IBNP

Involutive Bases for Noncommutative Polynomials

0.21

20 November 2026

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 methods for computing an involutive (Gröbner) basis B for an ideal J over a polynomial ring R in both the commutative and noncommutative cases.

Secondly, methods are provided to involutively reduce a given polynomial to its normal form in R/J.

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

© 2024–2025, Gareth Evans and Chris Wensley.

The IBNP package is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

Acknowledgements

This documentation was prepared with the GAPDoc [LN17] and AutoDoc [GH23] packages.

The procedure used to produce new releases uses the package GitHubPagesForGAP [Hor19] and the package ReleaseTools.

Contents

1 Introduction **4**
1.1 History 4

2 Using the packages GBNP and NMO **6**
2.1 Noncommutative polynomials (NPs) 6
2.2 Gröbner Bases 7
2.3 Orderings for monomials 7

3 Commutative Involutive Bases **9**
3.1 Reduction Paths 9
3.2 Commutative Involutive Divisions 10
3.3 Computing a Commutative Involutive Basis 14

4 Functions for Noncommutative Monomials **21**
4.1 Basic functions for monomials 21

5 Functions for Noncommutative Polynomials **25**
5.1 Basic functions for polynomials 25

6 Noncommutative Involutive Bases **27**
6.1 Noncommutative Involutive Divisions 27
6.2 Computing a Noncommutative Involutive Basis 32
6.3 The Disjoint Cones Conditions 33

References **37**

Index **38**

Chapter 1

Introduction

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

This package was started using GAP 4.13.1 and first released with GAP 4.15.0.

The package is loaded with the command

Example

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

The package may be obtained as a compressed `.tar` file `IBNP-0.16.tar.gz` from the GitHub release site: https://github.com/gap-packages/ibnp/releases/tag/v0.16. 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> TestPackage( "ibnp" );
Architecture: . . . . .
testing: . . . . .
. . .
#I  No errors detected while testing
```

1.1 History

The theoretical basis behind this package is the Ph.D. thesis "Noncommutative Involutive Bases" of Gareth Evans in 2005 [Eva05]. (The main concepts and results may also be found in the papers [Eva70] and [EW07].) We quote here the summary from that thesis.

The theory of Gröbner Bases originated in the work of Buchberger [Buc98] and is now considered to be one of the most important and useful areas of symbolic computation. A great deal of effort has been put into improving Buchberger's algorithm for computing a Gröbner Basis, and indeed in finding

alternative methods of computing Gröbner Bases. Two of these methods include the Gröbner Walk method [AGK97] and the computation of Involutive Bases [ZB96].

By the mid 1980's, Buchberger's work had been generalised for noncommutative polynomial rings by Bergman [Ber78] and Mora [Mor86]. This thesis provides the corresponding generalisation for Involutive Bases and (to a lesser extent) the Gröbner Walk, with the main results being as follows.

- *Algorithms for several new noncommutative involutive divisions are given, including strong; weak; global and local divisions.*
- *An algorithm for computing a noncommutative Involutive Basis is given. When used with one of the aforementioned involutive divisions, it is shown that this algorithm returns a noncommutative Gröbner Basis on termination.*
- *An algorithm for a noncommutative Gröbner Walk is given, in the case of conversion between two harmonious monomial orderings. It is shown that this algorithm generalises to give an algorithm for performing a noncommutative Involutive Walk, again in the case of conversion between two harmonious monomial orderings.*
- *Two new properties of commutative involutive divisions are introduced (stability and extendibility), respectively ensuring the termination of the Involutive Basis algorithm and the applicability (under certain conditions) of homogeneous methods of computing Involutive Bases.*

Source code for an initial implementation of an algorithm to compute noncommutative Involutive Bases is provided in Appendix B. This source code, written using ANSI C and a series of libraries (AlgLib) provided by MSSRC forms part of a larger collection of programs providing examples for this thesis, including implementations of the commutative and noncommutative Gröbner Basis algorithms [Buc98], [Mor86]; the commutative Involutive Basis algorithm for the Pommaret and Janet involutive divisions [ZB96]; and the Knuth-Bendix critical pairs completion algorithm for monoid rewrite systems [KB04].

The implementations described in the last paragraph formed a package *Involutive* in 2005. This was based on libraries developed by the *Multidisciplinary Software Systems Research Corporation* (MSSRC) which apparently no longer exists. This software was provided for the research by Larry Lambe who was an Honorary Professor in the Mathematics Department at Bangor at that time. (For an example of his work, see [LR97].)

It has long been our intention to make these algorithms available to the wider symbolic computation community, and this GAP package is the result. Involutive Bases are constructed, but Gröbner Walks will have to wait until a later version. In place of the AlgLib library functions we use the noncommutative polynomial operations provided by the GBNP [CK24] package.

Chapter 2

Using the packages GBNP and NMO

This package deals with polynomials in noncommutative algebras and to do so makes use of the noncommutative polynomial operations provided by the GBNP [CK24] package, and orderings provided by the NMO package, which is now included within GBNP. In this chapter we remind users how to call some of these operations.

2.1 Noncommutative polynomials (NPs)

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

- The first is a list m of monomials.
- The second 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 x,y,z (in this order), then k represents the monomial xzy^2x. There are various ways to print these, but the default uses variables $a,b,c,\ldots$. The zero polynomial is represented by `[[],[]]` and the polynomial 1 is represented by `[[[]],[1]]`. The algorithms are applicable for the algebra $\mathbb{F}[x_1,x_2,\ldots,x_t]$ of noncommutative 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 lexicographically - `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] ], [5,6,7,8,9] ];;
gap> PrintNP( Lp2 );
 5a^2 + 6aba + 7c + 8a^2 + 9cab
gap> Lp2 := CleanNP( Lp2 );
[ [ [ 3, 1, 2 ], [ 1, 2, 1 ], [ 1, 1 ], [ 3 ] ], [ 9, 6, 13, 7 ] ]
gap> ## note the degree lexicographic ordering
gap> PrintNP( Lp2 );
 9cab + 6aba + 13a^2 + 7c
gap> p2 := NP2GP( Lp2, A3 );
(9)*c*a*b+(6)*a*b*a+(13)*a^2+(7)*c
gap> PrintNPList( [ Lp1, Lp2, [ [], [] ], [ [ [] ], [9] ] ] );
 7a^2bc + 8bca
 9cab + 6aba + 13a^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 $\mathbb{Z}[a,b]$ 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 GAP 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 ] ], [ 9, 6, 13, 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 ] ], [ 9, 7, 6, 13 ] ]
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 ] ], [ 9, 6, 13, 7 ] ]
gap> GtNPoly( Lp1, Lp2 );
true
```

