

```
> 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)
```

```
Copyright 2006-2008 by Rafal Ablamowicz (*) and Bertfried Fauser ($)
```

```
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);**

Clipplus has been loaded. Definitions for type/climon and type/clipolynom now include &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 $m_1 m_2 m_r > m_1 m_1 m_r$ for every $m_1, m_2, m_1,$ and m_r in the Grassmann basis

In the following we sort Grassmann monomials in the basis for $\wedge R^3$ 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 $m = e^\alpha = (e_1^{\alpha_1} \&w e_2^{\alpha_2}) \&w (\dots) \&w (e_9^{\alpha_9})$ where $\alpha = [\alpha_1, \alpha_2, \dots, \alpha_9]$. Of course, each $\alpha_i = 0$ or 1 as $e_i^{\alpha_i} = \&w(e_i, e_i, \dots, e_i) = 0$ if $\alpha_i \geq 2$. We also agree that $e_i^0 = Id$ in the Grassmann algebra \wedge . We will refer to the list α also as a vector.

In the following let $m_1 = e^\alpha$ and $m_2 = e^\beta$.

'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 $m_1 \geq m_2$. Otherwise it returns false, in which case we say that $m_1 < m_2$. For example, it gives a lexicographic order on generators $e_1 > e_2 > \dots > e_9$ (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 $m1 \geq m2$. Otherwise it returns false, in which case we say that $m1 < m2$. For example, it gives the inverse lexicographic order on generators $e9 > e8 > \dots > e1$ (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 $m1 \geq m2$. Otherwise it returns false, in which case we say that $m1 < m2$. For example, it gives the reverse lexicographic order on generators $e1 > e2 > \dots > e9$ (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 $m1 \geq m2$. Otherwise it returns false, in which case we say that $m1 < m2$. For example, it gives the reverse lexicographic order on generators $e1 > e2 > \dots > e9$ (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]
      "Lex order gives" = [ e123, e12, e13, e1, e23, e2, e3, Id]
      "InvLex order gives" = [ e123, e23, e13, e3, e12, e2, e1, Id]
      "RevLex order gives" = [ Id, e1, e2, e12, e3, e13, e23, e123]
      "InvRevLex order gives" = [ Id, e3, e2, e23, e1, e13, e12, e123]
      "Deg[Lex] order gives" = [ e123, e12, e13, e23, e1, e2, e3, Id]
      "Deg[InvLex] order gives" = [ e123, e23, e13, e12, e3, e2, e1, Id]
      "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 := "Lex"

```

```

Monomial order Lex is admissible: true

```

```

      flag := true

```

```

      morder1 := "InvLex"

```

```

Monomial order InvLex is admissible: true

```

```

      flag := false

```

```

      morder1 := "RevLex"

```

```

Monomial order RevLex is admissible: false

```

```

      flag := false

```

```

      morder1 := "InvRevLex"

```

```

Monomial order InvRevLex is admissible: false

```

```

      flag := true

```

```

        morder1 := "GfG:-Deg[GfG:-Lex]"
Monomial order GfG:-Deg[GfG:-Lex] is admissible: true
        flag := true
        morder1 := "GfG:-Deg[GfG:-InvLex]"
Monomial order GfG:-Deg[GfG:-InvLex] is admissible: true
        flag := false
        morder1 := "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) → 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 $p1 = \text{wedge}(q,p2) + r$ where r is a polynomial such that $\text{GDivide}(r,p2) = r$, 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 $\wedge \dashrightarrow \wedge^{\text{op}}$.

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 remainder 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 $\wedge \mathbb{R}^n$, 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 $\wedge \mathbb{R}^n$, 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 $AP = \text{Clifford:-RC}(A, \text{reversion}(C))$, 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 $C = \text{Id}$, that is, $[A, \text{Id}, B]$.

Note: We temporarily set $B = \text{diag}(1, \dots, 1)$ 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 m_1 and m_2

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);
```

$$[Id, e_1, e_2, e_3, e_{12}, e_{13}, e_{23}, e_{123}]$$

$$p_1 := 7 e_2 + 5 e_{23}$$

$$ra := [7 e_2 + 5 e_{23}, -7 e_{12} + 5 e_{123}, 0, 7 e_{23}, 0, -7 e_{123}, 0, 0]$$

$$la := [7 e_2 + 5 e_{23}, 7 e_{12} + 5 e_{123}, 0, -7 e_{23}, 0, -7 e_{123}, 0, 0]$$

$$diff = [0, -7 e_{12}, 0, 7 e_{23}, 0, 0, 0, 0]$$

$$summ = [7 e_2 + 5 e_{23}, 5 e_{123}, 0, 0, 0, -7 e_{123}, 0, 0]$$

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(%));
```

$$m_1 := e_{1235}$$

$$m_2 := e_4we_3we_2$$

$$[e_{15}, e_{23}, -e_4]$$

$$e_{12345}$$

$$[-e_4, -e_3we_2, e_{15}]$$

>

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 RightNormalForm.

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 $k\langle S \rangle$ (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 $k\langle S \rangle$. A normal form of f w.r.t. G is an element h in $k\langle S \rangle$ 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 $k\langle S \rangle$ 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 $k\langle S \rangle$, and for every h, the normal form of f:

- if $h = 0$ then f belongs to I;
- if $h \neq 0$ 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.

$r := -e_2$

```
> r:=GDivide(r,p1,InvLex);  
r:=GDivide(r,p2,InvLex);  
r:=GDivide(r,p3,InvLex);
```

$-e_2 = -e_2$

$-e_2 = -e_2$

$-e_2 = -e_2$

```
>
```

Example 9: Procedure 'GGbasis' computes a Groebner basis for the ideal generated by a list of Grassmann polynomials $F = [p_1, p_2, p_3]$ 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(Clipford:-maxindex,F)));
```

$F := [2 e_1 - 3 e_1 e_3, e_3 + Id, 2 e_1]$

$n := 3$

```
> c1,c2,c3:=rd_clipolynom(n),rd_clipolynom(n),rd_clipolynom(n);
```

$c_1, c_2, c_3 := 5 Id + 5 e_1 e_3 - 5 e_2, 6 Id - 4 e_2 e_3, -5 Id$

```
> 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]);
```

$p_in := 10 e_1 e_2 - 15 e_1 e_3 - 15 e_1 e_2 e_3 + 6 e_3 + 6 Id - 4 e_2 e_3$

$p_out := 10 e_1 e_2 - 15 e_1 e_3 - 15 e_1 e_2 e_3 + 6 e_3 + 8 Id - 4 e_2 e_3 + 3 e_2$

Warning, procedure GReduce is obsolete and produces wrong results. Use GfG:-CanForm, GfG:-LeftNormalForm, or GfG:-RightNormalForm

$2 Id + 7 e_2$

```
> 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

$G := \left[-\frac{2 e_1}{3} + e_1 e_3, e_3 + Id, e_1 \right]$

```
> 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

$rem1 := 4 e_2$

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.