e2, e3, e23]$$

Here is a K_basis returned for S=Cl(B)f. Since for the current signature (1,3) we have that K = H and Cl(1,3)=H(2), the output from 'spinorKbasis' shown below has two basis vectors and their generators modulo f:

```
> K_basis:=spinorKbasis(SBgens,f,FBgens,'left');
```

$$K\_basis := \left[\left[\frac{Id}{2}+\frac{e14}{2}, \frac{e1}{2}+\frac{e4}{2}\right], [Id, e1], left\right]$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M
  2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 2.141000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

Example 6: Clifford algebra Cl(3,1) is isomorphic to R(4).

```
> restart:bench:=time():with(Clifford):eval(makealiases(4,'ordere
  d')):
> dim:=4:B:=linalg[diag](1,1,1,-1):#define form B for Cl(1,3)
```

```
clibasis:=cbasis(dim): #compute a Clifford basis for Cl(B)
> data:=clidata(B); #retrieve and display data about Cl(B)
```

$$data := \left[real, 4, simple, 'cmulQ'\left(\frac{Id}{2}+\frac{e1}{2}, \frac{Id}{2}+\frac{e34}{2}\right), [Id, e2, e3, e23], [Id], [Id, e2, e3, e23]\right]$$

```
> f:=data[4]:#assign pre-stored idempotent to f or use your own here
> sbasis:=minimalideal(clibasis,f,'left'):#compute a real basis in Cl(B)f
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
> Kbasis:=Kfield(sbasis,f); #compute a basis for the field K
```

$$Kbasis := \left[\left[\frac{Id}{4}+\frac{e1}{4}+\frac{e34}{4}+\frac{e134}{4}\right], [Id]\right]$$

```
> SBgens:=sbasis[2];#generators for a real basis in S
```

$$SBgens := [Id, e2, e3, e23]$$

```
> FBgens:=Kbasis[2]; #generators for K
```

$$FBgens := [Id]$$

Here is a K_basis returned for S=Cl(B)f. Since for the current signature (3,1) we have that K = R and Cl(3,1)=R(4), the output from 'spinorKbasis' shown below has four basis vectors and their generators modulo f:

```
> K_basis:=spinorKbasis(SBgens,f,FBgens,'left');
```

$$K\_basis := \left[\left[\frac{Id}{4}+\frac{e1}{4}+\frac{e34}{4}+\frac{e134}{4}, \frac{e2}{4}-\frac{e12}{4}+\frac{e234}{4}-\frac{e1234}{4}, \frac{e3}{4}-\frac{e13}{4}+\frac{e4}{4}-\frac{e14}{4}, \frac{e23}{4}+\frac{e123}{4}+\frac{e24}{4}+\frac{e124}{4}\right], [Id, e2, e3, e23], left\right]$$

```
> printf("Worksheet took %f seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM with Win XP Professional\n",time()-bench);
Worksheet took 0.498000 seconds to compute on Intel Pentium M 2.13 GHz 2GB RAM
with Win XP Professional
>
```

Comments:

- Using 'spinorKbasis' as shown above is the third step towards the computation of the spinor representation of Cl(Q) in a minimal ideal S over the field K. The first step was to find a real basis for S with 'minimalideal' and the second was to compute a basis for the field K with the procedure 'Kfield' (see minimalideal and Kfield for more information). The next step is to use procedure 'spinorKrepr' to find a matrix representation of Cl(Q) (see spinorKrepr).
- The K_bases found by the procedure 'spinorKbasis' in the above examples may now be entered into the procedure 'spinorKrepr' to find matrix representations R(2), C(2) and H(2) of the above Clifford algebras.

- The above examples may be redone for right minimal ideals with the same generators for the field K. This is because K is isomorphic to the intersection of the left and right minimal ideals Cl(B)f and fCl(B).

See Also: Clifford:-Kfield, Clifford:-Bsignature, Clifford:-`type/primitiveidemp`, Clifford:-spinorKrepr, Clifford:-matKrepr, Clifford:-minimalideal, Clifford:-`type/clipolynom`, Clifford:-clidata

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-squaremodf - compute the square modulo f of a basis element in a minimal ideal S

Calling Sequence:

squaremodf(p,f);

Parameters:

p - Clifford polynomial of type 'clipolynom'
f - a primitive idempotent generating S

Description:

- Procedure 'squaremodf' computes the square of a basis element element p in a left or right minimal ideal S=Cl(B)f or S=fCl(B) (see minimalideal for more help).
- The element p of the type 'clipolynom' is entered as the first argument while the idempotent f is entered as a second argument (see `type/clipolynom` and `type/idempotent` for more information).
- The procedure doesn't check whether f is primitive or not. See `type/primitiveidemp` for more help on primitive idempotents.
- The procedure returns 1 or -1 depending whether cmul(p,p) = f or cmul(p,p) = -f. It returns 0 if p is a nilpotent element (see `type/nilpotent` for more help on nilpotent elements).
- If the element p does not belong to S, an error message is returned.
- This procedure is needed to identify/verify squares of the basis elements of the spinor ideal S and is used by the procedure 'Kfield'. See Kfield for more information.
- Since the field K is isomorphic to the intersection Cl(Q)f and fCl(Q), it is not needed to specify if S is a right or a left minimal ideal in Cl(Q).

Examples:

```
> restart:with(Clifford):
> B:=linalg[diag](1,1,1): #define B of signature (3,0)
> f:=clidata(B)[4];#retrieve a primitive idempotent for this signature
```

$$f := \frac{Id}{2} + \frac{e1}{2}$$

```
> sbasis:=minimalideal(cbasis(3),f,'left');#this is a basis for S=Cl(B)f
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$sbasis := \left[\left[\frac{Id}{2} + \frac{e1}{2}, \frac{e2}{2} - \frac{e1we2}{2}, \frac{e3}{2} - \frac{e1we3}{2}, \frac{e2we3}{2} + \frac{e1we2we3}{2}\right], [Id, e2, e3, e2we3], left\right]$$

```
> map(squaremodf,sbasis[1],f); #apply 'squaremodf' to sbasis[1]
```

$$[1, 0, 0, -1]$$

From the above it is seen that sbasis[1][2] and sbasis[1][3] are nilpotent while sbasis[1][1] and sbasis[1][4] provide a basis for the field K isomorphic to C and, as such, are of the type 'fieldelement'. See `type/fieldelement` and Kfield for more information.

```
> squaremodf(e1,f); #element e1 does not belong to S, so an error message results
Error, (in Clifford:-squaremodf) either element e1 is not a basis element or it does not belong to the spinor space Cl(Q)f (or fCl(Q))
```

```
>
```

See Also: Clifford:-cbasis, Clifford:-clidata, Clifford:-minimalideal, Clifford:-`type/fieldelement`, Clifford:-Kfield

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-`convert/str_to_int` - convert strings `1`, `2`, etc., to numerals 1, 2, etc.

Calling Sequence:

convert(str, str_to_int);

map(convert,list,str_to_int);

Parameters:

str - a numerical string `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, or `0`
list - a list of numerical strings

Description:

- This useful conversion procedure converts numerical strings returned by the procedure 'extract' to numerals.
  It is used when option 'integers' is chosen in the procedure 'extract' (see extract for more help).
- This procedure is mostly for internal use.

Examples:

```
> restart:with(Clifford):
> convert(`1`,str_to_int);
```

$$1$$

```
> L:=extract(e1we2we3);
```

$$L := [1, 2, 3]$$

```
> map(convert,L,str_to_int);
```

$$[1, 2, 3]$$

```
>
```

See Also: Clifford:-extract

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-subs_clipolynom - substitute any Clifford polynomial p into a polynomial q(x)

Calling Sequence:

subs_clipolynom(p,q);
subs_clipolynom(p,q,'horner')

Parameters:

p - a Clifford polynomial of type 'clipolynom'
q - a polynomial in a single indeterminate x
'horner' - (optional) displays q with x replaced with p in 'horner' form

Description:

- Procedure 'subs_clipolynom' substitutes any Clifford polynomial p of type 'clipolynom' into an arbitrary polynomial q(x) in a single indeterminate x.
- If the optional argument 'horner' is used, polynomial q(p) is displayed in 'horner' form. Use 'eval' to evaluate it.
- Try `convert/horner` for more information on option 'horner'.

Examples:

```
> restart:with(Clifford):
> B:=linalg[diag](1$3);eval(makealiases(3)):
```

$$B := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
> q:=2*x^5-5*x^4-x^3+2*x^2-9;
```

$$q := 2\,x^5 - 5\,x^4 - x^3 + 2\,x^2 - 9$$

```
> convert(q,horner);
```

$$-9 + (2 + (-1 + (-5 + 2\,x)\,x)\,x)\,x^2$$

```
> expand(%);
```

$$2\,x^5 - 5\,x^4 - x^3 + 2\,x^2 - 9$$

Example 1: Let's define a Clifford polynomial p in Cl(B):

```
> p:=2-e1we2+e3+2*e1-e2we3;
```

$$p := 2 - e12 + e3 + 2\,e1 - e23$$

Let's now substitute p into q(x):

```
> subs_clipolynom(p,q);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$-322\,Id - 365\,e12 - 247\,e3 - 188\,e1 - 671\,e23 - 756\,e123$$

Let's now substitute p into q(x) expressed in 'horner' form:

```
> subs_clipolynom(p,q,'horner');
```

$$-9\,Id + Clifford{:}\text{-}cmul(Clifford{:}\text{-}cmul(2\,Id + Clifford{:}\text{-}cmul(-Id + Clifford{:}\text{-}cmul(-Id - 2\,e12 + 2\,e3 + 4\,e1 - 2\,e23, 2\,Id - e12 + e3 + 2\,e1 - e23), 2\,Id - e12 + e3 + 2\,e1 - e23), 2\,Id - e12 + e3 + 2\,e1 - e23), 2\,Id - e12 + e3 + 2\,e1 - e23)$$

```
> eval(%);
```

$$-322\,Id - 365\,e12 - 247\,e3 - 188\,e1 - 671\,e23 - 756\,e123$$

This procedure is useful when verifying, for example, that the given polynomial q(x) is the minimal polynomial of p, that is, q(p)=0.

```
> q:=climinpoly(p);
```

$$q := x^4 - 8\,x^3 + 18\,x^2 - 8\,x + 37$$

```
> subs_clipolynom(p,q,'horner');
```

$$37\,Id + Clifford{:}\text{-}cmul(-8\,Id + Clifford{:}\text{-}cmul(18\,Id + Clifford{:}\text{-}cmul(-6\,Id - e12 + e3 + 2\,e1 - e23, 2\,Id - e12 + e3 + 2\,e1 - e23), 2\,Id - e12 + e3 + 2\,e1 - e23), 2\,Id - e12 + e3 + 2\,e1 - e23)$$

```
> expand(%);
```

$$0$$

```
>
```

Example 2: Let p = 2*Id-3. Find minimal polynomial of p and check that it evaluates to 0 upon substituting p for x.