Chapter 3

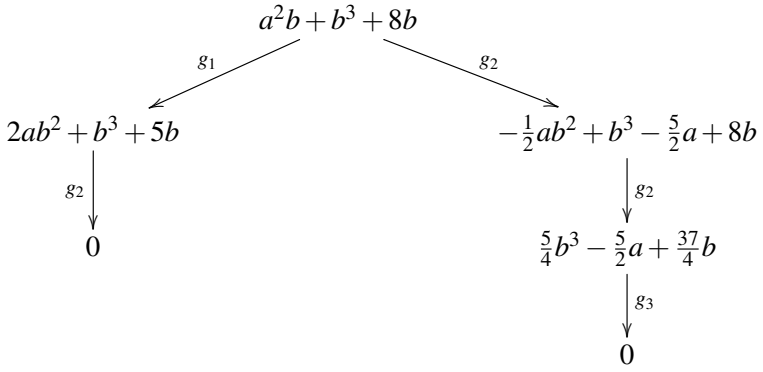
Commutative Involutive Bases

Given a Gröbner Basis G for an ideal J over a polynomial ring $\mathbb{R}$, the remainder of any polynomial $p \in \mathbb{R}$ with respect to G is unique. Although this remainder is unique, there may be many ways of finding it, as it is possible that several polynomials in G divide p, each giving a *reduction path* 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 $p \in \mathbb{R}$. 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 each polynomial. The set of multiplicative variables for w is denoted by $M_I(w)$. Then w is an *involutive divisor* of u, written $w \mid_I u$, if all variables in u' are in $M_I(w)$.

3.2.1 Example

Let $u := ab^3c$, $v := abc^3$ and $w := bc$ be three monomials over the polynomial ring $\mathbb{R} := \mathbb{Q}[a,b,c]$. Let an involutive division I partition the variables in $\mathbb{R}$ 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 $M_I(w)$, but the variables in v' (in particular the variable c) are not all in $M_I(w)$. So $w \mid_I u$ and $w \nmid_I v$.

3.2.2 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 Pommaret division.

Example

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

3.2.3 Selecting an Ordering

These three divisions are defined for a set of monomials, but we shall define a `DivisionRecord` below for a set of polynomials. The first step is therefore to select the leading monomials from this set, and that will bepend of the *ordering* chosen. We shall be using the orderings provided by the main GAP library as described in 2.3.

When calling `MonomialLexOrdering`, `MonomialGrlexOrdering` etc., it is essential to provide a list of indeterminates, as shown in the example. Otherwise some of the functions in this package will throw an error.

Example

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

3.2.4 PommaretDivision

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

Let $\mathbb{R} = \mathbb{F}[a_1,\ldots,a_n]$ with $a_1 > a_2 > \ldots > a_n$, and let w be a polynomial in $\mathbb{R}$ 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 $\mathbb{P}$ sets $M_{\mathbb{P}}(w) = \{a_1, a_2, \ldots, a_i\}$.

Because $M_{\mathbb{P}}(w)$ 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 $M_{\mathbb{P}}(u_i) = \{x\}$. Then u_6 involves y and z, so $M_{\mathbb{P}}(u_6) = \{x,y\}$, and similarly $M_{\mathbb{P}}(u_7) = \{x,y,z\}$.

Example

```
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> PommaretDivision( R, U, ord );
[ [ 1 ], [ 1 ], [ 1 ], [ 1 ], [ 1 ], [ 1, 2 ], [ 1 .. 3 ] ]
```

3.2.5 ThomasDivision

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

Let $\mathbb{R} = \mathbb{F}[a_1,\ldots,a_n]$ with $a_1 > a_2 > \ldots > a_n$, and let P be a set of polynomials $P = \{p_1,\ldots,p_m\}$ in $\mathbb{R}$ 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 $\mathbb{T}$ 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 ], [  ] ]
```

3.2.6 JanetDivision

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

Let $\mathbb{R} = \mathbb{F}[a_1,\ldots,a_n]$ with $a_1 > a_2 > \ldots > a_n$, and let P be a set of polynomials $P = \{p_1,\ldots,p_m\}$ in $\mathbb{R}$ 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 $\mathbb{J}$ 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 ] ]
```

3.2.7 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. The record constructed has three fields: the chosen division; a list of lists of positions of the multiplicative variables; and the set of polynomials.

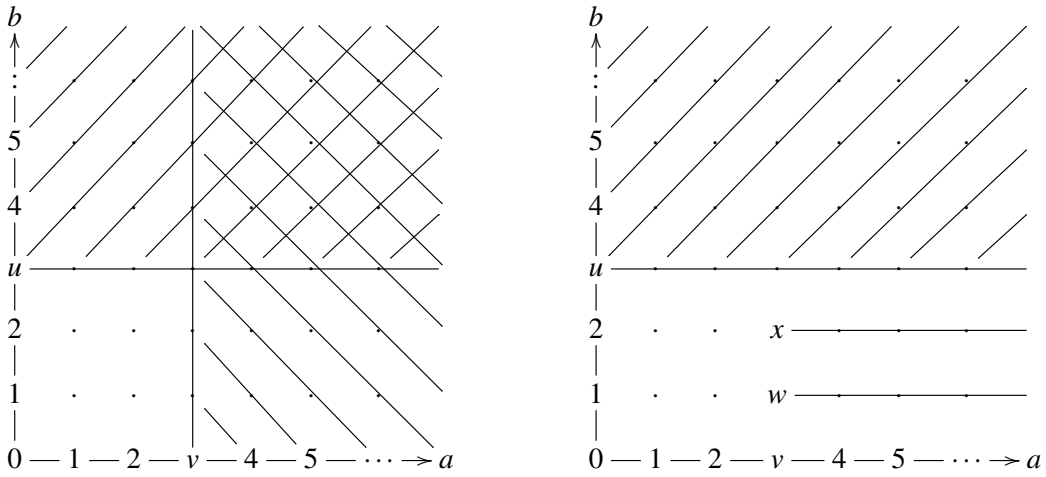
In the following example, polynomials $\{u = b^3 - 3a, v = a^3 - 3b\}$ define an ideal and form a Gröbner basis for that ideal. Using the Pommaret division, $M_{\mathbb{P}}(u) = \{a,b\}$ and $M_{\mathbb{P}}(v) = \{a\}$. The variable `drec2.mvars` in the listing below contains the *positions* of these 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( [a,b] );;
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* below the nodes (j,k) represent the monomials $a^j b^k$. The lead monomials of u and v are marked by these two names. 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 $\{a^j b, a^j b^2 \mid j \geq 3\}$ reduce by v involutively. The operation `InvolutiveBasis`, to be described below, produces two further polynomials, $w = vb = a^3 b - 3b^2$ and vb^2 which reduces by u to $x = a^3 b^2 - 9a$. 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.

