IBNP

Involutive Bases for Noncommutative Polynomials

0.11

18 September 2024

Gareth A. Evans

Christopher D. Wensley

Gareth A. Evans
Email: gareth@mathemateg.com
Address: Ysgol y Creuddyn
Ffordd Derwen, Bae Penrhyn
Llandudno, LL30 3LB
U.K.

Christopher D. Wensley
Email: cdwensley.maths@btinternet.com
Homepage: https://github.com/cdwensley

Abstract

The IBNP package provides ...

Bug reports, comments, suggestions for additional features, and offers to implement some of these, will all be very welcome.

Please submit any issues at https://github.com/gap-packages/ibnp/issues/ or send an email to the second author at cdwensley.maths@btinternet.com.

Copyright

Acknowledgements

This documentation was prepared with the GAPDoc [LN17] and AutoDoc [GH16] packages.

The procedure used to produce new releases uses the package GitHubPagesForGAP [Hor17] and the package ReleaseTools.

Contents

Chapter 1

Introduction

The IBNP package provides methods for computing an involutive (Gröbner) basis B for an ideal J over a polynomial ring $\mathscr{R}$ in both the commutative and noncommutative cases. Secondly, methods are provided to involutively reduce a given polynomial to its normal form in $\mathscr{R}/J$.

This package was first submitted to run with GAP 4.13.1.

The package is loaded with the command

Example

```
gap> LoadPackage( "ibnp" );
```

The package may be obtained either as a compressed `.tar` file or as a `.zip` file, `ibnp-version_number.tar.gz`, by ftp from one of the following sites:

- the IBNP GitHub release site: https://gap-packages.github.io/ibnp/.
- any GAP archive, e.g. https://www.gap-system.org/Packages/packages.html;

The package also has a GitHub repository at: https://github.com/gap-packages/ibnp.

Once the package is loaded, the manual `doc/manual.pdf` can be found in the documentation folder. The `html` versions, with or without MathJax, may be rebuilt as follows:

Example

```
gap> ReadPackage( "ibnp", "makedoc.g" );
```

It is possible to check that the package has been installed correctly by running the test files (this terminates the GAP session):

Example

```
gap> ReadPackage( "ibnp", "tst/testall.g" );
Architecture: . . . . .
testing: . . . . .
. . .
#I  No errors detected while testing
```

The main reference for this work is Evans' thesis [Eva05]. The main concepts and results may be found in the papers [Eva04] and [EW07].

Chapter 2

Using the packages GBNP and NMO

2.1 Non-commutative polynomials (NPs)

Recall that the main datatype used by the GBNP package is a list of non-commutative polynomials (NPs). The data type for a non-commutative polynomial (its NP format) is a list of two lists:

- The first list is a list m of monomials.
- The second list is a list c of coefficients of these monomials.

The two lists have the same length. The polynomial represented by the ordered pair $[m,c]$ is $\sum_i c_i m_i$. A monomial is a list of positive integers. They are interpreted as the indices of the variables. So, if $k = [1,3,2,2,1]$ and the variables are a,b,c (in this order), then k represents the monomial acb^2a. There are various ways to print these but the default uses $a,b,c,\ldots$. The zero polynomial is represented by `[[],[]]`. The polynomial 1 is represented by `[[[]],[1]]`. The algorithms work for the algebra $\mathbb{F}\langle\langle x_1,x_2,\ldots,x_t\rangle\rangle$ of non-commutative polynomials in `t` variables over the field $\mathbb{F}$. Accordingly, the list c should contain elements of $\mathbb{F}$.

The GBNP functions `GP2NP` and `NP2GP` convert a polynomial to NP format and back again. Polynomials returned by `NP2GP` print with their coefficients enclosed in brackets. Polynomials may also be printed using the function `PrintNP`. The function PrintNPList is used to print a list of NPs, with one polynomial per line. The function `CleanNP` is used to collect terms and reorder them. The default ordering is first by degree and then lexicographic - `MonomialGrlexOrdering`. Alternative orderings are available - see section 2.3.

Example

```
gap> A3 := FreeAssociativeAlgebraWithOne(Rationals,"a","b","c");;
gap> a := A3.1;; b := A3.2;; c := A3.3;;
gap> ## define a polynomial and convert to NP-format
gap> p1 := 7*a^2*b*c + 8*b*c*a;
(8)*b*c*a+(7)*a^2*b*c
gap> Lp1 := GP2NP( p1 );
[ [ [ 1, 1, 2, 3 ], [ 2, 3, 1 ] ], [ 7, 8 ] ]
gap> ## define an NP-poly; clean it; and convert to a polynomial
gap> Lp2 := [ [ [1,1], [1,2,1], [3], [1,1], [3,1,2] ], [5,6,7,6,5] ];;
gap> PrintNP( Lp2 );
 5a^2 + 6aba + 7c + 6a^2 + 5cab
gap> Lp2 := CleanNP( Lp2 );
```

```
[ [ [ 3, 1, 2 ], [ 1, 2, 1 ], [ 1, 1 ], [ 3 ] ], [ 5, 6, 11, 7 ] ]
gap> ## note the degree lexicographic ordering
gap> PrintNP( Lp2 );
 5cab + 6aba + 11a^2 + 7c
gap> p2 := NP2GP( Lp2, A3 );
(5)*c*a*b+(6)*a*b*a+(11)*a^2+(7)*c
gap> PrintNPList( [ Lp1, Lp2, [ [], [] ], [ [ [] ], [9] ] ] );
 7a^2bc + 8bca
 5cab + 6aba + 11a^2 + 7c
 0
 9
```

2.2 Gröbner Bases

The GBNP package computes Gröbner bases using the function `SGrobner`. In the example below the polynomials $\{p,q\}$ define an ideal in $\mathbb{Z}\langle\langle a,b\rangle\rangle$ which has a three element Gröbner basis.

Example

```
gap> p := [ [ [2,2,2], [2,1], [1,2] ], [1,3,-1] ];;
gap> q := [ [ [1,1], [2] ], [1,1] ];;
gap> PrintNPList( [p,q] );
 b^3 + 3ba - ab
 a^2 + b
gap> GB := SGrobner( [p,q] );;
gap> PrintNPList(GB);
 a^2 + b
 ba - ab
 b^3 + 2ab
```

2.3 Orderings for monomials

The three monomial orderings provided by the main library are `MonomialLexOrdering`, `MonomialGrlexOrdering` and `MonomialGrevlexOrdering`. The first of these is the default used by GBNP.

The NMO package is now part of the package GBNP. It provides a choice of orderings on monomials, including lexicographic and length-lexicographic ones.