```
> p:=2*Id-3;
```

$$p := 2\,Id - 3$$

```
> pol:=climinpoly(p);
```

$$pol := x + 1$$

```
> subs_clipolynom(p,pol);
```

$$0$$

```
> subs_clipolynom(p,pol,'horner');
```

$$0$$

```
>
```

See Also: Clifford:-climinpoly, Clifford:-cexp, Clifford:-sexp, Clif6:-`type/clipolynom`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-`type/symmatrix` - define a new type: a symmetric matrix

Calling Sequence:

type(m, symmatrix);

Parameters:

m - a name type 'matrix'

Description:

- The procedure checks if the matrix m is symmetric or not.
- The procedure returns 'true' or 'false' depending whether its argument is or is not of the type 'symmatrix'.
- See also related types: `type/antisymmatrix`, `type/diagmatrix`, and `type/climatrix`.

Examples:

```
> restart:with(Clifford):
> B:=linalg[diag](1,1,-1);
```

$$B := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

```
> type(B,diagmatrix);
```

$$true$$

```
> type(B,symmatrix);
```

$$true$$

```
> type(B,antisymmatrix);
```

$$false$$

```
> type(B,climatrix);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$false$$

```
> B:=linalg[matrix](2,2,[a,b,-b,c]);
```

$$B := \begin{bmatrix} a & b \\ -b & c \end{bmatrix}$$

```
> type(B,diagmatrix);
```

$$false$$

```
> type(B,symmatrix);
```

$$false$$

```
> type(B,antisymmatrix);
```

$$false$$

```
> type(B,climatrix);
```

$$false$$

```
> BA:=linalg[matrix](2,2,[0,b,-b,0]);
```

$$BA := \begin{bmatrix} 0 & b \\ -b & 0 \end{bmatrix}$$

```
> type(BA,diagmatrix);
```

$$false$$

```
> type(BA,symmatrix);
```

$$false$$

```
> type(BA,antisymmatrix);
```

$$true$$

```
> type(BA,climatrix);
```

$$false$$

```
> cliB:=linalg[matrix](2,2,[1+e1,2*e1we2-e2we3,e4,e5]);
```

$$cliB := \begin{bmatrix} 1 + e1 & 2\, e1we2 - e2we3 \\ e4 & e5 \end{bmatrix}$$

```
> type(cliB,diagmatrix);
```

$$false$$

```
> type(cliB,symmatrix);
```

$$false$$

```
> type(cliB,antisymmatrix);
```

$$false$$

```
> type(cliB,climatrix);
```

$$true$$

```
>
```

See Also: Clifford:-`type/climatrix`, Clifford:-`type/diagmatrix`, Clifford:-`type/antisymmatrix`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-`type/tensorprod` - define a new type 'tensorprod'

Calling Sequence:

type(p,tensorprod);

Parameters:

p - element of the type 'function' or 'anything'.

Description:

- The basis elements in the graded tensor product Cl(B1) &t Cl(B2) &t ... &t &t Cl(Br) of r Clifford algebras Cl(Bi), where Bi are quadratic forms, 1<=i<=r, are by definition of type *'tensorprod'*.
- Note that this is a different `&t` than the one defined in Bigebra package.
- Thus, any input like &t(e1,e2,e3) or e1 &t e2 &t e1we2 is of that type.
- See also GTP:-`type/gradedpolynom` and GTP:-`type/gradedmonom`.

Examples:

```
> restart:with(Clifford):with(GTP):
> type(e1 &t e1,tensorprod),type(Pi*(e1we2 &t e1 &t
  e2),tensorprod);
```

$$true, false$$

```
> type(2*&t(e1,e2,e3),tensorprod),type(&t(e1,e2,e3),tensorprod);
```

$$false, true$$

```
> type(2*&t(e1,e2,e3)+e2we3 &t e2we1,tensorprod);
```

$$false$$

```
> type(2*&t(e1,e2,e3)+e2we3 &t e2we1,tensorprod);
```

$$false$$

```
> dim:=2:K:=cbasis(2):L:=gbasis(K$2);
```

$$L := [Id \,\&t\, Id, Id \,\&t\, e1, Id \,\&t\, e2, Id \,\&t\, e1we2, e1 \,\&t\, Id, e1 \,\&t\, e1, e1 \,\&t\, e2, e1 \,\&t\, e1we2, e2 \,\&t\, Id, e2 \,\&t\, e1, e2 \,\&t\, e2, e2 \,\&t\, e1we2, e1we2 \,\&t\, Id, e1we2 \,\&t\, e1, e1we2 \,\&t\, e2, e1we2 \,\&t\, e1we2]$$

```
> map(type,L,tensorprod);
```

$$[true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true]$$

```
>
```

See Also: GTP:-`type/gradedpolynom`, GTP:-`type/gradedmonom`, GTP:-gbasis, GTP:-`type/gradedodd`, GTP:-grade, GTP:-`&t`, GTP:-gradedprod, GTP:-gprod, GTP:-`type/gradedeven`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-useproduct - allows user to select which Clifford product is used by 'cmul'

Calling Sequence:

useproduct(name);

Parameters:

name - name of the Clifford product to be used by the procedure 'cmul'. Allowed values of 'name' are: 'cmulNUM', 'cmulRS', 'cmulgen', or cmul_user_defined

Description:

- Procedure 'useproduct' allows user to select which procedure, that gives Clifford product, is used in 'cmul' internally. The choices are:

  (1) 'cmulRS' (default) - Computes Clifford product of any two Clifford polynomials of `type/clipolynom` using Rora-Stein cliffordization technique (see Bigebra for more information). This procedure is suitable for computations with symbolic bilinear form B especially when dim(V) >5. Procedure 'cmul' is then a wrapper functions which extends 'cmulNUM' to more than just two polynomials.

  (2) 'cmulNUM' - Computes Clifford product of any two Clifford polynomials `type/clipolynom` using recursively Chevalley's definition of the Clifford product (see 'cmul' for more information). This procedure is suitable when B has been defined as a numeric matrix. Procedure 'cmul' is then a wrapper functions which extends 'cmulNUM' to more than just two polynomials.

  (3) cmulgen - This procedure is a 'dummy procedure' which checks if the currently assigned name to the global variable _default_Clifford_product is one of 'cmulNUM', 'cmulRS', 'cmulgen', or 'cmul_user_defined'.

  (4) cmul_user_defined - This is a generic name for the Clifford product that user may have developed. It is expected to be a procedure that takes exactly three arguments: p1, p2, and K, where p1 and p2 are, for example, Clifford polynomials, while K is of type name, symbol, matrix, array (see some examples below).

- Thus, since the user can really change the internally used procedure in 'cmul', this procedure allows one to test a new code for Clifford product, or defined new Clifford products.

Examples:

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

Upon loading CLIFFORD, one of the environmental variables displayed by CLIFFORD_ENV is _default_Clifford_product. The default value of that variable is 'cmulRS' which gives the Clifford

product computed using the Rota-Stein cliffordization technique (see package Bigebra for more help on the subject).

```
> CLIFFORD_ENV();
```

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


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


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



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



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


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

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

To see that indeed procedure 'cmulRS' is used for internal computations, let's cause an error message in 'cmulgen':

```
> _default_Clifford_product;
```

$$Clifford{:}\text{-}cmulNUM$$

```
> cmulgen(e1,e2,B);
```

$$e1we2 + B_{1,2}\, Id$$

Hence, 'cmulgen' holds the name 'cmulRS'. Since 'cmulgen' applies internally name currently assigned to the global variable _default_Clifford_product to its arguments, and since 'cmulRS' (like 'cmulNUM') requires three arguments, the above error message has been returned.

It is possible to change the value of _default_Clifford_product to one of the following: 'cmulNUM', cmulgen, or _user_defined_Clifford_product. When the change is made, a warning message is printed as follows:

```
> _default_Clifford_product; #current value of the global
  variable
  useproduct(cmulNUM); #selecting 'cmulNUM'
  _default_Clifford_product; #current value of the global
  variable
```

$$Clifford{:}\text{-}cmulNUM$$

```
Warning, cmul will use cmulNUM; for help see pages ?cmul, ?Clifford:-intro, or
?cmulNUM
```

$$Clifford{:}\text{-}cmulNUM$$

```
> _default_Clifford_product;
  cmulgen(e1,e2,K);
```

$$Clifford{:}\text{-}cmulNUM$$

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

Since now we wanted to use 'cmulNUM', the above shows that instead of 'cmulRS', 'cmulNUM' is now used internally to compute Clifford product.

Let's assign now the generic name 'cmulgen' to the global variable

```
> useproduct(cmulgen);
Warning, cmul will use cmulgen; for help see pages ?cmul, ?Clifford:-intro, or
?cmulgen
```

```
> cmulgen(e1,e2);
Warning, to assign Clifford product, execute 'useproduct' with argument cmulRS,
cmulNUM, or cmul_user_defined first
```

$$Clifford{:}\text{-}cmulgen(e1, e2)$$

Of course nothing has happened since _default_Clifford_product = cmulgen which is a dummy

variable.

```
> _default_Clifford_product;
```

$$Clifford{:}\text{-}cmulgen$$

If user wants to create a new Clifford product, it can be assigned to a name _user_defined_Clifford_product. Of course, that generic name can be aliased to some shorter name. Here we just show an example where no procedure has been assigned yet to cmul_user_defined:

```
> useproduct(cmul_user_defined);
Warning, no computations with cmul can be peformed yet since cmul_user_defined
has not been defined as procedure. Select cmulRS, cmulNUM, or a new procedure
as argument to useproduct.
```

For example, we could define the following new Clifford product (for more help, see cmul_user_defined):

```
> cmul_user_defined:=proc(p1::{cliscalar,clibasmon,climon,clipoly
  nom},

  p2::{cliscalar,clibasmon,climon,clipolynom},
                                K::{name,symbol,matrix,array}) local
  a1,a2,p:
  a1:=scalarpart(p1)+vectorpart(p1,1):#assign to a1 only the
  scalar and the 1-vector parts of p1
  a2:=scalarpart(p1)+vectorpart(p2,1):#assign to a2 only the
  scalar and the 1-vector parts of p2
  RETURN(LC(a1,a2,K));
  end:
```

As it can be guessed from the above code, the result of cmul_user_defined(p1,p2,K) is just the left contraction of the scalar and 1-vector part of p2 by the scalar and the 1-vector part of p1 where the contraction is taken with respect to some form K.

```
> useproduct(cmul_user_defined);
Warning, cmul will use cmul_user_defined; for help see pages ?cmul,
?Clifford:-intro, or ?cmul_user_defined
```

```
> cmul(e1,e2+e3we4);
```

$$B_{1,2}\, Id$$

```
>
```

See Also: Clifford:-`type/clipolynom`, Clifford:-`type/climon`, Clifford:-`type/clibasmon`, Clifford:-`type/cliscalar`, Clifford:-cmulNUM, Clifford:-cmulRS, Clifford:-cmulgen, Clifford:-vectorpart, Clifford:-scalarpart

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-vectorpart - find the k-vector part in a Clifford polynomial

Calling Sequence:

vectorpart(p,k);

Parameters:

p - expression of the type 'cliscalar' or 'clipolynom'
k - a nonnegative integer

Description:

- Procedure 'vectorpart' computes the k-vector part of the given Clifford polynomial p where k is a nonnegative integer specified as the second argument. See `type/cliscalar` and `type/clipolynom` for more help on these basic types.
- When k = 0 then the procedure returns the scalar part of p times the identity element 'Id' of Cl(B).
- When the k-vector part is zero, the procedure returns zero.
- Note that vectorpart(2*Id + e1we2, 0) = 2*Id while scalarpart(2*Id + e1we2) = 2. See scalarpart for more help.
- Use procedure maxgrade to determine the maximum grade in p or whether p is of a certain grade.