3.2.8 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, using `drec2`, the polynomial p reduces only to $2a^3b + 9a^2 + 9ab$

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

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
```

3.2.9 LoggedIPolyReduce

▷ `LoggedIPolyReduce(`*`algebra, polynomial, DivisionRecord, order`*`)` (function)
▷ `LoggedIPolyReduceCP(`*`algebra, polynomial, DivisionRecord, order`*`)` (operation)

The global function `LoggedIPolyReduce` calls one of the operations `LoggedIPolyReduceCP` and `LoggedIPolyReduceNP` depending on whether the algebra is commutative or not. This function is similar to `IPolyReduce`, reducing a polynomial p using the current overlap record for a basis, and

an ordering. It's output, however, is a record containing, as well as the reduced polynomial r, logging information which shows how the reduction has been obtained:

$$p = r + \sum_i logs[i] * polys[i].$$

In the example `r.result` is equal to the previous result q, and the equation above is verified:

$$a^3b^3 + 2a^3b + 3ab^3 = (2a^3b + 9a^2 + 9ab) + (a^3 + 3a)(b^3 - 3a) + 3a(a^3 - 3b).$$

Example

```
gap> r := LoggedIPolyReduceCP( R, p, drec2, ord );
rec( logs := [ a^3+3*a, 3*a ], polys := [ b^3-3*a, a^3-3*b ],
  result := 2*a^3*b+9*a^2+9*a*b )
gap> r.result = q;
true
gap> p = r.result + r.logs[1]*r.polys[1] + r.logs[2]*r.polys[2];
true
```

3.2.10 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 reduces involutively to 0. This process is iterated until no more reductions are possible.

If no reduction takes place, so that the result is equal to the initial list of polynomials, then `true` is returned.

In the example we form `L3` by adding p to `L2`. On applying `IAutoreduceCP`, only p reduces, and the concatenation of `L2` with q is returned. Starting with `L4 = [u,v,w,x]`, there are no reductions, and `true` is returned.

Example

```
gap> L3 := Concatenation( L2, [p] );;
gap> IAutoreduceCP( R, L3, ord );
[ b^3-3*a, a^3-3*b, 2*a^3*b+9*a^2+9*a*b ]
gap> L4 := Concatenation( L2, [ a^3*b-3*b^2, a^3*b^2-9*a ] );;
gap> IAutoreduceCP( R, L4, ord );
true
```

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 $\mathbb{R} = \mathbb{F}[a_1,\ldots,a_n]$, such that P is autoreduced with respect to an involutive division I, and if p,q are two polynomials in $\mathbb{R}$, 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
```

Here we have returned to the example in section 3.2.7. Starting 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 is computed.

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

Example

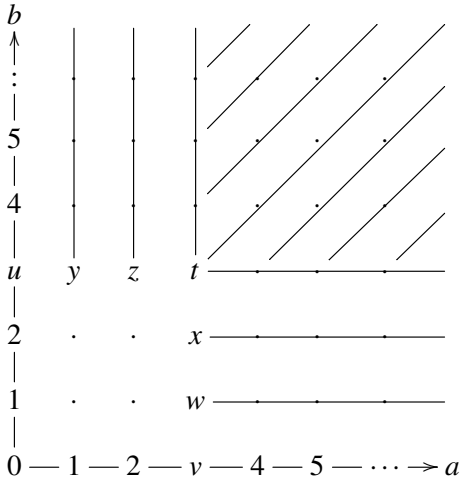
```
gap> CommutativeDivision := "Janet";;
gap> ibasJ := InvolutiveBasis( R, L2, ord );;
gap> ( ibasJ.mvars = ibasP.mvars ) and ( ibasJ.polys = ibasP.polys );
true
gap> CommutativeDivision := "Thomas";;
gap> ibasT := InvolutiveBasis( R, L2, ord );
rec( div := "Thomas",
  mvars := [ [ 2 ], [ 1 ], [ 2 ], [ 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 ] )
```

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

$$[u, v, y, w, z, x, t] \;=\; [\, b^3 - 3a,\; a^3 - 3b,\; ab^3 - 3a^2,\; a^3b - 3b^2,\; a^2b^3 - 9b,\; a^3b^2 - 9a,\; a^3b^3 - 9ab \,].$$

The multiplicative variables for these polynomials are $[\{b\}, \{a\}, \{b\}, \{a\}, \{b\}, \{a\}, \{a, b\}]$, so the

reduction diagram for the Thomas basis is:



The reduction of p with this basis is:

$$p = a^3b^3 + 2a^3b + 3ab^3 \xrightarrow{t} 2a^3b + 3ab^3 + 9ab \xrightarrow{w} 3ab^3 + 9ab + 6b^2 \xrightarrow{y} 9a^2 + 9ab + 6b^2.$$

Example

```
gap> r := LoggedIPolyReduceCP( R, p, ibasT, ord );
rec( logs := [ 0, 0, 3, 2, 0, 0, 1 ],
  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 ], result := 9*a^2+9*a*b+6*b^2 )
```

3.3.3 The process may abort

It may happen that the calculation of an involutive basis may start to produce an infinite polynomial basis. The constant `InvolutiveAbortLimit` is given an initial value of 20 which may be increased by the user. If the size of the basis being produced reaches this limit, then `fail` is returned.

For a simple example, we consider the Pommaret division and start with a set of polynomials $L := [x^3 - 5y, x^2 * y - 4y]$ all of whose leading monomials contain a power of x. Then, for each of these polynomials, x is the only involutive divisor, and prolongations all involve multiplying by y. The process aborts when the basis becomes

$$\{xy^3 - xy^2,\ x^2y^2 - xy^2,\ x^3y - xy^2,\ x^4 - xy\} \cup \{xy^i - xy^2 \mid 4 \leqslant i \leqslant 19\}.$$

(This may be verified on setting `InfoLevel(InfoIBNP)` to one.)

