THE NCAlgebra SUITE

NCAlgebra 3.7

Mathematica's Control System Professional
Noncommutative support 1.0

NCGB NonCommutative Groebner Bases 3.1

SYStems - ON THE WEB

`http://math.ucsd.edu/~ncalg`

Download the software and
Register there for updates

September 6, 2002

Contents

V NCGB: FOR THE MORE ADVANCED USER 169

VI NCGB: LISTS OF COMMANDS AND OTHER DETAILS 191

Preface

NCAlgebra is a package which runs under Mathematica. It is an algebra program designed to facilitate manipulation and reduction of noncommutative algebraic expressions. Specifically, it allows computer calculation in an ALGEBRA WITH TRANSPOSES OR ADJOINTS. Such computations are common in many areas but our background is operator theory and engineering systems so we are aiming at applications in these areas rather than at the theory of symbolic computation.

A noncommutative Gröbner basis package is also available which is C++ linked to NCAlgebra. At the moment we trust it under the Solaris operating system or Linux only. Someday we hope to support Microsoft Windows.

We have added files which allow one to use some of the functionality of the Mathematica package Control System Professional with noncommuting indeterminates.

Also included is a package for doing differentiation with complex variables. This package manipulates expressions in terms of the variable z and $\bar{z}$ rather than by using real and imaginary parts.

We are including a collection of files for doing system engineering. These are focused specifically on computations which occur in doing H^∞ control research. Our intent is not to produce a symbolic introduction to system theory but to focus on special areas of our own research with the hope that others will write more general systems packages. The interested user should read SYSDOC.dvi which is included and see the appendix on running SYSTEMS.

We see this package as a competitor to the yellow pad. Once you get used to it this might be considerably more effective for some types of hand calculations. Like Mathematica the emphasis is on interaction with the program and flexibility (see the section on Editing in the Helpful Tricks chapter (Chapter 5)).

NCAlgebra uses a slight modification of the Mathematica operation NonCommutativeMultiply (denoted by **). Many of the NCAlgebra functions are noncommutative versions of Mathematica functions used for simplification of commutative algebraic expressions. For example, the functions NCExpand and NCCollect extend the utility of the usual Mathematica functions Expand and Collect to algebraic expressions including noncommutative multiplications. NCExpand expands complicated (multi-parentheses) expressions and thus facilitates additive cancellation of terms. NCCollect does the opposite– it collects like terms. In addition, a number of more specialized functions are designed to assist in solving particular types of algebraic problems. These currently include capabilities for block matrix manipulation, multi-dimensional differentiation, and specialized applications in systems theory.

IMPORTANT NOTE/WARNING: The files which end with the suffix ".Extra" files which are provided with the NCAlgebra package are not supported!!! So the user must beware!!! They are a combination of routines which we may support in the future and "things that seemed like they might be useful sometimes but may not be of use or appropriate to use in every situation." See also APPENDIX F.

In Chapter 1 we present a few simple examples. These actually contain more than you need to know to have a good time with NCAlgebra. Once you have read about two pages you are already WELL INTO NCAlgebra.

Part I

NCAlgebra

NCALGEBRA

Version 3.7
(Mathematica 2.2, 3.0, and 4.0 compatible)

J. William Helton and Mark Stankus
Math Dept., UCSD

Robert L. Miller
General Atomic Corp.
La Jolla, California 92093

If you would like to try the NCAlgebra package or want updates go to the NCAlgebra web site

http://math.ucsd.edu/∼ncalg

or contact ncalg@ucsd.edu or MathSource@wri.com.

The basic NCAlgebra program was written by the authors and David Hurst, Daniel Lamm, Orlando Merino, Robert Obarr, Henry Pfister, Stan Yoshinobu, Phillipe Bergman, Dell Kronewitter, and Eric Rowell. Various additions include contributions by Dave Glickstein, Juan Camino, Jeff Ovall, Tony Mendes, and Tony Shaheen. The Mathematica C++ hybrid was developed with Kurt Schneider, Victor Shih and Mike Moore. Control demos are coauthored with Mike Walker. Simplification commands are based on theory developed by Bill Helton, John Wavrik and Mark Stankus. The beginnings of the program come from eran@slac. This program was written with support from the AFOSR, the NSF, the Lab for Mathematics and Statistics at UCSD, the UCSD Faculty Mentor Program and the US Department of Education.

0.1 ReleaseNotes NCAlgebra

ReleaseNotes NCAlgebra3.7 Sept. 2002

RandomMatrix[m,n,min,max,options] RandomMatrix allows you to create random matrices of size $m \times n$ with entries between min and max. The options allow the user to create matrices with integer or real entries. They also allow the creation of diagonal or symmetric matrices. Note: This is a utility that is not connected with noncommutative algebra.

Convexity We changed some defaults and fixed some bugs in NCConvexityRegion and NCMatrixOfQuadratic. Now it runs quickly on fairly big problems.

Path Setting Paths were set to make NCAlgebra more path independent.

CSP Our support for Control System Professional was updated to accomodate the second (2002) version of CSP.

ReleaseNotes NCAlgebra3.5 Sept. 2001

Basic Changes You no longer have to set every variable to be noncommutative. We have a command NCGuts which has an option called NCSetNC. When set to True, all letters are automatically noncommutative unless SetCommutative makes them commutative.

A further option of NCGuts allows one to use "**" to multiply matrices with noncommutative entries – the more cumbersome MatMult command is no longer needed. While this option seems dangerous to Bill, it makes many computations prettier and easier to type. If you don't trust the answer, then don't use the option.

Commands For Matricies With Noncommuting Entries We now have an LDU decomposition for matricies with noncommuting entries. Also, there is a command for computing the inverse of such matrices (however this only works under strong assumptions).

NCMatrixOfQuadratic gives a vector matrix factorization of a symmetric quadratic noncommutative function.

A Second Derivative Command NCHessian computes the Hessian of a function with noncommuting variables and coefficents.

Computing The Region Where A Noncommutative Function is Convex NCConvexityRegion is a command used to determine the region of formal noncommutative inequalities where a given noncommutative function is convex.

Basic Changes

`NCGuts:` NCGuts holds set of options geared for simplifying transposing, finding the inverse, and multiplying matrices conaining noncommuting variables.

NCStrongProduct1 $->$ False is the first option of NCGuts. When True, ** serves to multiply matrices with noncommutative entries as well as maintaining its original function. In addition, tp[ ] and tpMat are the same. The default setting is True.

NCStrongProduct2 $->$ False is the second option of NCGuts. When set to True, if M is a matrix with noncommutative entries, inv[M] returns a formula expression for the inverse of M. NCStrongProduct2 forces NCStrongProduct1.

NCSetNC $->$ False is the last option of NCGuts. When set to True, all letters are automatically noncommutative unless SetCommutative makes them commutative. This replaces the need for repeated calls to SetNonCommutative.

Commands For Matricies With Noncommuting Entries

`NCLDUDecomposition:` Given a square matrix M with noncommutative entries, this command finds the LDU decomposition of M. It returns a list of four elements, namely L, D, U, and P such that $PXP^T = LDU$. The first element is the lower triangular matrix L, the second element is the diagonal matrix D, the third element is the upper triangular matrix U, and the fourth is the permutation matrix P (the identity is returned if no permutation is needed). As an option, it may also return a list of the permutations used at each step of the LDU factorization as a fifth element.

`NCAllPermutationLDU:` NCAllPermutationLDU returns the LDU decomposition of a matrix after all possible column permutations are applied. The code cycles through all possible permutations and calls NCLDUDecomposition for each one. As an option, the permutations used for each LDU decomposition can also be returned.

`NCMatrixOfQuadratic:` NCMatrixOfQuadratic gives a vector matrix factorization of a symmetric quadratic noncommutative function. A three element list is the output. The first element is the left border vector, the second element is a symmetric coefficent matrix, and the third is the right border vector. The border vectors contain the variables in the given quadratic function and their transposes.

`NCIndependenceCheck:` NCIndependenceCheck verifies whether or not a given set of polynomials are independent or not. It analyzes each list of polynomials separately. There are three possible types of outputs for each list. Two of them correspond to NCIndependenceCheck successfully determining whether or not the list of polynomials is independent. The third type of output corresponds to an unsuccessful attempt at determining dependence or independence.

`NCBorderVectorGather:` NCBorderVectorGather can be used to gather the polynomial coefficents preceeding the elements given in a list of variables whenever they occur.

`NCPermutationMatrix:` NCPermutationMatrix returns the permutation matrix associated with the list of the first n integers. It gives the identity matrix with its columns re-ordered.

`NCMatrixToPermutation:` NCMatrixToPermutation returns the permutation associated with the permutation matrix, aMatrix. It is the inverse of NCPermutationMatrix.

`NCInverse:` NCInverse gives a symbolic inverse of a matrix with noncommutative entries.

A Second Derivative Command

`NCHessian:` NCHessian computes the Hessian of a function with noncommuting variables and coefficents. This is a second directional derivative which can be thought of as the second order term in the noncommutative Taylor expansion. Output will be a symmetric quadratic function with respect to the directions of differentiation.

Computing The Region Where A Noncommutative Function is Convex

`NCConvexityRegion:` This command is used to determine the region of formal noncommutative inequalities where a given noncommutative function is convex. NCConvexityRegion preforms three main operations. Given a noncommutative function F, the Hessian of F is computed with NCHessian. Then, using NCMatrixOfQuadratic, the Hessian is factored into vector matrix vector form. Finally, NCAllPermutationLDU finds the LDU decomposition of the symmetric coefficent matrix. The diagonal elements in the diagonal matrix in the LDU decomposition is returned.

ReleaseNotes NCAlgebra3.0

NCAlgebra 3.0 has several added functions.

1. LDU decomposition for block matices, to include a block Cholesky decompsition.
2. Formulas for inverses of block matrices.
3. A command which differentiates functions of the form
$$trace \;\; P(X, Y, etc)$$
$$log \;\; det \;\; P(X, Y, etc)$$
4. Support for the Mathematica toolbox Control System Professional. It gives CSP the ability to handle non-commuting objects.
5. A function which represents elements of an algebra as $n \times n$ matrices with commuting symbolic entries, or with inegers.
6. Online Help - While we have not set up help browsers at this time in the Mma style, one can get searchable online help by viewing NCBIGDOCUMENT.html with Netscape, etc. When you are in an NCAlgebra session just keep a web browser open with NCBIGDOCUMENT.html loaded in. The powerfulsearch features of these browsers allow you to look up things in the document.

An X in commands, e.g. NCXetc., always means that this command is experimental and we reserve the right to change it.

Chapter 1

The Most Basic NCAlgebra Commands — A short tutorial

NCAlgebra is a *Mathematica* package which allows one to do noncommutative algebraic computations. We will begin by presenting some calculations done with our package which should give some idea of what may be accomplished with NCAlgebra. We suggest that the reader later do these computations for himself with our package when it has been properly installed since it will provide a tutorial as well as an illustration.

In our package ** denotes noncommutative multiply, tp[x] denotes the transpose of an element x, and aj[x] denotes the adjoint of an element x. Note that the properties of transposes and adjoints that everyone constantly uses are built-in. The multiplicative identity is denoted Id in the program. At the present time, Id is set to 1. A element A may have an inverse, which will be denoted by inv[A], of it may have a left or right inverse, denoted inv[A] and invR[A], respectively.

The following examples are independent of each other, however they may be executed in one continuous session. At present, single-letter lower case variables are non-commutative by default and all others are commutative by default.

A Mathematica 3.0 user inside a notebook can use our special Palette by opening the file NCPalette.nb (see Section 2).

To begin start Mathematica and load NCAlgebra.m or NCGB.m.

```
In[1]:= a ** b - b ** a
Out[1]= a ** b - b ** a

In[2]:= A ** B - B ** A
Out[2]= 0

In[3]:= A ** b - b ** a
Out[3]= A b - b ** a

In[4]:= CommuteEverything[a ** b - b ** a]
Out[4]= 0

In[5]:= SetNonCommutative[A, B]
Out[5]= {False, False}

In[6]:= A ** B - B ** A
Out[6]= A ** B - B ** A
```

```
In[7]:= SetNonCommutative[A];SetCommutative[B]
Out[7]= {True}

In[8]:= A ** B - B ** A
Out[8]= 0
```

SNC is an alias for SetNonCommutative. So, SNC can be typed rather than the longer SetNonCommutative.

```
In[9]:= SNC[A];
In[10]:= A ** a - a ** A
Out[10]= A ** a - a ** A

In[11]:= SetCommutative[v];
In[12]:= v ** b
Out[12]= v b

In[13]:= NCCollect[a ** x + b ** x, x]
Out[13]= (a + b) ** x

In[14]:= NCExpand[(a + b) ** x]
Out[14]= a ** x + b ** x

In[15]:= NCCollect[tp[x] ** a ** x + tp[x] ** b ** x + z, {x, tp[x]}]
Out[15]= z + tp[x] ** (a + b) ** x

In[16]:= DirectionalD[x ** x, x, h]
Out[16]= h ** x + x ** h

In[17]:= Grad[tp[x] ** x + tp[x]**A**x + m**x, x]
         (* Here A is noncommutative and x represents a column vector*)
Out[17]= 2 x + A ** x + tp[A] ** x + tp[m]
```

Warning: Grad is trustworthy only on certain quadratics.

IMPORTANT: The Mathematica substitute commands \. → and \ :> are not reliable in NCAlgebra, so you must use our substitute command.

```
In[18]:= Substitute[x ** a ** b, a ** b -> c]
Out[18]= x ** c

In[19]:= Substitute[ tp[b ** a] + b ** a, b ** a -> p]
Out[19]= tp[a] ** tp[b] + p

In[21]:= SubstituteSymmetric[tp[b] ** tp[a] + w + a ** b, a**b->c]
Out[21]= c + w + tp[c]

In[23]:= MatMult[{{a, b}, {c, d}}, {{d, 2}, {e, 3}}]
Out[23]= {{a ** d + b ** e, 2 a + 3 b}, {c ** d + d ** e, 2 c + 3 d}}

In[24]:= tp[a ** b]
Out[24]= tp[b] ** tp[a]
```

```
In[25]:=tp[5]
Out[25]= 5

In[26]:= tp[2 + 3 I]
Out[26]= 2 + 3 I

In[27]:= tp[a]
Out[27]= tp[a]

In[28]:= tp[a + b]
Out[28]= tp[a] + tp[b]

In[29]:= tp[6 x]
Out[29]= 6 tp[x]

In[30]:= tp[tp[a]]
Out[30]= a

In[31]:= aj[5]
Out[31]= 5

In[32]:= aj[2 + 3 I]
Out[32]= 2 - 3 I

In[33]:= aj[a]
Out[33]= aj[a]

In[34]:= aj[a + b]
Out[34]= aj[a] + aj[b]

In[35]:= aj[6 x]
Out[35]= 6 aj[x]

In[36]:= aj[aj[a]]
Out[36]= a

In[37]:= Id
Out[37]= 1

In[38]:= inv[a ** b]
Out[38]= inv[b] ** inv[a]

In[39]:= inv[a] ** a
Out[39]= 1

In[40]:= a ** inv[a]
Out[40]= 1

In[41]:= a ** b ** inv[b]
Out[41]= a

In[42]:= invL[a] ** a
Out[42]= 1
```

```
In[43]:= a ** invR[a]
Out[44]= 1

In[45]:= a ** invL[a]
Out[45]= a ** invL[a]

In[46]:= invR[a] ** a
Out[46]= invR[a] ** a

In[47]:= f1 = 1 + inv[d] ** c ** inv[S - a] ** b - inv[d] ** c **
    inv[S - a + b ** inv[d] ** c] ** b - inv[d] ** c **
    inv[S - a + b ** inv[d] ** c] ** b ** inv[d] ** c **
    inv[S - a] ** b;
In[48]:= NCSimplifyRational[f1]
Out[48]= 1

In[49]:= f2 = inv[1 + 2 a] ** a;
In[50]:= NCSimplifyRational[f2]
Out[50]= (1 - inv[1 + 2 a]) / 2
```

NCSR is the alias for NCSimplifyRational.

```
In[51]:= f3 = a ** inv[1 - a];
In[52]:= NCSR[f3]
Out[52]= inv[1 - a] - 1

In[53]:= f4 = inv[1 - b ** a] ** inv[a];
In[54]:= NCSR[f4]
Out[54]= inv[a] ** inv[1 - a ** b]

In[55]:= NCSolve[a ** x == b, x]
Out[55]= {x -> inv[a] ** b}
```

Note: NCSolve applies to some linear equations in only one unknown.

1.1 Pretty Output

Beautifying NCAlgebra

```
In[104]:= SetOutput[ all -> True ];
In[105]:= inv[ tp[k] ]
             T -1
Out[105]= (k)


In[106]:= SetOutput[ all -> True ];
In[107]:= rt[x - inv[y + aj[z]]] + tp[x]
            T              * -1 1/2
Out[107]=  x  + (x - (y + z )  )


In[108]:= SetOutput[ all -> True, inv -> False ];
In[109]:= inv[1 + tp[1-x]]
```

```
                   T
Out[109]= inv[2 - x ]
```

TeX

```
In[110]:= mat = {{a + tp[b],aj[c]},{inv[d],e}}
Out[110]= {{a + tp[b], aj[c]}, {inv[d], e}}

In[111]:= Get["Extra.TeXForm"];LookAtMatrix[mat]
```

outputs

```
This is TeX, C Version 2.96 (no format preloaded)
(file.tex
LaTeX Version 2.09 <29 Apr 1987>
(/usr/local/lib/tex/macros/latex/article.sty
Document Style ‘article’. Released 4 September 1986.
(/usr/local/lib/tex/macros/latex/art12.sty)) (file.aux) [1] (file.aux)
Output written on file.dvi (1 page, 540 bytes).
Transcript written on file.log.
```

The above command takes the Mathematica matrix *mat*, converts it to LaTeX, outputs the string and surrounding LaTeX to a file, LaTeXs the file and displays the LaTeXed output to the screen using the program xdvi (this can be easily changed to other previewers such as dvipage or xpreview). This window contains the following formula in its display.

$$\begin{pmatrix} a+b^T & c^* \\ d^{-1} & e \end{pmatrix}$$

The LaTeX file generated can be displayed by typing `!!file.tex` which produces

```
\documentstyle [12pt]{article}
\begin{document}
$$
\pmatrix { a + b^T & c^{*} \cr d^{-1} & e \cr }
$$
\end{document}
```

```
In[113]:= LookAtLongExpression[Sum[a[i],{i,1,50}]];
```

generates a window which contains the following formula in its display.

$$a(1)+a(2)+a(3)+a(4)+a(5)+a(6)+a(7)+a(8)+$$
$$a(9)+a(10)+a(11)+a(12)+a(13)+a(14)+a(15)$$

$$
\begin{gathered}
+a(16)+a(17)+a(18)+a(19)+a(20)+a(21)+ \\
a(22)+a(23)+a(24)+a(25)+a(26)+a(27)+ \\
a(28)+a(29)+a(30)+a(31)+a(32)+a(33)+ \\
a(34)+a(35)+a(36)+a(37)+a(38)+a(39)+ \\
a(40)+a(41)+a(42)+a(43)+a(44)+a(45)+ \\
a(46)+a(47)+a(48)+a(49)+a(50)
\end{gathered}
$$

The above examples in this chapter demonstrate the most commonly used features in NCAlgebra.

Chapter 2

Getting Started in NCAlgebra and NCGB

This section contains instructions for getting started using NCAlgebra and NCGB in conjunction with Mathematica. Mathematica functions are documented in the book "Mathematica; A System for Doing Mathematics by Computer" by Stephen Wolfram. This book is not essential, but should be used as a reference in addition to this user's guide after you have some experience.

In order to use the NCAlgebra functions, you must first have access to a computer on which the Mathematica program is installed.

Then you must install NCAlgebra on your computer. It is assumed in this section that this has been done. How to do it is discussed in Section 29.

2.1 NC On the UNIX Platform

(1) Log on to the computer which has Mathematica.

(2) Move into the directory NC or into the directory where the file NCAlgebra.m resides.

(3) Type the command which executes the Mathematica program. (Usually it is just the command 'math'.)

You will see a display of the form

```
In[1] :=
```

which is a Mathematica prompt, asking for your inputs. When you type something followed by pressing the "return" key, Mathematica will respond with an output of the form

```
Out[1] = ...
```

The following are the initial NCAlgebra commands:
(4) Type:

```
Get["SetNCPath.m"]
Get["NCAlgebra.m"]
```

or if you want to run NCGB (as of Dec. 99 only on Solaris and Windows) type

```
Get["SetNCPath.m"]
Get["NCGB.m"]
```

"Get" above means that Mathematica should bring in the contents of the file specified (also you can use double arrows "<<") .

2.2 On the Windows platform

(1) Start Windows.

(2) Move into the directory NC/work by clicking on `MyComputer -> C: -> NC -> work`.

(3) Double click on the file GoNCAlgebra. This should start Mathematica and put you into the GoNCAlgebra notebook. The first cell of the notebook contains loading commands for the files SetNCPath.m and NCGB.m. Execute the first cell to start NCGB. If you insist on using only NCAlgebra, modify NCGB.m in the first cell to NCAlgebra.m.

In either Unix or Windows **Palettes** are recommended for the beginner who is in a notebook environment. To use them load in NCPalette.nb, which is a file in the NCAlgebra directory, via the command

```
NotebookOpen["NCAlgebra/NCPalette.nb"]
```

2.2.1 When trouble strikes

Your session may not start if you are **not in the NC directory.** Then you will have to edit the file SetNCPath.m as described inside that file. Also be sure you actually load SetNCPath.m, for example, if NC is a subdirectory of your "homedirectory" do this

```
Get["homedirectory/NC/SetNCPath.m"]
Get["NCAlgebra.m"]
```

You are now ready to begin a Mathematica session using NCAlgebra For example, you are at the place where you can do the examples in the previous chapter.

About the only thing which can go seriously wrong is that the path assignment statement in NCAlgebra.m does not reach the big collection of NC*.m files inside the NCAlgebra directory. The assignment statement in the file NCAlgebra.m is

```
System‘path = ""
```

and it may need to be adapted to the directories you have set up or to the naming conventions of your operating system.

The typical user will need to edit the file SetNCPath.m. Instructions for this are given in the file. You will probably have to edit a line which looks like this

```
$NC$Dir = "/home/jane/NC/";
```

to something which looks like this

```
$NC$Dir = "/home/dick/NC/";
```

For those knowledgeable in Mathematica, I typically use an "init.m" file in my home directory which looks like this

AppendTo[$ Path, "/home/dick/NC"]
¡¡SetNCPath.m

Notice that even in the Windows environment `/home/dick/NC` is a valid directory though the experienced DOS user would expect `\home\dick\NC`. The experienced Windows and Mathematica user would expect `\\home\\dick\\NC`, but we'll leave that for the Mathematica manual. The Mathematica language is quite platform independent.

2.3 On Screen Help

While we have not set up help browsers at this time in the Mma style, one can get searchable online help by viewing NCBIGDOCUMENT.html with Netscape (or another browser). When you are in an NCAlgebra session just keep a web browser open with NCBIGDOCUMENT.html loaded in. The powerful search features of these browsers allow you to look up things in the document.

Chapter 3

Installing NCGB

Here we will discuss the Groebner basis portion of the NCAlgebra package which is more technical. The Groebner basis portion is not just Mathematica code but also includes a binary kernel which was written in C++.

3.1 Installing NCGB the Easy Way

We assume that you have obtained NCGB by following the directions in "Getting NCAlgebra and NCGB" in part I.

It is possible that your installation is complete and the full power of NCGB is at your fingertips. For the sake of those who are not so lucky we will go through the following sequence of Mathematica input and offer remedies if your session does not behave as expected. Begin by typing

```
<<NCGB.m
```

This will load in a bunch of files some of which were loaded in earlier, some of which are new. The last few lines should include something like

```
LinkObject[/home/ncalg/NC/NCGB/Binary/p9c, 1, 1]
```

If you see an error at this point you may need to edit the file SetNCPath.m. The Windows user might have to comment out the two lines

```
(*
$NC$Binary$Dir$ = $NCDir$ <> "NCGB/Compile/debug/";
$NC$Binary$Name$ = "NCGB.exe";
*)
```

although we will try to do it for him or her. After editing and saving the file, restart Mathematica (or at least Stop the Kernel).

After `<<NCGB.m` executes correctly, enter the following two commands.

```
SetMonomialOrder[ a, b ];


NCProcess[ { a**b + 1 }, 1, "TestFile" ];
```

The execution of these commands should create a file TestFile.tex, latex it, and display a dvi window (a window containing mathematical formulas). **If you are in the Windows environment you will have to quit the dvi previewer program (yap) to continue your Mathematica session. The notebook will be locked up until you quit the dvi previewer.**

If there is still a problem, then perhaps you do not have latex or a dvi previewer. If this is the case see Section 29. Another possibility is that NCGB doesn't know where "latex" or either "xdvi" or "yap" is. If you *can* find both of these programs you can edit the file homedirectory/NC/NCGB/MmaSource/NCGBPlatformSpecific.m to tell NCGB where they live. A windows user might have to change (or uncomment) the line

```
NCGBSetLatexCommand[ "c:\\texmf\\miktex\\bin\\latex.exe" ];
```

or

```
NCGBSetDviCommand["c:\\texmf\\miktex\\bin\\yap.Exe"];
```

Editing this file, restarting Mathematica, and executing the above commands should open a dvi file automatically as mentioned above.

The dvi output is beautiful and worth the effort of getting a latex/xdvi/yap distribution. Nevertheless, the unmotivated NCGB user may make use of the NCGB commands which do not use latex/dvi without the latex/xdvi/yap distribution. For example, you can compute noncommutative Groebner bases without the latex distribution.

For more detailed tests of your new NCGB installation see Chapter 30.

3.2 Getting latex and a dvi previewer

Most UNIX systems come with the programs "latex" and "xdvi" which are used by some of the NCGB commands. A nice Windows version of these programs is freely available in the MikTeX latex distribution which can be found at

`http://www.miktex.de/`

3.3 Running SYStems, OldMmaGB

These are run by loading in the files SYStems.m and OldMmaGb.m. See Section 29 for more details.