Examples:

```
> restart:with(Clifford):_prolevel:=true:
> p:=3*Pi+e2we1we3+4-e2we1+e1+2*e3;
```

$$p := 3\,\pi + e2we1we3 + 4 - e2we1 + e1 + 2\,e3$$

```
> vectorpart(p,0);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$3\,\pi + 4$$

```
> scalarpart(p);
```

$$3\,\pi + 4$$

```
> vectorpart(p,1);
```

$$e1 + 2\,e3$$

```
> vectorpart(p,2);
```

$$-e2we1$$

```
> vectorpart(p,3);
```

$$e2we1we3$$

```
> vectorpart(p,4);
```

$$0$$

```
> vectorpart(p,5);
```

$$0$$

```
> p2:=2+2*e1we2-e3we4we5;
```

$$p2 := 2 + 2\ e1we2 - e3we4we5$$

```
> maxgrade(p2);
```

$$3$$

```
> for i from 0 to maxgrade(p2) do
  `vectorpart(p2,`||i||`)`=vectorpart(p2,i) od;
```

$$vectorpart(p2,0) = 2$$

$$vectorpart(p2,1) = 0$$

$$vectorpart(p2,2) = 2\ e1we2$$

$$vectorpart(p2,3) = -e3we4we5$$

```
> p3:=2*e1we2we3we4;
```

$$p3 := 2\ e1we2we3we4$$

```
> evalb(p3=vectorpart(p3,maxgrade(p3)));
```

$$true$$

```
>
```

See Also: Clifford:-scalarpart, Clifford:-`type/clipolynom`

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-version - display information about the current version of the 'CLIFFORD' package

Calling Sequence:

version();

Parameters:

no parameters needed

Description:

- Procedure 'version' displays information about the current version of the 'CLIFFORD' package.

Examples:

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

++

CLIFFORD - A Maple 11 Package for Clifford Algebras with "Bigebra"

(Version 10 with environmental variables given by CLIFFORD_ENV())

Last revised: December 20, 2007 (Source file: clifford_M11_08.mws)

Copyright 1995-2008 by Rafal Ablamowicz (*) and Bertfried Fauser ($)

(*) Department of Mathematics, Box 5054

Tennessee Technological University, Cookeville, TN 38505

tel: USA (931) 372-3569, fax: USA (931) 372-6353

rablamowicz@tntech.edu

http://math.tntech.edu/rafal/Cliff8/

($) Universit"at Konstanz, Fachbereich Physik, Fach M678

78457 Konstanz, Germany

Bertfried.Fauser@uni-konstanz.de

http://kaluza.physik.uni-konstanz.de/~fauser/

If you are a Clifford algebra pro, assign 'true' to '_prolevel' and see

how much faster your computations will be! But watch your syntax!

Use 'useproduct' to change value of _default_Clifford_product in Cl(B) from

cmulRS when B is symbolic to cmulNUM when B is numeric. Type ?cmul for help.

Type CLIFFORD_ENV() to see current values of environmental variables.

++++++++++++This is CLIFFORD version 11++++++++++++

There is a supplementary package 'Cliplus' that extends 'CLIFFORD':

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

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

```
>
```

See Also: CLIFFORD_ENV, Clifford:-makealiases

Last modified: December 20, 2007, RA/BF.

Function: Clifford:-wedge - wedge (exterior) product in a Grassmann algebra Ext(V) or in a Clifford algebra Cl(V,B)

Calling Sequence:

wedge(p1,p2,...pn);

p1 &w p2 &w ... &w pn;

Parameters:

p1, p2, ..., pn - expressions of the type 'cliscalar' or 'clipolynom'

Description:

- Procedure 'wedge' defines an exterior (wedge) product of any number of Clifford polynomials p1, p2, ..., pn. See `type/cliscalar` and `type/clipolynom` for more information on these basic types. There is a supplementary procedure Cliplus:-dwedge that computes dotted wedge product. Conversion from the undotted wedge product to the dotted wedge product is done by `convert/wedge_to_dwedge`.
- When only one argument is entered, the procedure returns it unevaluated.
- The infix form of this associative multiplication is &w. Thus, wedge(p1, p2, p3) = p1 &w p2 &w p3, etc.
- Via the procedure 'rmulm', wedge multiplication may be applied to matrices with entries in a Clifford algebra or in an exterior algebra. See rmulm for more help on multiplication of Clifford matrices of type 'climatrix' (see `type/climatrix` for more help too).
- In the case when input contains an index larger than the currently set dimension of V, an error message is returned. The dimension of V can be set by the user in two ways:
  (1) By changing the default value of a global variable dim_V which is set to 9 upon initialization of the package. To change the value of dim_V, just assign to it a positive integer between 1 and 9 (inclusive).
  (2) By defining a bilinear form B as a matrix of size dim by dim, then the value of dim_V is then set to dim by the procedure.
- If the value of dim_V is not a positive integer between 1 and 9 inclusive, an error message will be returned upon the first use of 'wedge'.
- When this procedure is invoked, it checks whether the bilinear form has been assigned. If yes, it then checks whether the size of B is different than the current value of dim_V. If not, computation proceeds. If yes, the size of B returned by linalg[coldim] is assigned to dim_V. If it amounts to assigning a smaller value to dim_V, a warning message is issued.
- Reducing value of dim_V may result in some terms in the output being truncated.
- Warning message may be suppressed by assigning 'false' to a global variable _warnings_flag. The latter is set to its default value of 'true' by the initialization file.

- To display values of all environmental variables used in 'CLIFFORD, use procedure CLIFFORD_ENV.

Examples:

```
> restart:with(Clifford):
  dim_V;              #default value of 9 set in the
  initialization procedure Clifford:-setup
  _warnings_flag;     #default value of 'false' set in the
  initialization file
  assigned(B);
```

$$9$$

$$true$$

$$false$$

Example 1: Some simple computations. Notice that output is automatically reordered using procedure reorder.

```
> wedge(1+e3we2-3*ejweiwe3,2*Id+e1we2we3+3*ejweiwe3);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

$$2\,Id - 2\,e2we3 + 3\,e3weiwej + e1we2we3$$

```
> wedge(1+e3we2,-3*ejweiwe3,2*Id+e1we2we3+3*ejwekwe4);
```

$$6\,e3weiwej$$

```
> wedge(2*Id+alpha1,5*Id-alpha2);
```

$$(2+\alpha 1)\,(5-\alpha 2)\,Id$$

```
>
```

Example 2: Using infix form of 'wedge':

```
> (1+e3we2-3*ejweiwe3) &w (2*Id+e1we2we3+3*ejweiwe3);
```

$$2\,Id - 2\,e2we3 + 3\,e3weiwej + e1we2we3$$

```
> (1+e3we2) &w (-3*ejweiwe3) &w (2*Id+e1we2we3+3*ejwekwe4);
```

$$6\,e3weiwej$$

```
>
```

Example 3: Let's test some error messages.

```
> dim_V:=2:
> wedge(3*Pi*e2we1we3,e4); ###<<<- Intended error message
Error, (in Clifford:-wedge) argument(s) contain(s) index larger then current
value of dim_V which is now 2. To complete computation, increase value of dim_V
or assign square matrix of size at least 4 by 4 to bilinear form B
```

This message is no longer displayed and computation can proceed by setting a larger value to dim_V or its alias 'dim':

```
> dim_V:=4:
> wedge(3*Pi*e2we1we3,e4);
```

$$-3\,\pi\,e1we2we3we4$$

If too large value has been assigned to dim_V or its alias 'dim', 'wedge' returns an error message:

```
> dim_V:=10:
> wedge(3*Pi*e2we1we3,e5); ###<<<- Intended error message
Error, (in Clifford:-wedge) value of dim_V must be a positive integer between 1 and 9, inclusive, but current value of dim_V is 10
```

One can change the value of dim_V by assigning a positive integer n, 1<= n <= 9, to its alias 'dim', or by defining a bilinear form B. Notice that when the value of dim_V is reduced, a warning message is printed.

```
> B:=matrix(5,5,[]):
> wedge(3*Pi*e2we1we3);
Warning, since B has been (re-)assigned, value of dim_V has been reduced by 'wedge' to 5
```

$$3\,\pi\, e2we1we3$$

However, when the value of dim_V is increased or not changed, no warning message is printed:

```
> B:=matrix(5,5,[]):
> wedge(3*Pi*e2we1we3,e5);
```

$$-3\,\pi\, e1we2we3we5$$

```
> B:=matrix(6,6,[]):
> wedge(3*Pi*e2we1we3,e5);
```

$$-3\,\pi\, e1we2we3we5$$

Printing of the warning message can be suppressed by assigning 'false' to the global variable _warnings_flag:

```
> _warnings_flag:=false;
  dim_V;
```

$$\_warnings\_flag := false$$

$$6$$

```
> B:=matrix(4,4,[]):
> wedge(3*Pi*e2we1we3,e4);
```

$$-3\,\pi\, e1we2we3we4$$

```
> dim_V;
```

$$4$$

```
>
```

Example 4: One can also apply 'wedge' product to matrices with entries in a Clifford algebra. Then, one uses the infix form of such product, namely Clifford:-`&m` or the long form 'rmulm'. In order to assign 'by hand' value to dim_V, we need to unassign B first. This way, we can proceed with computations in the Grassmann algebra Ext(V), where dim(V) = dim_V.

```
> B:='B':
  dim_V:=9;
  M:=matrix(2,2,[2*e1we2,e3,3+e4,e5]);
```

$$dim\_V := 9$$

$$M := \begin{bmatrix} 2\, e1we2 & e3 \\ 3 + e4 & e5 \end{bmatrix}$$

```
> M &wm M;
```

$$\begin{bmatrix} 3\ e3 + e3we4 & 2\ e1we2we3 + e3we5 \\ 6\ e1we2 + 2\ e1we2we4 + 3\ e5 - e4we5 & 3\ e3 - e3we4 \end{bmatrix}$$

```
> rmulm(M,M,wedge);
```

$$\begin{bmatrix} 3\ e3 + e3we4 & 2\ e1we2we3 + e3we5 \\ 6\ e1we2 + 2\ e1we2we4 + 3\ e5 - e4we5 & 3\ e3 - e3we4 \end{bmatrix}$$

```
>
```

See Also: Clifford:-`type/climatrix`, Clifford:-rmulm, Clifford:-`type/clipolynom`, Cliplus:-dwedge

Last modified: December 20, 2007, 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 relay on different filtrations. The dotted wedge product and the undotted one are related by the process of cliffordization which is used in CLIFFORD internally to compute the Clifford product in Cl(V,B). However, the cliffordization is performed in this case by an antisymmetric bilinear form B=F, say F(x,y)=-F(y,x), where x and y are 1-vectors in V.
- **NOTE:** Till now both types of algebras are expanded (formally) over the same basis Grassmann monomials which, according to CLIFFORD's convention, are written as 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, 2002.
[2] Ablamowicz, R. and Bertfried Fauser: “On the decomposition of Clifford algebras of arbitrary bilinear form”, Rafal Ablamowicz and Bertfried Fauser, in “Clifford Algebras and their Applications in Mathematical Physics”, Eds. Rafal Ablamowicz and Bertfried Fauser, Vol. 1: Algebra and Physics, Birkhäuser, Boston, 2000, pages 341--366 (see also http://www.birkhauser.com/cgi-win/isbn/0-8176-4182-3).
[3] “Clifford Algebras and their Applications in Mathematical Physics”, Eds. Rafal Ablamowicz and Bertfried Fauser, Vol. 1: Algebra and Physics, Birkhäuser, Boston, 2000, (ISBN 0-8176-4182-3) (for more information go to http://math.tntech.edu/rafal/mexico/mexico.html).
[4] Ablamowicz, R. and P. Lounesto: “On Clifford algebras of a bilinear form with an antisymmetric part,” with P. Lounesto, in “Clifford Algebras with Numeric and Symbolic computations”, Eds. R. Ablamowicz, P. Lounesto, and J. Parra, Birkhäuser, Boston, 1996, pages 167-188.
[5] “Clifford Algebras with Numeric and Symbolic Computations”, Eds. Rafal Ablamowicz, Pertti Lounesto, and J. Parra, Birkhäuser, Boston, 1996 (ISBN 0-8176-3907-1).

Examples:

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

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

$$[LCbig, RCbig, clibasis, clieval, cliexpand, climul, clirev, dottedcbasis, dwedge, makeclialiases]$$

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

```
> convert(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 *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 smilarly for other elements:

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

$$e1We2, e1We3, e2We3, e1We2We3$$

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

```
> d_bas;
```

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

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

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

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

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

$$[Id, e1We2, e1We3, e2We3]$$

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

$$[e1We2, e1We3, e2We3]$$

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

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

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

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

$$[Id, e1We2, e1We3, e2We3]$$

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

$$[e1We2, e1We3, e2We3]$$

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

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

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

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

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

$$-e1we2$$

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

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

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

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);
```

$$reversed\_e1We2 := -F_{1,2}\, Id - e1we2$$

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: Bigebra:-help, Cliplus:-dwedge, Clifford:-reversion, Clifford:-cbasis

Last modified: December 20, 2007, RA/BF.