Example

```
gap> CommutativeDivision := "Pommaret";;
gap> R0 := PolynomialRing( Rationals, ["x","y"] );;
gap> x := R0.1;;  y := R0.2;;
gap> ord0 := MonomialGrlexOrdering( [x,y] );;
gap> L0 := [ x^4-x*y, x^3*y-x*y^2 ];;
gap> ibasP := InvolutiveBasisCP( R0, L0, ord0 );
#I  reached the involutive abort limit 20
fail
gap> CommutativeDivision := "Janet";;
gap> ibasP := InvolutiveBasisCP( R0, L0, ord0 );
rec( div := "Janet", mvars := [ [ 1, 2 ], [ 1 ], [ 1 ], [ 1 ] ],
  polys := [ x*y^3-x*y^2, x^2*y^2-x*y^2, x^3*y-x*y^2, x^4-x*y ] )
```

3.3.4 A more detailed example

Here we consider Example 4.5.2 in the thesis [Eva05]. On setting `InfoLevel(InfoIBNP)` to `1` some of the intermediate calculations are displayed. The setting of the problem is a rational polynomial ring in three variables with the lex ordering $[x,y,z]$, using the Janet involutive division.

- the initial basis contains two polynomials $\{a = x^2 + y^3,\ b = x + z^3\}$
- the reduced Gröbner basis is $\{c = y^3 + z^6,\ b = x + z^3\}$, and the irreducible monomials are $\{z^k,\ yz^k,\ y^2z^k \mid k \geqslant 0\}$
- when starting to calculate an involutive basis, the autoreduction does nothing because x is not multiplicative for b
- the only prolongation is $bx = x^2 + xz^3$ which reduces, on subtracting $a + bz^3$ and changing the sign, to $c = y^3 + z^6$
- on restarting with basis $[c,b,a]$, autoreduction replaces a with $a - c = d = x^2 - z^6$
- there are then 3 prolongations $[by, bx, dy]$ and bx now reduces to zero
- $by = e = xy + yz^3$ is then added to the basis: $[c,b,e,d]$
- $dy = x^2y - yz^6$ reduces to zero on adding $e(-x + z^3)$
- on restarting with basis $[c,b,e,d]$ there are 4 prolongations $[by, ey, bx, dy]$ and of these only $f = ey = xy^2 + y^2z^3$ does not reduce to zero
- restarting with basis $[c,b,e,f,d]$, the 5 prolongations $[by, ey, fy, bx, dy]$ all reduce to zero.
- now see what happens when reducing $p = x^7 + y^7 + z^7$ using the basis $[c,b,e,f,d]$
- $x^7 - dx^5 = x^5z^6$ and $x^5z^6 - dx^3z^6 = x^3z^{12}$ and $x^3z^{12} - dxz^{12} = xz^{18}$ and $xz^{18} - bz^{18} = -z^{21}$
- $y^7 - cy^4 = -y^4z^6$ and $-y^4z^6 + cyz^6 = yz^{12}$, so p reduces to $-z^{21} + yz^{12} + z^7$.

Example

```
gap> SetInfoLevel( InfoIBNP, 1 );;
gap> CommutativeDivision := "Janet";;
gap> R3 := PolynomialRing( Rationals, [ "x", "y", "z" ] );;
gap> x := R3.1;; y := R3.2;; z := R3.3;;
gap> ord3 := MonomialLexOrdering( [x,y,z] );;
gap> F := [ y^3 + x^2, z^3 + x ];;
gap> gbas := GrobnerBasis( F, ord3 );
[ y^3+x^2, z^3+x, -z^6-y^3 ]
gap> rgbas := ReducedGrobnerBasis( F, ord3 );
[ z^6+y^3, z^3+x ]
gap> ibasF := InvolutiveBasisCP( R3, F, ord3 );
#I  restarting with basis:
[ z^3+x, y^3+x^2 ]
#I  division record for basis: rec(
div := "Janet",
mvars := [ [ 2, 3 ], [ 1, 2, 3 ] ],
polys := [ z^3+x, y^3+x^2 ] )
#I  prolongations = [ x*z^3+x^2 ]
```

```
#I  restarting with basis:
[ z^6+y^3, z^3+x, y^3+x^2 ]
#I  after autoreduction basis =
[ z^6+y^3, z^3+x, -z^6+x^2 ]
#I  division record for basis: rec(
div := "Janet",
mvars := [ [ 1, 2, 3 ], [ 3 ], [ 1, 3 ] ],
polys := [ z^6+y^3, z^3+x, -z^6+x^2 ] )
#I  prolongations = [ y*z^3+x*y, x*z^3+x^2, -y*z^6+x^2*y ]
#I  restarting with basis:
[ z^6+y^3, z^3+x, y*z^3+x*y, -z^6+x^2 ]
#I  division record for basis: rec(
div := "Janet",
mvars := [ [ 1, 2, 3 ], [ 3 ], [ 1, 3 ], [ 1, 3 ] ],
polys := [ z^6+y^3, z^3+x, y*z^3+x*y, -z^6+x^2 ] )
#I  prolongations = [ y*z^3+x*y, y^2*z^3+x*y^2, x*z^3+x^2, -y*z^6+x^2*y ]
#I  restarting with basis:
[ z^6+y^3, z^3+x, y*z^3+x*y, y^2*z^3+x*y^2, -z^6+x^2 ]
#I  division record for basis: rec(
div := "Janet",
mvars := [ [ 1, 2, 3 ], [ 3 ], [ 1, 3 ], [ 1, 3 ], [ 1, 3 ] ],
polys := [ z^6+y^3, z^3+x, y*z^3+x*y, y^2*z^3+x*y^2, -z^6+x^2 ] )
#I  prolongations = [ y*z^3+x*y, y^2*z^3+x*y^2, y^3*z^3+x*y^3, x*z^3+x^2, -y*z\
^6+x^2*y ]
rec( div := "Janet",
  mvars := [ [ 1, 2, 3 ], [ 3 ], [ 1, 3 ], [ 1, 3 ], [ 1, 3 ] ],
  polys := [ z^6+y^3, z^3+x, y*z^3+x*y, y^2*z^3+x*y^2, -z^6+x^2 ] )
gap> ## now for a reduction - reset the info level:
gap> SetInfoLevel( InfoIBNP, 2 );;
gap> p := x^7 + y^7 + z^7;;
gap> IPolyReduce( R3, p, ibasF, ord3 );
#I  reduced to: x^5*z^6+y^7+z^7
#I  reduced to: x^3*z^12+y^7+z^7
#I  reduced to: x*z^18+y^7+z^7
#I  reduced to: -z^21+y^7+z^7
#I  reduced to: -z^21-y^4*z^6+z^7
#I  reduced to: -z^21+y*z^12+z^7
-z^21+y*z^12+z^7
```

3.3.5 Using homogeneous polynomials

If the polynomials in an initial basis are not homogeneous then they may be made homogeneous by introducing an additional variable. The resulting involutive basis will contain only homogeneous polynomials. However, if these are de-homogenised by setting the additional variable equal to 1, the resulting basis may not be involutive.