3.4 Warning: NC* variables

The developers reserve the use of any variable beginning with the four letters NC.

Chapter 4

Examples

At this point looking at demos is very helpful. We recommend working thru the NCAlgebra demos which accompany this document. Some of them are postscript files and can be read (and can be found in the directory NC/DOCUMENTATION), or they can be run by loading NCPalette.nb. and then going to the Demo menu.

All demos can be found by listing all notebook files in the NC/NCAlgebra directory, that is list *.nb files. These are all the demos. The NCAlgebra website, `http://math.ucsd.edu/~ncalg` also contains some demos. As of August 1999, the following is a subset of the demos which are available:

`DemoBRL.nb`
`DemoGB1.nb`
`SingPert.nb`
`DemoGBM.nb*`
`DemoSimplify.nb`
`DmGBG1.nb`
`DmGBG2.nb`
`NCPalette.nb`
`PaletteSource.nb`

Chapter 5

Helpful Tricks

5.1 Expanding

NCExpand is the most common command applied to expressions. Often you must do it to achieve the most obvious cancellations. See page 14 of Chapter 1 or Section 5.1.1.

5.2 Simplifying Expressions

A person experienced in noncommutative calculations simplifies expressions in two ways:
1. Write the expression in the shortest possible form with special attention given to subexpressions with physical or special mathematical meaning.
2. The other is to expand expressions, apply simplifying rules repeatedly to each term, and see which terms cancel.

5.2.1 Simplifying Rules

The second method is the one which for commuting algebras has been developed to a high art in computer calculation. The idea is very simple and intuitive. Simplification is done with rules which replace complicated monomials with sums of simpler monomials, e.g.,

```
inv[1-x] ** x -> inv[1-x]-1
inv[a+b ** c] ** b ** c -> 1-inv[a+b ** c] ** inv[a]
```

throughout the expression to be simplified. When you use NCAlgebra you will often be making up such rules and substituting them in expressions. In a fixed collection of applications you can make your life easier if you save the rules and use them over and over again. The best way to do this is to put them in a function, say

```
MyRules=
```

```
        {inv[1-x_] :> inv[x]-1,
         inv[a+b ** c] ** b ** c -> 1-inv[a+b ** c] ** inv[a]};

    MySimplify[expr_]:=Substitute[expr, MyRules];
```

One of the trickier fine points is how to set the blanks in your rules. If you do not use blanks that's fine provided you always use the same letters and do not replace them with other notation in some equation. Clearly using blanks is much more powerful. The trick is how many. For example, x_ is ok here. APPENDIX E discusses this.

5.2.2 Orders

The next major point is not to go into a loop. To this end one must select an ordering, call it COM, on monomials. For mnemonic purposes it is best to select the ordering to reflect your intuitive idea of which monomials are more complicated than others. For example if all of your formulas involve polynomials in

```
            x, inv[x], inv[1-x ** y], inv[1-y ** x],
                      y,  inv[y]
```

a natural partial ordering is given by low degree < high degree

We then subdivide equivalence classes of this ordering with

```
                           x            inv[x]           inv[1-x ** y]
commutative expr  <                <                <
                           y            inv[y]           inv[1-y ** x]
```

then we subdivide equivalence classes of this ordering with lexicographical order, i.e , x < y.

A reasonable convention is that higher order expressions move RIGHT.

For example, a basic equality is

```
   inv[1-x ** y] ** x- x ** inv[1 - y ** x]==0 .
```

This translates to the rule

```
   inv[1-x ** y] ** x -> x ** inv[1-y ** x]
```

because inv[1-x ** y] is 'complicated' and we move it RIGHT. To harp on an earlier point we would suggest using the more powerful delayed assignment form of the rule:

```
inv[1-x__ ** y_] ** x__ :> x ** inv[1- y ** x]
```

IMPORTANT: these are the ordering conventions we use in NCSR. If you write rules consistent with them then you will then you can use them and NCSR without going into a loop. Indeed NCSR contains a "Gröbner basis" for reducing the set of polynomials in the expressions (inv).

Here is a summary of the ordering conventions ranked from most complicated to the least:

```
high degree>low degree
inv of complicated polynomials
inv of simple polynomials
complicated polynomials
simple polynomials
commuting elements and expressions in them.
```

REMEMBER HIGHER ORDER EXPRESSIONS MOVE RIGHT.

5.2.3 Automatic generation of rules

Automatic generation of rules is the subject of the NCGB part of this document. Since running the NCGB code requires C++, you may not have it. Here *NCSimplifyRationalX1[]* does the trick.

Lying around in the directory NC/NCAlgebra/OldmmaGB/ is a primative *NCSimplifyRationalX1[]* which works entirely under Mma. We don't support it since our efforts go to Mma C++ hybrids. We do not even recall its name. Anyone who resurrects it must be an intrepid adventurer.

5.3 Edit - For those without Notebooks

The failsafe command Edit does not get enough emphasis in the Mathematica literature. This command guarantees that Mathematica is never worse than a yellow pad. Whenever you have an expression 'expr' and the functions at your disposal are not doing what you want just enter

```
In[102]:=Edit[expr]
```

Mathematica throws you into a file containing expr. You can edit it with the vi or emacs editor or whatever is set up. Then exiting the file throws your edited expression into the Out[102] (see above). A truly remarkable feature is that

YOU CAN EDIT Mathematica FUNCTIONS (INCLUDING NCAlgebra FUNCTIONS) INTO EXPR, APPLYING DIFFERENT FUNCTIONS TO DIFFERENT PARTS OF EXPR, then these are automatically executed when you finish editing the file. A tutorial example of this extremely powerful feature is

```
Out[32]= x**y + x**z + x**y**x
```

```
In[33]:= Edit[%]
```

A new screen comes up and you can use your resident editor on it.

```
x**y + x**z + x**y**x
```

I usually make another copy of the expression for safety sake and make edits on one of them, while commenting out the second so it does not get read by Mathematica. This way if I make errors, I still have the original expression to fall back on and check with. This is especially useful when dealing with complicated expressions. For example, you could write

```
NCCollect[x ** y + x ** z,x] + x ** y **x;
(* x**y + x**z + x**y**x *)
```

Now quit editing and close the file, (e.g., :wq for vi).

```
Out[33]: x ** (y + z) + x ** y ** x
```

5.4 Conventions

The NCAlgebra files which are called by NCAlgebra.m start with NC. This makes moving them easier; cp NC* someplace/ where "someplace" is any directory of your choosing. Many operations on expressions start with NC .

Aliases are all caps like NCC for NCCollect or NCE (for NCExpand). The caps correspond exactly to the caps in the full function name. Exceptions are cases like Sub or SubSym where CAPs are followed by 2 lower case letters. This prevents ambiguities and two letter aliases.

Function names are written in a certain order: Command or action you wish taken comes first. The special properties of what you apply it to are second.

For example, let's look at NCSimplifyRational. The action is Simplify. The range of validity is "Rational" functions.

Files whose only function is to call other files have names which are all capital letters.

Chapter 6

NC Commands

Mathematica 3.0 has a lovely graphical user interface which uses Palettes. Mathematica Palettes display the most important commands and prompt the user. We have such a Palette for NCAlgebra and NCGB which contain most of the commands in this chapter. See the TEAR OFF Section in the back for a picture of the Mma Palettes for NCAlgebra and NCGB. To pop up this Palette, open a notebook, load NCAlgebra or NCGB, then open the file NCPalette.nb. If you are in a directory containing the file NCPalette.nb you can open it directly from a notebook.

6.1 Manipulating an expression

6.1.1 ExpandNonCommutativeMultiply[expr]

Aliases: **NCE,NCExpand**

Description: $ExpandNonCommutativeMultiply[expr]$ expands out NonCommutative-Multiply's in *expr*. It is the noncommutative generalization of the Mma Expand command.

Arguments: *expr* is an algebraic expression.

Comments / Limitations: None

6.1.2 NCCollect[expr, aListOfVariables]

Aliases: **NCC**

Description: $NCCollect[expr, aListOfVariables]$ collects terms of expression *expr* according to the elements of $aListOfVariables$ and attempts to combine them using a particular list of rules called *rulesCollect*. NCCollect is weaker than NCStrongCollect in that first-order and second-order terms are not collected together. NCCollect uses NCDecompose, and then NCStrongCollect, and then NCCompose.

Arguments: *expr* is an algebraic expression. $aListOfVariables$ is a list of variables.

Comments / Limitations: While $NCCollect[expr, x]$ always returns mathematically correct expressions, it may not collect x from as many terms as it should. If *expr* has been expanded in the previous step, the problem does not arise. If not, the pattern match behind NCCollect may not get entirely inside of every factor where x appears.

6.1.3 NCStrongCollect[expr, aListOfVariables]

Aliases: **NCSC**

Description: It collects terms of expression *expr* according to the elements of $aListOfVariables$ and attempts to combine them using the particular list of rules called rulesCollect. In the noncommutative case, the Taylor expansion, and hence the collect function, is not uniquely specified. This collect function often collects too much and while mathematically correct is often stronger than you want. For example, x will factor out of terms where it appears both linearly a quadratically thus mixing orders.

Arguments: *expr* is an algebraic expression. $aListOfVariables$ is a list of variables.

Comments / Limitations: Not well documented.

6.1.4 NCCollectSymmetric[expr]

Aliases: **NCCSym**

Description: None

Arguments: *expr* is an algebraic expression.

Comments / Limitations: None

6.1.5 NCTermsOfDegree[expr,aListOfVariables,indices]

Aliases: **None**

Description: $NCTermsOfDegree[expr, aListOfVariables, indices]$ returns an expression such that each term is homogeneous of degree given by the *indices* in the variables of $aListOfVariables$. For example, $NCTermsOfDegree[x**y**x + x**x**y + x**x + x**w, \{x, y\}, indices]$ returns $x**x**y + x**y**x$ if $indices = \{2, 1\}$, return $x**w$ if $indices = \{1, 0\}$, return $x**x$ if $indices = \{2, 0\}$ and returns 0 otherwise. This is like Mathematica's Coefficient command, but for the noncommuting case. However, it actually gives the terms and not the coefficients of the terms.

Arguments: *expr* is an algebraic expression, $aListOfVariables$ is a list of variables and indices is a list of positive integers which is the same length as *aList*.

Comments / Limitations: Not available before NCAlgebra 1.0

6.1.6 NCSolve[expr1==expr2,var]

Aliases: **None**

Description: $NCSolve[expr1 == expr2, var]$ solves some simple equations which are linear in the unknown *var*. Note that in the noncommutative case, many equations such as Lyapunov equations cannot be solved for an unknown. This obviously is a limitation on the NCSolve command.

Arguments: $expr1$ and $expr2$ are Mathematica expressions. *var* is a single variable.

Comments / Limitations: See description.

6.1.7 Substitute[expr,aListOfRules,(Optional On)]

Aliases: **Sub**

Description: It repeatedly replaces one symbol or sub-expression in the expression by another expression as specified by the rule. (See Wolfram's Mathematica 2.* book page 54.) More recently, we wrote the Transform command (§6.1.11) which apprears to be better.

Arguments: *expr* is an algebraic expression. $aListOfRules$ is a single rule or list of rules specifying the substitution to be made. On = save rules to Rules.temp, temporarily over-riding SaveRules[Off]. ‘Off’ cannot over-ride SaveRules[On].

Comments / Limitations: The symbols /. and //. are often used in Mathematica as methods for substituting one expression for another. This method of substitution often does not work when the expression to be substituted is a subexpression within a (noncommutative) product. This Substitute command is the noncommutative analogue to //.

6.1.8 SubstituteSymmetric[expr, aListOfRules, (optional On)]

Aliases: **SubSym**

Description: When a rule specifies that $a \rightarrow b$, then SubSym also makes the replacement $tp[a] \rightarrow tp[b]$.

Arguments: *expr* is an algebraic expression. $aListOfRules$ is a single rule or list of rules specifying the substitution to be made. On = save rules to Rules.temp, temporarily over-rides SaveRules[Off]. ’Off’ can not over-ride SaveRules[On].

Comments / Limitations: None

6.1.9 SubstituteSingleReplace[expr, aListOfRules, (optional On)]

Aliases: **SubSingleRep**

Description: Replaces one symbol or sub-expression in the expression by another expression as specified by the rule. (See Wolfram’s Mathematica 2.* page 54.)

Arguments: *expr* is an algebraic expression. $aListOfRules$ is a single rule or list of rules specifying the substitution to be made. On = save rules to Rules.temp, temporarily over-rides SaveRules[Off]. ‘Off’ can not over-ride SaveRules[On].

Comments / Limitations: The symbols /. and //. are often used in Mathematica as methods for substituting one expression for another. This method of substitution often does not work when the expression to be substituted is a subexpression within a (noncommutative) product. This Substitute command is the noncommutative analogue to /.

6.1.10 SubstituteAll[expr, aListOfRules, (optional On)]

Aliases: **SubAll**

Description: For every rule $a \to b$, SubAll also replaces,

$$tp[a] \to tp[b] \quad inv[a] \to inv[b] \quad rt[a] \to rt[b]\,.$$

Arguments: *expr* is an algebraic expression. *aListOfRules* is a single rule or list of rules specifying the substitution to be made. On = save rules to Rules.temp, temporarily over-riding SaveRules[Off]. 'Off' can not over-ride SaveRules[On].

Comments / Limitations: None

6.1.11 Transform[expr,aListOfRules]

Aliases: **Transform**

Description: None

Arguments: Transform is essentially a more efficient version of Substitute. It has the same functionality as Substitute.

Comments / Limitations: *expr* is an algebraic expression. *aListOfRules* is a single rule or list of rules specifying the substitution to be made.

Beware: Transform only applies rules once rather than repeatedly.

6.1.12 GrabIndeterminants[aListOfPolynomialsOrRules]

Aliases: **none**

Description: `GrabIndeterminants[L]` returns the indeterminates found in the list of (noncommutative) expressions or rules *L*. For example, `GrabIndeterminants[ { x**Inv[x]**x + Tp[Inv[x+a]], 3 + 4 Inv[a]**b**Inv[a] + x }]` returns

`{ x, Inv[x], Tp[Inv[x+a]], Inv[a], b }.`

Arguments: aListOfPolynomialsOrRules is a list of (noncommutative) expressions or rules.

Comments / Limitations:

6.1.13 GrabVariables[aListOfPolynomialsOrRules]

Aliases: **none**

Description: `GrabVariables[ aListOfPolynomialsOrRules ]` returns the variables found in the list of (noncommutative) expressions or rules aListOfPolynomialsOrRules. It is similar to the Mathematica command `Variables[]` which takes as an argument a list of polynomials in commutative variables or functions of variables. For example,

```
GrabVariables[ { x**Inv[x]**x + Tp[Inv[x+a]], 3 + 4 Inv[a]**b**Inv[a] + x }]
```

returns

```
{ x, a, b }.
```

Arguments: aListOfPolynomialsOrRules is a list of (noncommutative) expressions or rules.

Comments / Limitations:

6.1.14 NCBackward[expr]

Aliases: **NCB**

Description: It applies the rules

$$inv[Id - B**A]**B \to B**inv[Id - A**B]$$

$$inv[Id - B**A]**inv[A] \to inv[A]**inv[Id - A**B]$$

Arguments: *expr* is an algebraic expression.

Comments / Limitations: None

6.1.15 NCForward[expr]

Aliases: **NCF**

Description: It applies the rules

$$B**inv[Id - A**B] \to inv[Id - B**A]**B$$

$$inv[B]**inv[Id - B**A] \to inv[Id - B**A]**inv[A]$$

Arguments: *expr* is an algebraic expression.

Comments / Limitations: None

6.1.16 NCMonomial[expr]

Aliases: **None**

Description: NCMonomail changes the look of an expression by replacing nth integer powers of the NonCommutative variable x, with the product of n copies of x. For example, $NCMonomial[2x^2 + 5x^4]$ evaluates to $2x**x + 5x**x**x**x$ and $NCMonomial[(x^2)**z**x]$ evaluates to $x**x**z**x$.

Arguments: Any noncommutative expression.

Comments / Limitations: The program greatly eases the task of typing in polynomials. For example, instead of typing $x = x**x**x**x**x**x**x**x**x**x**y**x**x$, one can type $x = NCMono[(x^{12})**y**(x^2)]$. NCMono expands only integer exponents. This program will be (or has been, depending on the version of code which you have) superseded by NCMonomial and NCUnMonomial. NCMonomial implements the same functionality as NCMonomial and NCUnMonomial reverses the process. **Caution:** Mathematica treats $x**y^2$ as $(x**y)^2$ and so to have Mathematica acknowledge $x**y^2$ then input $x**(y^2)$ exactly. This has nothing to do with NCAlgebra or NCMonomial.

6.1.17 NCUnMonomial[expr]

Aliases: **None**

Description: NCUnMonomial reverses what NCMonomial does. NCUnMonomial changes the look of an expression by replacing a product of n copies of x with x^n. For example, $NCUnMonomial[2x**x + 5x**x**x**x]$ evaluates to $2x^2 + 5x^4$ and $NCUnMonomial[x**x**z**x]$ evaluates to $(x^2)**z**x$.

Arguments: Any noncommutative expression.

Comments / Limitations: See NCMonomial. NCAlgebra does not effectively manipulate expressions involving powers (such as (x^2))

6.2 Simplification

This area is under developement so stronger commands will appear in later versions. What we mean by simplify is not in the spirit of Mathematica's Simplify. They tend to factor expressions so that the expressions become very short. We expand expressions apply rules to the expressions which incorporate special relations the entries satisfy. Then we rely on cancelation of terms. The theoretical background lies in noncommutative Gröbner basis theory, and the rules we are implementing come from papers of Helton, Stankus and Wavrik [**IEEE TAC March 1998**].

The commands in this section are designed to simplify polynomials in a, b, $inv[S-a**b]$, $inv[S-b**a]$, $inv[S-a]$, $inv[S-b]$ and a few slightly more complicated inverses.

The commands in order of strength are NCSR, NCS1R, NCS2R. Of course, for a stronger the command, more rules get applied and so the command takes longer to run.

First, NCS1R normalizes $inv[S-a**b]$ to $S^{-1}*inv[1-\frac{(a**b)}{S}]$ provided S is a commutative expression (only works for numbers S in version 0.2 of NCAlgebra). The following list of rules are applied.

(0) $inv[-1+a] \rightarrow -inv[1-a]$
(1) $inv[1-a]\,(a-b)\,inv[1-b] \rightarrow inv[1-a] - inv[1-b]$
(2) $inv[1-ab]\,inv[b] \rightarrow inv[1-ba]\,a + inv[b]$
(3) $inv[1-ab]\,a\,b \rightarrow inv[1-ab] - 1$
(4) $a\,b\,inv[1-ab] \rightarrow inv[1-ab] - 1$
(5) $inv[c]\,inv[1-cb] \rightarrow inv[1-bc]\,inv[c]$
(6) $b\ inv[1-ab] \rightarrow inv[1-ba]b$

The command NCS2R increases the range of expressions to include $inv[poly]$, but the reductions for each of these inverses is considerably less powerful than for the case of $inv[1-ab]$.

An example: if $expr = a**inv[a+b]+inv[c-a]**(a-c)+inv[c+d]**(c+d+e)$, then the first reduction using the list of rules in NCSR gives $a**inv[a+b]+inv[c+d]**einv[a]**(a-b)**inv[b]$ and the second reduction gives $inv[b]-inv[a]$ which is the output from $NCSR[expr]$.

NCSimplify0Rational is an old attempt at simplification. We do not use it much.

6.2.1 NCSimplifyRational[expr], NCSimplify1Rational[expr], and NCSimplify2Rational[expr]

Aliases: **NCSR**

Description: The objective is to simplify expressions which include polynomials and inverses of very simple polynomials. These work by appling a collection of relations implemented as rules to expr. The core of NCSimplifyRational is NCSimplify1Rational and NCSimplify2Rational; indeed roughly NCSimplifyRational[expr] = NCSimplify1Rational[NCSimplify2Rational[expr]] together with some NCExpand's. NCSimplify1Rational[expr] contains one set of rules while NCSimplify2Rational[expr] contains another.

Arguments: *expr* is an algebraic expression.

Comments / Limitations: Works only for a specialized class of functions.

6.2.2 NCSimplify1Rational[expr]

Aliases: **NCS1R**

Description: It applies a collection of relations implemented as rules to *expr*. The goal is to simplify *expr*.

Arguments: *expr* is an algebraic expression.

Comments / Limitations: **WARNING:** NCS1R does not first do an ExpandNonCommutativeMultiply. Therefore, it may be the case that one can miss some simplification if *expr* is not expanded out. The solution, of course, is to call ExpandNonCommutativeMultiply before calling NCS1R. ExpandNonCommutativeMultiply is called from NCSR.

First, NCS1R normalizes $inv[S - a**b]$ to $S^{-1} * inv[1 - \frac{(a**b)}{S}]$ provided S is s a commutative expression (only works for numbers S in version 0.2 of NCAlgebra). The the following list of rules are applied.
(0) $inv[-1 + a] \rightarrow -inv[1 - a]$
(1) $inv[1 - a]\,(a - b)\,inv[1 - b] \rightarrow inv[1 - a] - inv[1 - b]$
(2) $inv[1 - ab]\,inv[b] \rightarrow inv[1 - ba]\,a + inv[b]$
(3) $inv[1 - ab]\,a\,b \rightarrow inv[1 - ab] - 1$
(4) $a\,b\,inv[1 - ab] \rightarrow inv[1 - ab] - 1$
(5) $inv[c]\,inv[1 - cb] \rightarrow inv[1 - bc]\,inv[c]$
(6) $b\ inv[1 - ab] \rightarrow inv[1 - ba]b$

In the notation of papers **[HW]**, **[HSW]**, these rules implement a superset of the union of the Gröbner basis for EB and the Gröbner basis for RESOL.

6.2.3 NCSimplify2Rational[expr]

Aliases: **NCS2R**

Description: You need this for expressions involving inv[polynomial] where the polynomial is not of the form $SId - X * *Y$

Arguments: *expr* is an algebraic expression.

Comments / Limitations: If the polynomial is too complicated, this may not help very much.

6.3 Vector Differentiation

6.3.1 DirectionalD[expr, aVariable, h]

Aliases: **DirD**

Description: Takes the Directional Derivative of expression *expr* with respect to the variable $aVariable$ in direction h.

Arguments: *expr* is an expression containing var. $aVariable$ is a variable. h is the direction which the derivative is taken in.

Comments / Limitations: None.

6.3.2 Grad[expr, aVariable]

Aliases: **Grad, NEVER USE Gradient**

Description: $Grad[expr, aVariable]$ takes the gradient of expression *expr* with respect to the variable $aVariable$. Quite useful for computations with quadratic Hamiltonians in H^{∞} control. BEWARE Gradient calls the Mma gradient and makes a mess.

Arguments: *expr* is an expression containing var. $aVariable$ is a variable.

Comments / Limitations: This only works reliably for quadratic expressions. It is not even correct on all of these. For example, $Grad[a**x + a**tp[x], x]$ returns $2tp[a]$. The reason is fundamental mathematics, not programming. If a is a row vector and x is a column vector, then $a**x$ is a number, but $a**tp[x]$ is not.

6.3.3 CriticalPoint[expr, aVariable]

Aliases: **Crit, Cri**

Description: It finds the value of $aVariable$ which makes the gradient of the expression $expr$ with respect to the variable $aVariable$ equal to 0.

Arguments: $expr$ is an expression containing $aVariable$. $aVariable$ is a variable.

Comments / Limitations: Uses the Grad and NCSolve functions. Both Grad and NCSolve are severely limited. Therefore, the CriticalPoint command has a very limited range of applications.

6.3.4 NCHessian[afunction, $\{X_1, H_1\}, \ldots, \{X_k, H_k\}$]

Aliases: **None.**

Description: `NCHessian[afunction,`$\{X_1, H_1\}, \{X_2, H_2\}, \ldots, \{X_k, H_k\}$ `]` computes the Hessian of a *afunction* of noncommutting variables and coefficients. The Hessian recall is the second derivative. Here we are computing the noncommutative directional derivative of a noncommutative function. Using repeated calls to `DirectionalD`, the Hessian of *afunction* is computed with respect to the variables $X_1, X_2, \ldots, X_k$ and the search directions H_1 , H_2 , … , H_k. The Hessian $\mathcal{H}\Gamma$ of a function Γ is defined by

$$\mathcal{H}\Gamma(\vec{X})[\vec{H}] := \frac{d^2}{dt^2}\Gamma(\vec{X} + t\vec{H})|_{t=0}$$

One can easily show that the second derivative of a hereditary symmetric noncommutative rational function Γ with respect to one variable X has the form

$$\mathcal{H}\Gamma(X)[H] = sym\left[\sum_{\ell=1}^{k} A_\ell H^T B_\ell H C_\ell\right],$$

where A_ℓ, B_ℓ, and C_ℓ are functions of X determined by Γ. (An analogous expression holds for more variables.) The Hessian will always be *quadratic* with respect to $\vec{H}$.

(A *noncommutative polynomial* in variables H_1, H_2, ..., H_k, is said to be *quadratic* if each monomial in the polynomial expression is of order two in the variables H_1, H_2, ..., H_k.)

Arguments: *afunction* is a function of the variables $X_1, X_2, \ldots, X_k$. The Hessian will be computed with respect to the search directions H_1 , H_2 , ... , H_k.
For example, suppose $F(x, y) = x + x**y + y**x$. Then,
`NCHessian`$[F, \{x, h\}, \{y, k\}]$ gives $2h**k+2k**h$ As another example, if $G(x, y, z) = inv[y] + z**x$, then `NCHessian`$[G, \{x, h\}, \{y, k\}, \{z, i\}]$ gives $2i**h + 2inv[y]**k**inv[y]**k**inv[y]$.
The results of `NCHessian` can be factored into the form $v^t M v$ by calling `NCMatrixofQuadratic`. (see `NCMatrixofQuadratic`).

Comments / Limitations: None.

6.4 Block Matrix Manipulation

By block matrices we mean matrices with noncommuting entries.