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:

- p1,p2,...,pn - 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:

- c1 : 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 relay 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 Grassmann 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:** Till 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, 2002.
[2] Ablamowicz, R. and Bertfried Fauser: “On the decomposition of Clifford algebras of arbitrary bilinear form”, Rafal Ablamowicz and Bertfried Fauser, in “Clifford Algebras and their Applications in Mathematical Physics”, Eds. Rafal Ablamowicz and Bertfried Fauser, Vol. 1: Algebra and Physics, Birkhäuser, Boston, 2000, pages 341--366 (see also http://www.birkhauser.com/cgi-win/isbn/0-8176-4182-3).
[3] “Clifford Algebras and their Applications in Mathematical Physics”, Eds. Rafal Ablamowicz and Bertfried Fauser, Vol. 1: Algebra and Physics, Birkhäuser, Boston, 2000, (ISBN 0-8176-4182-3) (for more information go to http://math.tntech.edu/rafal/mexico/mexico.html).
[4] Ablamowicz, R. and P. Lounesto: “On Clifford algebras of a bilinear form with an antisymmetric part,” with P. Lounesto, in “Clifford Algebras with Numeric and Symbolic computations”, Eds. R. Ablamowicz, P. Lounesto, and J. Parra, Birkhäuser, Boston, 1996, pages 167-188.
[5] “Clifford Algebras with Numeric and Symbolic Computations”, Eds. Rafal Ablamowicz, Pertti Lounesto, and J. Parra, Birkhäuser, Boston, 1996 (ISBN 0-8176-3907-1).

Examples:

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

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

$[LCbig, RCbig, clibasis, clieval, cliexpand, climul, clirev, dottedcbasis, dwedge, makeclialiases]$

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);
```

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

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

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

$$+ 2\, e1we3we4 + e1we4$$

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

```
>
```

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_d
  wedge,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),wedg
  e_to_dwedge,F));
```

$$e1we2we3we4 + F_{2,3}\, F_{1,4}\, Id - F_{1,3}\, F_{2,4}\, Id + F_{1,2}\, F_{3,4}\, Id - F_{2,4}\, e1we3 + F_{1,4}\, e2we3 + F_{3,4}\, e1we2 + F_{1,2}\, e3we4 + F_{2,3}\, e1we4 - F_{1,3}\, e2we4 = e1we2we3we4 + F_{2,3}\, F_{1,4}\, Id - F_{1,3}\, F_{2,4}\, Id + F_{1,2}\, F_{3,4}\, Id - F_{2,4}\, e1we3 + F_{1,4}\, e2we3 + F_{3,4}\, e1we2 + F_{1,2}\, e3we4 + F_{2,3}\, e1we4 - F_{1,3}\, e2we4$$

```
> ##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));%%-%;
```

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

$$+ e1we2we3we4 - F_{1,3}\, e2we4$$

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

$$0$$

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

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

$$0$$

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;
```

$$2\, Id - 3\, e2we3 + e1 + e4we5we6 = 2 + e1 - 3\, e2we3 + e4we5we6$$
$$2\, Id - 3\, e2we3 + e1 + e4we5we6 = 2 + e1 - 3\, e2we3 + e4we5we6$$

```
> 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**:

wedge(u,v) = convert(dwedge(convert(u,wedge_to_dwedge,F),convert(v,wedge_to_dwedge,F)),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
```

$$out2 := 6\,Id + 3\,e1 - 8\,e1we2we3 - 9\,e2we3 + 3\,e4we5we6 - 3\,e2we3we7 + 2\,e7 + e1we7 + 4\,e1we2we3we4we5we6 + e4we5we6we7$$

```
> out3:=wedge(u,v); #direct computation of the wedge product in undotted basis
```

$$out3 := 6\,Id + 3\,e1 - 8\,e1we2we3 - 9\,e2we3 + 3\,e4we5we6 - 3\,e2we3we7 + 2\,e7 + e1we7 + 4\,e1we2we3we4we5we6 + e4we5we6we7$$

```
> 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 'dottedbasis' 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
```

$$w\_dout1 := [0, g_{1,1}\, Id, (g_{1,2}+F_{1,2})\, Id, (g_{1,3}+F_{1,3})\, Id, g_{1,1}\, e2-(g_{1,2}+F_{1,2})\, e1, g_{1,1}\, e3-(g_{1,3}+F_{1,3})\, e1, (g_{1,2}+F_{1,2})\, e3-(g_{1,3}+F_{1,3})\, e2, g_{1,1}\, e2we3 - (g_{1,2}+F_{1,2})\, e1we3+(g_{1,3}+F_{1,3})\, e1we2+F_{2,3}\, g_{1,1}\, Id - F_{1,3}\, Id\, g_{1,2}+F_{1,2}\, Id\, g_{1,3}]$$

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
```

$$w\_dout := [0, g_{1,1}\, Id, Id\, g_{1,2}+F_{1,2}\, Id, Id\, g_{1,3}+F_{1,3}\, Id, g_{1,1}\, e2-e1\, g_{1,2}-e1\, F_{1,2}, g_{1,1}\, e3-e1\, g_{1,3}-e1\, F_{1,3}, e3\, g_{1,2}+F_{1,2}\, e3-e2\, g_{1,3}-F_{1,3}\, e2, g_{1,1}\, e2we3-e1we3\, g_{1,2}-e1we3\, F_{1,2}+e1we2\, g_{1,3}+e1we2\, F_{1,3}]$$

```
> map(simplify,w_dout-w_wout);
```

$$[0, 0, 0, 0, 0, 0, 0, 0]$$

```
>
```

This computation shows clearly the isomorphism 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
```

$$w\_out1 := -b\,(-g_{2,2}\,g_{1,3} + g_{2,3}\,g_{1,2})\,Id + b\,g_{2,2}\,e1we3 - b\,g_{1,2}\,e2we3 - b\,g_{2,3}\,e1we2 + a\,g_{2,3}\,e1 + a\,e1we2we3 - a\,g_{1,3}\,e2$$

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
```

$$d\_p1 := e1We2$$

$$d\_p2 := a\ e3 + b\ e2we3 + b\,F_{2,3}\,Id$$

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
```

$$d\_out1 := -b\,(g_{2,3}\,g_{1,2} + g_{2,3}\,F_{1,2} - g_{2,2}\,g_{1,3} - g_{2,2}\,F_{1,3} + F_{2,3}\,g_{1,2})\,Id + b\,g_{2,2}\,e1we3 - b\,g_{1,2}\,e2we3 - b\,g_{2,3}\,e1we2 + a\,(g_{2,3} + F_{2,3})\,e1 + a\,e1we2we3 - a\,(g_{1,3} + F_{1,3})\,e2 + F_{1,2}\,a\,e3$$

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
```

$$w\_out2 := -b\,Id\,g_{2,3}\,g_{1,2} + b\,Id\,g_{2,2}\,g_{1,3} + b\,g_{2,2}\,e1we3 - b\,g_{1,2}\,e2we3 - b\,g_{2,3}\,e1we2 + a\,g_{2,3}\,e1 + a\,e1we2we3 - a\,g_{1,3}\,e2$$

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);
```

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

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[h
elp]
```

[*&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(%);
```

$$Id\ \&\mathrm{dw}\ Id$$

$$Id$$

$$Id\ \&\mathrm{w}\ Id$$

$$Id$$

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

$$(Id\ \&\mathrm{dw}\ e1) + (e1\ \&\mathrm{dw}\ Id)$$

$$2\ e1$$

$$(Id\ \&\mathrm{w}\ e1) + (e1\ \&\mathrm{w}\ Id)$$

$$2\ e1$$

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

$$(Id\ \&\mathrm{dw}\ e2) + (e2\ \&\mathrm{dw}\ Id)$$

$$2\ e2$$

$$(Id\ \&\mathrm{w}\ e2) + (e2\ \&\mathrm{w}\ Id)$$

$$2\ e2$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$(Id \,\&\mathrm{dw}\, e1we2we3) + F_{2,3}\,(Id \,\&\mathrm{dw}\, e1) - F_{1,3}\,(Id \,\&\mathrm{dw}\, e2) + F_{1,2}\,(Id \,\&\mathrm{dw}\, e3)$$
$$+ (e1 \,\&\mathrm{dw}\, e2we3) + F_{2,3}\,(e1 \,\&\mathrm{dw}\, Id) - (e2 \,\&\mathrm{dw}\, e1we3) - F_{1,3}\,(e2 \,\&\mathrm{dw}\, Id)$$
$$+ (e1we2 \,\&\mathrm{dw}\, e3) + (e3 \,\&\mathrm{dw}\, e1we2) + F_{1,2}\,(e3 \,\&\mathrm{dw}\, Id) - (e1we3 \,\&\mathrm{dw}\, e2)$$
$$+ (e2we3 \,\&\mathrm{dw}\, e1) + (e1we2we3 \,\&\mathrm{dw}\, Id)$$
$$6\, F_{1,2}\, e3 + 8\, e1we2we3 - 6\, F_{1,3}\, e2 + 6\, F_{2,3}\, e1$$

$$(Id \,\&\mathrm{w}\, e1we2we3) + F_{2,3}\,(Id \,\&\mathrm{w}\, e1) - F_{1,3}\,(Id \,\&\mathrm{w}\, e2) + F_{1,2}\,(Id \,\&\mathrm{w}\, e3)$$
$$+ (e1 \,\&\mathrm{w}\, e2we3) + F_{2,3}\,(e1 \,\&\mathrm{w}\, Id) - (e2 \,\&\mathrm{w}\, e1we3) - F_{1,3}\,(e2 \,\&\mathrm{w}\, Id) + (e1we2 \,\&\mathrm{w}\, e3)$$
$$+ (e3 \,\&\mathrm{w}\, e1we2) + F_{1,2}\,(e3 \,\&\mathrm{w}\, Id) - (e1we3 \,\&\mathrm{w}\, e2) + (e2we3 \,\&\mathrm{w}\, e1)$$
$$+ (e1we2we3 \,\&\mathrm{w}\, Id)$$
$$8\, e1we2we3 + 2\, F_{2,3}\, e1 - 2\, F_{1,3}\, e2 + 2\, F_{1,2}\, e3$$

It is of utmost importance, that in these calculations we find that the usual loop tangle mult \citc 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^(grade_of_x) for the dotted bi-vector e1we2:

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

$$d\_p1 := (Id \,\&\mathrm{t}\, e1we2) + 2\, F_{1,2}\,(Id \,\&\mathrm{t}\, Id) + (e1 \,\&\mathrm{t}\, e2) - (e2 \,\&\mathrm{t}\, e1) + (e1we2 \,\&\mathrm{t}\, Id)$$

```
> 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`

Last modified: December 20, 2007, 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 <= n <= 9
k - the k-th grade, 0<= k <= 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:-makeclialiases for more help.
- When n=0 then [Id] is returned.