Applying this to the previous example, the resulting basis contains 10 polynomials which include homogenised versions of $[c, b, e, f]$, but not d.

Example

```
gap> SetInfoLevel( InfoIBNP, 0 );
gap> R4 := PolynomialRing( Rationals, [ "x", "y", "z", "t" ] );;
gap> x := R4.1;; y := R4.2;; z := R4.3;; t := R4.4;;
```

```
gap> H := [ x^2*t + y^3, x*t^2 + z^3 ];;
gap> ord4 := MonomialLexOrdering( [x,y,z,t] );;
gap> ibasH := InvolutiveBasisCP( R4, H, ord4 );
rec( div := "Janet",
  mvars := [ [ 1, 2, 3, 4 ], [ 1, 2, 3 ], [ 1, 3, 4 ], [ 1, 2, 3 ],
      [ 1, 2, 3 ], [ 1, 3, 4 ], [ 1, 3, 4 ], [ 1, 2 ], [ 1, 2 ], [ 1, 2 ] ],
  polys := [ y^3*t^3+z^6, x*t^2+z^3, x*t^3+z^3*t, x*z^3-y^3*t,
      x*z^3*t-y^3*t^2, x*y*t^3+y*z^3*t, x*y^2*t^3+y^2*z^3*t, x^2*t+y^3,
      x^2*z*t+y^3*z, x^2*z^2*t+y^3*z^2 ] )
```

Chapter 4

Functions for Noncommutative Monomials

A monomial, 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 Section 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 Predefined algebras

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

Example

```
gap> GeneratorsOfAlgebra( AlgebraIBNP );
[ (1)*<identity ...>, (1)*a, (1)*b ]
gap> Algebra2IBNP = AlgebraIBNP;
true
gap> A3 := Algebra3IBNP;
<algebra-with-one over Rationals, with 3 generators>
```

4.1.2 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", "c" );
gap> mon := [2,1,1,1,3,3,1];;
gap> PrintNM( mon );
```

```
ba^3c^2a
gap> L := [ [1,2,2], [3,1,2], [3,3,3], [2], [ ] ];;
gap> PrintNMList( L );
ab^2
cab
c^3
b
1
```

4.1.3 NM2GM

▷ NM2GM(*monomial, algebra*) (operation)
▷ NM2GMList(*list, algebra*) (operation)

Recall, from GBNP, that the functions NP2GP and NP2GPList convert a polynomial (or list of polynomials) in NP-format to an element of the algebra. This package provides additional functions NM2GM and NM2GMList which do the equivalent conversion for monomials.

Example

```
gap> m := NM2GM( mon, A3 );
(1)*b*a^3*c^2*a
gap> NM2GMList( [ mon, Reversed(mon), Concatenation(mon,mon) ], A3 );
[ (1)*b*a^3*c^2*a, (1)*a*c^2*a^3*b, (1)*(b*a^3*c^2*a)^2 ]
```

4.1.4 GM2NM

▷ GM2NM(*monomial*) (operation)
▷ GM2NMList(*list*) (operation)

Recall, from GBNP, that the functions GP2NP and GP2NPList convert a polynomial (or list of polynomials) to the equivalent NP-format. This package provides additional functions GM2NM and GM2NMList which do the equivalent conversion for monomials.

Example

```
gap> a:=A3.1;; b:=A3.2;; c:=A3.3;;
gap> p := (a*b*c)^2;;
gap> GM2NM(p);
[ 1, 2, 3, 1, 2, 3 ]
gap> GM2NMList( [ p, p^2, a^3, b^4, c^5 ] );
[ [ 1, 2, 3, 1, 2, 3 ], [ 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3 ], [ 1, 1, 1 ],
  [ 2, 2, 2, 2 ], [ 3, 3, 3, 3, 3 ] ]
```

4.1.5 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 := [2,1,1,1,3,3,1];;
gap> PrefixNM( mon, 3 );
[ 2, 1, 1 ]
gap> SubwordNM( mon, 3, 6 );
[ 1, 1, 3, 3 ]
gap> SuffixNM( mon, 3 );
[ 3, 3, 1 ]
```

4.1.6 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. The overlaps which are considered are 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
```

4.1.7 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*, use IsSubwordNM(small, large);.

Example

```
gap> m3 := [ 1, 1, 2 ];;                ## m3 = a^2b
gap> SubwordPosNM( m3, m1, 1 );
3                                       ## m1 = ba(a^b)ba^2
gap> SubwordPosNM( m3, m2, 1 );
1                                       ## m2 = (a^2b)ba^2
gap> SubwordPosNM( m3, m2, 2 );
0
```

```
gap> IsSubwordNM( [ 2, 1, 2 ], m1 );
false
```

4.1.8 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> mon4 := [2,2,2,2,1,1,1,3,3];;
gap> LeadVarNM( mon4 );
2
gap> LeadExpNM( mon4 );
4
gap> TailNM( mon4 );
[ 1, 1, 1, 3, 3 ]
```

4.1.9 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> GBNP.ConfigPrint( "a", "b", "c" );
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

5.1 Basic functions for polynomials

5.1.1 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. In the example this is v with degree 4

Example

```
gap> A2 := AlgebraIBNP;
<algebra-with-one over Rationals, with 2 generators>
gap> a := A2.1;; b := A2.2;;
gap> ord := NCMonomialLeftLengthLexicographicOrdering( A2 );;
gap> u := [ [ [1,1,2], [2,1], [1] ], [3,2,-1] ];;
gap> v := [ [ [1,1,2,1], [1,2,2], [2,1] ], [4,-2,1] ];;
gap> w := [ [ [2,1,2], [1,2], [2] ], [2,-1,3] ];;
gap> L3 := [ u, v, w ];;
gap> PrintNPList( L3 );
 3a^2b + 2ba - a
 4a^2ba - 2ab^2 + ba
 2bab - ab + 3b
gap> MaxDegreeNP( L3 );
4
```

5.1.2 ScalarMulNP

▷ ScalarMulNP(*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.