The Mathematica convention for handling vectors is tricky.

```
v={{1,2,4}}
```

is a 1×3 matrix or a row vector

```
v={{1},{2},{4}}
```

is a 3×1 matrix or a column vector

```
v={1,2,4}
```

is a vector but NOT A MATRIX. Indeed whether it is a row or column vector depends on the context. DON'T USE IT. Always remember to use TWO curly brackets on your vectors or there will probably be trouble.

As of NCAlgebra version 3.2 one can handle block matrix manipulation two different ways. One is the old way as described below where you use the command MatMult[A, B] to multiply block matrices A and B and tpMat[A] to take transposes. The other way is much more pleasing though still a little risky. First you use the **NCGuts**[] with the Options NCStrongProduct1 → True to change $**$ to make block matrices multiply corectly. Further invoke the Option NCStrongProduct2 → True to strengthen the power of $**$. Now one does not have to use MatMult and tpMat; just use $**$ and *tp* instead it recognizes matrix sizes and multiplies correctly.

6.4.1 MatMult[x, y, ...]

Aliases: **MM**

Description: MatMult multiplies matrices. The Mathematica code executed for $MatMult[x, y]$ is Inner[NonCommutativeMultiply, x, y, Plus];

Arguments: x is a block matrix, and y is a block matrix.

Comments / Limitations: MatMult can take any number of input parameters. For example, `MatMult[a, b, c, d]` will give the same result as `MatMult[a, MatMult[b, MatMult[c, d]] ]`.

6.4.2 ajMat[u]

Aliases: **None**

Description: $ajMat[u]$ returns the transpose of the block matrix u. The Mathematica code is $Transpose[Map[aj[\#]\&, u, 2]]$;

Arguments: u is a block $m \times n$ matrix.

Comments / Limitations: None

6.4.3 coMat[u]

Aliases: **None**

Description: $coMat[u]$ returns the transpose of the block matrix u. The Mathematica code is $[Map[co[\#]\&, u, 2]]$;

Arguments: u is a block $m \times n$ matrix

Comments / Limitations: None

6.4.4 tpMat[u]

Aliases: **None**

Description: $tpMat[u]$ returns the transpose of the block matrix u. The Mathematica is $Transpose[Map[tp[\#]\&, u, 2]]$;

Arguments: u is a block $m \times n$ matrix

Comments / Limitations: None

6.4.5 NCMToMatMult[expr]

Aliases: **None**

Description: Sometimes one develops an expression in which ** occurs between matrices. This command takes all ** and converts them to MatMult. The Mathematica code executed is expr//.NonCommutativeMultiply → MatMult;

Arguments: *expr* is an algebraic expression. This and its inverse (TimesToNCM) are important in manipulating block matrices. One can use

$$expr//.NonCommutativeMultiply \to MatMult$$

instead of this command, since that is all that this command amounts to.

Comments / Limitations: None

6.4.6 TimesToNCM[expr]

Aliases: **TTNCM**

Description: The Mathematica code executed is $expr/.Times \to NonCommutativeMultiply$

Arguments: *expr* is an algebraic expression.

Comments / Limitations: It changes commutative multiplication (Times) to NonCommutative multiplication.

6.4.7 Special Operations with Block Matrices

In 1999, we produced commands for LU decomposition and Cholesky decomposition of an inversion of matrices with noncommutative entries. These replace older commands *GaussElimination[X]* and *invMat2[mat]* for 2 × 2 block matrices which are no longer documented. The next 6 commands do that.

6.4.8 NCLDUDecomposition[aMatrix, Options]

Aliases: **None.**

Description: NCLDUDecomposition[X] yields the LDU decomposition for a square matrix X. It returns a list of four elements, namely L, D, U, and P such that $PXP^T = LDU$. The first element is the lower triangular matrix L, the second element is the diagonal matrix D, the third element is the upper triangular matrix U, and the fourth is the permutation matrix P (the identity is returned if no permutation is needed). As an option, it may also return a list of the permutations used at each step of the LDU factorization as a fifth element.

Suppose X is given by $X = \{\{a, b, 0\}, \{0, c, d\}, \{a, 0, d\}\}$. The command

$$\{lo, di, up, P\} = \texttt{NCLDUDecomposition}[X]$$

returns matrices, which in `MatrixForm` are:

$$lo = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & -b * *inv[c] & 1 \end{pmatrix} \qquad di = \begin{pmatrix} a & 0 & 0 \\ 0 & c & 0 \\ 0 & 0 & d + b * *inv[c] * *d \end{pmatrix}$$

$$up = \begin{pmatrix} 1 & inv[a] * *b & 0 \\ 0 & 1 & inv[c] * *d \\ 0 & 0 & 1 \end{pmatrix} \quad P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

As matrix X is 3×3, one can provide 2 permutation matrices. Let those permutations be given by $l_1 = \{3, 2, 1\}$ and $l_2 = \{1, 3, 2\}$, that means:

$$P1 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \quad P2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

just as in `NCPermutationMatrix`. The command

$$\{lo, di, up, P\} = \texttt{NCLDUDecomposition}[X, Permutation \rightarrow \{l1, l2\}]$$

returns matrices, which in `MatrixForm` are:

$$lo = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & -1 & 1 \end{pmatrix} \qquad di = \begin{pmatrix} d & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & b+c \end{pmatrix}$$

$$up = \begin{pmatrix} 1 & inv[d] ** a & 0 \\ 0 & 1 & inv[a] ** b \\ 0 & 0 & 1 \end{pmatrix} \quad P = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} = P_2\, P_1$$

It can be checked that P^T *lo di up* $P = X$:

$$MatMult[Transpose[P], lo, di, up, P] = \{\{a, b, 0\}, \{0, c, d\}, \{a, 0, d\}\}$$

Arguments: X is a square matrix n by n. The default Options are:
{`Permutation` → `False`, `CheckDecomposition` → `False`,
`NCSimplifyPivots` → `False`, `StopAutoPermutation` → `False`,
`ReturnPermutation` → `False`, `Stop2by2Pivoting` → `False` }. If permutation matrices are to be given, they should be provided as `Permutation` → $\{l_1,\ l_2,\ \cdots,\ l_n\}$, where each l_i is a list of integers (see the command `NCPermutationMatrix[]`). If `CheckDecomposition` is set to `True`, the function checks if PXP^T is identical to LDU. Where $P = P_1 P_2 \cdots P_n$, and each P_i is the permutation matrix associated with each l_i.

Often a prospective pivot will appear to be nonzero in Mathematica even though it reduces to zero. To ensure we are not pivoting with a convoluted form of zero, we simplify the pivot at each step. By default, `NCLDUDecomposition` converts the pivot from noncommutative to commutative and then simplifies the expression. If the commutative form of the pivot simplifies to zero, Mathematica scrolls down the diagonal looking for a pivot which does not simplify to zero. If all the diagonal entries simplify to zero utilizing the `CommuteEverything[]` command, the process is repeated using `NCSimplifyRational`.
This strategy is incorporated for two main reasons. One is that for large matrices it is much faster. Secondly, `NCSimplifyRational` does not always completely simplify complicated expressions. Setting `NCSimplifyPivots` → `True` bypasses `CommuteEverything` and immediately applies
`NCSimplifyRational` to each pivot. `NCLDUDecomposition` will automatically pivot if the current pivot at a particular iteration is zero. If the user utilized the `Permutation` option, then the permutation designated will be temporarily disregarded. However, `NCLDUDecomposition` will try and use the given permutation list for the next step. In this way,

`NCLDUDecomposition` follows the user permutation as closely as possible. If `StopAutoPermutation` → `True`, then `NCLDUDecomposition` will not automatically pivot and will strictly adhere to the user's permutation, attempting to divide by zero if need be. This will allow the user to determine which permutations are not possible. Because `NCLDUDecomposition` will automatically pivot when necessary by default, the `ReturnPermutation` was created so that the permutation used in the decomposition can be returned to the user for further analysis if set to `True`.

To explain the last option it is somewhat necessary for the user to have an idea of how the pivoting strategy works. The permutations used are always symmetrically applied. Because of this, we can only place other diagonal elements in the (1,1) position. However, it is possible to place any off diagonal element in the (2,1) position. Thus our strategy is to pivot only with diagonal elements if possible. If all the diagonal elements are zero, then a permutation matrix is used to place a nonzero entry in the (2,1) position which will automaticaly place a nonzero entry in the (1,2) position if the matrix is symmetric. Then, instead of using the (1,1) entry as a pivot, the 2×2 submatrix starting in the (1,1) position is used as a block pivot. This has the effect of creating an LDU decomposition where D is a block diagonal matrix with 1×1 and 2×2 blocks along the diagonal. (Note: The pivots are precisely the diagonal entries of D.) Setting `Stop2by2Pivoting` → `True` will halt 2×2 block pivoting, returning instead, the remaining undecomposed block with zeros along the diagonal as a final block diagonal entry.

Comments / Limitations: `NCLDUDecomposition` automatically assumes invertible any expressions (pivot) it needs to be invertible. Also, the 2×2 pivoting strategy assumes that the matrix is symmetric in that it only ensures that the (2,1) entry is nonzero (assuming by symmetry that the (1,2) is also zero). The pivoting strategy chooses its pivots based upon the smallest leaf count invoking the Mathematica command `LeafCount[]`. It will choose the smallest nonzero diagonal element basing size upon the leaf count. This strategy is incorporated in an attempt to find the simplest LDU factorization possible. If a 2×2 pivot is used and `ReturnPermutation` is set to `True` then at the end of the permutation list returned will be the string 2by2 permutation.

6.4.9 NCAllPermutationLDU[aMatrix]

Aliases: **None.**

Description: *NCAllPermutationLDU*[*aMatrix*] returns the LDU decomposition of a matrix for all possible permutations. The code cycles through all possible permutations and calls `NCLDUDecomposition` for each one.

Arguments: $aMatrix$ is a square matrix. The default options for `NCAllPermutationLDU` are: `PermutationSelection` → `False`, `CheckDecomposition` → `False`, `NCSimplifyPivots` → `False`, `StopAutoPermutation` → `False`, `ReturnPermutation` → `False`, `Stop2by2Pivoting` → `False`. All of these options have the same effect as in NCLDUDecomposition, except for PermutationSelection. PermutationSelection should be a list of numbers between 1 and the number of possible permutations. NCAllPermutationLDU will use this list to choose the permutations from its canonical list to decompose the matrix using NCLDUDecomposition. For example, `PermutationSelection` can be $\{1, \ldots, n\}$.

Comments / Limitations: The output is a list of all successful outputs from `NCLDUDecomposition`. Note that some permutations may lead to a zero pivot in the process of doing the LDU decomposition. In that case, the LDU decomposition is not well defined, actually in Mathematica one gets a lot of ∞ signs, but this output will not be included in the list of successful outputs.

6.4.10 NCInverse[aSquareMatrix]

Aliases: **None.**

Description: NCInverse[m] gives a symbolic inverse of a matrix with noncommutative entries.

Arguments: m is an $n \times n$ matrix with noncommutative entries.

Comments / Limitations: This command is primarily used symbolically and is not guarenteed to work for any specific examples. Usually the elements of the inverse matrix (m^{-1}) are huge expressions. We recommend using `NCSimplifyRational[NCInverse[`m`]]` to improve the formula you get. In some cases, `NCSimplifyRational[`$m^{-1}m$`]` does not provide the identity matrix, even though it does equal the identity matrix. The formula we use for `NCInverse[]` comes from the LDU decomposition. Thus in principle it depends on the order chosen for pivoting even if the inverse of a matrix is unique.

6.4.11 NCPermutationMatrix[aListOfIntegers]

Aliases: **None.**

Description: $NCPermutationMatrix[aListOfIntegers]$ returns the permutation matrix associated with the list of integers. It is just the identity matrix with its columns re-ordered.

Arguments: $aListOfIntegers$ is an encoding which specifies where the 1's occur in each column. e.g., *aListOfintegers* $= \{2, 4, 3, 1\}$ represents the permutation matrix

$$P = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Comments / Limitations: None.

6.4.12 NCMatrixToPermutation[aMatrix]

Aliases: **None.**

Description: NCMatrixToPermutation[aMatrix] returns the permutation associated with the *permutation matrix*, aMatrix. Basically, it is the inverse of `NCPermutationMatrix`.

Arguments: aMatrix must be matrix whose columns (or rows) can be permuted to yield the identity matrix. In other words, aMatrix must be a permutation matrix. For example, if m $= \{\{0, 0, 0, 1\}, \{1, 0, 0, 0\}, \{0, 0, 1, 0\}, \{0, 1, 0, 0\}\}$, then `NCPermutationMatrix`[m] gives $\{2, 4, 3, 1\}$.

Comments / Limitations: None.

6.4.13 NCCheckPermutation[SizeOfMatrix, aListOfPermutations]

Aliases: **None.**

Description: If `aListOfPermutations` is consistent with the matrix size, `SizeOfMatrix`, then the output is `valid permutation list`. If not, the output is `not valid permutation list`.

Arguments: The size of a square matrix (an integer) and a list of permutations.

Comments / Limitations: If the `SizeOfMatrix` is n, then `aListOfPermutations` must be a list of $n - 1$ permutations of the integers 1 through n. Since this command is generally called within the context of `NCLDUDecomposition` the list of permutations must correspond to a list that can be used within the command.

6.4.14 Diag[aMatrix]

Aliases: **None.**

Description: Returns the elements of the diagonal of a matrix.

Arguments: None.

Comments / Limitations: The code is Flatten[MapIndexed[Part,m]].

6.4.15 Cascade[P, K]

Aliases: **None**

Description: $Cascade[P, K]$ is the composition of P, K as is found is systems engineering.

Arguments: P is a 2×2 block matrix. K is a symbol.

Comments / Limitations: frequency response functions grow from this.

6.4.16 Chain[P]

Aliases: **None**

Description: $Chain[P]$ returns the chain matrix arising from P as is found in systems engineering.

Arguments: P is a block 2×2 matrix.

Comments / Limitations: Chain[ ] assumes appropriate matrices are invertible.

6.4.17 Redheffer[P]

Aliases: **None**

Description: $Redheffer[P]$ gives the inverse of chain.
Redheffer$[Chain[P]] = P = Chain[$Redheffer$[P]]$.

Arguments: P is a block 2×2 matrix.

Comments / Limitations: $Redheffer[P]$ assumes the invertiblity of the entries of P.

6.4.18 DilationHalmos[x]

Aliases: **None**

Description: $DilationHalmos[x]$ gives block 2×2 matrix which is the Halmos dilation of x

Arguments: x is a symbol

Comments / Limitations: $u = DilationHalmos[x]$ has the property u is unitary, that is, $MatMult[u, tpMat[u]] == IdentityMatrix[2]$ and $MatMult[tpMat[u], u] == IdentityMatrix[2]$.

6.4.19 SchurComplementTop[M]

Aliases: **None**

Description: $SchurComplementTop[M]$ returns the Shur Complement of the top diagonal entry of a block 2×2 matrix M.

Arguments: M is a block 2×2 matrix.

Comments / Limitations: Assumes invertibility of a diagonal entry.

6.4.20 SchurComplementBtm[M]

Aliases: **None**

Description: $SchurComplementBtm[M]$ returns the ShurComplement of the bottom diagonal entry of a block 2×2 matrix M.

Arguments: M is a block 2×2 matrix.

Comments / Limitations: Assumes invertibility of a diagonal entry.

6.5 Generating Random Matrices

6.5.1 RandomMatrix[m,n,min,max,options]

Aliases: **None.**

Description: `RandomMatrix[ m, n, min, max, options ]` returns a random matrix of size `m` by `n` with entries between the values `min` and `max`.

Arguments: `m` and `n` are the number of rows and columns of the matrix. `min` and `max` are the minimum and maximum values of the entries in the matrix. The options are `MatrixType` and `EntryType`. The option `MatrixType` has values `Any`, `Diagonal`, or `Symmetric`. The default is `MatrixType` → `Any` which returns an ordinary unrestricted matrix. `MatrixType` → `Diagonal` returns random diagonal matrices. `MatrixType` → `Symmetric` returns random symmetric matrices. The option `EntryType` has values `Integer` or `Real`. The default is `EntryType` → `Real` which produces real floating point numbers as entries for the random matrix. `EntryType` → `Integer` produces integers as entries.

Comments / Limitations: None.

6.6 Complex Analysis

6.6.1 A tutorial

The package in the file ComplexRules.m defines three objects:

- ComplexRules, transformation rules
- ComplexCoordinates, a function that applies rules to an expression.
- ComplexD[], takes complex derivatives.

The ComplexRules package is for handling complex algebra and differentiation. The algebra part of ComplexRules has been pretty much superceeded by the standard Mathematica command `ComplexExpand[]` so we advise using that. Our complex differentiation is still quite useful. ComplexRules.m may not work well with ReIm.m, see the warning at the end of this note.

```
In[1]:= <<ComplexRules`
In[2]:= y = Re[(e + w z )^2]^2


                        2 2
Out[2]= Re[(e + w z) ]
```

To rewrite this in terms of variables and their conjugates, apply the list of rules *ComplexRules* as follows

```
In[3]:= y //. ComplexRules


                2                                              2 2
        ((e + w z)  + (Conjugate[e] + Conjugate[w] Conjugate[z]) )
Out[3]= ------------------------------------------------------------
                                     4
```

You can get the same result with the function ComplexCoordinates[]:

```
In[4]:= ComplexCoordinates[y]


                2                                              2 2
        ((e + w z)  + (Conjugate[e] + Conjugate[w] Conjugate[z]) )
Out[4]= ------------------------------------------------------------
                                     4
```

Suppose that you know that in the expression above, e ranges in the unit circle of the complex plane, and that w is real. To simplify you can do this:

```
In[5]:= % /. {Conjugate[e]->1/e,Conjugate[w]->w}


                2    1                  2 2
        ((e + w z)  + (- + w Conjugate[z]) )
                         e
Out[5]= -------------------------------------
                          4
```

Complex derivatives are easy to produce with ComplexD[]:

```
In[6]:= ComplexD[ y , z]


                              2
Out[6]= w (e + w z) ((e + w z)

                                                 2
```

```
        + (Conjugate[e] + Conjugate[w] Conjugate[z]) )
```

Here is a differentiation with respect to Conjugate[w]:

```
In[7]:= ComplexD[ y , Conjugate[w]]

Out[7]= Conjugate[z] (Conjugate[e] + Conjugate[w] Conjugate[z])

                2                                              2
>    ((e + w z)  + (Conjugate[e] + Conjugate[w] Conjugate[z]) )
```

A mixed second order partial derivative is shown below:

```
In[8]:= ComplexD[ y , Conjugate[z] , z]

Out[8]= 2 w (e + w z) Conjugate[w]

>    (Conjugate[e] + Conjugate[w] Conjugate[z])
```

Repeated differentiation is also possible:

```
In[9]:= ComplexD[ y , {Conjugate[z],2}]

                       2                                           2
Out[9]= 2 Conjugate[w]  (Conjugate[e] + Conjugate[w] Conjugate[z])  +

                  2           2
>    Conjugate[w]  ((e + w z)  + (Conjugate[e] + Conjugate[w]

                              2
>    Conjugate[z]) )
```

Finally, we point out that it is possible that applying ComplexRules to an expression and applying ComplexCoordinates to it may yield different output (the same mathematically of course). Reason: ComplexCoordinates applies ComplexRules to the expression, in addition to a rule for transforming Abs[z] into Sqrt[z Conjugate[z]]. Example:

```
In[10]:= Abs[z^2 + 1]^2 //. ComplexRules


                 2 2
Out[10]= Abs[1 + z ]


In[11]:= ComplexCoordinates[ % ]


                2                 2
Out[11]= (1 + z ) (1 + Conjugate[z] )
```

ComplexD[] handles $Abs[]^2$ etc.:

```
In[12]:= ComplexD[ Abs[z^2 + 1]^2,z]


                              2
Out[12]= 2 z (1 + Conjugate[z] )
```

ComplexD[] also handles $Abs[]^1$ but the answer does not look as pretty:

```
In[13]:= ComplexD[ Abs[z^2 + 1],z]


                                 2
               z (1 + Conjugate[z] )
Out[13]= ----------------------------------
                  2                 2
        Sqrt[(1 + z ) (1 + Conjugate[z] )]
```

WARNING: The standard Mathematica package ReIm.m sets things so that expressions of complex variables “z” are rewritten in terms of Re[z], Im[z] (for example).

Compare this to the output of functions in the package ComplexRules.m, where the expressions of complex variables “z” are given in terms of z, Conjugate[z].

You may load both ReIm.m and ComplexRules.m, but keep in mind that the objectives of the packages conflict. Furthermore, programs that need ComplexRules to run will sometimes not work if ReIm.m has been loaded.

Mathematica can manipulate complex analysis via X + I Y where X and Y are commutative (e.g., numbers). However, it is often more convenient to calculate in terms of z and the conjugate of z. We implement a few commands in the file NCComplex.m. We discuss these commands below. One may also look at the file NCComplex.m for further documentation.

6.6.2 ComplexRules

Aliases: **None**

Description: *ComplexRules* is a set of replacement rules for writing expressions in terms of the variables and their complex conjugates. For example, use this with input containing numbers and variables, as well as operators/functions such as $+ - * /$, Re[], Im[], Conjugate[], Exp[], Power[], Sin[], Cos[] and others. Apply the command expr//.ComplexRules. Try the following example:
$Re[(1+zw)^2]^2$ //.ComplexRules

Arguments: None

Comments / Limitations: This only works for expressions defined with the commutative multiplication.

6.6.3 ComplexCoordinates[expr]

Aliases: **None**

Description: *ComplexCoordinates*[*expr*] expands *expr* in terms of the variables and their complex conjugates. The difference between *ComplexCoordinates*[*expr*] and *ComplexRules* is in the case $Abs[z]^2$//.ComplexRules. This case returns the same expression instead of z and Conjugate[z]. If you desire to use the latter expression, you can use ComplexCoordinates[expr]. This function replaces Abs[z] by Sqrt[z Conjugate[z]], after applying ComplexRules.

Arguments: *expr* is any expression with $+ - * /$, Re[], Im[], Conjugate[], Exp[], Power[], Sin[], Cos[] and others

Comments / Limitations: This only works for expressions defined with the commutative multiplication.

6.6.4 ComplexD[expr, aVariable]

Aliases: **None**

Description: $ComplexD[expr, aVariable]$ calculates the derivative of the complex expression *expr* with respect to the "complex" variable $aVariable$. You can also calculate the derivative with respect to $Conjugate[aVariable]$. Try these examples:

$ComplexD[Conjugate[Exp[z + 1/Conjugate[z]]^2], z];$
$ComplexD[Re[(1 + zw)^2]^2, w];$
$ComplexD[Abs[1/(e^2 - 1) - z]^2, z];$
$ComplexD[Conjugate[Exp[z + 1/Conjugate[z]]^2], Conjugate[z]];$

Here is a second order derivative:

$ComplexD[Conjugate[Exp[z + 1/Conjugate[z]]^2, z, 2];$

Arguments: *expr* is a complex expression. $aVariable$ is the variable in which to take the derivative with respect to.

Comments / Limitations: This only works for expressions defined with the commutative multiplication.

6.7 Setting symbols to commute or not commute

6.7.1 SetNonCommutative[A, B, C, ...]

Aliases: **SNC, SetNC**

Description: SetNonCommutative[A, B, C, ...] sets all the symbols A, B, C, ... to be noncommutative. The lower case letters a, b, c, ... are assumed noncommutative by Mathematica default as are functions of noncommutative variables. The functions *tp*[] and *aj*[] are set noncommutative by NCAlgebra for any argument, commutative or noncommutative. We may change this.

Arguments: Symbols separated by commas

Comments / Limitations: None

6.7.2 CommuteEverything[expr]

Aliases: **CE**

Description: It changes NonCommutativeMultiply to Times in *expr*.

Arguments: *expr* is an algebraic expression.

Comments / Limitations: Very useful for getting ideas in the middle of a complicated calcuation. If *expr* has you baffled, type $exprcom = CE[expr]$. *exprcom* is commutative and therefore is easy to analyze. Now *expr* is uneffected, so you can get back to working on it armed with new ideas.

6.7.3 SetCommutative[a, b, c, ...]

Aliases: **None**

Description: SetCommutative[a, b, c, ...] sets all the symbols *a*, *b*, *c*, ... to be commutative.

Arguments: Symbols separated by commas

Comments / Limitations: None

6.7.4 SetCommutingOperators[b,c]

Aliases: **None**

Description: SetCommutingOperators takes exactly two parameters. SetCommutingOperators[b, c] will implement the definitions which follow. They are in pseudo-code so that the meaning will not be obscured b ** c becomes c ** b if LeftQ[b, c]; and c ** b becomes b ** c if LeftQ[b, c];). See SetCommutingFunctions and LeftQ.

Arguments: *b*, *c* are symbols.

Comments / Limitations: **NOTE:** The above implementation will NOT lead to infinite loops.

WARNING: If one says SetCommutingOperators[b, c] and then sets only LeftQ[c,b], then neither of the above rules will be executed. Therefore, one must remember the order of the two parameters in the statement. One obvious helpful habit would be to use alphabetical order (i.e., say SetCommutingOperators[a,b] and not the reverse).

6.7.5 LeftQ[expr]

Aliases: **None**

Description: See SetCommutingFunctions and SetCommutingOperators.

Arguments: *expr* is an algebraic expression.

Comments / Limitations: None

6.7.6 CommutativeQ[X]

Aliases: **CQ**

Description: *CommutativeQ*[X] is True if X is commutative, and False if X is noncommutative.

Arguments: X is a symbol.

Comments / Limitations: See the description of SetNonCommutative for the defaults.

6.7.7 CommutativeAllQ[expr]

Aliases: **None**

Description: CommutativeAllQ[expr] is True if *expr* does not have any non-commuting sub-expressions, and False otherwise.

Arguments: *expr* is an algebraic expression.

Comments / Limitations: None

6.8 Operations on elements in an algebra

6.8.1 inv[x]

Aliases: **None**

Description: Inverse – a ** inv[a]=inv[a] ** a=Id

Arguments: x is a symbol.

Comments / Limitations: Warning: NCAlgebra does not check that $inv[x]$ exists or even that it makes sense (e.g. non-square matrices). This is the responsibility of the user.