Example

```
gap> Lp1;
[ [ [ 1, 1, 2, 3 ], [ 2, 3, 1 ] ], [ 7, 8 ] ]
gap> Lp2;
[ [ [ 3, 1, 2 ], [ 1, 2, 1 ], [ 1, 1 ], [ 3 ] ], [ 5, 6, 11, 7 ] ]
gap> GtNPoly( Lp1, Lp2 );
true
gap> ## select the lexicographic ordering and reorder p1, p2
gap> lexord := NCMonomialLeftLexicographicOrdering( A3 );;
gap> PatchGBNP( lexord );
```

```
LtNP patched.
GtNP patched.
gap> Lp1 := CleanNP( Lp1 );
[ [ [ 2, 3, 1 ], [ 1, 1, 2, 3 ] ], [ 8, 7 ] ]
gap> Lp2 := CleanNP( Lp2 );
[ [ [ 3, 1, 2 ], [ 3 ], [ 1, 2, 1 ], [ 1, 1 ] ], [ 5, 7, 6, 11 ] ]
gap> GtNPoly( Lp1, Lp2 );
false
gap> ## revert to degree lex order
gap> UnpatchGBNP();;
LtNP restored.
GtNP restored.
gap> Lp1 := CleanNP( Lp1 );
[ [ [ 1, 1, 2, 3 ], [ 2, 3, 1 ] ], [ 7, 8 ] ]
gap> Lp2 := CleanNP( Lp2 );
[ [ [ 3, 1, 2 ], [ 1, 2, 1 ], [ 1, 1 ], [ 3 ] ], [ 5, 6, 11, 7 ] ]
gap> GtNPoly( Lp1, Lp2 );
true
```

Chapter 3

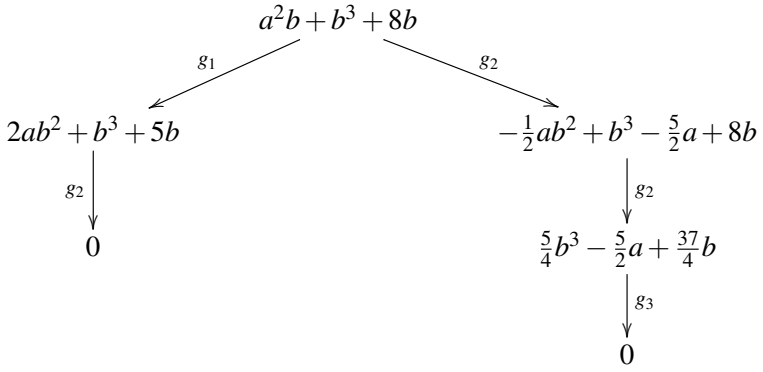
Commutative Involutive Bases

Given a Gröbner Basis G for an ideal J over a polynomial ring $\mathscr{R}$, the remainder of any polynomial $p \in \mathscr{R}$ with respect to G is unique. But, although this remainder is unique, there may be many ways of obtaining this remainder, as it is possible that several polynomials in G divide p, giving several *reduction paths* for p.

3.1 Reduction Paths

3.1.1 An Example

Consider the DegLex Gröbner basis $G := \{g_1, g_2, g_3\} = \{a^2 - 2ab + 3,\ 2ab + b^2 + 5,\ \frac{5}{4}b^3 - \frac{5}{2}a + \frac{37}{4}b\}$ over the polynomial ring $\mathbb{Q}[a,b]$, and consider the polynomial $p := a^2b + b^3 + 8b$. The remainder of p with respect to G is 0 (so that p is a member of the ideal J generated by G), but there are two ways of obtaining this remainder, as shown in the following diagram.



An *Involutive Basis* for J is a Gröbner Basis G such that there is only *one* possible reduction path for any polynomial $p \in \mathscr{R}$. In order to find such a basis, we restrict which reductions or divisions may take place by requiring, for each potential reduction of a polynomial p by a polynomial $g_i \in G$ (so that $LM(p) = LM(g_i) \times u$ for some monomial u), some extra conditions on the variables in u to be satisfied, namely that all variables in u have to be in a set of *multiplicative variables* for g_i, a set that is determined by a particular choice of an *involutive division*.

3.2 Commutative Involutive Divisions

Recall that a commutative monomial u is divisible by another monomial w if there exists a third monomial u' such that $u = wu'$. We use the notation $w \mid u$ and refer to w as a *conventional* divisor of u. An involutive division I partitions the variables in the polynomial ring into sets of *multiplicative* and *nonmultiplicative* variables for u. The set of multiplicative variables is denoted by $\mathscr{M}_I(u)$. Then w is an *involutive divisor* of u, written $w \mid_I u$, if all variables in u' are in $\mathscr{M}_I(u)$.

3.2.1 Example

Let $u := ab^3c$, $v := abc^3$ and $w := bc$ be three monomials over the polynomial ring $\mathscr{R} := \mathbb{Q}[a,b,c]$. Let an involutive division I partition the variables in $\mathscr{R}$ into the following two sets of variables for the monomial w: multiplicative = $\{a,b\}$; nonmultiplicative = $\{c\}$. It is true that w conventionally divides both monomials u and v, but w only involutively divides monomial u as, defining $u' := ab^2$ and $v' := ac^2$ (so that $u = wu'$ and $v = wv'$), we observe that all variables in u' are in the set of multiplicative variables for w, but the variables in v' (in particular the variable c) are not all in the set of multiplicative variables for w. So $w \mid_I u$ and $w \nmid_I v$.

3.2.2 PommaretDivision

▷ `PommaretDivision(alg, mons, order)` (operation)

Let $\mathscr{R} = \mathbb{Q}[a_1,\ldots,a_n]$, and let w be a polynomial in $\mathscr{R}$ with leading monomoial $a_1^{e_1}a_2^{e_2}\ldots a_n^{e_n}$ where e_i is the *first* non-zero exponent. The Pommaret involutive division $\mathscr{P}$ sets $\mathscr{M}_{\mathscr{P}}(w) = \{a_1,a_2,\ldots,a_i\}$.

Because $\mathscr{M}_{\mathscr{P}}(w)$ does not depend in any way on the other leading monomials in *polys*, this is a *global* division.

In The example the first five monomials u_i in U contain a power of x, so $\mathscr{M}_{\mathscr{P}}(u_i) = \{x\}$. Then u_6 involves y and z, so $\mathscr{M}_{\mathscr{P}}(u_6) = \{x,y\}$, and similarly $\mathscr{M}_{\mathscr{P}}(u_7) = \{x,y,z\}$.

Example

```
gap> R := PolynomialRing( Rationals, [ "x", "y", "z" ] );;
gap> x := R.1;; y := R.2;; z := R.3;;
gap> U := [ x^5*y^2*z, x^4*y*z^2, x^2*y^2*z, x*y*z^3, x*z^3, y^2*z, z ];
[ x^5*y^2*z, x^4*y*z^2, x^2*y^2*z, x*y*z^3, x*z^3, y^2*z, z ]
gap> ord := MonomialLexOrdering();;
gap> PommaretDivision( R, U, ord );
[ [ 1 ], [ 1 ], [ 1 ], [ 1 ], [ 1 ], [ 1, 2 ], [ 1 .. 3 ] ]
```

3.2.3 ThomasDivision

▷ `ThomasDivision(alg, mons, order)` (operation)

Let $\mathscr{R} = \mathbb{Q}[a_1,\ldots,a_n]$, and let P be a set of polynomials $P = \{p_1,\ldots,p_m\}$ in $\mathscr{R}$ with leading monomials $U = \{u_1,\ldots,u_m\}$ where $u_i = a_1^{e_i^1}a_2^{e_i^2}\ldots a_n^{e_i^n}$. The Thomas involutive division $\mathscr{T}$ sets a_i to be multiplicative for p_j and u_j if $e_j^i = \max_k e_k^i$ for all $1 \leqslant k \leqslant m$.