Example

```
gap> u2 := ScalarMulNP( u, 2 );;  PrintNP( u2 );
 6a^2b + 4ba - 2a
```

```
gap> x := [ [ [2,1] ], [5] ];;  PrintNP( x );
 5ba
gap> v2 := AddNP( v, x, 1, -2 );;  PrintNP( v2 );
 4a^2ba - 2ab^2 - 9ba
gap> w2 := MulNP( w, x );;  PrintNP( w2 );
 10bab^2a - 5ab^2a + 15b^2a
gap> u3 := BimulNP( [2,2], u, [1,1] );;  PrintNP( u3 );
 3b^2a^2ba^2 + 2b^3a^3 - b^2a^3
```

5.1.3 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 $w > u$, that $u < 2u$ and $v > v - 10ba$.

Example

```
gap> [ LtNPoly( w, u ), LtNPoly( u, u2 ) ];
[ false, true ]
gap> GtNPoly( v, v2 );
true
gap> ## LtNPoly and GtNPoly may be used within the Sort command:
gap> L4 := [u,v,u2,v2];
[ [ [ [ 1, 1, 2 ], [ 2, 1 ], [ 1 ] ], [ 3, 2, -1 ] ],
  [ [ [ 1, 1, 2, 1 ], [ 1, 2, 2 ], [ 2, 1 ] ], [ 4, -2, 1 ] ],
  [ [ [ 1, 1, 2 ], [ 2, 1 ], [ 1 ] ], [ 6, 4, -2 ] ],
  [ [ [ 1, 1, 2, 1 ], [ 1, 2, 2 ], [ 2, 1 ] ], [ 4, -2, -9 ] ] ]
gap> Sort( L4, GtNPoly );
gap> L4;
[ [ [ [ 1, 1, 2, 1 ], [ 1, 2, 2 ], [ 2, 1 ] ], [ 4, -2, 1 ] ],
  [ [ [ 1, 1, 2, 1 ], [ 1, 2, 2 ], [ 2, 1 ] ], [ 4, -2, -9 ] ],
  [ [ [ 1, 1, 2 ], [ 2, 1 ], [ 1 ] ], [ 6, 4, -2 ] ],
  [ [ [ 1, 1, 2 ], [ 2, 1 ], [ 1 ] ], [ 3, 2, -1 ] ] ]
```

5.1.4 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 `L4` is sorted, the fourth polynomial is the least.

Example

```
gap> LowestLeadMonomialPosNP( L4 );
4
```

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 $M_I^L(w)$, the *left multiplicative variables* for w, and $M_I^R(w)$, the *right multiplicative variables* for w, are defined by I.

6.1 Noncommutative Involutive Divisions

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

An *involutive rewriting system* I is *based on* $\mathbb{I}$ if $M_I^L(W)$ and $M_I^R(W)$ are determined using $\mathbb{I}$, in which case we may write $M_{\mathbb{I}}^L(W)$ and $M_{\mathbb{I}}^R(W)$ 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 $S = rws(\{x, y, z\},\ \{xy \to z,\ yz \to x\})$, so that $W = \{xy, yz\}$. Choose left and right multiplicative variables as shown in the following table:

ℓ	$M_I^L(\ell, W)$	$M_I^R(\ell, W)$
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 nonmultiplicative for yz. Thus the only involutive reduction is $xyzx \to_I z^2x$.

If an involutive division $\mathbb{I}$ 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 $\mathbb{I}$ 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 := GM2NMList( M6 );
[ [ 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* $\mathbb{L}$ 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 $\text{Prefix}(v, 1) = b = \text{Suffix}(u, 1)$, then $v[2] = a$ is assigned to be right nonmulitplicative for u. By symmetry, $u[2] = b$ is assigned to be right nonmulitplicative for v. The resulting sets are shown in the following table.

w	$M_{\mathbb{L}}^{L}(w,W)$	$M_{\mathbb{L}}^{R}(w,W)$
$u = ab^2$	$\{a,b\}$	$\{b\}$
$v = ba^2$	$\{a,b\}$	$\{a\}$

The following example takes W to be the list U6, continuing Example 5.5.12 in the thesis [Eva05]. As a is a subword of ab and ac, so b and c are right nonmultiplicative for a. Secondly, ab and cb have suffix b which is a prefix of bc, so c is right nonmultiplicative for ab and cb. Thirdly, ac, bc, c^2 all have suffix c, which is a prefix of cb and c^2, so b and c are both right nonmultiplicative for ac, bc, c^2.

Example

```
gap> M6;
[ (1)*a*b, (1)*a, (1)*b*c, (1)*a*c, (1)*c*b, (1)*c^2 ]
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`.

In the example, a is a prefix of ab and ac but is not a proper suffix of any monomial. However, bc has prefix b which is a suffix of ab and cb, so a and c and left nonmultiplicative for bc. Also cb and c^2 have prefix c which is a suffix of ac, bc, c^2, so all of a, b, c are left nonmultiplicative for cb and c^2.

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` can take values "Left", "Right", "LeftOverlap", "RightOverlap", "StrongLeftOverlop" or "StrongRightOverlap". The default is "LeftOverlap". The example shows how to select the left overlap 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 b is not right multiplicative for ca^2. The other terms are similar, 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 );;
[ [  ], [  ] ]
```

6.1.8 LoggedIPolyReduceNP

▷ LoggedIPolyReduceNP(*algebra, polynomial, DivisionRecord, order*) (operation)

This operation is called by the global function LoggedIPolyReduce when the algebra is noncommutative. This function is similar to IPolyReduceNP, reducing a polynomial p using the current overlap record for a basis, and an ordering. It's output, however, is a record containing, as well as the

reduced polynomial r, logging information L which shows how the reduction has been obtained:

$$p = r + \sum_{i,j} L[i][j][1] * L[i][j][2] * polys[i] * L[i][j][3].$$

In the example `r = logr.result` is equal to Lrp, and the equation above is then verified:

$$p = r + 5c^2a(ab^2 - c) + 7a^2b(bc^2 - a) + 6b^2c(ca^2 - b).$$

Example

```
gap> logr := LoggedIPolyReduce( A3, Lp, drec, ord );
rec( logs := [ [ [ 5, [ 3, 3, 1 ], [  ] ] ], [ [ 7, [ 1, 1, 2 ], [  ] ] ],
      [ [ 6, [ 2, 2, 3 ], [  ] ] ] ],
  polys := [ [ [ [ 1, 2, 2 ], [ 3 ] ], [ 1, -1 ] ],
      [ [ [ 2, 3, 3 ], [ 1 ] ], [ 1, -1 ] ],
      [ [ [ 3, 1, 1 ], [ 2 ] ], [ 1, -1 ] ] ],
  result := [ [ [ 3, 3, 1, 3 ], [ 2, 2, 3, 2 ], [ 1, 1, 2, 1 ] ], [ 5, 6, 7 ]
    ] )
gap> logr.result = Lrp;
true
gap> L := logr.logs;;
gap> p1 := ScalarMulNP( BimulNP( L[1][1][2], L3[1], L[1][1][3] ), L[1][1][1] );;
gap> p2 := ScalarMulNP( BimulNP( L[2][1][2], L3[2], L[2][1][3] ), L[2][1][1] );;
gap> q := AddNP( p1, p2, 1, 1 );;
gap> p3 := ScalarMulNP( BimulNP( L[3][1][2], L3[3], L[3][1][3] ), L[3][1][1] );;
gap> q := AddNP( q, p3, 1, 1 );;
gap> Lp = AddNP( q, Lrp, 1, 1 );
true
```

6.1.9 VerifyLoggedRecordNP

▷ `VerifyLoggedRecordNP(`*polynomial, LogRecord*`)` (operation)

This operation checks the identity in the equation displayed above.
For a more complicated example, see file `ibnp/tst/extras/reduce.tst`.