6.8.2 invL[x]

Aliases: **invL**

Description: Left inverse – invL[a] ** a=Id

Arguments: x is a symbol

Comments / Limitations: Warning. NCAlgebra does not check that $invL[x]$ exists. This is the responsibility of the user.

6.8.3 invR[x]

Aliases: **invR**

Description: invR[x] is the right inverse – a ** invR[a]=Id

Arguments: x is a symbol

Comments / Limitations: Warning. NCAlgebra does not check that $invR[x]$ exists. This is the responsibility of the user.

6.8.4 invQ[x]

Aliases: **None**

Description: invQ[m] = True forces $invR[m]$ and $invL[m]$ to be rewritten as $inv[m]$

Arguments: x is an expression.

Comments / Limitations: We never use this command.

6.8.5 ExpandQ[inv]

Aliases: **None**

Description: If $ExpandQ[inv]$ is set to $True$, an inverse of a product will be expanded to a product of inverses. If it is set to $False$, then a product of inverses will be rewritten to be a inverse of a product.

Arguments: inv

Comments / Limitations: None

6.8.6 ExpandQ[tp]

Aliases: **None**

Description: If $ExpandQ[tp]$ is set to $True$, a transpose of a product will be expanded to a product of transposes. If it is set to $False$, then a product of transposes will be rewritten to be a transpose of a product.

Arguments: tp

Comments / Limitations: None

6.8.7 OverrideInverse

Aliases: **None**

Description: $OverrideInverse$ is a variable which is either $True$ or $False$.

Arguments: If $OverrideInverse$ is set to $True$, then the replacement of invL and invR by inv (when x is invertible) is suppressed even if invQ is $True$. The default is $False$.

Comments / Limitations: None

6.8.8 aj[expr]

Aliases: **None**

Description: $aj[expr]$ takes the adjoint of the expression $expr$. Note that basic laws like $aj[a**b] = aj[b]**aj[a]$ are automatically executed.

Arguments: *expr* is an algebraic expression

Comments / Limitations: None

6.8.9 tp[expr]

Aliases: **None**

Description: $tp[expr]$ takes the transpose of expression $expr$. Note that basic laws like $tp[a**b] = tp[b]**tp[a]$ are automatically executed.

Arguments: *expr* is an algebraic expression

Comments / Limitations: None

6.8.10 co[expr]

Aliases: **None**

Description: $co[expr]$ takes the complex conjugate of $expr$. Note basic laws like co[a**b]=co[a]**co[b] and co[a]=aj[tp[a]]=tp[aj[a]]

Arguments: *expr* is an algebraic expression

Comments / Limitations: None

6.9 Convexity of a NC function

This chapter describes commands which do two things. One is compute the "region" on which a noncommutative function is matrix convex. The other is take a noncommutative quadratic function variables H_1, H_2, etc and give a Gram representation for it, that is, represent it as

$$V[H]^T M V[H]$$

a "vector" with the H_j entering linearly and M a matrix. Other commands are described here but they are subservient to **NCConvexityRegion[afunction,alist,options]** and would not be used independently of it. The commands in this chapter are not listed alphabetically but are listed in the presumed order of importance.

6.9.1 NCConvexityRegion[afunction,alistOfVars,opts]

Aliases: **None.**

Description: `NCConvexityRegion`[afunction,alistOfVars,opts] computes the "region" on which `afunction` is matrix convex with respect to `alistOfVars`. It performs three main operations. First it computes the Hessian with respect to *alistOfVars* (see `NCHessian`). Then, using `NCMatrixOfQuadratic`, the Hessian is factored into the form $v^t M v$. Finally, depending on the option `AllPermutation`, either `NCAllPermutationLDU` or `NCLDUDecomposition` is called to compute the LDU factorization of M, the default being `AllPermutation` → `NCLDUDecomposition`. If D ends up being diagonal, then a list of the diagonal elements of D is returned. If D ends up being block diagonal with 2×2 blocks, then a message is printed out and the list: $\{\{diagonalentries\}, \{subdiagonalentries\}, \{-subdiagonalentries\}\}$ is returned. The region of convexity of `afunction` with respect to `alistOfVars` equals the closure, in a certain sense, of the set of matrices which makes all `diagonal entries` positive. If there are non-zero `subdiagonal entries`, then `afunction` is typically not matrix convex on any open set. Options permit the user to select a range of different permutation matrices, thereby producing several possibly distinct diagonal matrices D.

EXAMPLE: `NCConvexityRegion`[$x**y + y**x, \{x, y\}$] gives:

L**D**tp[L] gave non-trivial blocks, so the output list is:

{{diagonal},{subdiagonal},{-subdiagonal}}

$\{\{\{0, 0\}, \{2\}, \{-2\}\}\}$

While, `NCConvexityRegion`[$x**y + y**x, \{x, y\}, AllPermutation \to True$] gives:

Middle matrix is size 2 X 2

At most 2 permutations possible.

{1}

L**D**tp[L] gave non-trivial blocks, so the output list is:

{{diagonal},{subdiagonal},{-subdiagonal}}

$\{\{\{0, 0\}, \{2\}, \{-2\}\}\}$

In both cases, `NCHessian`[$x**y + y**x, \{x, h\}, \{y, k\}$] gives

$2h**k + 2k**h$,

`NCMatrixOfQuadratic`[$2h**k + 2k**h, \{h, k\}$] gives

$\{\{\{h, k\}\}, \{\{0, 2\}, \{2, 0\}\}, \{\{h\}, \{k\}\}\}$,

and depending on if `AllPermutation` is set to `True` or `False` you have that either `NCLDUDecomposition`[$\{\{0, 2\}, \{2, 0\}\}$] gives

$\{\{\{1, 0\}, \{0, 1\}\}, \{\{0, 2\}, \{2, 0\}\}, \{\{1, 0\}, \{0, 1\}\}, \{\{1, 0\}, \{0, 1\}\}\}$

or `NCAllPermutationLDU`$[\{\{0,2\},\{2,0\}\}]$ gives

$\{\{\{\{1,0\},\{0,1\}\},\{\{0,2\},\{2,0\}\},\{\{1,0\},\{0,1\}\},\{\{1,0\},\{0,1\}\}\},$
$\{\{\{1,0\},\{0,1\}\},\{\{0,2\},\{2,0\}\},\{\{1,0\},\{0,1\}\},\{\{1,0\},\{0,1\}\}\}\}$

Arguments: *afunction* is a function whose variables are listed in *alistOfVars*, where *alistOfVars* should be of the form $\{x_1, x_2, \ldots, x_n\}$.

The default options for `NCConvexityRegion` are:

`NCSimplifyDiagonal` →`False`

`DiagonalSelection` → `False`

`ReturnPermutation` → `False`

`ReturnBorderVector` → `False`

`AllPermutation` → `False`

`NCSimplifyDiagonal` is an option geared toward a similar option used in `NCLDUDecomposition`. This will make sure that the pivots (or diagonal entries) are all first simplified with `NCSimplifyRational` before they are used to check that the pivots are all nonzero. Simplifying the pivots using `NCSimplifyRational` can be quite time consuming, so by default we commute everything and then use Mathematica simplification commmands. We do this only to convince ourselves that the pivot is nonzero. If all the pivots are zero using `CommuteEverything` we then revert to using `NCSimplifyRational` to verify our suspicions. Setting `NCSimplifyDiagonal` → `True` will bypass the commute everything step. (Note: Either way, the unsimplified form of the pivot is returned unless it is equal to zero.)

The option `AllPermutation` tells `NCConvexityRegion` which of `NCLDUDecomposition` or `NCAllPermutationLDU` to use. Setting `AllPermutation` to `True` will use `NCAllPermutationLDU`,while `False` uses `NCLDUDecomposition`. The default value is `AllPermutation` → `False`. The following pertains to the case where `AllPermutation` is set to `True`. If you decide to do this, then you should also set `DiagonalSelection` to the permutations you would like `NCAllPermutationLDU` to use. Since different permutations return different diagonals, some diagonals are simpler to work with than others. On the other hand, if `AllPermutation` is set to `False`, which it defaults to, then `NCConvexityRegion` calls `NCLDUDecomposition` and what follows does not apply as no permutations are used. Different permutations return different diagonals. Some diagonals are simpler to work with than others. Because of this, we allow the user to select a sampling of different permutations. The total number of permutations will not be known until M is computed. After M is computed, the total number of possible permutations will be printed on the screen. `DiagonalSelection` → $\{n\}$

returns the diagonals resulting from the first n permutations. `DiagonalSelection` → $\{k,n\}$ returns the diagonals resulting from the k^{th} through n^{th} permutations. Since the total number of permutations is assumed to be unknown by the user, if n is too high, then n is replaced by the total number of permutations. Also, not all of the permutations are permissible. Because of this, `NCLDUDecomposition` automatically pivots if an invalid permutation is used for a particular step. This means it is possible that not all the diagonals returned result from different permutations. For this reason there is the option `ReturnPermutation` which if entered as `True` returns the permutations used for each resulting factorization. Finally, the user may wish to analyze the border vectors and may do so by setting `ReturnBorderVector` to `True`. This will cause `NCConvexityRegion` to return the border vectors v from the v^tMv factorization of the hessian. Now v^t will have the form

$$L_{11}H_1, L_{12}H_1, \cdots, L_{1k_1}H_1, \cdots L_{n1}H_n, L_{n2}H_n, \cdots, L_{nk_n}H_n$$

So what will actually be returned is a list of the form

$$\{\{L_{11}, \ldots, L_{1k_1}\}, ..., \{L_{n1}, \ldots, L_{nk_n}\}\}.$$

This vector will be formed using a call to `NCBorderVectorGather`. Also, a call will be made to `NCIndependenceCheck` to determine, if possible, whether or not the elements of the above list are independent. The results of this check will be printed to the screen.

Comments / Limitations: None.

6.9.2 NCMatrixOfQuadratic[$\mathcal{Q}$, $\{H_1, \ldots, H_n\}$]

Aliases: **None.**

Description: `NCMatrixOfQuadratic[` $\mathcal{Q}$, $\{\mathrm{H}_1, H_2, \ldots, H_n\}$ `]` gives a vector matrix factorization of a symmetric quadratic function $\mathcal{Q}$ in noncommutative variables $\vec{H} = \{H_1, H_2, \ldots, H_n\}$ and their transposes.

`NCMatrixOfQuadratic`[$\mathcal{Q}$, $\{H_1, H_2, \ldots, H_n\}$], generates the list {`left border vector, coefficient matrix, right border vector`}. That is, Q is factored into the vector-matrix-vector product $V[\vec{H}]^T M_{\mathcal{Q}} V[\vec{H}]$. The vector $V[\vec{H}]$ is linear in $\vec{H}$ and is called a *border vector of the quadratic function* $\mathcal{Q}$. The matrix $M_{\mathcal{Q}}$ is called the *coefficient matrix of the quadratic function* $\mathcal{Q}$.

Arguments: Each term of $\mathcal{Q}$ is assumed to be a quadratic expression in terms of the variables $H_1, H_2, \ldots, H_n$ and their transposes ($\mathcal{Q}$ is homogeneous).

For example, suppose that $\mathcal{Q} = 3tp[x] * *y + 3\,tp[y] * *x$ and that $\vec{H} = \{x, y\}$. Then, `NCMatrixOfQuadratic`[$\mathcal{Q}$, $\vec{H}$] gives

$$\{\{\{tp[x], tp[y]\}\}, \{\{0, 3\}, \{3, 0\}\}, \{\{x\}, \{y\}\}\}.$$

In `MatrixForm`, this looks like

$$(tp[x] \quad tp[y]) * \begin{pmatrix} 0 & 3 \\ 3 & 0 \end{pmatrix} * \begin{pmatrix} x \\ y \end{pmatrix}.$$

In general, suppose $\mathcal{Q}$ is a quadratic function of two variables, $\vec{H} = \{H, K\}$, with all transpose elements H^T, K^T occuring before all non-transpose elements. Then `NCMatrixOfQuadratic` will return the *left border vector* $V[\vec{H}]^T$, the matrix $M_{\mathcal{Q}}$, and the *right vector* $V[\vec{H}]$ where

$$\mathrm{M}_{\mathcal{Q}} := \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1,\ell_1} & A_{1,\ell_1+1} & \cdots & A_{1,n} \\ A_{12}^T & A_{22} & \cdots & A_{2,\ell_1} & A_{2,\ell_1+1} & \cdots & A_{2,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ A_{1,\ell_1}^T & A_{2,\ell_1}^T & \cdots & A_{\ell_1,\ell_1} & A_{\ell_1,\ell_1+1} & \cdots & A_{\ell_1,n} \\ A_{1,\ell_1+1}^T & A_{2,\ell_1+1}^T & \cdots & A_{\ell_1,\ell_1+1}^T & A_{\ell_1+1,\ell_1+1} & \cdots & A_{\ell_1+1,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ A_{1,n}^T & A_{2,n}^T & \cdots & A_{\ell_1,n}^T & A_{\ell_1+1,n}^T & \cdots & A_{n,n} \end{pmatrix} \text{ and } \mathrm{V}[\vec{H}] := \begin{pmatrix} HL_1^1 \\ HL_2^1 \\ \cdots \\ HL_{\ell_1}^1 \\ KL_1^2 \\ \cdots \\ KL_{\ell_2}^2 \end{pmatrix}$$

for some L_i^j, $i = 1, \ldots, \ell_j$. The L_i^j, $i = 1, \ldots, \ell_j$ are called the *coefficients of the border vector.* The L_i^1 corresponding to H are distinct and only one may be the identity matrix (equivalently for the L_i^2 corresponding to K). The border vector V is the vector composed of H, K and L_i^j. The matrix $M_{\mathcal{Q}}$ is the matrix with $A_{i,j}$ entries.

Noncommutative quadratics which are *not hereditary* have a similar representation (which takes more space to write) for such a quadratic in H, K. For example, the border vector for a quadratic in H, H^T, K, K^T has the form

$$V[H, K] = V\,_1 V_2$$

where we have

$$V_1 = ((L_1^1)^T H^T, \cdots, (L_{\ell_1}^1)^T H^T, (L_1^2)^T K^T, \cdots, L(_{\ell_2}^2)^T K^T)$$

and

$$V_2 = (\tilde{L}_1^1 H, \cdots, \tilde{L}_{\ell_1}^1 H, \tilde{L}_1^2 K, \cdots, \tilde{L}_{\ell_2}^2 K).$$

We should emphasize that the size of the $M_{\mathcal{Q}}$ representation of a noncommutative quadratic functions $\mathcal{Q}[H_1, \ldots, H_k]$ depends on the particular quadratic and not only on the number of arguments of the quadratic. There are noncommutative quadratic functions in one variable which have a representation with M_Q a 102 × 102 matrix. The basic idea of `NCMatrixOfQuadratic` is that it searches for terms of form

$$Left**X**Middle**Y**Right$$

where $X = H_i$ or H_i^T and $Y = H_j$ or H_j^T for $1 \leq (i,j) \leq n$. Terms of the form $Left**X$ and $Y**Right$ are used to form the left and right vectors. Each time the search finds a unique *Right* ($Left$) term causes the length of the right (left) border vector to be increased by one. The term $Middle$ becomes the entries in the matrix M_Q.

Comments / Limitations: `NCMatrixOfQuadratic` will try to symmetrize the resulting matrix M_Q. If `NCMatrixOfQuadratic` is unable to do this, an error message will be printed and { `leftvector, matrix, rightvector` } will be returned, where `matrix` is not symmetric and `leftvector` is not necessarily the transpose of `rightvector`. The vector-matrix-vector product should still be equal to the orginal quadratic expression.

6.9.3 NCIndependenceCheck[aListofLists,variable]

Aliases: **None.**

Description: `NCIndependenceCheck[aListofLists,variable]` is aimed at verifying whether or not a given set of polynomials are independent or not. It analyzes each list of polynomials in *aListofLists* separately. There are three possible types of outputs for each list in *aListofLists*. Two of them correspond to `NCIndependenceCheck` successfully determining whether or not the list of polynomials is independent. The third type of output corresponds to an unsuccessful attempt at determining dependence or independence. If a particular list is determined to be independent, *True* will be returned. If a list is determined to be dependent, a list beginning with *False* containing a set of coefficients which demonstrate independence will be returned. Finally, if `NCIndependenceCheck` cannot determine dependence or independence, it returns a list beginning with *Undetermined* containing other information which is illustrated below and described further in Comments/Limitations.

Arguments: *aListofLists* is a list containing a list of the polynomials which are suspected of being dependent. The argument *variable* will be subscripted and used to return the coefficient dependencies for each list. Below is an example of a list of four lists. The first two are dependent, the third is independent, and the fourth is undetermined.

Suppose you have four lists:

$$\begin{aligned}
List1 &= \{7, 6a, a, abd^2, d, b, 12a, d, 4a^2d, a^2, 5a^2, b^2, b\} \\
List2 &= \{50, 8a, a, abd^2, d, b, 12a, d, 4a^2d, a^2, 16a^2, 40b^2, b\} \\
List3 &= \{4a, 5b + c, c\} \\
List4 &= \{x ** y, y ** x\}
\end{aligned}$$

Then `NCIndependenceCheck[List1,List2,List3,List4,`λ`]` returns $\{NewList1, NewList2, NewList3, NewList4\}$ where:

$$\begin{aligned}
NewList1 &= \{False, \\
&\quad \{0, -\frac{\lambda_3}{6} - 2\lambda_7, \lambda_3, 0, -\lambda_8, -\lambda_{13}, \lambda_7, \lambda_8, -\frac{5\lambda_{11}}{4}, 0, \lambda_{11}, 0, \lambda_{13}\}\} \\
NewList2 &= \{False, \\
&\quad \{0, -\frac{\lambda_3}{8} - \frac{3\lambda_7}{2}, \lambda_3, 0, -\lambda_8, -\lambda_{13}, \lambda_7, \lambda_8, -4\lambda_{11}, 0, \lambda_{11}, 0, \lambda_{13}\}\} \\
NewList3 &= True \\
NewList4 &= \{Undetermined, -\lambda_2 x ** y + \lambda_2 y ** x, \{-\lambda_2, \lambda_2\}\}
\end{aligned}$$

In particular, what the above says is that $List1.Newlist1[[2]] = 0$, and $List2.Newlist2[[2]] = 0$ (where "." refers to the vector dot product). Therefore, the set of polynomials in $List1$ and $List2$ are dependent. $List3$ is independent. Note that $List4$ is clearly indpendent in the noncommutating case, and dependent in the commuting case. When such phenomena occur, `NCIndependenceCheck` is unable to determine whether or not the list of polynomials is independent. However, it does return to the user, a set of dependencies for the λ_i's which must hold in order for the polynomials to sum to zero.

Comments / Limitations: *IndependenceCheck* first uses the *CommuteEverything* command to make the problem feasible. Therefore it is possible that polynomials are dependent if variables commute, and independent if not. So in this case, or when the the expression does not collapse to zero when using the commuting coefficients with the non commuting polynomials, then the list $\{Undertermined, expression, list\}$ is returned. The list element *expression* is the sum of the polynomials with their corresponding λ's. And finally, *list* yields a list of the dependencies for the coefficents.

6.9.4 NCBorderVectorGather[alist,varlist]

Aliases: **None.**

Description: `NCBorderVectorGather[alist,varlist]` can be used to gather the polynomial coefficents preceeding the elements given in *varlist* whenever they occur in *alist*. That is to say, *alist* is a vector with variable entries. Each entry should end with some term from *varlist* (or the transpose of some term from *varlist*). Then for each element of *varlist* the coefficients that appear in front of that element in *alist* are gathered together and placed inside a list. The list returned will be a list of lists, each entry a list of the coefficients corresponding to the respective entries in *varlist* and their transposes if they occur.

Arguments: The first argument *alist* is a list of polynomials, all of which end in terms from elements of the second argument, *varlist*, or in their transpose. *alist* need not be ordered in a particular way with respect to *varlist*. The preceeding is best explained in the following example.
Suppose $List =$

$$\{A**B**k, B**B**tp[h], B**tp[A]**k, B**C**tp[h], A**tp[h], B**h, C**h\}$$

Then `NCBorderVectorGather[List,{k,h}]` returns the following list

$$\{\{A**B, B**tp[A]\}, \{B, C\}, \{B**B, B**C, A\}\}$$

Note that the vectors are gather in the pattern $k, tp[k], h, tp[h]$. This pattern will be the same despite the length of *avarlist*.

Comments / Limitations: None.

6.10 NCGuts

This section details the command NCGuts, which expands the meaning of "**", tp[], and inv[].

6.10.1 NCStrongProduct1

Aliases: **None.**

Description: NCStrongProduct1 is an option of NCGuts. When True, ** serves to multiply matrices as well as maintaining its original function with noncommutative entries. This replaces the command MatMult. For example,

$$\mathrm{MatMult}[\{\{a,b\},\{c,d\}\},\{\{x\},\{y\}\}]$$

is the same as

$$\{\{a,b\},\{c,d\}\} * *\{\{x\},\{y\}\}.$$

In addition, *tp* and tpMat are the same. NCStrongProduct1 → False is the default.

Arguments: None.

Comments / Limitations: None.

6.10.2 NCStrongProduct2

Aliases: **None.**

Description: NCStrongProduct2 is an option of NCGuts. When set to true, if m is a matrix with noncommutative entries, inv[m] returns a formula expression for the inverse of m. The considerable limitations of NCInverse are still limitations in inv[m]. NCStrongProduct2 forces NCStrongProduct1. In other words, NCGuts[NCStrongProduct2-¿True] makes "**" multiply matrices with noncommutative entries, just as NCGuts[NCStrongProduct1-¿True] does. NCStrongProduct2 → False is the default.

Arguments: None.

Comments / Limitations: None.

6.10.3 NCSetNC

Aliases: **None.**

Description: NCSetNC is an option of NCGuts. When set to false, all letters are automatically noncommutative unless SetCommutative makes them commutative. NCSetNC → False is the default.

Arguments: None.

Comments / Limitations: None.

6.11 Setting Properties of an element in an algebra

6.11.1 SetInv[a, b, c, ...]

Aliases: **None**

Description: $SetInv[a, b, c, \ldots]$ sets all the symbols a, b, c, ... to be invertible (i.e. $invQ[a]$, $invQ[b]$, $invQ[c]$, ... are set $True$).

Arguments: Symbols separated by commands

Comments / Limitations: If one does not set x to be invertible before the first use of $invL[x]$ or $invR[x]$, then NCAlgebra may not make the substitution from $invL[x]**x$ to 1 or from $x**invR[x]$ to 1 automatically.

6.11.2 SetSelfAdjoint[Symbols]

Aliases: **None**

Description: SetSelfAdjoint[a, b, ...] will set a, b, ... to be self-adjoint. The rules tp[a] := a, tp[b] :=b, ... and aj[a] := a, aj[b] := b, ... will be automatically applied. See SelfAdjointQ.

Arguments: Symbols is one or more symbols separated by commas.

Comments / Limitations: If one does not set x to be self adjoint before the first use of $aj[x]$, then NCAlgebra may not make the substitution from $aj[x]$ to x automatically. Similary for tp.

6.11.3 SelfAdjointQ[aSymbol]

Aliases: **None**

Description: $SelfAdjointQ[x]$ will return $True$ if $SetSelfAdjoint[x]$ was executed previously. See $SetSelfAdjoint$.

Arguments: $aSymbol$ is a symbol

Comments / Limitations: None

6.11.4 SetIsometry[Symbols]

Aliases: **None**

Description: $SetIsometry[a, b, \ldots]$ will set a, b, ... to be isometries. If set the rules tp[a] ** a := Id, tp[b] ** b :=Id, ... and aj[a] ** a := Id; aj[b] ** b := Id; ... will be automatically applied. See IsometryQ.

Arguments: Symbols is one or more symbols separated by commas.

Comments / Limitations: If one does not set x to be an isometry before the first use of $aj[x]$, then NCAlgebra may not make the substitution from $aj[x] * *x$ to 1 automatically. Similarly for tp.

6.11.5 IsometryQ[aSymbol]

Aliases: **None**

Description: $IsometryQ[x]$ will return $True$ if $SetIsometry[x]$ was executed previously. See $SetIsometry$.

Arguments: *aSymbol* is a symbol.

Comments / Limitations: None

6.11.6 SetCoIsometry[Symbols]

Aliases: **None**

Description: $SetCoIsometry[a, b, \ldots]$ will set a, b, ... to be co-isometries. The rules a ** tp[a] := Id, b ** tp[b] :=Id, ... and a ** aj[a] := Id, b ** aj[b] := Id, ... will be automatically applied. See $CoIsometryQ$.

Arguments: $Symbols$ is one or more symbols separated by commas.

Comments / Limitations: If one does not set x to be a coisometry before the first use of $aj[x]$, then NCAlgebra may not make the substitution from $x * *aj[x]$ to 1 automatically.

6.11.7 CoIsometryQ[aSymbol]

Aliases: **None**

Description: $CoIsometryQ[x]$ will return $True$ if $SetCoIsometry[x]$ was executed previously. See $SetCoIsometry$.

Arguments: *aSymbol* is a symbol.

Comments / Limitations: None

6.11.8 SetUnitary[Symbols]

Aliases: **None**

Description: $SetUnitary[a, b, \ldots]$ will set a, b, ... to be isometries and co-isometries. Also effects on $UnitaryQ$. See $SetIsometry$ and $SetCoIsometry$.

Arguments: *Symbols* is one or more symbols separated by commas.

Comments / Limitations: If one does not set x to be a unitary before the first use of $aj[x]$, then NCAlgebra may not make the substitution from $x * *aj[x]$ to 1 or from $aj[x] * *x$ to 1 automatically.

6.11.9 UnitaryQ[aSymbol]

Aliases: **None**

Description: $UnitaryQ[x]$ will return $True$ if $SetUnitary[x]$ was executed previously. Caution: If one executes $SetIsometry[x]; SetCoIsometry[x]$; then x is unitary, but $UnitaryQ$ remains uneffected. See $SetUnitary$.

Arguments: *aSymbol* is a symbol.