In the example, using the same seven monomials, the highest powers of $[x,y,z]$ are $[5,2,3]$ respectively. So x is multiplicative only for u_1, y is multiplicative for $\{u_1,u_3,u_6\}$, and z is multiplicative only for u_4 and u_5. Note that two of the monomials have no multiplicative variable.

Example

```
gap> ThomasDivision( R, U, ord );
[ [ 1, 2 ], [  ], [ 2 ], [ 3 ], [ 3 ], [ 2 ], [  ] ]
```

3.2.4 JanetDivision

▷ `JanetDivision(alg, mons, order)` (operation)

Let $\mathscr{R} = \mathbb{Q}[a_1,\ldots,a_n]$, and let P be a set of polynomials $P = \{p_1,\ldots,p_m\}$ in $\mathscr{R}$ with leading monomials $U = \{u_1,\ldots,u_m\}$ where $u_i = a_1^{e_i^1} a_2^{e_i^2} \ldots a_n^{e_i^n}$. The Janet involutive division $\mathscr{J}$ sets a_n to be multiplicative for u_j provided $e_j^n = \max_k e_k^n$ for all $1 \leqslant k \leqslant m$. To determine whether a_i is multiplicative for u_j, let $L = [e_j^{i+1}, e_j^{i+2}, \ldots, e_j^n]$. Let S be the subset of $\{1,\ldots,m\}$ containing those k such that $[e_k^{i+1}, e_k^{i+2}, \ldots, e_k^n] = L$. Then a_i is multiplicative for u_j provided $e_j^i = \max_{k\in S} e_k^i$.

In the example, recall that the exponent lists for the seven monomials are

$$[5,2,1], \quad [4,1,2], \quad [2,2,1], \quad [1,1,3], \quad [1,0,3], \quad [0,2,1], \quad [0,0,1].$$

As with the Thomas division, $\max_k e_k^3 = 3$ and z is multiplicative only for u_4 and u_5.

For y, $L = [1]$ when $k \in \{1,3,6,7\}$ and $\max_{\{1,3,6,7\}} e_k^2 = 2$, so y is multiplicative for u_1, u_3 and u_6, but not for u_7. $L = [2]$ only for u_2, so y is multiplicative for u_2. $L = [3]$ for u_4 and u_5, and $e_4^2 > e_5^2$, so y is multiplicative for u_4 but not u_5.

For x, $L = [2,1]$ for $k \in \{1,3,6\}$ and $e_1^1 = 5$ is greater than e_3^1 and e_6^1, so x is multiplicative for u_1. The other values for L, namely $[1,2], [1,3], [0,3]$ and $[0,1]$, occur just once each, so x is multiplicative for u_2, u_4, u_5 and u_7.

Example

```
gap> JanetDivision( R, U, ord );
[ [ 1, 2 ], [ 1, 2 ], [ 2 ], [ 1, 2, 3 ], [ 1, 3 ], [ 2 ], [ 1 ] ]
```

3.2.5 Selecting a Division

The global variable `CommutativeDivision` is a string which can take values "Pommaret", "Thomas" or "Janet". The default is "Pommaret". The example shows how to select the Janet division.

Example

```
gap> CommutativeDivision := "Janet";
"Janet"
```

3.2.6 DivisionRecord

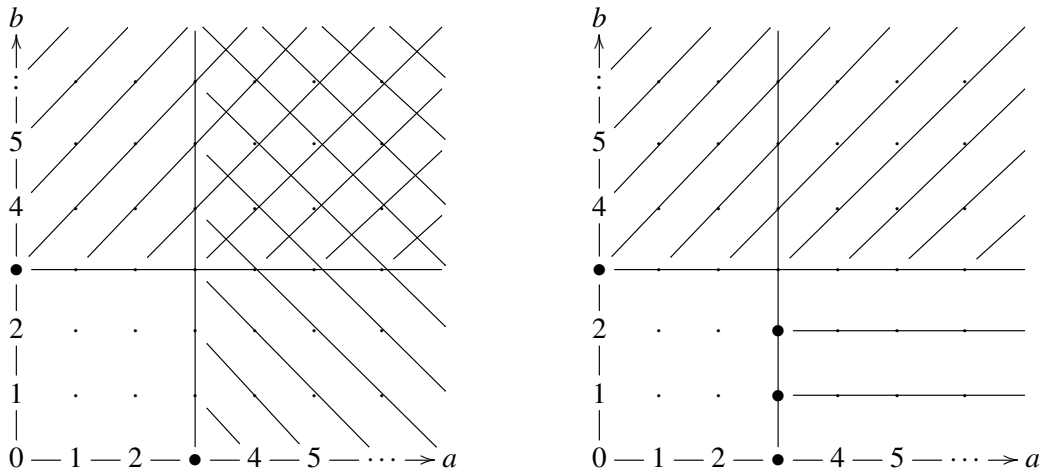
▷ `DivisionRecord(alg, polys, order)` (function)
▷ `DivisionRecordCP(alg, polys, order)` (operation)

The global function `DivisionRecord` calls one of the operations `DivisionRecordCP` and `DivisionRecordNP`, depending on whether the algebra is commutative or not. In the commutative case, this function finds the sets of multiplicative variables for a set of polynomials using one of the involutive divisions listed above.

In the following example, polynomials $\{u = b^3 - a, v = a^3 - b\}$ define an ideal and form a Gröbner basis for that ideal. Multiplicative variables for u are $\{a, b\}$, while those for v are just $\{a\}$. The variable `drec2.mvars` in the listing below contains the *positions* of these variables in the generating set $\{a, b\}$.

Example

```
gap> R := PolynomialRing( Rationals, [ "a", "b" ] );;
gap> a := R.1;; b := R.2;;
gap> L2 := [ b^3 - 3*a, a^3 - 3*b ];;
gap> ord := MonomialGrlexOrdering();;
gap> GB2 := ReducedGroebnerBasis( L2, ord );;
gap> GB2 = L2;
true
gap> CommutativeDivision := "Pommaret";;
gap> drec2 := DivisionRecordCP( R, L2, ord );
rec( div := "Pommaret", mvars := [ [ 1, 2 ], [ 1 ] ],
  polys := [ b^3-3*a, a^3-3*b ] )
```



In the *reduction diagrams* above the nodes (j,k) represent the monomials $a^j b^k$. The lead monomials of u and v are marked by bullets. In the left hand diagram the two shaded areas indicate those monomials which are conventionally reducible by u and by v, so that the doubly shaded area contains those monomials which are conventionally reducible by both. For an involutive division, this must be avoided.

In the right-hand diagram we see that u involutively divides the same set of monomials in the main shaded area. On the other hand v just involutively divides monomials $\{a^j \mid j \geq 3\}$. So none of

the monomials $\{a^jb, a^jb^2 \mid j \geq 3\}$ reduce by v involutively. The operation `InvolutiveBasis`, to be described below, produces two further polynomials, $w = vb = a^3b - 3b^2$ and vb^2 which reduces by u to $x = a^3b^2 - 9a$. Both w and x have multiplicative variables $\{a\}$ and the monomials which they can reduce lie on the two horizantal line segments in the right-hand diagram. In this way, all the conventionally reducible monomials are involutively reducible by just one of $\{u, v, w, x\}$.