Example

```
gap> VerifyLoggedRecordNP( Lp, logr );
true
```

6.1.10 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.

If the set of polynomials is already reduced, then `true` is returned.

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
gap> IAutoreduceNP( A3, R4, ord );
true
```

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 that a Gröbner basis starting from `L3` is rather large, so add a fourth polynomial $a^2b - c$ 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 for this manual, so add a fourth poly
gap> K4 := Concatenation( L3, [ [ [ [1,1,2], [3] ], [1,-1] ] ] );;
gap> PrintNPList( K4 );
 ab^2 - c
 bc^2 - a
 ca^2 - b
 a^2b - c
gap> gbas := SGrobner( K4 );;
gap> PrintNPList( gbas );
 b - a
 c - a
 a^3 - a
gap> ## so the only reduced elements are {1,a,a^2} with a^3=a
gap> ibasK := InvolutiveBasis( A3, K4, ord );
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( ibasK.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, ibasK, ord );;
gap> PrintNP( Lr );
 18a^2
```

In this simple example the left division produces the same basis, while the right and right overlap divisions *do not* produce (as might be expected) a mirror image basis.

Example

```
gap> NoncommutativeDivision := "RightOverlap";;
gap> ribasK := InvolutiveBasis( A3, K4, ord );;
gap> PrintNPList( ribasK.polys );
 a^3 - a
 c - a
 b - a
```

6.3 The Disjoint Cones Conditions

The *disjoint right cones condition* for a set of monomials W requires that, for each monomial $w_i \in W$, at least one variable in every monomial $w_j \in W$ is right nonmultiplicative for w_i. The *disjoint left cones condition* is the mirror image of this.

6.3.1 StrongLeftOverlapDivision

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

The strong left overlap division is the extension of the left overlap division obtained by enforcing the disjoint right cones condition. This is achieved by considering all pairs $[w_i, w_j]$ and, if no variable in w_j is right nonmultiplicative for w_i, then $w_j[1]$ is removed from the list of right multiplicative variables for w_i.

In the example, the involutive basis using the left overlap division contains six polynomials with leading monomials $[c^2, cb, ca, ba, ac, ab]$ and with corresponding right non-multiplicative variables $[[a,b,c],[a],[b,c],[b,c],[a,b,c],[a]]$. Every monomial contains either b or c. When using the strong left overlap division, the first change is in the case $i = j = 2$ when neither b nor c is non-multiplicative

for cb. So c is made non-multiplicative for w_2. Similarly c is made non-multiplicative for $w_6 = ab$. The right multiplicative variables are now $[[\,],[b],[a],[a],[\,],[b]]$, and `InvolutiveBasisNP` continues with this information.

Example

```
gap> P4 := [ [ [ [1,2], [3] ], [1,-2] ],
>            [ [ [2,1], [3] ], [1,-2] ],
>            [ [ [1,3], [2] ], [1,-2] ],
>            [ [ [3,1], [2] ], [1,-2] ] ];;
gap> PrintNPList( P4 );
 ab - 2c
 ba - 2c
 ac - 2b
 ca - 2b
gap> NoncommutativeDivision := "LeftOverlap";;
gap> ibasP := InvolutiveBasisNP( A3, P4, ord );
rec( div := "LeftOverlap",
  mvars :=
    [
      [ [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ]\

        ], [ [  ], [ 2, 3 ], [ 1 ], [ 1 ], [  ], [ 2, 3 ] ] ],
  polys := [ [ [ [ 3, 3 ], [ 2, 2 ] ], [ 1, -1 ] ],
      [ [ [ 3, 2 ], [ 2, 3 ] ], [ 1, -1 ] ],
      [ [ [ 3, 1 ], [ 2 ] ], [ 1, -2 ] ], [ [ [ 2, 1 ], [ 3 ] ], [ 1, -2 ] ],
      [ [ [ 1, 3 ], [ 2 ] ], [ 1, -2 ] ], [ [ [ 1, 2 ], [ 3 ] ], [ 1, -2 ] ]
    ] )
gap> PrintNPList( ibasP.polys );
 c^2 - b^2
 cb - bc
 ca - 2b
 ba - 2c
 ac - 2b
 ab - 2c
gap> ## check that cbc reduces to b^3 and abc reduces to 2b^2
gap> IPolyReduce( A3, GP2NP( c*b*c ), ibasP, ord );
[ [ [ 2, 2, 2 ] ], [ 1 ] ]
gap> IPolyReduce( A3, GP2NP( a*b*c ), ibasP, ord );
[ [ [ 2, 2 ] ], [ 2 ] ]
gap> ## now apply the strong left overlap division - two polynomials are added
gap> NoncommutativeDivision := "StrongLeftOverlap";;
gap> sbasP := InvolutiveBasisNP( A3, P4, ord );
rec( div := "StrongLeftOverlap",
  mvars :=
    [
      [ [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ],
          [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ] ],
      [ [  ], [  ], [  ], [ 2 ], [ 1 ], [ 1 ], [  ], [ 2 ] ] ],
  polys := [ [ [ [ 3, 2, 3 ], [ 2, 2, 2 ] ], [ 1, -1 ] ],
      [ [ [ 1, 2, 3 ], [ 2, 2 ] ], [ 1, -2 ] ],
      [ [ [ 3, 3 ], [ 2, 2 ] ], [ 1, -1 ] ],
      [ [ [ 3, 2 ], [ 2, 3 ] ], [ 1, -1 ] ],
      [ [ [ 3, 1 ], [ 2 ] ], [ 1, -2 ] ], [ [ [ 2, 1 ], [ 3 ] ], [ 1, -2 ] ],
      [ [ [ 1, 3 ], [ 2 ] ], [ 1, -2 ] ], [ [ [ 1, 2 ], [ 3 ] ], [ 1, -2 ] ]
```

```
      ] )
gap> PrintNPList( sbasP.polys );
 cbc - b^3
 abc - 2b^2
 c^2 - b^2
 cb - bc
 ca - 2b
 ba - 2c
 ac - 2b
 ab - 2c
```

6.3.2 StrongRightOverlapDivision

▷ StrongRightOverlapDivision(*alg, mons, order*) (operation)

This operation is the mirror image of StrongLeftOverlapDivision.