Comments / Limitations: None

6.11.10 SetProjection[Symbols]

Aliases: **None**

Description: $SetProjection[a, b, \ldots]$ will set a, b, ... to be projections. The rules a ** a := a, b ** b :=b, ... will be automatically applied. Caution: If one wants x to be a self-adjoint projection, then one must execute $SetSelfAdjoint[x]; SetProjection[x]$. See $ProjectionQ$.

Arguments: $Symbols$ is one or more symbols separated by commas.

Comments / Limitations: If one does not set x to be a projection before the first use of x, then NCAlgebra may not make the substitution from $x**x$ to x.

6.11.11 ProjectionQ[S]

Aliases: **None**

Description: $ProjectionQ[x]$ will return true if $SetProjection[x]$ was executed previously. See $SetProjection$.

Arguments: S is a symbol.

Comments / Limitations: None

6.11.12 SetSignature[Symbols]

Aliases: **None**

Description: When $SetSignature[a]$ and $SetSelfAdjoint[a]$ are executed, the rule a ** a := -1 will be automatically applied. See $SetSelfAdjoint$ and $SignatureQ$.

Arguments: $Symbols$ is one or more symbols separated by commas.

Comments / Limitations: If one does not set x to be a signature matrix and self adjoing before the first use of x, then NCAlgebra may not make the substitution from $x**x$ to -1.

6.11.13 SignatureQ[Symbol]

Aliases: **None**

Description: $SignatureQ[x]$ will return $True$ if $SetSignature[x]$ was executed previously. See $SetSignature$.

Arguments: *Symbol* is a symbol.

Comments / Limitations: None

6.12 Setting Properties of functions on an algebra

6.12.1 SetSesquilinear[Functions]

Aliases: **SetSesq**

Description: $SetSesquilinear[a, b, c, \ldots]$ sets a, b, c, ... to be functions of two variables which are linear in the first variable and conjugate linear in the second variable. See SetBilinear.

Arguments: $Functions$ is one or more symbols separated by commas.

Comments / Limitations: None

6.12.2 SesquilinearQ[aFunction]

Aliases: **None**

Description: $SesquilinearQ[x]$ will return $True$ if $SetSesquilinear[x]$ was executed previously. See $SetSesquilinear$.

Arguments: $aFunction$ is a symbol.

Comments / Limitations: None

6.12.3 SetBilinear[Functions]

Aliases: **None**

Description: $SetBilinear[a, b, c, \ldots]$ sets a, b, c, ... to be functions of two variables which is linear in the first variable and linear in the second variable. See SetSesquilinear.

Arguments: Functions is one or more symbols separated by commas.

Comments / Limitations: None

6.12.4 BilinearQ[aFunction]

Aliases: **None**

Description: $BilinearQ[x]$ will return True if $SetBilinear[x]$ was executed previously. See $SetBilinear$.

Arguments: $aFunction$ is a symbol.

Comments / Limitations: None

6.12.5 SetLinear[Functions]

Aliases: **None**

Description: $SetLinear[b, c, d, \ldots]$ sets b, c, d, ... to be functions of one variable which are linear. See $LinearQ$.

Arguments: *Functions* is one or more symbols separated by commas.

Comments / Limitations: None

6.12.6 LinearQ[aFunction]

Aliases: **None**

Description: $LinearQ[x]$ will return $True$ if $SetLinear[x]$ was executed previously. See $SetLinear$.

Arguments: $aFunction$ is a symbol.

Comments / Limitations: None

6.12.7 SetConjugateLinear[Functions]

Aliases: **None**

Description: $SetConjugateLinear[b, c, d, \ldots]$ sets b, c, d, … to be functions of one variable which are conjugate linear. See $ConjugateLinearQ$.

Arguments: $Functions$ is one or more symbols separated by commas.

Comments / Limitations: None

6.12.8 ConjugateLinearQ[aFunction]

Aliases: **None**

Description: $ConjugateLinearQ[x]$ will return True if $SetConjugateLinear[x]$ was executed previously. See $SetConjugateLinear$.

Arguments: aFunction is a symbol.

Comments / Limitations: None

6.12.9 SetIdempotent[Functions]

Aliases: **None**

Description: $SetIdempotent[b, c, d, \ldots]$ sets b, c, d, … to be functions of one variable such that, for example, $b[b[z\_]] := z$; Common examples are inverse, transpose and adjoint. See IdempotentQ.

Arguments: $Functions$ is one or more symbols separated by commas.

Comments / Limitations: None

6.12.10 IdempotentQ[aFunction]

Aliases: **None**

Description: $IdempotentQ[x]$ will return $True$ if $SetIdempotent[x]$ was executed previously and False otherwise. See $SetIdempotent$.

Arguments: $aFunction$ is a symbol.

Comments / Limitations: None

6.12.11 SetCommutingFunctions[aFunction, anotherFunction]

Aliases: **None**

Description: SetCommutingFunctions takes exactly two parameters. SetCommutingFunctions[b, c] will implement the definitions $b[c[z\_\_\_]] := c[b[z]]$ /; $Not[LeftQ[b,c]]$; and $c[b[z\_\_\_]] := b[c[z]]$ /; LeftQ[b, c]; Common examples are the adjoint commuting with the transpose. Note: The above implementation will NOT lead to infinite loops. WARNING: If one says SetCommutingFunctions[b, c] and then sets only LeftQ[c,b], then neither of the above rules will be executed. Therefore, one must remember the order of the two parameters in the statement. One obvious helpful habit would be to use alphabetical order (i.e. say SetCommutingFunctions[aj, tp] and not the reverse). See $CommutatingOperators$ and $LeftQ$.

Arguments: *aFunction* and *anotherFunction* are symbols.

Comments / Limitations: None

6.12.12 SetNonCommutativeMultiplyAntihomomorphism[Functions]

Aliases: **None**

Description: $SetNonCommutativeMultiplyAntihomomorphism[b,c,d,\ldots]$ sets b, c, d, ... to be functions of one variable such that, for example, $b[anything1 * *anything2]$ becomes $b[anything2]{*}{*}b[anything1]$ if $ExpandQ[b]$ is True; b[anything2] ** b[anything1] becomes b[anything1 ** anything2] if $ExpandQ[b]$ is False; Common examples are inverse, transpose and adjoint. NOTE: The synonym NCAntihomo is easier to type.

Arguments: *Functions* is one or more symbols separated by commas.

Comments / Limitations: None

6.13 Manipulating an Expression — less useful commands

6.13.1 NCTermArray[expr,aList,anArray]

Aliases: **None**

Description: $NCTermArray[expr, aList, anArray]$ creates an array $anArray$ whose contents represent the terms of $expr$ sorted by degree. The variables $anArray[”variables”]$, $anArray[”types”]$ and elements such as $anArray[x{*}{*}x, y]$ and $anArray[x{*}{*}x{*}{*}x, y{*}{*}y]$ to hold the terms with 2 x‘s and 1 y and 3 x’s and 2 y’s, respectively (assuming that $aList$ = {x,y}). You can reconstruct $expr$ from $anArray$ via ReconstuctTaylor[anArray].

Arguments: $expr$ is an algebraic expression, $aList$ is a list of variables and $anArray$ is a symbol.

Comments / Limitations: Not available before NCAlgebra 1.0

The following is an example of the above command.

```
In[22]:= expr = x ** z ** x ** w + x ** z ** y ** w + z ** x ** x ** w +
    z ** x ** y ** w;


In[23]:= NCTermArray[expr,{x,y},foo]

Out[23]= foo

In[24]:= ??foo
Global‘foo

foo["types"] = {{1, 1}, {2, 0}}

foo["variables"] := {x, y}

foo[x, y] = x ** z ** y ** w + z ** x ** y ** w

foo[x ** x, 1] = x ** z ** x ** w + z ** x ** x ** w

foo[x___] := 0
```

6.13.2 NCReconstructFromTermArray[anArray]

Aliases: **None**

Description: $NCTermArray[expr, aList, anArray]$;

$$newexpr = NCReconstructFromTermArray[anArray];$$

sets *newexpr* equal to *expr*.

Arguments: *anArray* is a symbol previously filled by NCTermArray

Comments / Limitations: Not available before NCAlgebra 1.0

If we continue with the example above we have the following

```
In[24]:= NCReconstructFromTermArray[foo]


Out[24]= x ** z ** x ** w + x ** z ** y ** w + z ** x ** x ** w +

>    z ** x ** y ** w
```

However, now one can also do some manipulation before reconstructing as shown below.

```
In[25]:= foo[x,y] = NCC[foo[x,y],y**w]
Out[25]= (x ** z + z ** x) ** y ** w

In[26]:= foo[x**x,1] = NCC[foo[x**x,1],x**w]

Out[26]= (x ** z + z ** x) ** x ** w

In[27]:= ??foo
Global`foo

foo["types"] = {{1, 1}, {2, 0}}

foo["variables"] := {x, y}

foo[x, y] = NCC[x ** z ** y ** w + z ** x ** y ** w, y ** w]

foo[x ** x, 1] = NCC[x ** z ** x ** w + z ** x ** x ** w, x ** w]
```

```
foo[x___] := 0

In[27]:= NCReconstructFromTermArray[foo]
Out[27]= (x ** z + z ** x) ** x ** w + (x ** z + z ** x) ** y ** w
```

6.13.3 NCCompose[aVerySpecialList]

Aliases: **NCCom**

Description: $NCCompose[NCDecompose[poly, a]]$ will reproduce $poly$. For example, $NCCompose[NCDecompose[poly, a, b], 1, 0]$ will reconstruct the elements of poly which are of order 1 in a and of order 0 in b.

Arguments: Not documented yet.

Comments / Limitations: Called within NCCollect. **The average user would never use this.**

6.13.4 NCDecompose[expr, listofsymbols]

Aliases: **NCDec**

Description: NCDecompose[poly, a] or NCDecompose[poly, a, b, c, ...] will produce a list of elements of poly in which elements of the same order of a (or the same order of a, b, c, ...) are collected together.

Arguments: Not documented yet.

Comments / Limitations: Called within NCCollect. **The average user would never use this.**

6.14 Utilities

Most of these utilities are for saving things. They probably do not work nor will you wish to use them in the Notebook environment.

6.14.1 CEEP

Aliases: **None**

Description: The “CEEP” file tells Mathematica that you want to record the functions you use during the session for later use or examination. It prompts you for a file name. Say you respond MYSESSION. CEEP records your session – two different ways into two different files – the first file (e.g. MYSESSION.m and MYSESSION.ex) records the In[] and Out[] lines of code you see on the screen and the second file (which contains the suffix .ex – MYSESSION.ex in the above example) saves just the commands which you type.

Functions stored in these files can be brought into a Mathematica session at a later time by typing `<<MYSESSION.ex` and it executes. This file can also be modified in a text editor external to the Mathematica program. The “NCAlgebra.m” file contains the instructions to load NCAlgebra “packages” which allow the manipulation of noncommutative expressions.

Also, when using UNIX via a UNIX shell (rather than a Mathematica notebook), UNIX has a 'script' utility which can be used. Type 'man script' to find out more.

Arguments: None

Comments / Limitations: Fails inside a Notebook

6.14.2 SaveRules[expression, 'optional tag → ”message”']

Aliases: **SaveR**

Description: Its main purpose is to control the Rules.temp file which records the rules used any time a Substitute command is used.

Arguments: $SaveRules[On]$ turns on the Rules.temp for continuous recording of rules. $SaveRules[Off]$ turns off the continuous record feature of Substitute commands, but any Substitute command can make a record in Rules.temp by using On as its optional argument. $SaveRules[expression]$ will save the evaluated form of $expression$ to the Rules.temp. $SaveRules[expression, tag \to "message"]$ will save the evaluated form of 'expression' to the Rules.temp file with a explanatory message.

Comments / Limitations: SaveRules["ccc"], where ccc is a string, can be used to include comments into the Rules.temp file. SaveRules[mathematical expression] will record the mathematical expression without its definitions.

6.14.3 SaveRulesQ[]

Aliases: **SaveRQ**

Description: SaveRulesQ[] indicates the status of the continuous recording feature of the Substitute commands into the Rules.temp file by message and returns True if continuous records are being made and False if continuous records are not being made.

Arguments: None

Comments / Limitations: Messages can be suppressed or enabled by typing Off[SaveRulesQ::Off] and On[SaveRulesQ::On].

6.14.4 FunctionOnRules[Rules, Function1, Function2, (optional On)]

Aliases: **FORules**

Description: It maps $Function1$ onto the left hand side and $Function2$ onto the right hand side of each rule in a set of rules, and returns the new set of rule. For example, FunctionOnRules[{ a→ x, b→ y }, Sin, Cos] gives {Sin[a] → Cos[x], Sin[b] → Cos[y] }

Arguments: *Rules* is a single rule or list of rules. $Function1$ and $Function2$ are any built-in Mathematica function, NCAlgebra function, pure function, or user-defined function.

Comments / Limitations: None

Chapter 7

Pretty Output and Tex Commands

There are 2 ways of producing pretty output. The most practical for a small session is described in the Pretty Output section below. The fancier way is to produce TEX displays. TEX displays can be done in several ways.

One is by using the Mathematica TEX setting features. Just follow their directions; we have installed a few special NCAlgebra features via the file NCTeXForm.m. These features are automatically loaded. If you are useing a notebook and have trouble it is possibly because you do not have mma's notebook.sty file in the correct place. That's not an NCAlgebra problem but is between you and Mma. We did put some suggestions on this in Section 37.7.

Another is TeX produced in real time on the screen. Most of this section is devoted to describing how to produce output in TEX. We have a very effective way of doing this interactively which we highly recommend.

Older methods we developed for TEX are in **NCOLDDOC** which calls the files *OldCommandDoTex.tex* and *MoreDetailsOnTex.tex* which we do not support. pressions together with the context to a file.

7.0.5 SetOutput[optionlist,. . .]

Aliases: **None**

Description: SetOutput displays noncommutative expressions in a special format without affecting the internal representation of the expression. For example, SetOutput[all → True] turns on all display beatifications listed below. Options are set by typing,

- all → True, to use the new format.
- opt → False, to return to the default format.
- dot → True, '**' is displayed as '.'(i.e., a dot)

- dot → False, '**' defaults as '**'
- aj → True, aj[X] is displayed as X^*
- aj → False, aj[X] is displayed as aj[X]
- inv → True, inv[X] is displayed as X^{-1}
- inv → False, inv[X] is displayed as inv[X]
- tp → True, tp[X] is displayed as X^T
- tp → False, tp[X] is displayed as tp[X]

Examples : (Generic) SetOutput[opt→True] turns on all beautifications of the output related to the attribute opt. (One never types opt. The letters "opt" here stand for any of the listed attributes.) So, for example, one might try the command SetOutput[dot→True, aj→True]; aj[rt[x ** y]]

Arguments: Options are: all, dot, rt, tp, inv, aj; You may input a comma-separated sequence of options in any order without first forming a Mathematica list.

Comments / Limitations: BEWARE: do not use with NCSimplifyRational. Also working with parts of a "pretty output" may not work (since it changes the Head structure of the output). You can turn on or off options individually at any time. The nesting order of the final display is determined first by any NCAlgebra-defined or user-defined functions, and then by any Mathematica specified order. Only after the Out[#] is assigned are the SetOutput options applied to the OutputForm of the expression. So the internal form of the expression is NEVER altered. Nevertheless, the displayed form may be unexpected for several reasons, among them being; 1. f[g[x]] may be defined elsewhere to always display as g[f[x]]. In this case, SetOutput will display the optional forms of g[f[x]], not that of the inputed f[g[x]]. 2. Mathematica establishes precedences for functions and operations in order to minimize the overall use of parantheses. This can have unusual, but not unmanageable effects on the displayed form of an expression. For example, $(x**y)^2$ displays as x^2**y ,but is represented internally as the equivalent of $(x**y)^2$.

7.1 NC enhancement of Mathematica's TeX settings

The Mathematica program provides a command called TeXForm which can be used to output your expression generated by Mathematica into TEX format. We have extended this facility to include basic NC constructs, by adding them via the file NCTeXForm.m. Examples of what this accomplishes are

tp[x] will have the TeX'ed form x^T
aj[x] will have the TeX'ed form x^*
rt[x] will have the TeX'ed form $x^{\frac{1}{2}}$
inv[x] will have the TeX'ed form x^{-1}.

As you might guess if you want to add more its easy. To use this NCTeXForm.m must be loaded; thats all. These are used by NCAlgebra but not NCGB. NCGB *TeX* fonts are more advanced than those of NCAlgebra and they are stored in

```
NC/NCGB/MmmaSource/ToStringForTeX.m
```

7.2 Simple TeX Commands

In this section we describe some handy additional TEX display features we have added to Mathematica. These act in addition to Mathematica's TEX setting with NCAlgebra, like TeXForm or Format[, TeXForm]. Mathematica's TEX conversion however does not need this and is fine with NCAlgebra, since we have added special NCAlgebra and NCGB notation to the Mathematica-to-TEX dictionary. If you are content with that, there is no reason to read this part of the document. Also the NCProcess TEX spreadsheets are automatic and are not related to this section of the document.

These are very useful commands and we recommend using them. We support only UNIX at this point. You **must do a little installation work** to use our fancier TEX commands. Mathematica 3.0 creates TEX output that uses a specific Mathematica file called a style file (called notebook.sty). Some of the TEX produced needs to know where this file is kept. A description of how to inform TEX where this file is stored is found in Section 7.5. Also you must load the file Extra.TeXForm into Mma for two of the commands.

To use these TeXUtilities.m must be loaded; thats all:

```
<<TeXUtilities.m
```

7.3 Real time Tex

7.3.1 SeeTeX[] or SeeTeX[anInteger]

Aliases: **None**

Description: SeeTeX[] tells Mathematica that you would always like to have your outputs displayed in TEX. Inputting SeeTeX[] gives you additional displays which accompany the rest of your Mathematica session unless you turn off SeeTeX using the NoTeX[] (see below). Each output in your session causes a seperate window to open on the screen and this window displays the TEX version of the output. When too many TeX displays are present, Mathematica automatically makes the oldest display disappear. One can set the maximum number of TEX windows to be N which appear using $SeeTeX[N]$. SeeTeX[] is the same as SeeTeX[5] the first time that it is called. See also NoTeX, KillTeX and Keep.

Arguments: *anInteger* is an integer.

Comments / Limitations: Not available before NCAlgebra 2.1. At this time, SeeTeX is implemented by creating a directory called TeXSession. The directory TeXSession is filled with files as the session continues. The files corresponding to the k-the output are master$k$.tex and out$k$.tex. The file master$k$.tex is boring. The file out$k$.tex contains the TEX for the output Out[k]. See also the command SeeTex. These files are kept until the KillTeX[] command (§7.3.3) is invoked or until you reload NCAlgebra.m. **BEWARE LOADING NCAlgebra.m AUTOMATICALLY DELETES THE DIRECTORY** Also, the directory contains a few dvi files and the windows are produced automatically by running xdvi on master$k$.dvi. When the TeX windows are removed automatically, the corresponding dvi file is automatically removed. The function Keep[k] (§7.3.5) prevents master$k$.dvi and the associated window from being deleted.

7.3.2 NoTeX[]

Aliases: **None**

Description: NoTeX[] tells Mathematica to stop putting windows on the screen.

Arguments: None

Comments / Limitations: Not available before NCAlgebra 2.1. The command NoTeX[] stops TeX files from being added to the directory TeXSession. It does not remove any files from the directory TeXSession. BUG: If one does SeeTeX[4], then puts up 4 windows, then NoTeX[], then SeeTeX[], the program forgets that the 4 windows are there.

7.3.3 KillTeX[]

Aliases: **None**

Description: KillTeX[] removes all of the files in the directory TeXSession. See also SeeTeX(§7.3.1)

Arguments: No arguments

Comments / Limitations: Not available before NCAlgebra 2.1

7.3.4 See[aListOfIntegers]

Aliases: **None**

Description: See[aListOfIntegers]

Arguments: See[aListOfIntegers] allows one to create a TEX file which displays multiple Out statements. For example, See[{5,14,9}] creates a window with `Out[5]=`, followed by the TEX form of Out[5],`Out[14]=`, followed by the TEX form of Out[14], `Out[9]=`, followed by the TEX form of Out[9]

Comments / Limitations: $aListOfIntegers$ is a list of integers

Not available before NCAlgebra 2.1 If the See[{5,14,9}] command is invoked as the 19th command (e.g., In[19] := See[{5,14,9}]), then the TeXed formulas Out[5], Out[14] and Out[19] will be in the file master19.tex. This might help a user in writing a paper based on the session, since it can be used to bring together important formulas. This command requires that the SeeTeX be called before the use of See.

7.3.5 Keep[anInteger]

Aliases: **None**

Description: $Keep[anInteger]$ tells Mathematica that it should not automatically destroy the window corresponding to the $anInteger$th output.

Arguments: $anInteger$ is an integer.

Comments / Limitations: Not available before NCAlgebra 2.1. The file master$k$.dvi is also kept. The TeX files are always kept whether or not you invoke Keep (WARNING: See KillTeX (§7.3.3)).. This command requires that the SeeTeX be called before the use of “See”.

7.3.6 Kill[anInteger]

Aliases: **None**

Description: Kill[anInteger]

Arguments: Kill[k] removes the window and the dvi file corresponding to Out[k].

Comments / Limitations: Not available before NCAlgebra 2.1. The file master$k$.dvi is deleted. This command requires that the SeeTeX be called before the use of the command “See”.

7.4 One shot commands

The following two commmands require the loading of the file `Extra.TeXForm`.

7.4.1 LookAtMatrix[aMatrix]

Aliases: **None**

Description: LookAtMatrix[aMatrix] takes the Mathematica matrix *aMatrix*, converts it to TEX outputs the string and surrounding LATEXto a file, TeXs the file and displays the TeXed output to the screen using the program dvipage (this can be easily changed to other previewers such as xdvi or xpreview).

Arguments: $aMatrix$ is a matrix.

Comments / Limitations: Must have loaded the file Extra.TeXForm

7.4.2 LookAtLongExpression[anExpression]

Aliases: **None**

Description: LookAtLongExpression[anExpression] takes the Mathematica expression *anExpression*, converts it to TEX outputs the string and surrounding LATEXto a file, TeXs the file and displays the TeXed output to the screen using the program dvipage (this can be easily changed to other previewers such as xdvi or xpreview).

Arguments: $anExpression$ is an expression

Comments / Limitations: Must have loaded the file Extra.TeXForm

7.5 Informing TEX about notebook.sty, etc.

See the installation Appendix for how to install our special TEX features.

7.6 HTML output - not Supported

We (Kurt Schneider) made the NCAlgebra web page via HTML output from NCGB which he arranged. He could do it but the rest of us have not tried it. Kurt's work was in 1996. Since then the ways of HTML have changed as has the NCAlgebra web page. We have provided some info in the NCGBDOCUMENT to taunt the user.

Chapter 8

An NCAlgebra demo - The Bounded Real Lemma

The Bounded Real Lemma

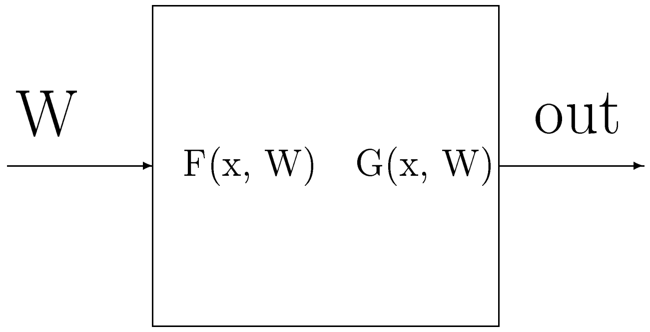
• Introduction

This example derives the Bounded Real Lemma for a linear system for a linear system.
This is a special case of this lemma for a more general system described in [BHW].
First we load the program:

```
In [1]: = <<NCAlgebra.m
```

```
You have already loaded NCAlgebra.m
```

• System Definition



where: $\frac{dx}{dt} = F(x,W) = Ax + BW$;
out = G(x,W) = Cx+DW.

• Demo Initialization

1) To set all matrices as noncommutative we use the well known command:

```
In [2] : =   SetNonCommutative[F, G, A, B, C, D, x, W]
```

```
Out [2] ={False, False, False, False, False, False, False, False}
```

2) Next we define F, and G:

```
In [3] : = F[x_,W_]:=A**x+B**W
           G[x_,W_]:=C**x+D**W
           out= G[x,W]
```

Out [5] =C∗∗x+D∗∗W

3) Recall definition of Energy Hamiltonian H

$$\boxed{\mathrm{H{=}out}^T\ \mathrm{out}\text{-}\gamma^2 W^T\mathrm{W}+\frac{(p.F+(p.F)^T)}{2}}$$

where $e(x) = x^T\mathrm{XX}x$

$$p = \nabla e(x) = 2(x.\ \mathrm{XX})^T$$

In [6] : = **SetNonCommutative[p]**

Out [6] = {False}

In [7] : = **H=tp[out]∗∗out-(($\gamma \wedge 2$s)*tp[W]∗∗W)+ (p∗∗F[x,W]+tp[F[x,W]]∗∗tp[p])/2**

Out [7] = $\frac{1}{2}$ (p∗∗(A∗∗x+B∗∗W)+(tp[W]∗∗tp[B]+ tp[x]∗∗tp[A])∗∗tp[p])+
(tp[W]∗∗tp[D]+tp[x]∗∗tp[C])∗∗ (C∗∗x+D∗∗W)- γ^2 tp[W]∗∗W

In [8] : = **SetNonCommutative[e,XX]**

Out [8] = {False, False}

In [9] : = **e[x_]:=tp[x]∗∗XX∗∗x**

In [10] : = **tp[XX]=XX**

Out [10] = XX

In [11] : = **p=2 tp[XX∗∗x]**

Out [11] = 2 tp[x]∗∗XX

• Lemma

Assume $(-\gamma^2 + D^T D)^{-1}$ Exists.