The polynomial $p = a^3b^3 + 2a^3b + 3ab^3$ reduces involutively as follows.

$$p \xrightarrow{u} 3a^4 + 2a^3b + 3ab^3 \xrightarrow{v} 2a^3b + 3ab^3 + 9ab \xrightarrow{w} 3ab^3 + 9ab + 6b^2 \xrightarrow{u} 9a^2 + 9ab + 6b^2$$

This reduction is computed in section 3.3.2.

3.2.7 IPolyReduce

▷ `IPolyReduce(`*`algebra, polynomial, DivisionRecord, order`*`)` (function)
▷ `IPolyReduceCP(`*`algebra, polynomial, DivisionRecord, order`*`)` (operation)

The global function `IPolyReduce` calls one of the operations `IPolyReduceCP` and `IPolyReduceNP` depending on whether the algebra is commutative or not. This function reduces a polynomial p using the current overlap record for a basis, and an ordering.

In the example the polynomial p reduces only to $9a^2 + 9ab + 2a^3b$ because the polynomial x is not available to reduce $2a^3b$ to $6b^2$.

Example

```
gap> p := a^3*b^3 + 2*a^3*b + 3*a*b^3;;
gap> q := IPolyReduce( R, p, drec2, ord );
2*a^3*b+9*a^2+9*a*b
```

3.2.8 IAutoreduce

▷ `IAutoreduce(`*`alg, polys, order`*`)` (function)
▷ `IAutoreduceCP(`*`alg, polys, order`*`)` (operation)

The global function `IAutoreduce` calls one of the operations `IAutoreduceCP` and `IAutoreduceNP` depending on whether the algebra is commutative or not. This function applies `IPolyReduce` to a list of polynomials recursively until no more reductions are possible. More specifically, this function involutively reduces each member of a list of polynomials with respect to all the other members of the list, removing the polynomial from the list if it is involutively reduced to 0. This process is iterated until no more reductions are possible.

In the example we form `L` by adding p to `L2`. Applying `IAutoreduceCP` only p reduces, and the concatenation of `L2` with q is returned.

Example

```
gap> L := Concatenation( L2, [p] );;
gap> IAutoreduceCP( R, L, ord );
[ b^3-3*a, a^3-3*b, 2*a^3*b+9*a^2+9*a*b ]
```

3.3 Computing a Commutative Involutive Basis

The involutive algorithm for constructing an involutive basis uses *prolongations* and *autoreduction*.

3.3.1 Prolongations and Autoreduction

Given a set of polynomials P, a *prolongation* of $p \in P$ is a product pa_i where the generator a_i is *not* multiplicative with respect to the current involutive division.

A set of polynomials P is said to be *autoreduced* if no polynomial $p \in P$ contains a term which is involutively divisible by some polynomial $p' \in P \setminus \{p\}$.

We denote by $rem_I(p,Q)$ the involutive remainder of polynomial p with respect to a set of polynomials Q. Here is the *Commutative Autoreduction Algorithm*:

```
Input: a set of polynomials P = {p_1,p_2,...,p_n} and an involutive division I
  while there exists p_i in P such that rem_I(p_i, P\{p_i}) <> p_i do
    q := Rem_I(p_i, P\{p_i});
    P := P\{p_i};
    if (q<>0) then
      P := P union {q};
    fi;
  od;
  return P;
```

It can be shown that if P is a set of polynomials over a polynomial ring $\mathscr{R} = R[a_1,\ldots,a_n]$, such that P is autoreduced with respect to an involutive division I, and if p,q are two polynomials in $\mathscr{R}$, then $rem_I(p,P) + rem_I(q,P) = rem_I(p+q,P)$.

Given an involutive division I and an admissible monomial ordering O, an autoreduced set of polynomials P is a *locally involutive basis* with respect to I and O if any prolongation of any $p_i \in P$ involutively reduces to zero using P. Further, P is an *involutive basis* with respect to I and O if any multiple $p_i t$ of any $p_i \in P$ by any term t involutively reduces to zero using P.

The *Commutative Involutive Basis Algorithm*:

```
Input: a basis F = {f_1,f_2,...,f_m} for an ideal J
       over a commutative polynomial ring R[a_1,...,a_n];
       an admissible monomial ordering O;
       a continuous and constructive involutive division I
Output: an involutive basis G = {g_1,g_2,...,g_k} for J (if it terminates)
  G := { };
  F := autoreduction of F with respect to O and I;
  while G = { } do
    P := set of all prolongations f_i*a_j, 1<=i<=m, 1<=j<=n;
    q := 0;
    while (P <> { }) and (q=0) do
      p := a polynomial in P with minimal lead monomial w.r.t. O;
      P := P \ {p};
      q := rem_I(p,F);
    od;
    if (q <> 0) then  ## new basis element found
      F := autoreduction of (F union {q})
    else  ## all the prolongations have reduced to zero
      G := F;
```

```
    fi;
  od;
  return G;
```

3.3.2 InvolutiveBasis

▷ `InvolutiveBasis(`*alg, polys, order*`)` (function)
▷ `InvolutiveBasisCP(`*alg, polys, order*`)` (operation)

The global function `InvolutiveBasis` calls one of the operations `InvolutiveBasisCP` and `InvolutiveBasisNP` depending on whether the algebra is commutative or not. This function finds an involutive basis for the ideal generated by a set of polynomials, using a chosen ordering, and returns a division record.

Any involutive basis returned by this algorithm is a Gröbner basis, and remainders are involutively unique with respect to this basis.

Example

```
gap> ibasP := InvolutiveBasis( R, L2, ord );
rec( div := "Pommaret", mvars := [ [ 1, 2 ], [ 1 ], [ 1 ], [ 1 ] ],
  polys := [ b^3-3*a, a^3-3*b, a^3*b-3*b^2, a^3*b^2-9*a ] )
gap> r := IPolyReduce( R, p, ibasP, ord );
9*a^2+9*a*b+6*b^2
```

In the example above we start with $F = \{u, v\}$. There is only one prolongation, $P = \{w = vb\}$ and F becomes $\{u, v, w\}$. At the second iteration, $P = \{vb, wb\}$; vb reduces to zero; wb reduces to $x = a^3b^2 - 9a$; and F becomes $\{u, v, w, x\}$. At the third iteration $P = \{vb, wb, xb\}$ and all three of these reduce to zero, so $G = F$ is returned.

It is then shown that the multiplicative variables for $\{u, v, w, x\}$ are $\{\{a, b\}, \{a\}, \{a\}, \{a\}\}$.

Finally, the full reduction r of p, as described at the end of section 3.2.6, is computed.