Examples:

```
> restart:with(Clifford):with(Cliplus);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
```

```
clude &C and &C[K]. Type ?cliprod for help.
```

$[LCbig, RCbig, clibasis, clieval, cliexpand, climul, clirev, dottedcbasis, dwedge, makeclialiases]$

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);
```

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

Using optional argument in 'cliexpand', we can convert Grassmann basis to Clifford basis in Cl(K):

```
> map(cliexpand,Gbasis,K);
```

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

Using optional argument in 'cliexpand', we can convert Grassmann basis to Clifford basis in Cl(-K):

```
> map(cliexpand,Gbasis,-K);
```

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

```
>
```

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

Last modified: December 20, 2007, 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 in
clude &C and &C[K]. Type ?cliprod for help.
```

$$[LCbig, RCbig, clibasis, clieval, cliexpand, climul, clirev, dottedcbasis, dwedge, makeclialiases]$$

Example: Some computations with an arbitrary form B:

```
> p:=e1 &C e2;type(p,cliprod);clieval(p);
```

$$p := e1 \ \&C \ e2$$

$$true$$

$$e1we2 + B_{1,2}\, Id$$

```
> p:=`&C`[B](e1,e2);type(p,cliprod);clieval(p);
```

$$p := \&C_B(e1, e2)$$

$$true$$

$$e1we2 + B_{1,2}\, Id$$

```
> p:=`&C`[K](e1,e2);type(p,cliprod);clieval(p);
```

$$p := \&C_K(e1, e2)$$

$$true$$

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

```
> p:=`&C`[-K](e1,e2);type(p,cliprod);clieval(p);
```

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

$$true$$

$$e1we2 - K_{1,2}\, Id$$

```
> q:=`&C`[K](e1,e2,e3);clieval(q);
```

$$q := \&C_K(e1, e2, e3)$$

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

```
> q:=`&C`[-K](e1,e2,e3);clieval(q);
```

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

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

```
> q:=`&C`[K](e1,e2,Id);clieval(q);
```

$$q := \&C_K(e1, e2, Id)$$

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

```
> q:=`&C`[-K](e1,e2,Id);clieval(q);
```

$$q := \&C_{-K}(e1, e2, Id)$$

$$e1we2 - K_{1,2}\, Id$$

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

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

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

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

$$0$$

```
> 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)
```

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

```
> 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

Last modified: December 20, 2007, 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 in
clude &C and &C[K]. Type ?cliprod for help.
```

$$[LCbig, RCbig, clibasis, clieval, cliexpand, climul, clirev, dottedcbasis, dwedge, makeclialiases]$$

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)
```

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

```
> map(cliexpand,L,B); #the above basis expressed in terms of the
  Clifford basis in Cl(B)
```

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

```
> map(cliexpand,L,K); #the above basis expressed in terms of the Clifford basis in Cl(K)
```

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

```
> map(cliexpand,L,-K); #the above basis expressed in terms of the Clifford basis in Cl(-K)
```

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

```
> 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)
```

$$p1 := 1 - 2\, e2 + (e3\ \&\mathrm{C}\ e4) - B_{3,4}\, Id$$

$$p1 := 1 - 2\, e2 + \&C_B(e3, e4) - B_{3,4}\, Id$$

```
> 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)
```

$$[Id, e1, e2, e3, \&C_H(e1, e2) - H_{1,2}\, Id, \&C_H(e1, e3) - H_{1,3}\, Id, \&C_H(e2, e3) - H_{2,3}\, Id, \&C_H(e1, e2, e3) - H_{2,3}\, e1 + H_{1,3}\, e2 - H_{1,2}\, e3]$$

```
> 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
```

$$p1 := 1 - 2\, e2 + \&C_H(e3, e4) - H_{3,4}\, Id$$

```
> 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)
```

$$[Id, e1, e2, e3, \&C_{-H}(e1, e2) + H_{1,2}\,Id, \&C_{-H}(e1, e3) + H_{1,3}\,Id, \&C_{-H}(e2, e3) + H_{2,3}\,Id, \&C_{-H}(e1, e2, e3) + H_{2,3}\,e1 - H_{1,3}\,e2 + H_{1,2}\,e3]$$

```
> 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
```

$$p1 := 1 - 2\,e2 + \&C_{-H}(e3, e4) + H_{3,4}\,Id$$

```
> 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/cliprod`, Clifford:-`type/clibasmon`, Cliplus:-clieval, Cliplus:-clibasis, Clifford:-makealiases

Last modified: December 20, 2007, 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,clibasmo
n);
```

$$e1\ \&C\ e2, false$$

$$2\,(e1\ \&C\ e2), false$$

$$2\,(e1\ \&C\ e2) + 3\ \&C(e1, e3, e4), false$$

```
> p1,type(p1,climon);p2,type(p2,climon);p3,type(p3,climon);
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &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,clipol
ynom);
```

$$e1\ \&C\ e2, false$$

$$2\,(e1\ \&C\ e2), false$$

$$2\,(e1\ \&C\ e2) + 3\ \&C(e1, e3, e4), true$$

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);
```

$$p1 := \&C_B(e1, e2)$$

$$p2 := 2\ \&C_B(e1, e2)$$

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

$$p4 := \&C_{-B}(e1, e2)$$

```
> p1,type(p1,cliprod);p2,type(p2,cliprod);
p3,type(p3,cliprod);p4,type(p4,cliprod);
```

$$\&C_B(e1, e2), true$$

$$2\ \&C_B(e1, e2), false$$

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

$$\&C_{-B}(e1, e2), true$$

```
> p1,type(p1,clibasmon);p2,type(p2,clibasmon);
p3,type(p3,clibasmon);p4,type(p4,clibasmon);
```

$$\&C_B(e1, e2), false$$

$$2\ \&C_B(e1, e2), false$$

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

$$\&C_{-B}(e1, e2), false$$

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,typ
  e(p4,climon);
  p1,type(p1,clipolynom);p2,type(p2,clipolynom);p3,type(p3,clipol
  ynom);
```

$$\_warnings\_flag := false$$

$$\&C_B(e1, e2), false$$

$$2\ \&C_B(e1, e2), true$$

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

$$\&C_{-B}(e1, e2), false$$

$$\&C_B(e1, e2), false$$

$$2\ \&C_B(e1, e2), false$$

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

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`
```

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

```
> cmul(p1,p1);cmul(p1,p2);cmul[-B](p4,p4);
```

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

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

$$-Id\ B_{2,2}\ B_{1,1} - B_{2,1}\ \&C_{-B}(e1, e2) - B_{1,2}\ \&C_{-B}(e1, e2)$$

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);
```

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

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

```
> &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
```

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

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

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

```
> 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!
```

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

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

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

```
> &c[B](p1,p1,p1); #short for of the Clifford product can be used
  as well
```

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

```
>
```

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);
```

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

```
>
```

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]);
```

$$B:=\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}$$

```
> 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);
```

$$2\ \&C_B(e1, e2) + 6\ B_{2,3}\ B_{1,1}\ e2 + 6\ B_{1,3}\ B_{2,2}\ e1 - 6\ B_{1,2}\ B_{3,2}\ e1 + 6\ e1\ B_{3,1}\ B_{2,2} - 6\ e1\ B_{3,2}\ B_{2,1} + 6\ e2\ B_{3,2}\ B_{1,1} - 6\ e3\ B_{2,2}\ B_{1,1} + 4\ B_{2,1}\ \&C_B(e1, e2) + 4\ B_{1,2}\ \&C_B(e1, e2) - 6\ B_{2,3}\ B_{1,2}\ e1 - 4\ Id\ B_{2,2}\ B_{1,1} - 6\ B_{2,1}\ B_{2,3}\ e1 + 6\ B_{2,1}\ \&C_B(e1, e2, e3) + 6\ B_{1,2}\ \&C_B(e1, e2, e3)$$

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

```
> p1 &c p2;
```

$$2\ \&C_B(e1, e2) + 6\ B_{2,3}\ B_{1,1}\ e2 + 6\ B_{1,3}\ B_{2,2}\ e1 - 6\ B_{1,2}\ B_{3,2}\ e1 + 6\ e1\ B_{3,1}\ B_{2,2}$$

$$- 6\, e1\, B_{3,2}\, B_{2,1} + 6\, e2\, B_{3,2}\, B_{1,1} - 6\, e3\, B_{2,2}\, B_{1,1} + 4\, B_{2,1}\, \&C_B(e1, e2) + 4\, B_{1,2}\, \&C_B(e1, e2)$$
$$- 6\, B_{2,3}\, B_{1,2}\, e1 - 4\, Id\, B_{2,2}\, B_{1,1} - 6\, B_{2,1}\, B_{2,3}\, e1 + 6\, B_{2,1}\, \&C_B(e1, e2, e3)$$
$$+ 6\, B_{1,2}\, \&C_B(e1, e2, e3)$$

One use 'clicollect' to collect the above output:

```
> clicollect(%);
```

$$-4\, Id\, B_{2,2}\, B_{1,1} - 6\, (-B_{3,1}\, B_{2,2} + B_{3,2}\, B_{2,1} - B_{1,3}\, B_{2,2} + B_{2,1}\, B_{2,3} + B_{2,3}\, B_{1,2} + B_{1,2}\, B_{3,2})\, e1$$
$$+ 6\, B_{1,1}\, (B_{2,3} + B_{3,2})\, e2 - 6\, e3\, B_{2,2}\, B_{1,1} + 2\, (2\, B_{1,2} + 2\, B_{2,1} + 1)\, \&C_B(e1, e2)$$
$$+ 6\, (B_{2,1} + B_{1,2})\, \&C_B(e1, e2, e3)$$

Mixed basis elements are also allowed:

```
> p2:=p1+2*e2we3;
```

$$p2 := Id + 2\, (e1\ \&\mathrm{C}\ e2) + 3\, \&\mathrm{C}(e1, e2, e3) + 2\, e2we3$$

```
> cmul(p2,p2);
```

$$Id + 4\, \&C_B(e1, e2) - 9\, Id\, B_{3,3}\, B_{2,2}\, B_{1,1} + 6\, B_{2,3}\, B_{1,1}\, e2 + 6\, B_{1,3}\, B_{2,2}\, e1 - 6\, B_{1,2}\, B_{3,2}\, e1$$
$$+ 6\, e1\, B_{3,1}\, B_{2,2} - 6\, e1\, B_{3,2}\, B_{2,1} + 6\, e2\, B_{3,2}\, B_{1,1} - 12\, e3\, B_{2,2}\, B_{1,1} + 6\, e2\, B_{3,3}\, B_{2,1}$$
$$- 6\, B_{2,3}\, B_{2,1}\, e3 - 4\, B_{2,3}\, Id - 4\, B_{2,3}\, \&C_B(e1, e2) + 4\, B_{3,2}\, \&C_B(e1, e2) - 4\, B_{3,2}\, B_{1,2}\, Id$$
$$+ 4\, B_{2,1}\, \&C_B(e1, e2) + 4\, B_{1,2}\, \&C_B(e1, e2) + 9\, B_{3,2}\, B_{1,1}\, \&C_B(e2, e3) - 6\, B_{2,3}\, B_{1,2}\, e1$$
$$+ 9\, B_{3,1}\, B_{2,2}\, \&C_B(e1, e3) - 9\, B_{3,2}\, B_{2,1}\, \&C_B(e1, e3) - 9\, B_{2,1}\, B_{2,3}\, \&C_B(e1, e3)$$
$$+ 9\, B_{1,3}\, B_{2,2}\, \&C_B(e1, e3) - 9\, B_{2,3}\, B_{1,2}\, \&C_B(e1, e3) - 9\, B_{1,2}\, B_{3,2}\, \&C_B(e1, e3)$$
$$- 4\, Id\, B_{2,2}\, B_{1,1} + 4\, B_{3,2}\, \&C_B(e2, e3) + 4\, B_{2,1}\, \&C_B(e2, e3) + 4\, B_{1,2}\, \&C_B(e2, e3)$$
$$- 4\, B_{2,3}\, \&C_B(e2, e3) + 4\, B_{2,3}^{\ 2}\, Id + 6\, \&C_B(e1, e2, e3) + 4\, \&C_B(e2, e3) - 4\, B_{2,1}\, B_{2,3}\, Id$$
$$+ 9\, B_{2,3}\, B_{1,1}\, \&C_B(e2, e3) - 6\, B_{2,1}\, B_{2,3}\, e1 - 4\, Id\, B_{3,3}\, B_{2,2} + 4\, Id\, B_{1,3}\, B_{2,2} - 4\, Id\, B_{2,3}\, B_{1,2}$$
$$- 6\, B_{1,2}\, B_{3,2}\, e3 + 6\, B_{1,2}\, B_{3,3}\, e2 + 6\, e3\, B_{1,3}\, B_{2,2} - 6\, e3\, B_{2,3}\, B_{1,2} - 12\, e1\, B_{3,3}\, B_{2,2}$$
$$+ 4\, Id\, B_{3,1}\, B_{2,2} - 4\, Id\, B_{3,2}\, B_{2,1} + 6\, e3\, B_{3,1}\, B_{2,2} - 6\, e3\, B_{3,2}\, B_{2,1} + 12\, B_{2,1}\, \&C_B(e1, e2, e3)$$
$$+ 12\, B_{1,2}\, \&C_B(e1, e2, e3) + 12\, B_{3,2}\, \&C_B(e1, e2, e3) + 9\, B_{3,3}\, B_{2,1}\, \&C_B(e1, e2)$$
$$+ 9\, B_{3,3}\, B_{1,2}\, \&C_B(e1, e2)$$

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);
```