Then the linear system described in the introduction above is:

i) finite gain dissipative with gain bounded by γ^2.

ii) Energy function $x^T XXx$.

if and only if

XX is a positive semi definite solution to the Riccati inequality:

$$XXA + A^T XX + C^T C + (XXB + C^T D)(-\gamma^2 + D^T D)^{-1}(-B^T XX - D^T C) \leq 0$$

• Proof (by computation)

In [12] : = **H**

Out [12] = $(\text{tp[W]**tp[D]+tp[x]**tp[C]})\text{** } (\text{C**x+D**W})\text{-}\gamma^2 \text{ tp[W]**W+}$
$\frac{1}{2}\,(2\ (\text{tp[W]**tp[B]+tp[x]**tp[A]})\text{** XX**x+ 2 tp[x]**XX**}(\text{A**x+B**W}))$

In [13] : = **SetOutput[all→True]**

In[13] was obtain by clicking on "SetOutputPretty" in the Main Commands section of NCPalette.

The strategy here is to maximize H and see if it is < 0

1) Differentiate to find the "worst" input W. Use the commmand Grad on the NCPalette or type in

In [14] : = **Grad[H,W]**

Out [14] = -2 W γ^2+2 B^T.XX.x+2 D^T.C.x+2 D^T.D.W

2) find the critical W using the command NCSolve or

In [15] : = **NCSolve[%==0,W]**

Out [15] = $\{\mathtt{W} \to 2(2\gamma^2 - 2\mathtt{D}^\mathtt{T}.\mathtt{D})^{-1}.\mathtt{B}^\mathtt{T}.\mathtt{XX.x} + 2(2\gamma^2 - 2\mathtt{D}^\mathtt{T}.\mathtt{D})^{-1}.\mathtt{D}^\mathtt{T}.\mathtt{C.x}\}$

3) plug the critical point back into H

In [16] : = **Substitute[H, %]**

Out [16] = $((2\mathtt{x}^\mathtt{T}.\mathtt{XX.B}.(2\gamma^2 - 2\mathtt{D}^\mathtt{T}.\mathtt{D})^{-1} + 2\mathtt{x}^\mathtt{T}.\mathtt{C}^\mathtt{T}.\mathtt{D}.(2\gamma^2 - 2\mathtt{D}^\mathtt{T}.\mathtt{D})^{-1}).\mathtt{D}^\mathtt{T} + \mathtt{x}^\mathtt{T}.\mathtt{C}^\mathtt{T}).$
$(\mathtt{C.x} + \mathtt{D}.(2(2\gamma^2 - 2\mathtt{D}^\mathtt{T}.\mathtt{D})^{-1}.\mathtt{B}^\mathtt{T}.\mathtt{XX.x} + 2(2\gamma^2 - 2\mathtt{D}^\mathtt{T}.\mathtt{D})^{-1}.\mathtt{D}^\mathtt{T}.\mathtt{C.x}))+$
$\gamma^2(2\mathtt{x}^\mathtt{T}.\mathtt{XX.B}.(2\gamma^2 - 2\mathtt{D}^\mathtt{T}.\mathtt{D})^{-1} + 2\mathtt{x}^\mathtt{T}.\mathtt{C}^\mathtt{T}.\mathtt{D}.(2\gamma^2 - 2\mathtt{D}^\mathtt{T}.\mathtt{D})^{-1}).$
$(-2(2\gamma^2 - 2\mathtt{D}^\mathtt{T}.\mathtt{D})^{-1}.\mathtt{B}^\mathtt{T}.\mathtt{XX.x} - 2(2\gamma^2 - 2\mathtt{D}^\mathtt{T}.\mathtt{D})^{-1}.\mathtt{D}^\mathtt{T}.\mathtt{C.x})+$

$$\begin{aligned}&\tfrac{1}{2}(2((2x^{T}.XX.B.(2\gamma^2-2D^{T}.D)^{-1}+2x^{T}.C^{T}.D.(2\gamma^2-2D^{T}.D)^{-1}).B^{T}+x^{T}.A^{T}).XX.x+\\&\quad 2x^{T}.XX.(A.X+B.(2(2\gamma^2-2D^{T}.D)^{-1}.B^{T}.XX.x+2(2\gamma^2-2D^{T}.D)^{-1}.D^{T}.C.x)))\end{aligned}$$

4) Expand the expression using NCExpand

In [17] : = **NCExpand[%]**

Out [17] =
$$\begin{aligned}&x^{T}.XX.A.x+x^{T}.A^{T}.XX.x+x^{T}.C^{T}.C.x+\\&\quad 4x^{T}.XX.B.(2\gamma^2-2D^{T}.D)^{-1}.B^{T}.XX.x+4x^{T}.XX.B.(2\gamma^2-2D^{T}.D)^{-1}.D^{T}.C.x+\\&\quad 4x^{T}.C^{T}.D.(2\gamma^2-2D^{T}.D)^{-1}.B^{T}.XX.x+4x^{T}.C^{T}.D.(2\gamma^2-2D^{T}.D)^{-1}.D^{T}.C.x-\\&\quad 4\gamma^2x^{T}.XX.B.(2\gamma^2-2D^{T}.D)^{-1}.(2\gamma^2-2D^{T}.D)^{-1}.B^{T}.XX.x-\\&\quad 4\gamma^2x^{T}.XX.B.(2\gamma^2-2D^{T}.D)^{-1}.(2\gamma^2-2D^{T}.D)^{-1}.D^{T}.C.x-\\&\quad 4\gamma^2x^{T}.C^{T}.D.(2\gamma^2-2D^{T}.D)^{-1}.(2\gamma^2-2D^{T}.D)^{-1}.B^{T}.XX.x-\\&\quad 4\gamma^2x^{T}.C^{T}.D.(2\gamma^2-2D^{T}.D)^{-1}.(2\gamma^2-2D^{T}.D)^{-1}.D^{T}.C.x+\\&\quad 4x^{T}.XX.B.(2\gamma^2-2D^{T}.D)^{-1}.D^{T}.D.(2\gamma^2-2D^{T}.D)^{-1}.B^{T}.XX.x+\\&\quad 4x^{T}.XX.B.(2\gamma^2-2D^{T}.D)^{-1}.D^{T}.D.(2\gamma^2-2D^{T}.D)^{-1}.D^{T}.C.x+\\&\quad 4x^{T}.C^{T}.D.(2\gamma^2-2D^{T}.D)^{-1}.D^{T}.D.(2\gamma^2-2D^{T}.D)^{-1}.B^{T}.XX.x+\\&\quad 4x^{T}.C^{T}.D.(2\gamma^2-2D^{T}.D)^{-1}.D^{T}.D.(2\gamma^2-2D^{T}.D)^{-1}.D^{T}.C.x\end{aligned}$$

5) try to simplify the expression using NCSimplifyRational

In [18] : = **NCSimplifyRational [%]**

Out [18] =
$$\begin{aligned}&x^{T}.XX.A.x+x^{T}.A^{T}.XX.x+x^{T}.C^{T}.C.x+2x^{T}.XX.B.(2\gamma^2-2D^{T}.D)^{-1}.B^{T}.XX.x+\\&\quad 2x^{T}.XX.B.(2\gamma^2-2D^{T}.D)^{-1}.D^{T}.C.x+2x^{T}.C^{T}.D.(2\gamma^2-2D^{T}.D)^{-1}.B^{T}.XX.x+\\&\quad 2x^{T}.C^{T}.D.(2\gamma^2-2D^{T}.D)^{-1}.D^{T}.C.x\end{aligned}$$

6) Collect common terms and Arrange the expression as much as possible

In [19] : = `NCCollectSymmetric[%, inv[2γ ∧ 2 − 2tp[D] ∗ ∗D]]`

Out [19] =
$$\begin{aligned}&(x^{T}.XX.B+x^{T}.C^{T}.D).(2\gamma^2-2D^{T}.D)^{-1}.(2B^{T}.XX.x+2D^{T}.C.x)+x^{T}.XX.A.x+\\&\quad x^{T}.A^{T}.XX.x+x^{T}.C^{T}.C.x\end{aligned}$$

In [20] : = **NCCollectSymmetric[%, {x, XX}]**

Out [20] =
$$\begin{aligned}&x^{T}.(XX.A+A^{T}.XX).x+x^{T}.C^{T}.C.x+\\&\quad x^{T}.(XX.B+C^{T}.D).(2\gamma^2-2D^{T}.D)^{-1}.(2B^{T}.XX+2D^{T}.C).x\end{aligned}$$

In [21] : = **NCCollect [%, 2]**

Out [21] = $\mathtt{x^T.(XX.A + A^T.XX).x + x^T.C^T.C.x +}$
$\mathtt{2x^T.(XX.B + C^T.D).(2\gamma^2 - 2D^T.D)^{-1}.(B^T.XX + D^T.C).x}$

To pull out the "2" from the inv[ ] expression one needs to be a bit tricky and use:

In [22] : = $\mathtt{Substitute[\%, inv[2\gamma \wedge 2 - 2tp[D] * *D] - > inv[\gamma \wedge 2 - tp[D] * *D]/2]}$

Out [22] = $\mathtt{x^T.(XX.A + a^T.XX).x + x^T.C^T.C.x + x^T.(XX.B + C^T.D).(\gamma^2 - D^T.D)^{-1}.(B^T.XX + D^T.C).x}$

7) Let $x = 1$

In [23] : = **Substitute[%, x->1]**

Out [23] = $\mathtt{XX.A + A^T.XX + C^T.C + (XX.B + C^T.D).(\gamma^2 - D^T.D)^{-1}.(B^T.XX + D^T.C)}$

8) Let D=0 for clarity

In [24] : = **Substitute[%, d->0]**

Out [24] = $\mathtt{XX.A + A^T.XX + C^T.C + XX.B.(\gamma^2)^{-1}.B^T.XX}$

The last output is the expression of the well known Riccati equation which proves the Theorem.

Chapter 9

Aliases

The following is a list of aliases. For example, `NCC[x,y]` is equivalent to (but easier to type than) `NCCollect[x,y]`.

```
NCDec -> NCDecompose
NCCom -> NCCompose
NCC   -> NCCollect
NCSC  -> NCStrongCollect
NCSym -> NCCollectSymmetric

CAR   -> ComplexAlgebraRules
RCAR  -> ReverseComplexAlgebraRules

DirD  -> DirectionalD
DirDP -> DirectionalDPolynomial
Cri   -> CriticalPoint
Crit  -> CriticalPoint
GradPoly -> Grad

ExprToTeXFile -> ExpressionToTeXFile

NCForward -> NCInverseForward
NCBackward -> NCInverseBackward
NCF   -> NCInverseForward
NCB   -> NCInverseBackward
NCEI  -> NCExpandInverse
NCETP -> NCExpandTranspose
```

```
MM    -> MatMult
NCMTMM -> NCMToMatMult
tpM   -> tpMat
GauE  -> GaussElimination

NCExpand -> ExpandNonCommutativeMultiply,
ENCM  -> ExpandNonCommutativeMultiply
NCE   -> ExpandNonCommutativeMultiply

TTNCM -> TimesToNCM
CE    -> CommuteEverything
CQ    -> CommutativeQ

SNC   -> SetNonCommutative
NCM   -> NonCommutativeMultiply
SetNC -> SetNonCommutative

NCSR  -> NCSimplifyRational
NCS0R -> NCSimplify0Rational
NCS1R -> NCSimplify1Rational
NCS2R -> NCSimplify2Rational
NCIE  -> NCInvExtractor
MSR   -> MakeSimplifyingRule

NCSolve -> NNCSolveLinear1

Sub   -> Substitute
SubR  -> SubstituteReverse
SubRev -> SubstituteReverse
SubSym -> SubstituteSymmetric
SubRSym -> SubstituteReverseSymmetric
SubRevSym -> SubstituteReverseSymmetric
SubSingleRep -> SubstituteSingleReplace
SubAll -> SubstituteAll
SaveR  -> SaveRules
```

```
SaveRQ -> SaveRulesQ
FORules -> FunctionOnRules

NCHDP -> NCHighestDegreePosition
NCHD  ->NCHighestDegree
LPR   -> LeftPatternRule

LinDGKF -> LinearDGKF
```

Part II

NONCOMMUTATIVE CONTROL SYSTEM PROFESSIONAL

Mathematica Control System Professional Noncommutative Support

Version 1.0
(Mathematica 2.2 and 3.0 compatible)

J. William Helton and F. Dell Kronewitter
Math Dept., UCSD

This package adds non-commuting capabilities
to some of the functionality of Mathematica's
Control System Professional.

If you would like to try the NCAlgebra package or want updates go to the NCAlgebra web site.

http://math.ucsd.edu / ~ ncalg

or contact ncalg@ucsd.edu or MathSource@wri.com.

Chapter 10

State Space Systems Constructions

The commands in this section facilitate working with linear dynamic systems.

Most of the entries below require Mathematica's *Control System Professional* package available from Wolfram Research. To change the (commutative) standard Control System Professional Package for use with NCAlgebra first load in the Control System Professional and then simply load in the file `NCControl.m`

```
<< NCControl.m
```

The file

NC/NCAlgebra/NCControlSPDemo.nb

contains several examples which illustrate the use of CSP package with non commutative systems.

The linear system

$$\left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]$$

is written in Control/Mathematica notation as

```
StateSpace[ A, B, C, D].
```

where A, B, C, and D are matrices made up of symbolic noncommuting indeterminates. For example,

```
Series1 = StateSpace[ {{a}}, {{b}}, {{c}}, {{d}}]
```

or

```
Series1 = StateSpace[ {{a11, a12}, {a21,a22}}, {{b1}, {b2}}, {{c1, c2}}, {{d}}].
```

10.1 System Interconnections

The following commands allow one to connect two systems in various ways

10.1.1 SeriesConnect[System1, System2]

Aliases: **none**

Description: SeriesConnect[$sys1$, $sys2$] creates a system which is the series connection of the two linear dynamic systems, $sys1$ and $sys2$.

Arguments: System1,System2

Comments / Limitations: More complete documentation can be found in the Control Systems Professional manual.

10.1.2 FeedbackConnect[System1, System2]

Aliases: **none**

Description: $FeedbackConnect[System1, System2]$ creates a system which is the feedback connection of the two linear dynamic systems, $System1$ and $System2$.

Arguments: System1,System2

Comments / Limitations: More complete documentation can be found in the Control Systems Professional manual.

10.1.3 ParallelConnect[System1, System2]

Aliases: **none**

Description: $ParallelConnect[System1, System2]$ creates a system which is the parallel connection of the two linear dynamic systems, $System1$ and $System2$.

Arguments: System1,System2

Comments / Limitations: More complete documentation can be found in the Control Systems Professional manual.

10.2 Continuous vs. Discrete

The following commands allow one to determine the whether a system is discrete or continuous.

10.2.1 ContinuousTimeQ[System1]

Aliases: **none**

Description: ContinuousTimeQ[$System1$] returns $True$ if $System1$ is a continuous dynamic system and $False$ otherwise.

Arguments: System1

Comments / Limitations: More complete documentation can be found in the Control Systems Professional manual.

10.2.2 DiscreteTimeQ[System1]

Aliases: **none**

Description: $DiscreteTimeQ[System1]$ returns $True$ if $System1$ is a discrete dynamic system and $False$ otherwise.

Arguments: System1

Comments / Limitations: More complete documentation can be found in the Control Systems Professional manual.

10.3 Transfer Function

The following command will return the transfer function of a system

10.3.1 TransferFunction[System1]

Aliases: **none**

Description: $TransferFunction[System1]$ returns the transfer function associated with the state space representation of $System1$.

Arguments: System1

Comments / Limitations: More complete documentation can be found in the Control Systems Professional manual. Changes to the CSP II package, since the CSP I require that inside NCControl.m we set the option

ReductionMethod → NCInverse

for TransferFunction. For systems with commutative entries one still gets a correct answer from TransferFunction even with this option set. However, it may look different from that produced by CSP. Thus for a commutative $[A, B, C, D]$ system, one may want use the option

ReductionMethod → DeterminantExpansion

which gives the same answer as the CSP default.

10.4 Systems from Systems

The following commands will return the new system associated with the argument

10.4.1 Dual[System1]

Aliases: **none**

Description: $Dual[System1]$ returns the dual system to $System1$

Arguments: System1

Comments / Limitations: More complete documentation can be found in the Control Systems Professional manual.

10.4.2 InverseSystem[System1]

Aliases: **none**

Description: $InverseSystem[System1]$ returns the system which is the inverse of $System1$

Arguments: System1

Comments / Limitations: This function does not require the Control Systems Professional manual.

Part III

NONCOMMUTATIVE GRÖBNER BASES–NCGB

NONCOMMUTATIVE GRÖBNER BASIS PACKAGE

Version 3.1
(Mathematica 2.2 and 3.0 and Gnu g++2.7.2 compatible)

M. Stankus
Math Dept., UCSD

J. William Helton
Math Dept., UCSD
La Jolla, California 92093

If you would like to try the NonCommutative Gröbner Basis package or want updates look on the NCAlgebra homepage

http://math.ucsd/∼ ncalg

The program was written with contributions from Dell Kronewitter, Eric Rowell, Juan Camino, Dave Glickenstein, Kurt Schneider, Victor Shih and Mike Moore, and with support from the AFOSR, the NSF, the Lab for Mathematics and Statistics at UCSD, the Ford Motor Co., the UCSD Faculty Mentor Program and the US Department of Education.

February 1996
September 1997
April 1999
October 1999
October 2001

??This is only the introductory parts of the NCGB manual. For the other parts (which are more technical), look at the NCAlgebra webpage.

ReleaseNotesNCGB

ReleaseNotes NCGB2.02.1.tex

The commands *CleanUpBasisQ*[] and *Iterations*[] were removed since they did not serve any real purpose.

ReleaseNotes NCGB3.0.tex

NCGB has been seriously overhauled. The file stucture was made more modular. This facilitates mantainance and the forthcoming NCGB stand alone version. The C++ code was considerably revised to compile under both UNIX and Visual C++.

Added are:

1. Stronger change of variable commands. (Still experimental)
2. A command to compute the coefficients of the "noncommutative" Hilbert series.
3. A facility for handling psuedoinverses; See NCMakeRelations
4. Some files are loaded automatically on demand
5. Commands which save typing, NCAutomaticOrder and NCAddTranspose
6. A method for automatically changing elimination orders when solving algebraic problems, NCXWholeProcess

Part IV

NCGB: Easy Introduction

Chapter 11

Introduction

We think of this package as being useful for at least 4 things:

1. Simplifying complicated expressions (Chapter 12).
2. Eliminating unknowns from collections of polynomial equations and sorting the result. Indeed the package is aimed at discovering algebraic theorems and appealing formulas semi-automatically (Chapters 19 and 18 with examples in Chapters 17 and 20).
3. Finding small bases for ideals in a noncommuting algebra (Chapter 24).
4. Producing Groebner Bases in noncommuting situations (Chapter 12

This package can be used with the NCAlgebra package to add powerful automatic methods for handling collections of equations in noncommuting variables.

Most commutative algebra packages contain commands based on Gröbner Bases and uses of Gröbner Basis. For example, in Mathematica, the Solve command puts collections of equations in a "canonical" form which, for simple collections, readily yields a solution. Likewise, the Mathematica Eliminate command tries to convert a collection of polynomial equations (e.g., $\{p_j(x_1, \ldots, x_n) = 0 : 1 \leq j \leq k_1\}$) in unknowns $x_1, x_2, \ldots x_n$ to a "triangular" form in unknowns, that is, a new collection of equations like

$$\begin{aligned} q_1(x_1) &= 0 && (11.1)\\ q_2(x_1, x_2) &= 0 && (11.2)\\ q_3(x_1, x_2) &= 0 && (11.3)\\ q_4(x_1, x_2, x_3) &= 0 && (11.4)\\ &\ldots && (11.5)\\ q_{k_2}(x_1, \ldots, x_n) &= 0. && (11.6) \end{aligned}$$

Here the polynomials $\{q_j : 1 \leq j \leq k_2\}$ generate the same ideal that the polynomials $\{p_j : 1 \leq j \leq k_1\}$ generate. Therefore, the set of solutions to the collection of polynomial equations $\{p_j = 0 : 1 \leq j \leq k_1\}$ equals the set of solutions to the collection of polynomial equations $\{q_j = 0 : 1 \leq j \leq k_2\}$. This canonical form greatly simplifies the task of solving collections of polynomial equations by facilitating backsolving for x_j in terms of $x_1, \ldots, x_{j-1}$.

The user who is not acquainted at all with Gröbner Basis should still be able to read and use most of the material which is contained within this document.

In [**FMora**]**, c.f.** [**TMora**], F. Mora described a version of the Gröbner basis algorithm which applies to noncommutative free algebras. We refer to this algorithm as Mora's algorithm and as the Gröbner Basis Algorithm. This strategy also puts collections of equations into a "canonical form" which we believe has considerable possibilities in the noncommutative case.

How to read this document

To learn how to install the program or use someone else's installation, read Chapter 2 or Chapter 3.

The first thing you should do is read Part 1 to see examples of the basic commands.

If you are interested in simplification of expressions, you should read Chapter 12. Simplification is discussed in the papers [**HW**] and [**HSW**].

If you are interested in proving theorems and want to understand the ideas, you should read Chapter 19.

If you are interested in proving theorems and want to see examples, you should read Chapter 17. and Chapter 20 to see examples of the software in action.

If you want to understand the commands which were used to do the example in Chapter 17, then read Chapter 18.

If you want to compute Gröbner Bases, read Chapter 12.3 or read Chapter 23 without first reading anything else.

In addition,

(1) The computer commands given in the text are generally shown in verbatim (e.g., `NCProcess1`) or italics.

(2) The Mathematica variables are in bold face (e.g., **MathVariable**).

(3) Filenames are in bold face (e.g., **output.txt**).

Chapter 12

Simple Demos of Basic Commands

In this chapter, we will give a number of demonstrations of how one would use our computer program to simplify expressions. Demonstrations for proving theorems using this program are given in Chapters 17, 18 and 20.

Throughout this document, we shall use the word "relation" to mean a polynomial in noncommuting indeterminates.[1]

12.1 To start a C++ GB session

The first step is to start Mathematica:

```
% math
Mathematica 2.2 for SPARC
Copyright 1988-93 Wolfram Research, Inc.
```

The next step is to load the appropriate software. [2]

```
In[1]:= <<NCGB.m

Hi there !!!!!!!
NCSetRule.m loaded
NCPInverses.m loaded
NCMono.m loaded
NCSolve.m loaded
NCMatMult.m loaded
NCAliasFunctions.m loaded
NCAlias.m loaded
Starting Main
LinkObject[p9c, 1, 1]
```

---

[1] If an analyst saw the equation $AB = 1$ for matrices A and B, then he might say that A and B satisfy the polynomial equation $x\,y - 1 = 0$. An algebraist would say that $x\,y - 1$ is a relation.

[2] When the file "NCGB.m" is loaded, it loads the file NCAlgebra.m which in turn loads lots or few files depending on how one has set the environmental variable \$NC\$LongLoadTime\$. The default is \$NC\$LongLoadTime\$=True. To save time you can set \$NC\$LongLoadTime\$=False before loading NCGB.m.

12.1.1 NCGBSetIntegerOverflow[False]

Here is a technical point which has implications. By default, $C++$ stores only a small number of integers and if longer integers occur in a computer run it will make a mistake. NCGB, which you are using, does not have this problem because of some potentially time consuming dynamic storage allocation. If you are sure your runs have small integers (between ±2 billion on a Sun), then you might want to override this NCGB feature to save run time. There are two ways to do this. One is type the command **NCGBSetIntegerOverflow**[**True**] before loading NCGB.m. The other is to edit a line in the file NCGB.m to read **$NCGB$IntegerOverflow=True**. These commands actually switch which code you are using. If you are in the middle of a session and wish to switch to just type **NCGBSetIntegerOverflow**[**True**] or **NCGBSetIntegerOverflow**[**True**] and reload NCGB.m.

12.2 Simplifying Expressions

Suppose we want to simplify the expression $a^3b^3 - c$ assuming that we know $ab = 1$ and $ba = b$.
First NCAlgebra requires us to declare the variables to be noncommutative.

```
In[2]:= SetNonCommutative[a,b,c]
```

Now we must set an order on the variables a, b and c.

```
In[3]:= SetMonomialOrder[{a,b,c}]
```

Later we explain what this does, in the context of a more compliated example where the command really matters. Here any order will do. We now simplify the expression $a^3b^3 - c$ by typing

```
In[4]:= NCSimplifyAll[{a**a**a**b**b**b -c}, {a**b-1,b**a- b}, 3]
```

After messages appear on the screen (which indicate that the computation is taking place), you get the answer as the following Mathematica output.

```
Out[4]= {1 - c}
```

The number 3 indicates how hard you want to try (how long you can stand to wait) to simplify your expression.

12.3 Making a Groebner Basis

A reader who has no explicit interest in Groebner Bases might want to skip this section. Readers who lack background in Gröbner Basis may want to read [CLS]. This section does indicate what underlies the simlification commands in Chapter 12. For more on the subject see 13.2.

Before making a Gröbner Basis, one must declare which variables will be used during the computation and must declare a "monomial order" which can be done using the commands described in Chapter 21. A user does not need to know theoretical background related to monomials orders. Indeed, as we shall see in Chapter 17, for many engineering problems, it suffices to know which variables correspond to quantities which are known and which variables correspond to quantities which are unknown. If one is solving for a variable or desires to prove that a certain quantity is zero, then one would want to view that variable as unknown. For simple mathematical problems, one can take all of the variables to be known. At this point in the exposition we assume that we have set a monomial order.

```
In[1]:= <<NCGB.m
```

```
In[2]:= SetNonCommutative[a,b,x,y]
```

```
In[3]:= SetMonomialOrder[a,b,x,y]
```

```
In[4]:= ourGB = NCMakeGB[{y**x - a, y**x - b, x**x - a, x**x**x - b}, 10]
```

The result is:

```
Out[5]= {-a+x**x,-a+b,-a+y**x,-a+a**x,-a+x**a,-a+y**a,-a+a**a}
```

Our favorite format (as can be seen from the output to the screen) for displaying lists of relations is ColumnForm.

```
In[5]:= ColumnForm[%]
Out[5]= -a + x ** x
        -a + b
        -a + y ** x
        -a + a ** x
        -a + x ** a
        -a + y ** a
        -a + a ** a
```

Someone not familiar with GB's might find it instructive to note this output GB triangularizes the input equations to the extent that we have a compatibility condition on a, namely $a^2 - a = 0$; we can solve for b in terms of a; there is one equation involving only y and a; and there are three equations involving only x and a. Thus if we were in a concrete situation with a and b, given matrices, and x and y, unknown matrices we would expect to be able to solve for large pieces of x and y independently and then plug them into the remaining equation $yx - a = 0$ to get a compatibility condition.