If, instead of the Pommaret division, we use Thomas or Janet we obtain:

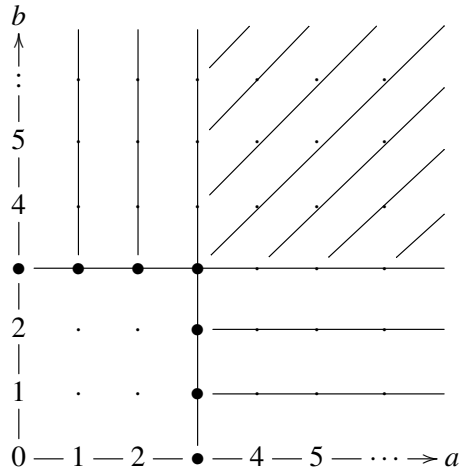
Example

```
gap> CommutativeDivision := "Thomas";;
gap> ibasT := InvolutiveBasis( R, L2, ord );
rec( div := "Thomas",
  mvars := [ [ 2 ], [ 1 ], [ 2 ], [ 1 ], [ 2 ], [ 1 ], [ 1, 2 ] ],
  polys := [ b^3-3*a, a^3-3*b, a*b^3-3*a^2, a^3*b-3*b^2, a^2*b^3-9*b,
      a^3*b^2-9*a, a^3*b^3-9*a*b ] )
gap> CommutativeDivision := "Janet";;
gap> ibasJ := InvolutiveBasis( R, L2, ord );;
gap> ibasJ.mvars = ibasP.mvars;
true
gap> ibasJ.polys = ibasP.polys;
true
```

The Janet division gives the same involutive basis as Pommaret, but the Thomas Division produces 7, rather than 4 polynomials:

$$[\, b^3 - 3a,\ a^3 - 3b,\ ab^3 - 3a^2,\ a^3b - 3b^2,\ a^2b^3 - 9b,\ a^3b^2 - 9a,\ a^3b^3 - 9ab\,].$$

The multiplicative variables for these polynomials are $[\{b\},\{a\},\{b\},\{a\},\{b\},\{a\},\{a,b\}]$, so the reduction diagram for the Thomas basis is:



Chapter 4

Functions for Noncommutative Monomials

A word, such as ab^2a is represented in GBNP as the list $[1,2,2,1]$. Polynomials have a more complicated structure, for example $6ab^2a - 7ab + 8ba$ is represented in GBNP by $[[[1,2,2,1],[1,2],[2,1]],[6,-7,8]]$, which is a list of monomials followed by the corresponding list of coefficients. Polynomials are dealt with in the following chapter.

As shown in Scetion 2.1, GBNP has functions `PrintNP` and `PrintNPList` to print a polynomial and a list of polynomials. Here we provide equivalent functions for monomials.

4.1 Basic functions for monomials

4.1.1 PrintNM

▷ `PrintNM(`*monomial*`)` (operation)

▷ `PrintNMList(`*list*`)` (operation)

Recall, from GBNP, that the actual letters printed are controlled by the operation `GBNP.ConfigPrint`.

Example

```
gap> GBNP.ConfigPrint( "a", "b" );
gap> mon := [2,1,1,1,2,2,1];;
gap> PrintNM( mon );
ba^3b^2a
gap> L := [ [1,2,2], [2,1,2], [1,1,1], [2], [ ] ];;
gap> PrintNMList( L );
ab^2
bab
a^3
b
1
```

4.1.2 PrefixNM

▷ PrefixNM(*monomial, posint*) (operation)
▷ SubwordNM(*monomial, posint, posint*) (operation)
▷ SuffixNM(*monomial, posint*) (operation)

These are the three operations which pick a sublist from a monomial list.

Example

```
gap> mon := [2,1,1,1,2,2,1];;
gap> PrefixNM( mon, 3 );
[ 2, 1, 1 ]
gap> SubwordNM( mon, 3, 6 );
[ 1, 1, 2, 2 ]
gap> SuffixNM( mon, 3 );
[ 2, 2, 1 ]
```

4.1.3 SuffixPrefixPosNM

▷ SuffixPrefixPosNM(*monomial, monomial, posint, posint*) (operation)

The operation SuffixPrefixPosNM(left, right, start, limit) looks for overlaps of type *suffix of left = prefix of right*. The size of the smallest such overlap is returned. Which overlaps are considered is controlled by the third and fourth arguments. We commence by looking at the overlap of size *start* and go no further than the overlap of size *limit*. When no overlap exists, 0 is returned. To test all possibilities, *start* should be 1 and *limit* should be $min(|left|, |right|) - 1$. It is the user's responsibility to make sure that these bounds are correct - no checks are made.

Example

```
gap> m1 := [2,1,1,1,2,2,1,1];;          ## m1 = ba^3b^2a^2
gap> m2 := [1,1,2,2,1,1];;              ## m2 = a^2b^2a^2
gap> SuffixPrefixPosNM( m1, m2, 1, 5 );  ## overlap is a
1
gap> SuffixPrefixPosNM( m1, m2, 2, 5 );  ## overlap is a^2
2
gap> SuffixPrefixPosNM( m1, m2, 3, 5 );  ## no longer an overlap
0
gap> SuffixPrefixPosNM( m2, m1, 1, 5 );  ## overlap is ba^2
3
```

4.1.4 SubwordPosNM

▷ SubwordPosNM(*monomial, monomial, posint*) (operation)
▷ IsSubwordNM(*monomial, monomial*) (operation)

The operation SubwordPosNM(small, large, start); answers the question for monomials *Is small a subword of large?*. The value returned is the start position in *large* of the first subword

found. When no subword is found, 0 is returned. The search commences at position *start* in *large* so, to test all possibilities, the third argument should be 1.

To just ask whether or not *small* is a subword of *large*, just use `IsSubwordNM( small, large);`.

Example

```
gap> m3 := [ 1, 1, 2 ];;
gap> SubwordPosNM( m3, m1, 1 );
3
gap> SubwordPosNM( m3, m2, 1 );
1
gap> SubwordPosNM( m3, m2, 2 );
0
gap> IsSubwordNM( [ 2, 1, 2 ], m1 );
false
```

4.1.5 LeadVarNM

▷ `LeadVarNM(`*monomial*`)` (operation)
▷ `LeadExpNM(`*monomial*`)` (operation)
▷ `TailNM(`*monomial*`)` (operation)

Given the word $w = b^4a^3c^2$, represented by $[2,2,2,2,1,1,1,3,3]$, the *lead variable* is b or 2, and the *lead exponent* is 4. Removing b^4 from w leaves the *tail* a^3c^2.

Example

```
gap> mon := [2,2,2,2,1,1,1,3,3];;
gap> LeadVarNM( mon );
2
gap> LeadExpNM( mon );
4
gap> TailNM( mon );
[ 1, 1, 1, 3, 3 ]
```

4.1.6 DivNM

▷ `DivNM(`*monomial, monomial*`)` (operation)

The operation `DivNM( large, small);` for two monomials returns all the ways that *small* divides *large* in the form of a list of pairs of monomials *[left,right]* so that *large* = *left*small*right*. In the example we search for subwords *ab* of $m = abcababc$, returning $[[abcab,c],[abc,abc],[1,cababc]]$.

Example

```
gap> m := [ 1, 2, 3, 1, 2, 1, 2, 3 ];;
gap> d := [ 1, 2 ];;
gap> PrintNMList( [ m, d ] );
abcababc
ab
```

```
gap> divs := DivNM( m, d );
[ [ [ 1, 2, 3, 1, 2 ], [ 3 ] ], [ [ 1, 2, 3 ], [ 1, 2, 3 ] ],
  [ [  ], [ 3, 1, 2, 1, 2, 3 ] ] ]
gap> PrintNMList( divs[1] );
abcab
c
```

Chapter 5

Functions for Noncommutative Polynomials

A word,

5.1 Basic functions for polynomials

5.1.1 Predefined algebras

For convenience of use in examples, three algebras over the rationals, `AlbebraIBNP` and `AlgebrakIBNP` with $k \in [2,3,4]$, are predefined by the package.