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

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

```
> Cliplus:-clieval(p2);
```

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

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

```
> clicollect(cmul(p2,p1));
```

$$-(-1 + 9\, B_{3,3}\, B_{2,2}\, B_{1,1} + 4\, B_{2,3}\, B_{2,1} + 4\, B_{2,2}\, B_{1,1} - 4\, B_{1,3}\, B_{2,2} + 4\, B_{2,3}\, B_{1,2} + 2\, B_{2,3}$$

$$+4\,B_{1,2}\,B_{3,2}+4\,B_{3,2}\,B_{2,1}-4\,B_{3,1}\,B_{2,2})\,Id$$
$$-6\,(B_{2,3}\,B_{1,2}+B_{3,2}\,B_{2,1}+B_{1,2}\,B_{3,2}-B_{3,1}\,B_{2,2}-B_{1,3}\,B_{2,2}+B_{3,3}\,B_{2,2}+B_{2,3}\,B_{2,1})\,e1$$
$$+6\,(B_{3,3}\,B_{2,1}+B_{3,2}\,B_{1,1}+B_{3,3}\,B_{1,2}+B_{2,3}\,B_{1,1})\,e2$$
$$-6\,(2\,B_{2,2}\,B_{1,1}+B_{2,3}\,B_{1,2}+B_{1,2}\,B_{3,2}+B_{2,3}\,B_{2,1}-B_{1,3}\,B_{2,2}-B_{3,1}\,B_{2,2}+B_{3,2}\,B_{2,1})\,e3$$
$$+6\,(B_{3,2}+2\,B_{2,1}+1+2\,B_{1,2})\,\&C_B(e1,e2,e3)$$
$$+(9\,B_{3,2}\,B_{1,1}+4\,B_{1,2}+4\,B_{2,1}+2+9\,B_{2,3}\,B_{1,1})\,\&C_B(e2,e3)-$$
$$(9\,B_{2,3}\,B_{2,1}+9\,B_{3,2}\,B_{2,1}-9\,B_{1,3}\,B_{2,2}+9\,B_{2,3}\,B_{1,2}+9\,B_{1,2}\,B_{3,2}-9\,B_{3,1}\,B_{2,2}+4\,B_{2,2})$$
$$\&C_B(e1,e3)+(9\,B_{3,3}\,B_{2,1}+4\,B_{1,2}+4+4\,B_{2,1}+9\,B_{3,3}\,B_{1,2}+4\,B_{3,2})\,\&C_B(e1,e2)$$

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);
```

$$\begin{bmatrix} Id, & e1, & e2, & e3, & \&C_B(e1,e2), & \&C_B(e1,e3), & \&C_B(e2,e3), & \&C_B(e1,e2,e3) \\ e1, & Id, & e1we2, & e1we3, & e2, & e3, & \&C_B(e1,e2,e3), & \&C_B(e2,e3) \\ e2, & -e1we2, & Id, & e2we3, & -e1, & -\&C_B(e1,e2,e3), & e3, & -\&C_B(e1,e3) \\ e3, & -e1we3, & -e2we3, & Id, & \&C_B(e1,e2,e3), & -e1, & -e2, & \&C_B(e1,e2) \\ \&C_B(e1,e2), & -e2, & e1, & \&C_B(e1,e2,e3), & -Id, & -\&C_B(e2,e3), & \&C_B(e1,e3), & -e3 \\ \&C_B(e1,e3), & -e3, & -\&C_B(e1,e2,e3), & e1, & \&C_B(e2,e3), & -Id, & -\&C_B(e1,e2), & e2 \\ \&C_B(e2,e3), & \&C_B(e1,e2,e3), & -e3, & e2, & -\&C_B(e1,e3), & \&C_B(e1,e2), & -Id, & -e1 \\ \&C_B(e1,e2,e3), & \&C_B(e2,e3), & -\&C_B(e1,e3), & \&C_B(e1,e2), & -e3, & e2, & -e1, & -Id \end{bmatrix}$$

```
> printf("Worksheet took %f seconds to compute on Pentium M 2.13
  GHz 2GB RAM machine with Win XP Professional\n",time()-bench);
Worksheet took 2.856000 seconds to compute on Pentium M 2.13 GHz 2GB RAM machin
e with Win XP Professional
>
```

See Also: Clifford:-cmul, Clifford:-clicollect, Clifford:-`type/cliprod`, Cliplus:-clieval, Cliplus:-cliexpand, Cliplus:-clibasis

Last modified: December 20, 2007, 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 e1 &C e2 &C e3, it will return e3 &C e2 &C e1. 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 in
clude &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);
```

$$\text{"clirev(p)"} = \&C(e3, e2, e1)$$
$$\text{"reversion(p)"} = \&C(e3, e2, e1)$$
$$\text{"clirev(p1)"} = e2\ \&C\ e1$$
$$\text{"reversion(p1)"} = e2\ \&C\ e1$$
$$\text{"clirev(p2)"} = e2we3\ \&C\ e1$$
$$\text{"reversion(p2)"} = e2we3\ \&C\ e1$$

```
> pp:=`&C`[B](e1,e2,e3);
  p11:=`&C`[K](e1,e2,e3);
  p22:=`&C`[-K](e1,e2,e3);
```

$$pp := \&C_B(e1, e2, e3)$$
$$p11 := \&C_K(e1, e2, e3)$$
$$p22 := \&C_{-K}(e1, e2, e3)$$

```
> 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:-clirev) optional (or default B) parameter in clirev 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=clirev) is defined at the time when the package 'Cliplus' is loaded, user can enter name 'reversion', yet maple returns always its macro name 'clirev'. For example,

```
> 'reversion';
```

$$Cliplus{:}\text{-}clirev$$

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
```

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

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
```

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

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
```

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

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:-clirev) optional (or default B) parameter in clirev 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
```

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

p1 and p3 now should be the same:

```
> p1 - p3;
```

$$0$$

Reversion is always an anti-automoprhism 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);
```

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

We want to show that

reversion(cmul(p1,p2)) = cmul(reversion(p2),reversion(p1)).

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

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

```
> 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);
```

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

```
> 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);
```

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

We want to show that

reversion(cmul(p1,p2)) = cmul(reversion(p2),reversion(p1)).

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

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

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

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

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

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

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

$$Revp1p2 := -7\,K_{2,3}\,e1 - 7\,K_{3,2}\,e1 + K_{1,2}\,e3 + 2\,e3 + 6\,K_{1,3}\,K_{2,1}\,e2 + 8\,K_{2,1}\,K_{2,3}\,Id + 8\,K_{3,2}\,K_{1,2}\,Id - 2\,\&C_K(e1, e3) + 6\,K_{1,2}\,\&C_K(e1, e2, e3) + 6\,K_{2,1}\,\&C_K(e1, e2, e3) + 3\,K_{1,1}\,K_{2,3}\,Id - 6\,e3\,K_{2,1}^{2} + 2\,K_{3,3}\,Id + 4\,K_{3,3}\,e2 + 8\,K_{2,3}\,K_{1,2}\,Id - 3\,K_{3,3}\,K_{1,2}\,Id - 6\,K_{2,3}\,K_{1,1}\,e2 - 3\,K_{3,1}\,\&C_K(e1, e2) - 8\,K_{2,3}\,\&C_K(e1, e2) - 3\,K_{1,3}\,\&C_K(e1, e2) - 8\,K_{3,2}\,\&C_K(e1, e2) + 3\,K_{3,3}\,\&C_K(e1, e2) - 6\,K_{3,2}\,K_{1,1}\,e2 + 6\,K_{2,2}\,K_{1,1}\,e3 - 12\,e3\,K_{2,1}\,K_{1,2} + 6\,K_{3,1}\,e2\,K_{2,1} - 6\,K_{1,3}\,K_{2,2}\,e1 + 8\,K_{3,2}\,K_{2,1}\,Id + 3\,K_{1,3}\,e2 - 6\,K_{3,1}\,K_{2,2}\,e1 + 8\,K_{2,2}\,\&C_K(e1, e3) + 3\,K_{2,1}\,\&C_K(e1, e3) + 3\,K_{1,2}\,\&C_K(e1, e3) + 3\,K_{3,1}\,e2 + K_{2,1}\,e3 + 4\,K_{2,3}\,Id + 6\,K_{1,2}\,K_{1,3}\,e2 + 3\,K_{3,2}\,K_{1,1}\,Id - 3\,K_{3,3}\,Id\,K_{2,1} + 6\,K_{3,1}\,K_{1,2}\,e2 + 4\,K_{3,2}\,Id - 4\,\&C_K(e2, e3) + 3\,\&C_K(e1, e2, e3) - 3\,K_{1,1}\,\&C_K(e2, e3) - 8\,K_{2,1}\,\&C_K(e2, e3) - 8\,K_{1,2}\,\&C_K(e2, e3) - 6\,K_{1,2}^{2}\,e3$$

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

$$0$$

Another short example:

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

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

```
> reversion(p1);
```

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

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

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

Mixed input can be used in 'reversion':

```
> reversion(p2);
```

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

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

```
> cliexpand(%);
```

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

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

```
> clieval(%);
```

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

```
> printf("Worksheet took %f seconds to compute on Pentium M 2.13
  GHz 2GB RAM machine with Win XP Professional\n",time()-bench);
Worksheet took 1.810000 seconds to compute on Pentium M 2.13 GHz 2GB RAM machin
e with Win XP Professional
>
```

See Also: Clifford:-cmul, Clifford:-clicollect, Clifford:-`type/cliprod`, Cliplus:-clieval, Cliplus:-cliexpand, Clifford:-reversion

Last modified: December 20, 2007, 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 it to assign such expressions to a descriptive lhs, see below.

Examples:

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

$[LCbig, RCbig, clibasis, clieval, cliexpand, climul, clirev, dottedcbasis, dwedge, makeclialiases]$

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);
```

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

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

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

```
> 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);
```

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

```
> 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 *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 smilarly 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

Last modified: December 20, 2007, RA/BF.

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:

- p1,p2,...,pn - 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:

- c1 : 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 relay 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:** Till 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, 2002.
[2] Ablamowicz, R. and Bertfried Fauser: “On the decomposition of Clifford algebras of arbitrary bilinear form”, Rafal Ablamowicz and Bertfried Fauser, in “Clifford Algebras and their Applications in Mathematical Physics”, Eds. Rafal Ablamowicz and Bertfried Fauser, Vol. 1: Algebra and Physics, Birkhäuser, Boston, 2000, pages 341--366 (see also http://www.birkhauser.com/cgi-win/isbn/0-8176-4182-3).
[3] “Clifford Algebras and their Applications in Mathematical Physics”, Eds. Rafal Ablamowicz and Bertfried Fauser, Vol. 1: Algebra and Physics, Birkhäuser, Boston, 2000, (ISBN 0-8176-4182-3) (for more information go to http://math.tntech.edu/rafal/mexico/mexico.html).
[4] Ablamowicz, R. and P. Lounesto: “On Clifford algebras of a bilinear form with an antisymmetric part,” with P. Lounesto, in “Clifford Algebras with Numeric and Symbolic computations”, Eds. R. Ablamowicz, P. Lounesto, and J. Parra, Birkhäuser, Boston, 1996, pages 167-188.
[5] “Clifford Algebras with Numeric and Symbolic Computations”, Eds. Rafal Ablamowicz, Pertti Lounesto, and J. Parra, Birkhäuser, Boston, 1996 (ISBN 0-8176-3907-1).