12.4 Reducing a polynomial by a GB

Now we reduce a polynomial or ListOfPolynomials by a GB or by any ListofPolynomials2. First we convert ListOfPolynomials2 to rules subordinate to the monomial order which is currently in force in our session.

For example, let us continue the session above with

```
In[9]:= ListOfRules2 = PolyToRule[ourGB];
Out[9]= {x**x->a,b->a,y**x->a,a**x->a,x**a->a,y**a->a,
             a**a->a}
```

To reduce ListOfPolynomials by ListOfRules2 use the command

```
             Reduction[ ListofPolynomials, ListofRules2]
```

For example, to reduce the polynomial `poly = a**x**y**x**x + x**a**x**y + x**x**y**y` in our session type

```
In[10]:= Reduction[ { poly }, ListOfRules2 ]
```

12.4.1 Simplification via GB's

The way the previously described command *NCSimplifyAll* works is

```
    NCSimplifyAll[ ListofPolynomials, ListofPolynomials2] =
                Reduction[ ListofPolynomials,
                         PolyToRule[NCMakeGB[ListofPolynomials2,10]]]
```

Chapter 13

NCGB Facilitates Natural Notation

Now we turn to a more complicated (though mathematically intuitive) notation. Also we give some more examples of Simplification and GB manufacture. We shall use the variables

$$y, Inv[y], Inv[1-y], a \text{ and } x\,. \tag{13.1}$$

In NCAlgebra, lower case letters are noncommutative by default, and functions of noncommutative variables are noncommutative, so the `SetNonCommutative` command, while harmless, is not necessary. Using $Inv[]$ has the advantage that our TeX display commands recognize it and treat it wisely. Also later we see that the command `NCMakeRelations` generates defining relations for $Inv[]$ automatically.

13.1 A Simplification example

We want to simplify a polynomial in the variables of (13.1). We begin by setting the variables noncommutative with the following command.

```
In[5]:= SetNonCommutative[y, Inv[y], Inv[1-y], a, x]
```

Next we must give the computer a precise idea of what we mean by "simple" versus "complicated". This formally corresponds to specifying an order on the indeterminates. If $Inv[y]$ and $Inv[1-y]$ are going to stand for the inverses of y and $1-y$ respectively, as the notation suggests, then the order

$$y < Inv[y] < Inv[1-y] < a < x$$

sits well with intuition, since the matrix y is "simpler" than $(1-y)^{-1}$. [1] To set this order input [2]

```
In[6]:= SetMonomialOrder[{ y, Inv[y], Inv[1-y], a, x}]
```

Suppose that we want to connect the Mathematica variables $Inv[y]$ with the mathematical idea of the inverse of y and $Inv[1-y]$ with the mathematical idea of the inverse of $1-y$. Then just type [3] in the defining relations for the inverses involved.

---

[1] There are many orders which "sit well with intuition". Perhaps the order $Inv[y] < y < Inv[1-y] < a < x$ does not set well, since, if possible, it would be preferable to express an answer in terms of y,rather than y^{-1}.

[2] This sets a graded lexicographic on the monic monomials involving the variables y, $Inv[y]$, $Inv[1-y]$, a and x with $y < Inv[y] < Inv[1-y] < a < x$.

[3] See also §13.4.1

```
In[7]:= resol = {y ** Inv[y] == 1,   Inv[y] ** y == 1,
                 (1 - y) ** Inv[1 - y] == 1,   Inv[1 - y] ** (1 - y) == 1}

Out[7]= {y ** Inv[y] == 1, Inv[y] ** y == 1,
        (1 - y) ** Inv[1 - y] == 1, Inv[1 - y] ** (1 - y) == 1}
```

As an example of simplification, we simplify the two expressions $x**x$ and $x + Inv[y] * *Inv[1-y]$ assuming that y satisfies *resol* and $x**x = a$. The following command computes a Gröbner Basis for the union of *resol* and $\{x^2 - a\}$ and simplifies the expressions $x**x$ and $x + Inv[y] * *Inv[1-y]$ using the Gröbner Basis. Experts will note that since we are using an iterative Gröbner Basis algorithm which may not terminate, we must set a limit on how many iterations we permit; here we specify *at most* 3 iterations.

```
In[8]:= NCSimplifyAll[{x**x,x+Inv[y]**Inv[1-y]},Join[{x**x-a},resol],3]

Out[8]= {a, x + Inv[1 - y] + Inv[y]}
```

We name the variable $Inv[y]$, because this has more meaning to the user than would using a single letter. $Inv[y]$ has the same status as a single letter with regard to all of the commands which we have demonstrated.

Next we illustrate an extremely valuable simplification command. The following example performs the same computation as the previous command, although one does not have to type in *resol* explicitly. More generally one does not have to type in relations involving the definition of inverse explicitly. Beware, `NCSimplifyRationalX1` picks its own order on variables and completely ignores any order that you might have set.

```
In[9]:= <<NCSRX1.m
In[10]:= NCSimplifyRationalX1[{x**x**x,x+Inv[z]**Inv[1-z]},{x**x-a},3]
Out[11]= {a ** x, x + Inv[1 - z] + inv[z]}
```

WARNING: Never use inv[ ] with NCGB since it has special properties given to it in NCAlgebra and these are not recognized by the C++ code behind NCGB

13.2 MakingGB's and Inv[], Tp[]

Here is another GB example. This time we use the fancy `Inv[]` notation.

```
In[1]:= <<NCGB.m

In[2]:= SetNonCommutative[y, Inv[y], Inv[1-y], a, x]

In[3]:= SetMonomialOrder[{ y, Inv[y], Inv[1-y], a, x}]

In[4]:= resol = {y ** Inv[y] == 1,   Inv[y] ** y == 1,
                 (1 - y) ** Inv[1 - y] == 1,   Inv[1 - y] **
(1 - y) == 1}
```

The following commands makes a Gröbner Basis for *resol* with respect to the monomial order which has been set.

```
In[8]:= NCMakeGB[resol,3]
Out[8]= {1 - Inv[1 - y] + y ** Inv[1 - y], -1 + y ** Inv[y],
>    1 - Inv[1 - y] + Inv[1 - y] ** y, -1 + Inv[y] ** y,
>    -Inv[1 - y] - Inv[y] + Inv[y] ** Inv[1 - y],
>    -Inv[1 - y] - Inv[y] + Inv[1 - y] ** Inv[y]}
```

13.3 Simplification and GB's revisited

Changing polynomials to rules

The following command converts a list of relations to a list of rules subordinate to the monomial order specified above.

```
In[9]:= PolyToRule[%]
Out[9]= {y ** Inv[1 - y] -> -1 + Inv[1 - y], y ** Inv[y] -> 1
,
>    Inv[1 - y] ** y -> -1 + Inv[1 - y], Inv[y] ** y -> 1,
>    Inv[y] ** Inv[1 - y] -> Inv[1 - y] + Inv[y],
>    Inv[1 - y] ** Inv[y] -> Inv[1 - y] + Inv[y]}
```

Changing rules to polynomials

The following command converts a list of rules to a list of relations.

```
In[10]:= PolyToRule[%]
Out[10]= {1 - Inv[1 - y] + y ** Inv[1 - y], -1 + y ** Inv[y],
>    1 - Inv[1 - y] + Inv[1 - y] ** y, -1 + Inv[y] ** y,
>    -Inv[1 - y] - Inv[y] + Inv[y] ** Inv[1 - y],
>    -Inv[1 - y] - Inv[y] + Inv[1 - y] ** Inv[y]}
```

Simplifying using a GB revisited

We can apply the rules in §13.3 repeatedly to an expression to put it into "canonical form." Often the canonical form is simpler than what we started with.

```
In[11]:= Reduction[{Inv[y]**Inv[1-y] - Inv[y]}, Out[9]]
Out[11]= {Inv[1 - y]}
```

13.4 Saving lots of time when typing

13.4.1 Saving time when typing relations involving inverses:NCMakeRel

One can save time in inputting various types of starting relations easily by using the command *NCMakeRelations.*

```
In[12]:= <<NCMakeRelations.m
In[13]:= NCMakeRelations[{Inv,y,1-y}]
Out[13]= {y ** Inv[y] == 1, Inv[y] ** y == 1, (1 - y) ** Inv[1 -
y] == 1,
    Inv[1 - y] ** (1 - y) == 1}
```

WARNING

It is traditional in mathematics to use only single characters for indeterminates (e.g., x, y and α). However, we allow these indeterminate names as well as more complicated constructs such as

$$Inv[x], Inv[y], Inv[1 - x ** y] \text{ and } Rt[x] \, .$$

In fact, we allow $f[expr]$ to be an indeterminate if *expr* is an expression and f is a Mathematica symbol which has no Mathematica code associated to it (e.g., $f = Dummy$ or $f = Joe$, but NOT $f = List$ or $f = Plus$). Also one should never use $inv[m]$ to represent m^{-1} in the input of any of the commands explained within this document, because NCAlgebra has already assigned a meaning to $inv[m]$. It knows that $inv[m] ** m$ is 1 which will transform your starting set of data prematurely.

Besides *Inv* many more functions are facilitated by NCMakeRelations, see Section 25.0.1.

13.4.2 Saving time working in algebras with involution: NCAddTranspose, NCAddAdjoint

One can save time when working in an algebra with transposes or adjoints by using the command NCAddTranpose[ ] or NCAddAdjoint[ ]. These commands "symmetrize" a set of relations by applying tp[ ] or aj[ ] to the relations and returning a list with the new expressions appended to the old ones. This saves the user the trouble of typing both $a = b$ and $tp[a] = tp[b]$.

```
NCAddTranspose[ { a + b , tp[b] == c + a } ]
```

returns

```
{ a + b , tp[b] == c + a, b == tp[c] + tp[a], tp[a] + tp[b] }
```

13.4.3 Saving time when setting orders: NCAutomaticOrder

One can save time in setting the monomial order by not including all of the indeterminants found in a set of relations, only the variables which they are made of. $NCAutomaticOrder[aMonomialOrder, aListOfPolynomials]$ inserts all of the indeterminants found in $aListOfPolynomials$ into $aMonomialOrder$ and sets this order. NCAutomaticOrder[$aListOfPolynomials$] inserts all of the indeterminants found in $aListOfPolynomials$ into the ambient monomial order. If x is an indeterminant found in $aMonomialOrder$ then any indeterminant whose symbolic representation is a function of x will appear next to x.

```
NCAutomaticOrder[{{a},{b}},  { a**Inv[a]**tp[a] + tp[b]}]
```

would set the order to be $a < tp[a] < Inv[a] \ll b < tp[b]$.

Chapter 14

Demo on NCGB - Matrix Computation

(The functions used in this notebook require the C++ NCGB module.)

The Partially Prescribed Inverse Problem

This is a type of problem known as a matrix completion problem. This particular one was suggested by Hugo Woerdeman. We are grateful to him for discussions.

Problem:

Given matrices a, b, c, and d, we wish to determine under what conditions there exists matrices x, y, z, and w such that the block matrices

$$\begin{pmatrix} a & x \\ y & b \end{pmatrix} \qquad \begin{pmatrix} w & c \\ d & z \end{pmatrix}$$

are inverses of each other. Also, we wish to find formulas for x, y, z, and w.

This problem was solved in a paper by W.W. Barrett, C.R. Johnson, M. E. Lundquist and H. Woerderman [**BJLW**] where they showed it splits into several cases depending upon which of a, b, c and d are invertible. In our example, **we assume that a, b, c and d are invertible** and discover the result which they obtain in this case.

First we set all of our variables to be noncommutative and set up the relations which make matrices a, b, c, and d invertible. (Inverses in this particular problem are taken to be two sided.) Strong invertibility relations help when one is trying to get an idea of the solution of the problem.

```
In[1] =  SetNonCommutative[a,b,c,d,w,x,y,z,Inv[a],Inv[b],Inv[c],Inv[d]]

Out[1] =  {False,False,False,False,False,False,False, False,False,False,False,False}
```

Then we define the relations we are interested in. The two relations **oneway, otherway** set our block matrices as inverses of each other. The relations **inverses** invoke the assumption that **a,b,c,** and **d** are invertible by defining their inverses.

```
In[2]:=  first = {{a,x}, {y,b}}
         second = {{w,c},{d,z}}

Out[2] = {{a,x}, {y,b}}

Out[3] = {{w,c},{d,z}}

In[4]:= oneway = MatMult[first,second] - IdentityMatrix[2]
          otherway = MatMult[second,first] - IdentityMatrix[2]

Out [4] = {{-1+a**w+x**d,a**c+x**z}, {b**d+y**w,-1+b**z+y**c}}

Out [5] = {{-1+c**y+w**a,c**b+w**x}, {d**a+z**y,-1+d**x+z**b}}

In[6]:=  inverses = {-1 + a ** Inv[a], -1 + Inv[a] ** a,
           -1 + b ** Inv[b], -1 + Inv[b] ** b,
           -1 + c ** Inv[c], -1 + Inv[c] ** c,
           -1 + d ** Inv[d], -1 + Inv[d] ** d
          }

          allRelations =Join[ Flatten[{ oneway, otherway }], inverses]

Out [6] = {-1+a**Inv[a],-1+Inv[a]**a, -1+b**Inv[b],-1+Inv[b]**b,
              -1+c**Inv[c],-1+Inv[c]**c,-1+d**Inv[d],-1+Inv[d]**d}

Out [7] = {-1+a**w+x**d,a**c+x**z, b**d+y**w,-1+b**z+y**c,
              -1+c**y+w**a,c**b+w**x,d**a+z**y,-1+d**x+z**b,
              -1+a**Inv[a],-1+Inv[a]**a, -1+b**Inv[b],-1+Inv[b]**b,
              -1+c**Inv[c],-1+Inv[c]**c, -1+d**Inv[d],-1+Inv[d]**d}
```

Specify to NCAlgebra which variables are known and which are unknown. To GB fans this sets the monomial order indicated around the middle of the first page of the output.

```
In[8]:= SetKnowns[a, Inv[a], b, Inv[b], c, Inv[c], d, Inv[d]]
          SetUnknowns[{z}, {x, y, w}]}
```

Tell NCAlgebra to solve for our unknown variables

```
In[10]:= answer= NCProcess[allRelations, 4, “DemoGBMA” ];
```

```
Outputting results to the stream OutputStream[“DemoGBMA.tex”, 11]
Done outputting results to the stream OutputStream[“DemoGBMA.tex”, 11]
```

The TeX output which appears shows that, if a, b, c and d are invertible, then one can find x, y, z and w such that the matrices above are inverses of each other if and only if z b z = z + d a c.

The TeX output also gives formulas for x, y and w in terms of z.

Input =
$-1 + a\,w + x\,d$
$a\,c + x\,z$
$b\,d + y\,w$
$-1 + b\,z + y\,c$
$-1 + c\,y + w\,a$
$c\,b + w\,x$
$d\,a + z\,y$
$-1 + d\,x + z\,b$
$-1 + a\,a^{-1}$
$-1 + a^{-1}\,a$
$-1 + b\,b^{-1}$
$-1 + b^{-1}\,b$
$-1 + c\,c^{-1}$
$-1 + c^{-1}\,c$
$-1 + d\,d^{-1}$
$-1 + d^{-1}\,d$
File Name = DemoGBMA
NCMakeGB Iterations = 4
NCMakeGB on Digested Iterations = 5
SmallBasis Iterations = 5
SmallBasis on Knowns Iterations = 6
Deselect→ {}
UserSelect→ {}
UserUnknowns→ {}
NCShortFormulas→-1
RR→True
RRByCat→False
SB→False
SBByCat→True
DegreeCap→-1
DegreeSumCap→-1
DegreeCapSB→-1
DegreeSumCapSB→-1
NCCV→True
THE ORDER IS NOW THE FOLLOWING:
$a < a^{-1} < b < b^{-1} < c < c^{-1} < d < d^{-1} \ll z \ll x < y < w$

YOUR SESSION HAS DIGESTED

THE FOLLOWING RELATIONS

THE FOLLOWING VARIABLES HAVE BEEN SOLVED FOR:
$\{w, x, y\}$
The corresponding rules are the following:

$w \rightarrow a^{-1}\, d^{-1}\, z\, b\, d$

$x \rightarrow d^{-1} - d^{-1}\, z\, b$

$y \rightarrow c^{-1} - b\, z\, c^{-1}$

The expressions with unknown variables $\{\}$
and knowns $\{a, b, c, d, a^{-1}, b^{-1}, c^{-1}, d^{-1}\}$

$a\, a^{-1} \rightarrow 1$

$b\, b^{-1} \rightarrow 1$

$c\, c^{-1} \rightarrow 1$

$d\, d^{-1} \rightarrow 1$

$a^{-1}\, a \rightarrow 1$

$b^{-1}\, b \rightarrow 1$

$c^{-1}\, c \rightarrow 1$

$d^{-1}\, d \rightarrow 1$

USER CREATIONS APPEAR BELOW

SOME RELATIONS WHICH APPEAR BELOW

MAY BE UNDIGESTED

THE FOLLOWING VARIABLES HAVE NOT BEEN SOLVED FOR:
$\{a, b, c, d, z, a^{-1}, b^{-1}, c^{-1}, d^{-1}\}$

1.0 The expressions with unknown variables $\{z\}$
and knowns $\{a, b, c, d\}$

$z\, b\, z \rightarrow z + d\, a\, c$

The time for preliminaries was 0:00:06
The time for NCMakeGB 1 was 0:00:01
The time for Remove Redundant 1 was 0:00:03
The time for the main NCMakeGB was 0:00:21
The time for Remove Redundant 2 was 0:00:09
The time for reducing unknowns was 0:00:03

```
The time for clean up basis was 0:00:05
The time for SmallBasis was 0:03:44
The time for CreateCategories was 0:00:01
The time for NCCV was 0:00:19
The time for RegularOutput was 0:00:02
The time for everything so far was 0:04:57
```

Chapter 15

To Run NCGB - Template.nb

Making a Groebner basis and NCProcess template

```
In [1]:= << NCGB.m

           You have already loaded NCGB.m

In [2]:= SetNonCommutative [a,b,c,d,w,x,y,z, Inv[a]    ]

Out [2] = {False, False, False, False, False, False, False, False, False,}

In [3]:= inputPolys =
            {-1+a**w+x**d, a**c+x**z, b**d+y**w, -1+b**z+y**c,
            -1+c**y+w**a,c**b+w**x, d**a+z**y,
            -1+d**x+z**b-1 + a**Inv[a]**b,-1 +Inv[a] ** a }

Out [3] = {-1+a**w+x**d, a**c+x**z, b**d+y**w, -1+b**z+y**c,
             -1+c**y+w**a,c**b+w**x, d**a+z**y,
             -2+d**x+z**b+a**Inv[a]**b,-1+ Inv[a]**a}

In [4]:= SetMonomialOrder [a, Inv[a], b, c, d, {z}, {x, y, w}]
```

Execute one of the cells below depending on whether you wish the output to be sorted and specially formated or not.

Now we compute a partial Groebner basis

```
In [5]:= grobnerBasis = NCMakeGB [inputPolys, 4]

Out [5] = {-1+a**w+x**d, a**c+x**z, -1+c+x**z, -1+c**y+w**a, d**a+z**y,
```

```
-1+Inv[a]**a, -1+a**Inv[a], -1+b, d+y**w, -1+z+y**c, c+w**x,
-w-c**d+w**a**w, -c**z+w**a**c, -z**d+d**a**w,
-z+z**z-d**a**c,-d**a**c**z+z**d**a**c, -1+z+d**x}
```

Now we compute a Groebner basis, reduce some redundant polynomials, e result, and display it in TeX. This may take a long time.

equivalentPolySet = NCProcess[inputPolys, 4, “MyTexFileName”];

Usually much faster than this is

equivalentPolySet = NCProcess[inputPolys, 4, SBByCat → False, “MyTexFileName”];

Also an unreduced but sorted partial GB is

equivalentPolySet = NCProcess[inputPolys, 4, SBByCat → False RR → False, “MyTexFileName”];

Chapter 16

NCProcess: What It Does

What you saw in Chapter 14 was an example of the command NCProcess and of our favorite form for output called a "spreadsheet". Now we give a little bit of an idea of what it does. Later we explain NCProcess in much more detail, since we will not be providing descriptions of how to use commands or much else in this chapter. See Chapters 12 and 17. For more on concepts see Chapter 19. The example in Chapter 17 is a very good illustration of these ideas. The description of how NCProcess is called is given in the previous chapters and Chapter 18.

We use the abbreviations *GB* and *GBA* to refer to Gröbner Basis and Gröbner Basis Algorithm respectively. We begin by describing a facet of noncommutative GB's which we have not yet described. GB's are very effective for eliminating, or solving for, variables.

16.1 NCProcess: Input and Output

The commands which we use heavily are called NCProcess1 and NCProcess2. NCProcess1 and NCProcess2 are variants on a more general command called NCProcess. A person can make use of NCProcess1 and NCProcess2 without knowing any of the NCProcess options.

The NCProcess commands are based upon a GBA and will be described in §18.2. GBA's are very effective at eliminating or solving for variables. Also we have algorithms for removing Redundant equations for the output which can be turned on and off as options. NCProcess2 removes them much more agressively than NCProcess1, but is much slower and it may remove an equation you really like. Also we have algorithms for removing Redundant equations for the output which can be turned on and off as options. A person can use this practical approach to performing computations and proving theorems *without knowing anything about* GBA's or the options.

The **input** to NCProcess command one needs:

I1. A list of knowns.

I2. A list of unknowns (together with an order which gives you priorities for eliminating them).

I3. A collection of equations in these knowns and unknowns. [1]

I4. A number of iterations.

The knowns (I1) are set using the *SetKnowns* command, The unknowns (I2) are set using the *SetUnknowns* command, For example, $SetKnowns[A, B, C]$ sets A, B and C known and $SetUnknowns[x, y, z]$ sets x, y and z unknown. Also, in this case, the algorithms sets the highest prioirity on eliminating z, then y and then x. Some readers might recall this is exactly the information needed as input to NCMakeGB.

The **output** of the NCProcess commands is a list of expressions which are **mathematically equivalent to the equations which are input** (in step I3). That is, the output equations and input equations have exactly the same set of solutions as the input equations. When using NCProcess1, this equivelent list hopefully has solved for some unknowns. The output is presented to the user as

O1. Unknowns which have been solved for and equations which yield these unknowns.

O2. Equations involving no unknowns.

O3. Equations selected or created by the user. [2] For example, in the context of S1 below, one would want to select the equation E_{17}. There are also times during a strategy when one wants to introduce new variables and equations. This is illustrated in Chapter 17.

O4. Equations involving only one unknown.

O5. Equations involving only 2 unknowns. etc.

We say that an equation which is in the output of an NCProcess command is *digested* if it occurs in items O1, O2 or O3 and is *undigested* otherwise. Often, in practice, the digested polynomial equations are those which are well understood.

16.1.1 When to stop

Often one makes a run of NCProcess1 gets some likable equations, puts them in to another run of NCProcess.

The digested equations (those in items O1, O2 and O3) often contain the necessary conditions of the desired theorem and the main flow of the proof of the converse. If the starting polynomial equations follow as algebraic consequences of the digested equations, then we should stop. One might run NCProcess2 at this point inorder to get few equations.

---

[1] In future sections we will refer to two collections of equations: the relation mentioned above as well as a set of **user selected** relations.

[2] These do not exist in the first run. A user-selected equation is a polynomial equation which the user has selected. The algorithm described in §19.3.4 treats these equations as "digested." This, for example, implies that they are given the highest priority in eliminating other equations when NCProcess runs. For example, equations which one knows can be solved by Matlab can be selected.

16.2 Changing Variables

Often to solve a problem one must change variables. Our belief is that for system engineering problems the changes of variables are fairly simple.

The nature of a simple effective class of changes of variables is explained in [HS99] and sketched in Chapter 35 The NCProcess output prompts for changes of variables if `NCCOV` → `True` by placing parentheses in carefully selected places. The experimental commands described in §35 actually automates this change of variable business to some extent.

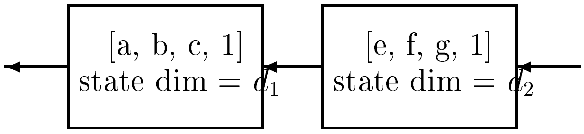
Chapter 17

NCProcess: An Example

The commands `NCProcess1` and `NCProcess2` were briefly described in §16. In this chapter, we derive a theorem due to Bart, Gohberg, Kaashoek and Van Dooren. The reader can skip the statement of this theorem (§17.1) if he wishes and go directly to the algebraic problem statement (§17.2).

17.1 Background

Theorem([BGKvD]) A minimal factorization



of a system $[A, B, C, 1]$ corresponds to projections P_1 and P_2 satisfying $P_1 + P_2 = 1$,

$$AP_2 = P_2AP_2 \quad (A - BC)P_1 = P_1(A - BC)P_1 \tag{17.1}$$

provided the state dimension of the $[A, B, C, 1]$ system is d_1+d_2. (which has the geometrical interpretation that A and $A - BC$ have complimentary invariant subspaces).

We begin by giving the algebraic statement of the problem. Suppose that these factors exist. By the Youla-Tissi statespace isomorphism theorem, there is map

$$(m_1, m_2): \quad \text{Statespace of the product} \longrightarrow \text{Statespace of the original} \tag{17.2}$$

which intertwines the original and the product system. Also minimality of the factoring is equivalent to the existence of a two sided inverse $(n_1^T, n_2^T)^T$ to (m_1, m_2). These requirements combine to imply that each of the following expressions is zero.

17.2 The Problem

Minimal factors exist if and only if there exist m_1, m_2, n_1, n_2, a, b, c, e, f and g such that the following polynomials is zero.

$$(FAC) \qquad \begin{array}{cc} Am_1 - m_1 a - m_2 f c & Am_2 - m_2 e \\ B - m_1 b - m_2 f & -c + C\,m_1 \\ n_1 m_1 - 1 & n_2 m_2 - 1 \\ n_1 m_2 & n_2 m_1 \\ -g + C\,m_2 & m_1 n_1 + m_2 n_2 - 1 \end{array}$$

Each of these expressions must equal 0. Here A, B and C are known.