Example

```
gap> GeneratorsOfAlgebra( Algebra2IBNP );
[ (1)*<identity ...>, (1)*a, (1)*b ]
gap> GeneratorsOfAlgebra( Algebra3IBNP );
[ (1)*<identity ...>, (1)*a, (1)*b, (1)*c ]
gap> GeneratorsOfAlgebra( Algebra4IBNP );
[ (1)*<identity ...>, (1)*a, (1)*A, (1)*b, (1)*B ]
gap> AlgebraIBNP = Algebra2IBNP;
true
```

5.1.2 MaxDegreeNP

▷ `MaxDegreeNP(`*polylist*`)` (operation)

Given an `FAlgList`, this function calculates the degree of the lead term for each element of the list and returns the largest value found.

Example

```
gap> A2 := AlgebraIBNP;
<algebra-with-one over Rationals, with 2 generators>
gap> a := A2.1;; b := A2.2;;
gap> ord := NCMonomialLeftLengthLexicographicOrdering( A2 );;
gap> t := [ [ [1,2,1,1,2,1], [2,2,1,2], [2,1,1,2] ], [6,7,8] ];;
```

```
gap> u := [ [ [1,1,2,1], [1,2,2], [2,1] ], [4,-2,1] ];;
gap> v := [ [ [2,1,2], [1,2], [2] ], [2,-1,3] ];;
gap> w := [ [ [1,1,2], [2,1], [1] ], [3,2,-1] ];;
gap> L4 := [ t, u, v, w ];;
gap> PrintNPList( L4 );
 6aba^2ba + 7b^2ab + 8ba^2b
 4a^2ba - 2ab^2 + ba
 2bab - ab + 3b
 3a^2b + 2ba - a
gap> MaxDegreeNP( L4 );
6
```

5.1.3 MyScalarMulNP

▷ MyScalarMulNP(*pol*, *const*) (operation)

Arithmetic with polynomials is performed using the GBNP functions AddNP, MulNP and BiMulNP. We find it convenient to add here a function which multiplies a polynomial by an element of the underlying field of the algebra.

Does this actually get used?

Better to move it to GBNP?

Example

```
gap> w2 := MyScalarMulNP( w, 2 );;
gap> PrintNP( w2 );
 6a^2b + 4ba - 2a
```

5.1.4 LtNPoly

▷ LtNPoly(*pol1*, *pol2*) (operation)

▷ GtNPoly(*pol1*, *pol2*) (operation)

These two functions generalise the GBNP functions LtNP and GtNP which (confusingly) apply only to monomials. They compare a pair of polynomials with respect to the monomial ordering currently being used. In the example we check that $v > w$, that $w < 2w$ and $u < u + ba$.

Example

```
gap> LtNPoly( v, w );
false
gap> LtNPoly( w, w2 );
true
gap> u2 := AddNP( u, [ [ [2,1] ], [1] ], 1, 1 );;
gap> PrintNPList( [u,u2] );
 4a^2ba - 2ab^2 + ba
 4a^2ba - 2ab^2 + 2ba
gap> LtNPoly( u, u2 );
true
```

```
gap> ## LtNPoly and GtNPoly may be used within the Sort command:
gap> L5 := [u,v,w,u2,w2];
[ [ [ [ 1, 1, 2, 1 ], [ 1, 2, 2 ], [ 2, 1 ] ], [ 4, -2, 1 ] ],
  [ [ [ 2, 1, 2 ], [ 1, 2 ], [ 2 ] ], [ 2, -1, 3 ] ],
  [ [ [ 1, 1, 2 ], [ 2, 1 ], [ 1 ] ], [ 3, 2, -1 ] ],
  [ [ [ 1, 1, 2, 1 ], [ 1, 2, 2 ], [ 2, 1 ] ], [ 4, -2, 2 ] ],
  [ [ [ 1, 1, 2 ], [ 2, 1 ], [ 1 ] ], [ 6, 4, -2 ] ] ]
gap> Sort( L5, GtNPoly );
gap> L5;
[ [ [ [ 1, 1, 2, 1 ], [ 1, 2, 2 ], [ 2, 1 ] ], [ 4, -2, 2 ] ],
  [ [ [ 1, 1, 2, 1 ], [ 1, 2, 2 ], [ 2, 1 ] ], [ 4, -2, 1 ] ],
  [ [ [ 2, 1, 2 ], [ 1, 2 ], [ 2 ] ], [ 2, -1, 3 ] ],
  [ [ [ 1, 1, 2 ], [ 2, 1 ], [ 1 ] ], [ 6, 4, -2 ] ],
  [ [ [ 1, 1, 2 ], [ 2, 1 ], [ 1 ] ], [ 3, 2, -1 ] ] ]
```

5.1.5 LowestLeadMonomialPosNP

▷ LowestLeadMonomialPosNP(*polylist*) (operation)

Given a list of polynomials, this function looks at all the leading monomials and returns the position of the smallest lead monomial with respect to the monomial ordering currently being used. In the example, since L5 is sorted, the fifth polynomial is the least.

Example

```
gap> LowestLeadMonomialPosNP( L5 );
5
```

Chapter 6

Noncommutative Involutive Bases

When applying a noncommutative rewriting system we conventionally apply a rule $\ell \to r$ to a word w if and only if w has the form $w = u\ell v$, where u or v may be the empty word ε. Then w reduces to urv.

An *involutive monoid rewriting system I* will restrict these conventional reductions by imposing a limitation on the letters allowed in u and v. Sets $\mathscr{M}_I^L(w)$, the *left multiplicative variables* for w, and $\mathscr{M}_I^R(w)$, the *right multiplicative variables* for w, are defined by I.

6.1 Noncommutative Involutive Divisions

An *involutive division* $\mathscr{I}$ is a procedure for determining, given an arbitrary set of monomials W, sets of left and right multiplicative letters $\mathscr{M}_I^L(\ell, W)$ and $\mathscr{M}_I^R(\ell, W)$ for any $\ell \in W$. Then set $\mathscr{M}_I^L(W) = \{\mathscr{M}_I^L(\ell, W) \mid \ell \in W\}$ and $\mathscr{M}_I^R(W) = \{\mathscr{M}_I^R(\ell, W) \mid \ell \in W\}$.

An *involutive rewriting system I* is *based on* $\mathscr{I}$ if $\mathscr{M}_I^L(W)$ and $\mathscr{M}_I^R(W)$ are determined using $\mathscr{I}$, in which case we may write $\mathscr{M}_{\mathscr{I}}^L(W)$ and $\mathscr{M}_{\mathscr{I}}^R(W)$ for these sets of letters.

A word ℓ is an *involutive divisor* of w, written $\ell \mid_I w$, if

- $w = u\ell v$;
- either $u = \varepsilon$, or the last letter of u is left multiplicative for ℓ;
- and either $v = \varepsilon$, or the first letter of v is right multiplicative for ℓ.

When this is the case, w *involutively reduces* to urv by the rule $\ell \to r$.