When we compute an involutive basis rbasP using the right overlap division we find that rbasP.polys = ibasP.polys. However there is just one left multiplicative variable for each of the polynomials and the left disjoint cones condition is already satisfied. So, when using the strong right overlap division, we get the same basis.

Example

```
gap> NoncommutativeDivision := "RightOverlap";;
gap> rbasP := InvolutiveBasisNP( A3, P4, ord );
rec( div := "RightOverlap",
  mvars := [ [ [ 2 ], [ 2 ], [ 2 ], [ 2 ], [ 1 ], [ 1 ] ],
      [ [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ], [ 1 .. 3 ],
          [ 1 .. 3 ] ] ],
  polys := [ [ [ [ 3, 3 ], [ 2, 2 ] ], [ 1, -1 ] ],
      [ [ [ 3, 2 ], [ 2, 3 ] ], [ 1, -1 ] ],
      [ [ [ 3, 1 ], [ 2 ] ], [ 1, -2 ] ], [ [ [ 2, 1 ], [ 3 ] ], [ 1, -2 ] ],
      [ [ [ 1, 3 ], [ 2 ] ], [ 1, -2 ] ], [ [ [ 1, 2 ], [ 3 ] ], [ 1, -2 ] ]
    ] )
gap> NoncommutativeDivision := "StrongRightOverlap";;
gap> srbasP := InvolutiveBasisNP( A3, P4, ord );;
gap> ( rbasP.polys = srbasP.polys ) and ( rbasP.mvars = srbasP.mvars );
true
```

References

[AGK97] B. Amrhein, O. Gloor, and W. Küchlin. On the walk. *Theoret. Comput. Sci.*, 187 (1-2):179-202, 1997. 5

[Ber78] G. M. Bergman. The diamond lemma for ring theory. *Adv. in Math.*, 29 (2):178-218, 1978. 5

[Buc98] B. Buchberger. An algorithmic criterion for the solvability of a system of algebraic equations. Translation of Ph.D. thesis by M. Abramson and R. Lumbert. In B. Buchberger and F. Winkler, editors, *Gröbner Bases and Applications*, volume 251 of *Proc. London Math. Soc.*, page 535-545. Cambridge University Press, 1998. 4, 5

[CK24] A. Cohen and J. Knopper. *GBNP - computing Gröbner bases of noncommutative polynomials (Version 1.1.0)*. Discrete Algebra and Geometry (DAM) group at the Department of Mathematics and Computer Science of Eindhoven University of Technology, 2001-2024. GAP package, https://github.com/gap-packages/gbnp. 5, 6

[Eva70] G. A. Evans. Noncommutative involutive bases. In J. Leech, editor, *Proc. Int. Conf. Applications of Computer Algebra*, page 263-297. Pergamon Press, 1970. 4

[Eva05] G. A. Evans. *Noncommutative Involutive Bases*. PhD thesis, University of Wales, Bangor, 2005. https://arxiv.org/pdf/math/0602140.pdf. 4, 18, 28, 29

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

[GH23] S. Gutsche and M. Horn. *AutoDoc - Generate documentation from GAP source code (Version 2023.06.19)*, 2023. GAP package, https://github.com/gap-packages/AutoDoc. 2

[Hor19] M. Horn. *A GitHub Pages generator for GAP packages (Version 0.3)*, 2019. GAP package, https://github.com/gap-system/GitHubPagesForGAP/. 2

[KB04] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, page 263-297. Pergamon Press, 2004. 5

[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

[LR97] L. A. Lambe and D. E. Radford. *Introduction to the Quantum Yang-Baxter Equation and Quantum Groups: An Algebraic Approach*, volume 423 of *Mathematics and its Applications*. Kluwer Academic, 1997. 5

[Mor86] T. Mora. Gröbner bases for non-commutative polynomial rings. In J. Calmet, editor, *AAECC-3: Proc. 3rd Int. Conf. on Algebraic Algorithms and Error-Correcting Codes (Grenoble, France, July 15-19, 1985)*, volume 223 of *Lecture Notes in Comput. Sci.*, page 353-362. Springer, 1986. 5

[ZB96] A. Y. Zharkov and Y. A. Blinkov. Involution approach to investigating polynomial systems. *Math. Comput. Simulation*, (42) 4-6:323-332, 1996. 5

Index

AlgebraIBNP, 21

CommutativeDivision, 10
conventional divisor, 10

disjoint cones, 33
DivisionRecord, 12
DivisionRecordCP, 12
DivisionRecordNP, 29
DivNM, 24

GitHub repository, 4
GM2NM, 22
GM2NMList, 22
GP2NP, 6
GtNPoly, 26

homogeneous polynomials, 19

IAutoreduce, 14
IAutoreduceCP, 14
IAutoreduceNP, 31
involutive divisor, 10
InvolutiveAbortLimit, 17
InvolutiveBasis, 16
InvolutiveBasisCP, 16
InvolutiveBasisNP, 32
IPolyReduce, 13
IPolyReduceCP, 13
IPolyReduceNP, 30
IsSubwordNM, 23

JanetDivision, 11

LeadExpNM, 24
LeadVarNM, 24
LeftDivision, 28
LeftOverlapDivision, 28
LoggedIPolyReduce, 13
LoggedIPolyReduceCP, 13
LoggedIPolyReduceNP, 30
LowestLeadMonomialPosNP, 26
LtNPoly, 26

MaxDegreeNP, 25

NM2GM, 22
NM2GMList, 22
NoncommutativeDivision, 29
NP2GP, 6

orderings, 10

PommaretDivision, 10
PrefixNM, 22
PrintNM, 21
PrintNMList, 21

RightDivision, 28
RightOverlapDivision, 29

ScalarMulNP, 25
StrongLeftOverlapDivision, 33
StrongRightOverlapDivision, 35
SubwordNM, 22
SubwordPosNM, 23
SuffixNM, 22
SuffixPrefixPosNM, 23

TailNM, 24
ThomasDivision, 11

VerifyLoggedRecordNP, 31