Examples:

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

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

$[LCbig, RCbig, clibasis, clieval, cliexpand, climul, clirev, dottedcbasis, dwedge, makeclialiases]$

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);
```

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

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

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

$$+ 2\, e1we3we4 + e1we4$$

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

```
>
```

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_d
  wedge,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),wedg
  e_to_dwedge,F));
```

$$e1we2we3we4 + F_{2,3}\, F_{1,4}\, Id - F_{1,3}\, F_{2,4}\, Id + F_{1,2}\, F_{3,4}\, Id - F_{2,4}\, e1we3 + F_{1,4}\, e2we3 + F_{3,4}\, e1we2 + F_{2,3}\, e1we4 - F_{1,3}\, e2we4 + F_{1,2}\, e3we4 = e1we2we3we4 + F_{2,3}\, F_{1,4}\, Id - F_{1,3}\, F_{2,4}\, Id + F_{1,2}\, F_{3,4}\, Id - F_{2,4}\, e1we3 + F_{1,4}\, e2we3 + F_{3,4}\, e1we2 + F_{2,3}\, e1we4 - F_{1,3}\, e2we4 + F_{1,2}\, e3we4$$

```
> ##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));%%-%;
```

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

$$- F_{1,3}\, e2we4 + F_{1,2}\, e3we4$$

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

$$0$$

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

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

$$0$$

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;
```

$$2\, Id - 3\, e2we3 + e1 + e4we5we6 = 2 + e1 - 3\, e2we3 + e4we5we6$$
$$2\, Id - 3\, e2we3 + e1 + e4we5we6 = 2 + e1 - 3\, e2we3 + e4we5we6$$

```
> 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**:

wedge(u,v) = convert(dwedge(convert(u,wedge_to_dwedge,F),convert(v,wedge_to_dwedge,F)),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
```

$$out2 := 6\,Id + 3\,e1 - 9\,e2we3 + 3\,e4we5we6 + 2\,e7 + e4we5we6we7 - 8\,e1we2we3 + 4\,e1we2we3we4we5we6 - 3\,e2we3we7 + e1we7$$

```
> out3:=wedge(u,v); #direct computation of the wedge product in undotted basis
```

$$out3 := 6\,Id + 3\,e1 - 9\,e2we3 + 3\,e4we5we6 + 2\,e7 + e4we5we6we7 - 8\,e1we2we3 + 4\,e1we2we3we4we5we6 - 3\,e2we3we7 + e1we7$$

```
> 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 smilarly 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 'dottedbasis' 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
```

$$w\_dout1 := [0, g_{1,1}\, Id, (g_{1,2}+F_{1,2})\, Id, (g_{1,3}+F_{1,3})\, Id, g_{1,1}\, e2-(g_{1,2}+F_{1,2})\, e1, g_{1,1}\, e3-(g_{1,3}+F_{1,3})\, e1, (g_{1,2}+F_{1,2})\, e3-(g_{1,3}+F_{1,3})\, e2, g_{1,1}\, e2we3 - (g_{1,2}+F_{1,2})\, e1we3+(g_{1,3}+F_{1,3})\, e1we2+F_{2,3}\, g_{1,1}\, Id - F_{1,3}\, Id\, g_{1,2}+F_{1,2}\, Id\, g_{1,3}]$$

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
```

$$w\_dout := [0, g_{1,1}\, Id, Id\, g_{1,2}+F_{1,2}\, Id, Id\, g_{1,3}+F_{1,3}\, Id, g_{1,1}\, e2-e1\, g_{1,2}-e1\, F_{1,2}, g_{1,1}\, e3-e1\, g_{1,3}-e1\, F_{1,3}, e3\, g_{1,2}+F_{1,2}\, e3-e2\, g_{1,3}-F_{1,3}\, e2, g_{1,1}\, e2we3-e1we3\, g_{1,2}-e1we3\, F_{1,2}+e1we2\, g_{1,3}+e1we2\, F_{1,3}]$$

```
> map(simplify,w_dout-w_wout);
```

$$[0, 0, 0, 0, 0, 0, 0, 0]$$

```
>
```

This computation shows clearly the isomorphism 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
```

$$w\_out1 := -b\,(-g_{2,2}\,g_{1,3} + g_{2,3}\,g_{1,2})\,Id + b\,g_{2,2}\,e1we3 - b\,g_{1,2}\,e2we3 - b\,g_{2,3}\,e1we2 + a\,g_{2,3}\,e1 - a\,g_{1,3}\,e2 + a\,e1we2we3$$

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
```

$$d\_p1 := e1We2$$

$$d\_p2 := a\ e3 + b\ e2we3 + b\,F_{2,3}\,Id$$

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
```

$$d\_out1 := -b\,(g_{2,3}\,g_{1,2} + g_{2,3}\,F_{1,2} - g_{2,2}\,g_{1,3} - g_{2,2}\,F_{1,3} + F_{2,3}\,g_{1,2})\,Id + b\,g_{2,2}\,e1we3 - b\,g_{1,2}\,e2we3 - b\,g_{2,3}\,e1we2 + a\,(g_{2,3} + F_{2,3})\,e1 - a\,(g_{1,3} + F_{1,3})\,e2 + F_{1,2}\,a\,e3 + a\,e1we2we3$$

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
```

$$w\_out2 := -b\,Id\,g_{2,3}\,g_{1,2} + b\,Id\,g_{2,2}\,g_{1,3} + b\,g_{2,2}\,e1we3 - b\,g_{1,2}\,e2we3 - b\,g_{2,3}\,e1we2 + a\,g_{2,3}\,e1 - a\,g_{1,3}\,e2 + a\,e1we2we3$$

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);
```

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

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[h
elp]
```

[*&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(%);
```

$$Id\ \&\mathrm{dw}\ Id$$

$$Id$$

$$Id\ \&\mathrm{w}\ Id$$

$$Id$$

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

$$(Id\ \&\mathrm{dw}\ e1) + (e1\ \&\mathrm{dw}\ Id)$$

$$2\ e1$$

$$(Id\ \&\mathrm{w}\ e1) + (e1\ \&\mathrm{w}\ Id)$$

$$2\ e1$$

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

$$(Id\ \&\mathrm{dw}\ e2) + (e2\ \&\mathrm{dw}\ Id)$$

$$2\ e2$$

$$(Id\ \&\mathrm{w}\ e2) + (e2\ \&\mathrm{w}\ Id)$$

$$2\ e2$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$(Id \,\&\mathrm{dw}\, e1we2we3) + F_{2,3}\,(Id \,\&\mathrm{dw}\, e1) - F_{1,3}\,(Id \,\&\mathrm{dw}\, e2) + F_{1,2}\,(Id \,\&\mathrm{dw}\, e3)$$
$$+ (e1 \,\&\mathrm{dw}\, e2we3) + F_{2,3}\,(e1 \,\&\mathrm{dw}\, Id) - (e2 \,\&\mathrm{dw}\, e1we3) - F_{1,3}\,(e2 \,\&\mathrm{dw}\, Id)$$
$$+ (e1we2 \,\&\mathrm{dw}\, e3) + (e3 \,\&\mathrm{dw}\, e1we2) + F_{1,2}\,(e3 \,\&\mathrm{dw}\, Id) - (e1we3 \,\&\mathrm{dw}\, e2)$$
$$+ (e2we3 \,\&\mathrm{dw}\, e1) + (e1we2we3 \,\&\mathrm{dw}\, Id)$$
$$6\, F_{2,3}\, e1 + 6\, F_{1,2}\, e3 - 6\, F_{1,3}\, e2 + 8\, e1we2we3$$

$$(Id \,\&\mathrm{w}\, e1we2we3) + F_{2,3}\,(Id \,\&\mathrm{w}\, e1) - F_{1,3}\,(Id \,\&\mathrm{w}\, e2) + F_{1,2}\,(Id \,\&\mathrm{w}\, e3)$$
$$+ (e1 \,\&\mathrm{w}\, e2we3) + F_{2,3}\,(e1 \,\&\mathrm{w}\, Id) - (e2 \,\&\mathrm{w}\, e1we3) - F_{1,3}\,(e2 \,\&\mathrm{w}\, Id) + (e1we2 \,\&\mathrm{w}\, e3)$$
$$+ (e3 \,\&\mathrm{w}\, e1we2) + F_{1,2}\,(e3 \,\&\mathrm{w}\, Id) - (e1we3 \,\&\mathrm{w}\, e2) + (e2we3 \,\&\mathrm{w}\, e1)$$
$$+ (e1we2we3 \,\&\mathrm{w}\, Id)$$
$$8\, e1we2we3 + 2\, F_{2,3}\, e1 - 2\, F_{1,3}\, e2 + 2\, F_{1,2}\, e3$$

It is of utmost importance, that in these calculations we find that the usual loop tangle mult \citc 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^(grade_of_x) for the dotted bi-vector e1we2:

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

$$d\_p1 := (Id \,\&\mathrm{t}\, e1we2) + 2\, F_{1,2}\,(Id \,\&\mathrm{t}\, Id) + (e1 \,\&\mathrm{t}\, e2) - (e2 \,\&\mathrm{t}\, e1) + (e1we2 \,\&\mathrm{t}\, Id)$$

```
> 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`

Last modified: December 20, 2007, 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 relay on different filtrations. The dotted wedge product and the undotted one are related by the process of cliffordization which is used in CLIFFORD internally to compute the Clifford product in Cl(V,B). However, the cliffordization is performed in this case by an antisymmetric bilinear form B=F, say F(x,y)=-F(y,x), where x and y are 1-vectors in V.
- **NOTE:** Till now both types of algebras are expanded (formally) over the same basis Grassmann monomials which, according to CLIFFORD's convention, are written as 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, 2002.
[2] Ablamowicz, R. and Bertfried Fauser: “On the decomposition of Clifford algebras of arbitrary bilinear form”, Rafal Ablamowicz and Bertfried Fauser, in “Clifford Algebras and their Applications in Mathematical Physics”, Eds. Rafal Ablamowicz and Bertfried Fauser, Vol. 1: Algebra and Physics, Birkhäuser, Boston, 2000, pages 341--366 (see also http://www.birkhauser.com/cgi-win/isbn/0-8176-4182-3).
[3] “Clifford Algebras and their Applications in Mathematical Physics”, Eds. Rafal Ablamowicz and Bertfried Fauser, Vol. 1: Algebra and Physics, Birkhäuser, Boston, 2000, (ISBN 0-8176-4182-3) (for more information go to http://math.tntech.edu/rafal/mexico/mexico.html).
[4] Ablamowicz, R. and P. Lounesto: “On Clifford algebras of a bilinear form with an antisymmetric part,” with P. Lounesto, in “Clifford Algebras with Numeric and Symbolic computations”, Eds. R. Ablamowicz, P. Lounesto, and J. Parra, Birkhäuser, Boston, 1996, pages 167-188.
[5] “Clifford Algebras with Numeric and Symbolic Computations”, Eds. Rafal Ablamowicz, Pertti Lounesto, and J. Parra, Birkhäuser, Boston, 1996 (ISBN 0-8176-3907-1).