For example, let $M = rws(\{x, y, z\},\ \{xy \to z,\ yz \to x\})$, so that $W = \{xy, yz\}$. Choose left and right multiplicative variables as shown in the following table:

ℓ	$\mathscr{M}_I^L(\ell, W)$	$\mathscr{M}_I^R(\ell, W)$
xy	$\{x, y, z\}$	$\{y, z\}$
yz	$\{y, z\}$	$\{x\}$

We consider reductions of $w = xyzx$. Conventionally, both rules may be used., giving reductions z^2x and x^3 respectively. Involutively, we see that $xy \mid_I xyzx$ because z is right multiplicative for xy, but $yz \nmid_I xyzx$ because x is left nonmultiplkiucative for yz. Thus the only involutive reduction is $xyzx \to_I z^2x$.

If an involutive division $\mathscr{I}$ determines the left and right multiplicative variables for a word $\ell in W$ *independently* of the set W, then the division is known as a *global involutive division*. Otherwise $\mathscr{I}$ is a *local involutive division*.

6.1.1 LeftDivision

▷ `LeftDivision(alg, mons, order)` (operation)

Given a word w, the *left division* ◁ assigns all letters to be left multiplicative for w, and all letters to be right nonmultiplicative for w. The example is taken from Example 5.5.12 in the thesis [Eva05].

Example

```
gap> A3 := Algebra3IBNP;;
gap> a:=A3.1;;  b:=A3.2;; c:=A3.3;;
gap> ord := NCMonomialLeftLengthLexicographicOrdering( A3 );;
gap> M6 := [ a*b, a, b*c, a*c, c*b, c^2 ];;
gap> U6 := [ [1,2], [1], [2,3], [1,3], [3,2], [3,3] ];;
gap> LeftDivision( A3, U6, ord );
[ [ [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ] ],
  [ [  ], [  ], [  ], [  ], [  ], [  ] ] ]
```

6.1.2 RightDivision

▷ `RightDivision(alg, mons, order)` (operation)

Given a word w, the *right division* ▷ assigns all letters to be left nonmultiplicative for w, and all letters to be right multiplicative for w.

Example

```
gap> RightDivision( A3, U6, ord );
[ [ [  ], [  ], [  ], [  ], [  ], [  ] ],
  [ [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ] ] ]
```

6.1.3 LeftOverlapDivision

▷ `LeftOverlapDivision(alg, mons, order)` (operation)

Let $W = \{w_1, \ldots, w_m\}$. The *left overlap division* $\mathscr{L}$ assumes, to begin with, that all letters are left and right multiplicative for every w_i. It then assigns some letters to be right nonmultiplicative as follows.

- Suppose $w_j \in W$ is a *subword*, but not a suffix, of a (different) word $w_i \in W$. Then, for some k, we have $w_j = \text{Subword}(w_i, k, k + deg(w_j) - 1)$. Assign the letter in position $k + deg(w_j) \in w_i$ to be right nonmultiplicative for w_j.
- Suppose a proper *prefix* of w_i is equal to a proper *suffix* of a (not neccessarily different) w_j, and that w_i is not a proper subword of w_j, or vice versa. Then, for some k, we have $\text{Prefix}(w_i, k) = \text{Suffix}(w_j, k)$. Assign the letter in position $k+1$ in w_i to be right nonmultiplicative for w_j.

Fox example, consider the rewriting system with rules $\{ab^2 \to b,\ ba^2 \to a\}$, so that the leading monomials are $\{u = ab^2, v = ba^2\}$. Neither monomial is a subword of the other, so the first rule above does

not apply. Since $\mathrm{Prefix}(v,1) = b = \mathrm{Suffix}(u,1)$, then $v[2] = b$ is assigned to be right nonmulitplicative for u. By symmetry, $u[2] = a$ is assigned to be right nonmulitplicative for v. The resulting sets are shown in the following table.

w	$\mathscr{M}^L_{\mathscr{L}}(w,W)$	$\mathscr{M}^R_{\mathscr{L}}(w,W)$
$u = ab^2$	$\{a,b\}$	$\{a\}$
$v = ba^2$	$\{a,b\}$	$\{b\}$

The following example continues Example 5.5.12 in the thesis [Eva05].

Example

```
gap> LeftOverlapDivision( A3, U6, ord );
[ [ [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ] ],
  [ [ 1, 2 ], [ 1 ], [ 1 ], [ 1 ], [ 1, 2 ], [ 1 ] ] ]
```

6.1.4 RightOverlapDivision

▷ `RightOverlapDivision(`*alg, mons, order*`)` (operation)

This division is the mirror image of `LeftOverlapDivision`.

Example

```
gap> RightOverlapDivision( A3, U6, ord );
[ [ [ 1 .. 3 ], [ 1 .. 3 ], [ 2 ], [ 1 .. 3 ], [  ], [  ] ],
  [ [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ] ] ]
```

6.1.5 Selecting a Division

The global variable `NoncommutativeDivision` which can take values "Left", "Right", "LeftOverlap" or "RightOverlap". The default is "LeftOverlap". The example shows how to select the left division.

Example

```
gap> NoncommutativeDivision := "LeftOverlap";
"LeftOverlap"
```

Other divisions may be added in due course.

6.1.6 DivisionRecordNP

▷ `DivisionRecordNP(`*alg, mons, order*`)` (operation)

This operation is called by the global function `DivisionRecord` when the algebra is noncommutative.

This operation finds the sets of multiplicative variables for a set of polynomials using one of the involutive divisions listed above. As in the commutative case, a three-field record `drec` is returned: `drec.div` is the division string; `drec.mvars` is a two-element list, the first listing the sets of left

multiplicative variables, and the second listing the sets of right multiplicative variables; `drec.polys` is the list of polynomials in NP-format.

Example

```
gap> L3 := [ [ [ [1,2,2], [3] ], [1,-1] ],
>            [ [ [2,3,3], [1] ], [1,-1] ],
>            [ [ [3,1,1], [2] ], [1,-1] ] ];;
gap> PrintNPList( L3 );
 ab^2 - c
 bc^2 - a
 ca^2 - b
gap> drec := DivisionRecord( A3, L3, ord );
rec( div := "LeftOverlap",
  mvars := [ [ [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ] ],
      [ [ 1, 2 ], [ 2, 3 ], [ 1, 3 ] ] ],
  polys := [ [ [ [ 1, 2, 2 ], [ 3 ] ], [ 1, -1 ] ],
      [ [ [ 2, 3, 3 ], [ 1 ] ], [ 1, -1 ] ],
      [ [ [ 3, 1, 1 ], [ 2 ] ], [ 1, -1 ] ] ] )
```

6.1.7 IPolyReduceNP

▷ `IPolyReduceNP(`*algebra, polynomial, DivisionRecord, order*`)` (operation)

This operation is called by the global function `IPolyReduce` when the algebra is noncommutative. This function reduces a polynomial p using the current overlap record for a basis, and an ordering.

In the example $p = 5c^2a^2b^2 + 6b^2c^2a^2 + 7a^2b^2c^2$. The monomial $c^2a^2b^2$ reduces to c^2ac by $ab^2 \to c$, since there are no letters to the right, but not by $ca^2 \to b$ since ca^2 is not right multiplicative by b. The other terms are simiolar, and p reduces to $5c^2ac + 6b^2cb + 7a^2ba$.

Example

```
gap> ## choose a polynomial to reduce
gap> p := 5*c^2*a^2*b^2 + 6*b^2*c^2*a^2 + 7*a^2*b^2*c^2;;
gap> ## convert to NP format and reduce
gap> Lp := GP2NP( p );
[ [ [ 3, 3, 1, 1, 2, 2 ], [ 2, 2, 3, 3, 1, 1 ], [ 1, 1, 2, 2, 3, 3 ] ],
  [ 5, 6, 7 ] ]
gap> Lrp := IPolyReduce( A3, Lp, drec, ord );;
gap> ## convert back to a polynomial
gap> rp := NP2GP( Lrp, A3 );
(5)*c^2*a*c+(6)*b^2*c*b+(7)*a^2*b*a
gap> ## p-rp should now belong to the ideal and reduce to 0
gap> q := p - rp;;
gap> Lq := GP2NP( q );;
gap> Lrq := IPolyReduce( A3, Lq, drec, ord );;
gap> rq := NP2GP( Lrq, A3 );
<zero> of ...
```

6.1.8 IAutoreduceNP

▷ IAutoreduceNP(*alg, polys, order*) (operation)

This operation is called by the global function IAutoreduce when the algebra is noncommutative. This function applies IPolyReduceNP to a list of polynomials recursively until no more reductions are possible. More specifically, this function involutively reduces each member of a list of polynomials with respect to all the other members of the list, removing the polynomial from the list if it is involutively reduced to 0. This process is iterated until no more reductions are possible.

In the example we form L4 by adding Lp to L3. Applying IAutoreduceNP only p reduces, and the concatenation of L3 with Lrp is returned.

Example

```
gap> L4 := Concatenation( L3, [Lp] );;
gap> R4 := IAutoreduceNP( A3, L4, ord );;
gap> PrintNPList( R4 );
 5c^2ac + 6b^2cb + 7a^2ba
 ca^2 - b
 bc^2 - a
 ab^2 - c
```

6.2 Computing a Noncommutative Involutive Basis

The involutive algorithm for constructing an involutive basis in the noncommutative case also uses *prolongations* and *autoreduction*.

6.2.1 InvolutiveBasisNP

▷ InvolutiveBasisNP(*alg, polys, order*) (operation)

This operation is called by the global function InvolutiveBasis when the algebra is noncommutative. This function finds an involutive basis for the ideal generated by a set of polynomials, using a chosen ordering.

In the example we find thqt a Gröbner basis starting from L3 is rather large. so add three more polynomials $[a^2b-c, b^2c-a, c^2a-b]$ defining the ideal. The resulting Gröbner basis then has just three terms. We then calculate an involutive basis, which has just seven terms. We also find the reduced form of p to be $18a^2$.

Example

```
gap> gbas := SGrobner( L3 );;
gap> Length( gbas );
64
gap> ## that's too large an example to continue with!
gap> K3 := [ [ [ [1,1,2], [3] ], [1,-1] ],
>            [ [ [2,2,3], [1] ], [1,-1] ],
>            [ [ [3,3,1], [2] ], [1,-1] ] ];;
gap> L6 := Concatenation( L3, K3 );;
gap> gbas := SGrobner( L6 );;
```

```
gap> PrintNPList( gbas );
 b - a
 c - a
 a^3 - a
gap> ## so the only reduced elements are {1,a,a^2}
gap> ibas := InvolutiveBasis( A3, L6, ord );
rec( div := "LeftOverlap",
  mvars := [ [ [ 1, 2 ], [ 1, 2 ], [ 1, 2 ], [ 1, 2 ], [ 1, 2 ], [ 1, 2 ] ],
      [ [  ], [  ], [  ], [ 1 ], [ 2 ], [  ] ] ],
  polys := [ [ [ [ 2, 1, 1, 2 ], [ 1, 2 ] ], [ 1, -1 ] ],
      [ [ [ 1, 2, 2, 1 ], [ 2, 1 ] ], [ 1, -1 ] ],
      [ [ [ 2, 1, 2 ], [ 2 ] ], [ 1, -1 ] ],
      [ [ [ 2, 1, 1 ], [ 1 ] ], [ 1, -1 ] ],
      [ [ [ 1, 2, 2 ], [ 2 ] ], [ 1, -1 ] ],
      [ [ [ 1, 2, 1 ], [ 1 ] ], [ 1, -1 ] ] ] )
rec( div := "LeftOverlap",
  mvars :=
    [
      [ [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ],
          [ 1 .. 3 ], [ 1 .. 3 ] ],
      [ [ 2, 3 ], [ 2, 3 ], [ 2, 3 ], [ 2, 3 ], [ 2, 3 ], [ 2, 3 ], [ 2, 3 ]
        ] ],
  polys := [ [ [ [ 3, 1, 1 ], [ 1 ] ], [ 1, -1 ] ],
      [ [ [ 2, 1, 1 ], [ 1 ] ], [ 1, -1 ] ],
      [ [ [ 1, 1, 1 ], [ 1 ] ], [ 1, -1 ] ],
      [ [ [ 3, 1 ], [ 1, 1 ] ], [ 1, -1 ] ],
      [ [ [ 2, 1 ], [ 1, 1 ] ], [ 1, -1 ] ], [ [ [ 3 ], [ 1 ] ], [ 1, -1 ] ],
      [ [ [ 2 ], [ 1 ] ], [ 1, -1 ] ] ] )
gap> PrintNPList( ibas.polys );
 ca^2 - a
 ba^2 - a
 a^3 - a
 ca - a^2
 ba - a^2
 c - a
 b - a
gap> Lr := IPolyReduce( A3, p, ibas, ord );;
gap> PrintNP( Lr );
 18a^2
```

In this simple example the left division produces the same basis, while the right and right overlap divisions produce (as might be expected) a mirror image basis.

Example

```
gap> NoncommutativeDivision := "RightOverlap";;
gap> ibas := InvolutiveBasis( A3, L6, ord );;
gap> PrintNPList( ibas.polys );
 a^2c - a
 a^2b - a
 a^3 - a
 ac - a^2
```

```
ab - a^2
c - a
b - a
```

Chapter 7

Examples

To be added.

References

[Eva04] G. A. Evans. Noncommutative involutive bases. In Q.-N. Trann, editor, *Proc. Int. Conf. Applications of Computer Algebra*, page 49--64. Beaumont, Texas, USA, 2004. 4

[Eva05] G. A. Evans. *Noncommutative Involutive Bases*. PhD thesis, University of Wales, Bangor, 2005. https://arxiv.org/pdf/math/0602140.pdf. 4, 24, 25

[EW07] G. A. Evans and C. D. Wensley. Complete involutive rewriting systems. *Symbolic Comput.*, 42:1034--1051, 2007. 4

[GH16] S. Gutsche and M. Horn. *AutoDoc - Generate documentation from GAP source code (Version 2016.12.04)*, 2016. GAP package, https://github.com/gap-packages/AutoDoc. 2

[Hor17] M. Horn. *GitHubPagesForGAP - Template for easily using GitHub Pages within GAP packages (Version 0.2)*, 2017. GAP package, https://gap-system.github.io/GitHubPagesForGAP/. 2

[LN17] F. Lübeck and M. Neunhöffer. *GAPDoc (Version 1.6)*. RWTH Aachen, 2017. GAP package, https://www.math.rwth-aachen.de/~Frank.Luebeck/GAPDoc/index.html. 2

Index