Examples:

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

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

$$[LCbig, RCbig, clibasis, clieval, cliexpand, climul, clirev, dottedcbasis, dwedge, makeclialiases]$$

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

```
> convert(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 *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 smilarly for other elements:

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

$$e1We2, e1We3, e2We3, e1We2We3$$

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

```
> d_bas;
```

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

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

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

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

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

$$[Id, e1We2, e1We3, e2We3]$$

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

$$[e1We2, e1We3, e2We3]$$

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

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

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

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

$$[Id, e1We2, e1We3, e2We3]$$

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

$$[e1We2, e1We3, e2We3]$$

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

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

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

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

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

$$-e1we2$$

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

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

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

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);
```

$$reversed\_e1We2 := -F_{1,2}\, Id - e1we2$$

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

Last modified: December 20, 2007, 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/clipolynom`. For completness, 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/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: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 _| v in Cl(V,B).

```
> u:=2*e1we3+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/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

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

Ir is possible to use 'LC' with an optional parameter of type 'name':

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

$$4\,K_{3,1}\,K_{1,2}\,e3 - 4\,K_{3,1}\,K_{1,3}\,e2 - 4\,K_{3,2}\,K_{1,1}\,e3 + 4\,K_{3,2}\,K_{1,3}\,e1 + 4\,K_{3,3}\,K_{1,1}\,e2 - 4\,K_{3,3}\,K_{1,2}\,e1 + 2\,K_{3,1}\,K_{1,2}\,Id - 2\,K_{3,2}\,K_{1,1}\,Id + 2\,K_{4,1}\,e2we3 - 2\,K_{4,2}\,e1we3 + 2\,K_{4,3}\,e1we2 + K_{4,1}\,e2 - K_{4,2}\,e1 + 2\,e1we2we3 + e1we2$$

For example, it is well known that the Clifford product cmul(x,v) = x _| 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);
```

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

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

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

```
> out1-out2;
```

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

```
>
```

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

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

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

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

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

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

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

$$-2\,H_{2,1}\,e2+2\,H_{2,2}\,e1+e1we2we3+H_{3,1}\,e2-H_{3,2}\,e1$$

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

$$0$$

```
> printf("Worksheet took %f seconds to compute on Pentium M 2.13
  GHz 2GB RAM machine with Win XP Professional\n",time()-bench);
Worksheet took 0.314000 seconds to compute on Pentium M 2.13 GHz 2GB RAM machin
e with Win XP Professional
>
```

Example 2: 'LC' 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 '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 in
clude &C and &C[K]. Type ?cliprod for help.
```

$$[LCbig, RCbig, clibasis, clieval, cliexpand, climul, clirev, dottedcbasis, dwedge, makeclialiases]$$

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

$$v := 2\,e1we2we3+e1we2$$

```
> '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);
```

$$out3 := 4\,B_{3,1}\,B_{1,2}\,e3-4\,B_{3,1}\,B_{1,3}\,e2-4\,B_{3,2}\,B_{1,1}\,e3+4\,B_{3,2}\,B_{1,3}\,e1+4\,B_{3,3}\,B_{1,1}\,e2$$

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

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

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

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

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

```
> out5:=clieval(out3);
```

$$out5 := 4\, B_{3,1}\, B_{1,2}\, e3 - 4\, B_{3,1}\, B_{1,3}\, e2 - 4\, B_{3,2}\, B_{1,1}\, e3 + 4\, B_{3,2}\, B_{1,3}\, e1 + 4\, B_{3,3}\, B_{1,1}\, e2$$
$$- 4\, B_{3,3}\, B_{1,2}\, e1 + 2\, B_{3,1}\, B_{1,2}\, Id - 2\, B_{3,2}\, B_{1,1}\, Id + 2\, B_{4,1}\, e2we3 - 2\, B_{4,2}\, e1we3$$
$$+ 2\, B_{4,3}\, e1we2 + B_{4,1}\, e2 - B_{4,2}\, e1 + 2\, e1we2we3 + e1we2$$

```
> out5-out4;
```

$$0$$

```
>
```

Now let's see the mixed input:

```
> out6:=LC(u,vv,B);
```

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

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

```
> clieval(out6)-out5;
```

$$0$$

```
> out7:=LC(uu,v,B);
```

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

```
> 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
```

$$K_{2,3}\,K_{1,4}\,Id - K_{2,4}\,K_{1,3}\,Id + K_{1,2}\,\&C_K(e3, e4)$$

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

$$K_{2,3}\,K_{1,4}\,Id - K_{2,4}\,K_{1,3}\,Id - K_{1,2}\,\&C_{-K}(e3, e4)$$

```
> printf("Worksheet took %f seconds to compute on Pentium M 2.13
  GHz 2GB RAM machine with Win XP Professional\n",time()-bench);
Worksheet took 0.609000 seconds to compute on Pentium M 2.13 GHz 2GB RAM machin
e with Win XP Professional
>
```

See Also: Clifford:-cmul, Clifford:-clicollect, Cliplus:-clieval, Cliplus:-cliexpand, Clifford:-makealiases, Clifford:-wedge

Last modified: December 20, 2007, RA/BF.

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

Calling Sequence:

makeclialiases(p);
makeclialiases[K](p);
makeclialiases(p,s);
makeclialiases[K](p,s);

Parameters:

p - positive integer such that 1<= p <= 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 in
clude &C and &C[K]. Type ?cliprod for help.
```

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

```
> type(e1 &C e2,cliprod);
  type(`&C`[K](e1,e2),cliprod);
  type(`&C`[-K](e1,e2),cliprod);
```

true

true

true

```
> makeclialiases(2);
```

```
makeclialiases[K](2);
makeclialiases[-K](2);
```

$$\text{alias}(e12 = e1 \,\&C\, e2, e21 = e2 \,\&C\, e1)$$

$$\text{alias}(e12 = \&C_K(e1, e2), e21 = \&C_K(e2, e1))$$

$$\text{alias}(e12 = \&C_{-K}(e1, e2), e21 = \&C_{-K}(e2, e1))$$

Command 'eval' is not built-in into 'makeclialiases' (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 in
clude &C and &C[K]. Type ?cliprod for help.
```

$$[LCbig, RCbig, clibasis, clieval, cliexpand, climul, clirev, dottedcbasis, dwedge, makeclialiases]$$

```
> makeclialiases(3,'ordered');
```

$$\text{alias}(e12 = e1 \,\&C\, e2, e13 = e1 \,\&C\, e3, e23 = e2 \,\&C\, e3, e123 = \&C(e1, e2, e3))$$

```
> eval(%);
```

$$e12, e13, e23, e123$$

```
> makeclialiases[X](3,'ordered');
```

$$\text{alias}(e12 = \&C_X(e1, e2), e13 = \&C_X(e1, e3), e23 = \&C_X(e2, e3), e123 = \&C_X(e1, e2, e3))$$

```
> eval(%);
```

$$e12, e13, e23, e123$$

```
> makeclialiases[-X](3,'ordered');
```

$$\text{alias}(e12 = \&C_{-X}(e1, e2), e13 = \&C_{-X}(e1, e3), e23 = \&C_{-X}(e2, e3), e123 = \&C_{-X}(e1, e2, e3))$$

```
> eval(%);
```

$$e12, e13, e23, e123$$

```
>
```

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

See Also: Cliplus:-clieval, Cliplus:-clibasis, Clifford:-makealiases

Last modified: December 20, 2007, 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 |_ 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 in
clude &C and &C[K]. Type ?cliprod for help.
```

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

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

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

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

For example, it is well known that the Clifford product cmul(v,x) = v |_ 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);
```

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

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

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

```
> 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);
```

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

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

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

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

$$0$$

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

```
> restart:with(Clifford):with(Cliplus);
  v:=2*e1we3+e4+Id;u:=2*e1we2we3+e1we2;
Cliplus has been loaded. Definitions for type/climon and type/clipolynom now in
clude &C and &C[K]. Type ?cliprod for help.
```

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

makeclialiases]

$$v := 2\, e1we3 + e4 + Id$$

$$u := 2\, e1we2we3 + e1we2$$

```
> '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 u
```

$$2\, e1we2we3 + e1we2$$

$$2\, e1we3 + e4 + Id$$

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

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

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

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

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

$$\begin{aligned} out4 := {} & 4\, K_{3,1}\, K_{2,3}\, e1 - 4\, K_{3,1}\, K_{1,3}\, e2 - 4\, K_{2,1}\, K_{3,3}\, e1 + 4\, K_{2,1}\, K_{1,3}\, e3 + 4\, K_{1,1}\, K_{3,3}\, e2 \\ & - 4\, K_{1,1}\, K_{2,3}\, e3 + 2\, K_{3,4}\, e1we2 - 2\, K_{2,4}\, e1we3 + 2\, K_{1,4}\, e2we3 + 2\, e1we2we3 \\ & + 2\, K_{2,1}\, K_{1,3}\, Id - 2\, K_{1,1}\, K_{2,3}\, Id + K_{2,4}\, e1 - K_{1,4}\, e2 + e1we2 \end{aligned}$$

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

```
> out5:=clieval(out3);
```

$$\begin{aligned} out5 := {} & 4\, K_{3,1}\, K_{2,3}\, e1 - 4\, K_{3,1}\, K_{1,3}\, e2 - 4\, K_{2,1}\, K_{3,3}\, e1 + 4\, K_{2,1}\, K_{1,3}\, e3 + 4\, K_{1,1}\, K_{3,3}\, e2 \\ & - 4\, K_{1,1}\, K_{2,3}\, e3 + 2\, K_{3,4}\, e1we2 - 2\, K_{2,4}\, e1we3 + 2\, K_{1,4}\, e2we3 + 2\, e1we2we3 \\ & + 2\, K_{2,1}\, K_{1,3}\, Id - 2\, K_{1,1}\, K_{2,3}\, Id + K_{2,4}\, e1 - K_{1,4}\, e2 + e1we2 \end{aligned}$$

```
> out5-out4;
```

$$0$$

Now let's see the mixed input:

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

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

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

```
> clieval(out6)-out5;
```

$$0$$

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

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

```
> out7-out6;
```

$$0$$

```
>
```

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

```
> RCbig(`&C`[K](e1,e2),`&C`[K](e3,e4)); ##<<<--- Error because
  contraction is w.r.t. B
Error, (in Cliplus:-RCbig) optional (or default B) parameter in RCbig differs
from indices encountered in its cliprod arguments. Found these names as indices
of &C: {B, K}
```

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

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

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

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

```
>
```

See Also: Clifford:-cmul, Clifford:-clicollect, Cliplus:-clieval, Cliplus:-cliexpand, Clifford:-makealiases, Clifford:-wedge

Last modified: December 20, 2007, RA/BF.

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

Calling Sequence:

none

Parameters:

none

Description:

- Procedure 'init' is the initialization procedure for the 'Cliplus' package. It is executed automatically when the package is loaded.
- At the time of loading, the following macros are defined:

  - macro(Clifford:-cmul = Cliplus:-climul) - extends Clifford:-cmul from 'CLIFFORD'
  - macro(Clifford:-cmulQ = Cliplus:-climul) - extends Clifford:-cmulQ from 'CLIFFORD'
  - macro(`&c` = Cliplus:-climul) - extends Clifford:-`&c` from 'CLIFFORD'
  - macro(`&cQ` = Cliplus:-climul) - extends Clifford:-`&cQ` from 'CLIFFORD'
  - macro(Clifford:-reversion = Cliplus:-clirev) - extends Clifford:-reversion from 'CLIFFORD'
  - macro(Clifford:-LC = Cliplus:-LCbig) - extends Clifford:-LC from 'CLIFFORD'
  - macro(Clifford:-RC = Cliplus:-RCbig) - extends Clifford:-RC from 'CLIFFORD'
- To see all environmental variables that are defined and used by 'CLIFFORD', use procedure Clifford:-CLIFFORD_ENV.
- User can change values of these environmental variables by making simple assignments. See below.

Examples:

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

```
> CLIFFORD_ENV();


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

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



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



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



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

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

Last modified: December 20, 2007, RA/BF.