The problem is to solve these equations. That is, we want a constructive theorem which says when and how they can be solved.

17.3 Solution via a Prestrategy

We now apply a strategy to see how one might discover this theorem. The formalities of what a strategy is are not important here. This chapter is designed to illustrate `NCProcess` and allied commands. For a description of the formalities of a strategy see [HS] or for a sketch see Chapter 19.

Before running `NCProcess1`, we must declare A, B and C to be knowns and the remaining variables to be unknowns. The "**" below denotes matrix multiplication.

```
In[1]:=Get["NCGB.m"];
In[2]:=SetNonCommuative[A,B,C0,m1,m2,n1,n2];
In[3]:=SetKnowns[A,B,C];
In[4]:=SetUnknowns[m1,m2,n1,n2,a,b,c,e,f,g];
```

We now set the variable **FAC** equal to the list of polynomials in §17.2.

```
In[5]:=FAC = {A**m1 - m1**a - m2**f**c,
              A**m2 - m2**e,
              B - m1**b - m2**f,
              -c + C0**m1,
              -g + C0**m2,
              n1**m1 - 1,
              n1**m2,
              n2**m1,
              n2**m2 - 1,
              m1**n1 + m2**n2 - 1};
```

The commands above and below will be explained in Chapter 18.

The command which produces the output in the file **Spreadsheet1.dvi** is the following.

```
In[6]:= result = NCProcess1[FAC,2,"Spreadsheet1"];
```

Here `NCProcess1` is being applied to the set of relations **FAC** for 2 iterations. The `NCProcess1` command has two outputs, one will be stored in **result** and the other will be stored in the file **Spreadsheet1.dvi**. The **Spreadsheet1.dvi** file appears below and is likely to be more interesting and useful than the value of **result**. The file created by `NCProcess` is a list of equations whose solution set is the same as the solution set for **FAC**. (We added the <=== appearing below after the spreadsheet was created.) The → below can be read as an equal sign.

THE ORDER IS NOW THE FOLLOWING:
$A < B < C \ll m_1 \ll m_2 \ll n_1 \ll n_2 \ll a \ll b \ll c \ll e \ll f \ll g$

YOUR SESSION HAS DIGESTED
THE FOLLOWING RELATIONS

THE FOLLOWING VARIABLES HAVE BEEN SOLVED FOR:
$\{a, b, c, e, f, g\}$

The corresponding rules are the following:

$a \to n_1\, A\, m_1$

$b \to n_1\, B$

$c \to C\, m_1$

$e \to n_2\, A\, m_2$

$f \to n_2\, B$

$g \to C\, m_2$

USER CREATIONS APPEAR BELOW

SOME RELATIONS WHICH APPEAR BELOW
MAY BE UNDIGESTED

THE FOLLOWING VARIABLES HAVE NOT BEEN SOLVED FOR:
$\{m_1, m_2, n_1, n_2\}$

2.0 The expressions with unknown variables $\{n_1, m_1\}$
and knowns $\{A, B, C\}$

$n_1\, m_1 \to 1$

$(1 - m_1\, n_1)\, A\, m_1 \; + -1 (1 - m_1\, n_1)\, B C\, m_1 \; = \; 0$ <===

2.0 The expressions with unknown variables $\{n_1, m_2\}$
and knowns $\{A\}$

$n_1\, m_2 \to 0$

$n_1\, A\, m_2 \to 0$

2.0 The expressions with unknown variables $\{n_2, m_1\}$
and knowns $\{A, B, C\}$

$n_2\, m_1 \to 0$

$n_2\, B\, C\, m_1 \to n_2\, A\, m_1$

2.0 The expressions with unknown variables $\{n_2, m_2\}$
and knowns $\{\}$

$n_2\, m_2 \to 1$

4.0 The expressions with unknown variables $\{n_2, n_1, m_2, m_1\}$
and knowns $\{\}$

$m_2\, n_2 \to 1 - 1 m_1\, n_1$ <===

The above "spreadsheet" indicates that the unknowns a, b, c, e, f and g are solved for and states their values. The following are facts about the output: (1) there are no equations in 1 unknown, (2) there are 4 categories of equations in 2 unknowns and (3) there is one category of equations in 4 unknowns. A user must observe that the first equation [1] which we marked with <=== becomes an equation in the unknown quantity $m_1 n_1$ when multiplied on the right by n_1. This motivates the creation of a new variable P defined by setting

$$P_1 = m_1 n_1 \,. \tag{17.3}$$

The user may notice at this point that the second equation marked with <=== is an equation in only one unknown quantity $m_2 n_2$ once the above assignment has been made and P_1 is considered known[2]. These observations lead us to "select" (see footnote corresponding to O2 in §18.2) the equations $m_1 n_1 - P_1$ and $m_2 n_2 - 1 + m_1 n_1$. Since we selected an equation in $m_1 n_1$ and an equation in $m_2 n_2$, it is reasonable to select the the equations $n_1 m_1 - 1$, and $n_2 m_2 - 1$ because they have exactly the same unknowns. While useless at this point we illustrate the command `GetCategory` with the following examples

```
In[10]:= GetCategory[{n1,m1},NCPAns ]
Out[10]= { n1**m1 - 1 }

In[11]:= GetCategory[{n1,m1,n2,m2},NCPAns ]
Out[11]= {m2**n2 + m1**n1 - 1 }
```

Run `NCProcess1` again [3] with (17.3) added and P_1 declared known as well as A, B and C declared known. See Chapter 18.4 for the precise call. The output is:

---

[1] This polynomial is not written as a rule since it has a collected form as described in §22.2. This collected form can be used to assist a person in finding decompositions (see §[**HS**]).

[2] If the user does not notice it at this point, it will become very obvious with an additional run of `NCProcess1`.

[3] There is limit of 2 iterations.

THE ORDER IS NOW THE FOLLOWING:
$A < B < C < P_1 \ll m_1 \ll m_2 \ll n_1 \ll n_2 \ll a \ll b \ll c \ll e \ll f \ll g$

YOUR SESSION HAS DIGESTED
THE FOLLOWING RELATIONS

THE FOLLOWING VARIABLES HAVE BEEN SOLVED FOR:
$\{a, b, c, e, f, g\}$
The corresponding rules are the following:
$a \to n_1\, A\, m_1$
$b \to n_1\, B$
$c \to C\, m_1$
$e \to n_2\, A\, m_2$
$f \to n_2\, B$
$g \to C\, m_2$

The expressions with unknown variables $\{\}$
and knowns $\{A, B, C, P_1\}$
$P_1\, P_1 \to P_1$
$-1 P_1\, A\, (1 + -1 P_1) \;=\; 0$
$A\, P_1 + -1 P_1\, A + -1 (1 + -1 P_1)\, BC\, P_1 = 0$

USER CREATIONS APPEAR BELOW

$m_1\, n_1 \to P_1$
$n_1\, m_1 \to 1$
$n_2\, m_2 \to 1$
$m_2\, n_2 \to 1 + -1 m_1\, n_1$

SOME RELATIONS WHICH APPEAR BELOW
MAY BE UNDIGESTED

THE FOLLOWING VARIABLES HAVE NOT BEEN SOLVED FOR:
$\{m_1, m_2, n_1, n_2\}$

2.0 The expressions with unknown variables $\{n_1, m_1\}$
and knowns $\{P_1\}$
$\Uparrow\;\; m_1\, n_1 \to P_1$
$\Uparrow\;\; n_1\, m_1 \to 1$

2.0 The expressions with unknown variables $\{n_2, m_2\}$
and knowns $\{\}$
$\Uparrow\;\; n_2\, m_2 \to 1$

4.0 The expressions with unknown variables $\{n_2, n_1, m_2, m_1\}$
and knowns $\{\}$
$\Uparrow\;\; m_2\, n_2 \to 1 + -1 m_1\, n_1$

Note that the equations in the above display which are in the undigested section (i.e., below the lowest set of bold lines) are repeats of those which are in the digested section (i.e., above the lowest set of bold lines). The symbol $\Uparrow$ indicates that the polynomial equation also appears as a user select on the spreadsheet. We relist these particular equations simply as a convenience for categorizing them. We will see how this helps us in §17.4. Since all equations are digested, we have finished using `NCProcess1` (see S4). As we shall see, this output spreadsheet leads directly to the theorem about factoring systems.

17.4 The end game

The first step of the end game is to run NCProcess2 on the last spreadsheet which was produced in §17.3. The aim of this run of NCProcess2 is to shrink the spreadsheet as aggressively as possible without destroying important information. The spreadsheet produced by NCProcess2 is the same as the last spreadsheet which was produced [4] in §17.3.

Note that it is necessary that all of the equations in the spreadsheet have solutions, since they are implied by the original equations. The equations involving only knowns play a key role. In particular, they say precisely that, there must exist a projection P_1 such that

$$P_1\,A\,P_1 = P_1\,A \text{ and } P_1\,B\,C\,P_1 = P_1\,A - A\,P_1 + B\,C\,P_1 \tag{17.4}$$

are satisfied.

The converse is also true and can be verified with the assistance of the above spreadsheet. To do this, we assume that the matrices A, B, C and P_1 are given and that (17.4) holds, and wish to define m_1, m_2, n_1, n_2, a, b, c, e, f and g such that each of the equations in the above spreadsheet hold. If we can do this, then each of the equations from the starting polynomial equations (FAC) given in §17.2 will hold and we will have shown that a minimal factorization of the $[A, B, C, 1]$ system exists.

(1) Since $P_1^2 = P_1$, it is easy to show that there exists (not necessarily square) matrices m_1 and n_1 such that $n_1 m_1 = 1$ and $m_1 n_1 = P_1$. These are exactly the equations in the $\{n_1, m_1\}$-Category of the above spreadsheet.

(2) Since $(1 - P_1)^2 = 1 - P_1$, it is easy to show that there exists (not necessarily square) matrices m_2 and n_2 such that $n_2 m_2 = 1$ and $m_2 n_2 = 1 - P_1$. These are exactly the equations in the $\{n_2, m_2\}$-Category of the above spreadsheet together with the equations in the $\{n_2, m_2, n_1, m_1\}$-Category of the above spreadsheet.

(3) Since we have defined m_1, m_2, n_1 and n_2, we can define a, b, c, e, f and g by setting $a = n_1\,A\,m_1$, $b = n_1\,B$, $c = C\,m_1$, $e = n_2\,A\,m_2$, $f = n_2\,B$ and $g = C\,m_2$. These are exactly the equations in the singleton category.

Here we have used the fact that we are working with matrices and not elements of an abstract algebra.

With the assignments made above, every equation in the spreadsheet above holds. Thus, by backsolving through the spreadsheet, we have constructed the factors of the original system $[A, B, C, 1]$. This proves

Theorem ([BGKvD]) The system $[A, B, C, 1]$ can be factored if and only if there exists a projection P_1 such that $P_1\,A\,P_1 = P_1\,A$ and $P_1\,B\,C\,P_1 = P_1\,A - A\,P_1 + B\,C\,P_1$.

---

[4] It is not hard to see that `NCProcess2` would not have an effect, since the set of equations found on the previous spreadsheet can be easily seen to be minimal. We include the run here for pedagogical reasons.

Note that the known equations can be neatly expressed in terms of P_1 and $P_2 = 1 - P_1$. Indeed, it is easy to check with a little algebra that these are equivalent to (17.1). It is a question of taste, not algebra, as to which form one chooses.

For a more complicated example of an end game, see §20.4.

17.4.1 Concluding Remarks

We saw that this factorization problem could be solved with a 2-prestrategy. It was not a 1-prestrategy because there was at least at one point in the session where the user had to make a decision about an equation in two unknowns. On the other hand, the assignment (17.3) was a motivated unknown. We will see in §19.3.1 that this is a 1-prestrategy. For example, the equation

$$(1 - m_1\,n_1)\,Am_1\; - (1 - m_1\,n_1)\,BC\,m_1 \tag{17.5}$$

in the two unknowns m_1 and n_1 can be transformed into an equation in the one unknown m_1n_1 by multiplying by n_1 on the right:

$$(1 - m_1\,n_1)\,Am_1n_1\; - (1 - m_1\,n_1)\,BC\,m_1n_1 \tag{17.6}$$

If we do not restrict ourselves to the original variables but allow constructions of new variables (according to certain very rigid rules), then the factorization problem is solvable using a generalization of a 1-prestrategy, called a 1-strategy. Section 5 of [**HS**] describes 1-strategies.

The brevity of this presentation suppresses some of the advantages and some of the difficulties. For example, one might not instantly have all of the insight which leads to the second spreadsheet. In practice, a session in which someone “discovers” this theorem might use many spreadsheets.

Chapter 18

NCProcess: The Commands

Now that the user has a feel for what we are doing we describe the NCProcess commands one runs over and over again to produce the example in Chapter 17. We are currently working on this command so it is changing with time.

18.1 SetKnowns and SetUnknowns

In most mathematics and engineering appications, certain quantities are considered known and others are considered unknown. The goal is usually to solve for unknowns in terms of the knowns, or to solve for some unknowns in terms of others. In a prestrategy session, one must declare which variables are knowns and which ones are unknowns. While this declaration evolves through the course of a session, it is, at any moment, a part of the computing environment. Indeed, before any of the *NCProcess* commands can be called, it is necessary to set all variables to either be known or be unknown. Mathematically, the variables which are declared unknown are those which one wants to solve for.

An example is :

Example 18.1
```
SetNonCommutative[A,B,C,a,b,c];
SetKnowns[A,B,C];
SetUnknowns[{a,b,c}];
```

After the above three commands have been executed, the user can run any of the **NCProcess** commands (`NCProcess1`, `NCProcess2` or `NCProcess`) on any equations in the variables A, B, C, a, b, c. Here, A, B and C are knowns and a, b and c are unknowns. In the case of `NCProcess` and `NCProcess1`, setting the knowns and unknowns as above has the effect that the **NCProcess** command will try to solve for a, b, c and produce equations from which they have been eliminated. Also the spreadsheet displayed by the **NCProcess** command does bookkeeping based on what is known and unknown.

The above three commands have imposed an order on the variables A, B, C, a, b, c which expresses our priorities for eliminating them. We use the notation

$$A < B < C \ll a < b < c$$

to denote the order imposed in the example. The **NCProcess** command will try hardest to eliminate c and the least to solve for A.

If you are running a prestrategy session and stop making progress one of the first things to try is changing the order.

The $\ll$ means that the **NCProcess** commands go to much greater lengths [1] towards eliminating a, b, c than it does for A, B, C.

A fancier example of such prioritizing is :

Example 18.2
```
SetNonCommutative[A,B,C,a,b,c,e,f,g];
SetKnowns[A,B,C];
SetUnknowns[{a,b,c},{d,e,f}];
```

This produces the ordering

$$A < B < C \ll a < b < c \ll d < e < f \,,$$

There is an alternative to *SetKnowns* and *SetUnknowns*. The command `SetMonomialOrder[{A,B,C},{a,b,c},{d,e,f}]` has exactly the same effect as the commands in Example 18.2.

One can proceed with an *NCProcess* command, only after an ordering is set.

18.2 NCProcess

The workhorse commands of a strategy are the NCProcess1 and NCProcess2 commands. These two commands fit the general mold of the NCProcess command. In particular, each option of NCProcess is also an option of NCProcess1 and NCProcess2. See §18.6.

18.2.1 NCProcess[aListOfPolynomials,iterations,fileName, Options]

Aliases: **None**

Description: *NCProcess[aListOfPolynomials,iterations,fileName]* finds a new generating set for the ideal generated by *aListOfPolynomials* along the lines of the demo presented in Chapter 19. The spreadsheets presented in §19.3.1 are the contents of the files *fileName*, and are produced by the command *NCProcess*.

In addition to creating the file *fileName*, *NCProcess* returns as Mathematica output a list consisting of three lists.

---

[1] '$\ll$' are called multigraded lexicographic orders. Intuitively, we think of A, B and C as corresponding to variables in some engineering problem which represent quantities which are known and think of a, b, c, d, e and f as corresponding to variables in the engineering problem which represent quantities which are unknown. The fact that d, e and f are in the top level indicates that we are very interested in solving for d, e and f in terms of A, B, C, a, b and c, but are not willing to solve for b in terms of expressions involving either d, e or f.

More precise discussion of mulitgraded lex orderings are in Chapter 21.

(1) A partial GB for the digested relations.

(2) The digested relations in the simplified second partial GB.

(3) The undigested relations in the simplified second partial GB.

In practice, the user runs *NCProcess*, then looks at the file *fileName* in order to get ideas for the next step. When the user decides on the next step, he can use some of the lists of Mathematica output in addition to some new relations as inputs for the next call to `NCProcess`. There are many options for `NCProcess`.

Arguments: *aListOfPolynomials* is a list of polynomials. *iterations* is a natural number. [2] *fileName* is a character string. If *fileNames*'s last four characters are not ".dvi", then ".dvi" is appended to *fileName*.

Comments / Limitations: Not available before NCAlgebra 1.2 **If you are using NCProcess in the Windows environment you will have to quit the dvi previewer to continue your Mathematica session.**

The command `NCProcess` calls `NCMakeGB[aListOfPolynomials,iters]`. `NCMakeGB` is an algorithm for producing a partial Gröbner basis. This produces many new relations whose solution set is the same as the solution set for *aListOfPolynomials*. Typically, many of the relations follow from other relations within the same output. There are many options for *NCProcess* which remove "mathematically redundant" relations before generating the spreadsheet in *fileName* and lists (2) and (3) of the Mathematica output. The various options as well as the default options for *NCProcess* are described in §22.

18.2.2 Examples

Here are some examples of how the *NCProcess* commands are called.

```
NCProcess1[aListOfPolys,2,"filname"]

list1 = NCProcess1[aListOfPolys,2,"filname",
                                       DegreeCap->8, DegreeSumCap->12]

list2 = NCProcess2[aListOfPolys,2,"filname",UserSelect->anotherList,
                                       DegreeCap->6, DegreeSumCap->10]
```

---

[2] Up to four iteration numbers can be specified. When only one is given, *NCProcess* uses that number to choose default values for the other iteration numbers. If the user specifies four values, I_1, I_2, I_3, I_4, then I_1 is the iterations for `NCMakeGB`, I_2 is the iterations for `NCMakeGB` on the digesteds, I_3 is the iterations for `SmallBasis`, and I_4 is the iterations for the knowns in `SmallBasis`.

18.3 Commonly Used NCProcess Options and Commands

18.3.1 UserSelect → aListOfPolynomials

A valuable option for *NCProcess* is *UserSelect*. Using the option

$$UserSelect \rightarrow aListOfPolynomials$$

forces the elements of the list *aListOfPolynomials* to appear in a "User Selected" part of the spreadsheet generated during this call to *NCProcess*. It also affects the order in which the polynomials are used inside `NCMakeGB` as well as reduction algorithms described in §22.1. The user selected polynomials are processed early and polynomials processed later tend to be eliminated.

18.3.2 DegreeCap→ $aNumber1$ and DegreeSumCap→ $aNumber2$

One way to reduce the run time for *NCProcess* is to use the options capping the degree of the polynomials that are produced in the course of running *NCProcess*.

This is valuable since the user will ordinarily know that a polynomial of a very high degree will not be useful to him, hence there is no reason to produce it. It is not the time that it takes to produce a large polynomial that is the primary factor; rather, it is the reduction algorithms that will get bogged down trying to remove it. Degree caps can prevent the algorithm from ever producing polynomials over a certain degree, or combining polynomials over a certain degree, and the user will still be left with a generating set for the ideal generated by the input relations. There are two different options associated with degree caps. For instance,

$$\text{DegreeCap} \rightarrow 8$$

would prevent a polynomial of degree 8 or higher from combining with a polynomial of higher degree.

$$\text{DegreeSumCap} \rightarrow 10$$

would prevent two polynomials whose degrees add up to 10 or more from combining. Degree caps could prevent an important relation from being created, so when there is a lack of progress, raising the degree caps as well as the iteration number would be the next step.

WE URGE USE OF DEGREE CAPS. THEY SAVE A LOT OF TIME.

In addition, *DegreeCap* and *DegreeSumCap* are options which cap the degree of polynomials occurring in `NCMakeGB`. Indeed, calling DegreeCap in *NCProcess* just sets the DegreeCap for running `NCMakeGB` inside of *NCProcess*.

18.3.3 MainUnknowns→ $aListOfIndeterminates$

If $aListOfIndeterminates$ are given, then the output of `NCProcess` will only include equations whose unknowns are all contained in aListOfIndeterminates or are functions of them.

18.3.4 NCShortFormula→ *Length*

If NCShortFormula is set to an integer, *Length*, the output of NCProcess will only contain expressions whose Mathematica defined `LeafCount[ ]` is less than *Length*. This could be useful in producing TEX output when *very* long expressions are involved. This option also reduces run time significantly since there is much less output to process. We often have good luck with NCShortFormulas→300. If set to -1 no length elimination will be done.

WARNING The two options given above, NCShortFormula and MainUnknowns, actually eliminate polynomials from the output of NCProcess. If one of these options is set the system of equations which is the output of NCProcess is not equivalent to the system of equations which is input as is usually the case.

18.3.5 Getting Categories

$GetCategory[aListOfVariables, NCPAns]$ retrieves the polynomials in the category stored in $NCPAns$ corresponding to the list of variables in the list $aListOfVariables$. See Chapter 17.3. To be recognized immediately after an NCProcess run $aListOfVariables$ must equal a list of unknowns which corresponds to a category in that NCProcess run. The NCProcess stores all category information in NCPAns. The next NCProcess starts by clearing NCPAns and writes the category information it produces in NCPAns.

18.4 Typical use of the NCProcess command

Now we demonstrate the $NCProcess$ command by "solving" FAC as already discussed in Chapter 17. While we show an interactive session the user probably will want to type the commands into a file and load them into your session. This is standard practice among Mathematica users and saves lots of time.

As shown in §17.2, the prestrategy starts as follows.

```
In[1]:=Get["NCGB.m"];
In[2]:=SetNonCommutative[A,B,C0,m1,m2,n1,n2];
In[3]:=SetKnowns[A,B,C];
In[4]:=SetUnknowns[m1,m2,n1,n2,a,b,c,e,f,g];
In[5]:=FAC = {A**m1 - m1**a - m2**f**c,
              A**m2 - m2**e,
              B - m1**b - m2**f,
              -c + C0**m1,
              -g + C0**m2,
              n1**m1 - 1,
              n1**m2,
              n2**m1,
              n2**m2 - 1,
              m1**n1 + m2**n2 - 1};
In[6]:= result = NCProcess1[FAC,2,"Spreadsheet1"];
```

The first command simply loads the noncommutative Gröbner basis package into Mathematica. Then we set the variables to be noncommutative.

The third and fourth commands sets the monomial order that will be used in solving for and eliminating variables as explained in Chapter 21. The list **FAC** is the set of relations that describe the problem as explained in §17.2. The call to $NCProcess1$ will run two iterations on **FAC** before creating the file "Spreadsheet1.dvi" and returning the final **result** as Mathematica output.

The file "Spreadsheet1.dvi" is the same as the one which was produced in Chapter 19. It exists outside of Mathematica and is only for viewing. It cannot be used to create a list of polynomials for Mathematica. The strategy that the user follows at this point was described in §17.3. Recall that it led the user to select the relations

$$m_1\, n_1 - P_1,\ n_1\, m_1 - 1,\ n_2\, m_2 - 1,\ m_2\, n_2 - 1 + m_1\, n_1$$

and to declare the new variable P_1 to be known.

Next we describe how this is input to $NCProcess1$ in order to produce the second spreadsheet from §17.3.

The Mathematica output of any of the $NCProcess$ commands is a list containing three lists. In the example above, that list is named **result**. Recall that in Mma the way to get sublist number j inside **result** is to type $result[[j]]$.

The second round of this strategy session is

```
In[7]:= SetKnowns[A,B,C,P1];
In[8]:= digested = result[[2]]
In[9]:= undigested = result[[3]]
In[10]:= interesting = {m1**n1-P1,n1**m1-1,n2**m2-1,m2**n2-1+m1**n1};
In[11]:= nextinput = Union[digested,interesting,undigested]
In[12]:= NCProcess1[nextinput,2,"Spreadsheet2",
                    UserSelect->discovered,
                    DegreeCap->3,DegreeSumCap->6];
```

Now we discuss the input to $NCProcess1$ above. The variables **digested** and **undigested** hold the the last two lists from the list **result** which the first $NCProcess1$ command returned, and **discovered** includes the new definition of P_1. The union of these three lists is taken as the input for the second $NCProcess1$ run. Once again we used two iterations. This is a good number of iterations with which to start. More iterations can be used if there is a lack of progress. The next argument is a string which refers to a file name which will store the spreadsheet. Unlike the first call to `NCProcess1`, this call uses non-default values for the options. The *UserSelect* option (as described in §18.3.1) causes the **discovered** relations to be given high priority. The DegreeCaps are set in order to save time (as explained in §18.3.2).

The file "Spreadsheet2.dvi" is the same as the second spreadsheet which was produced in Chapter 19. As we saw in §17.4, this spreadsheet leads directly to the main theorem about factoring systems.

For another demonstration of strategies using $NCProcess$ see Chapter 20.

18.5 Details of NCProcess

The following section gives psuedocode for the NCProcess1 and NCProcess2 commands.

This psuedocode uses the function `NCMakeGB` which implements a Gröbner Basis Algorithm which returns partial GB's which are reduced. In particular, running the function `NCMakeGB` for 0 iterations on a set F does not compute any S-polynomials, but does produce a set G which is reduced. G will be called a reduced form of F.

18.5.1 NCProcess1 command

The input to NCProcess1 is a set of starting equations *start*, a number of iterations n for the GBA and a collection of user selects.

The steps NCProcess1 takes are:

I. Preparation for the main call to `NCMakeGB`

(1) Run the GBA on the equations in *start* which do not involve any unknown variables together with the user selects for at most $n+1$ iterations. Let A denote this partial GB.

(2) Shrink A using the RemoveRedundantProtected operation. Call this shrunken set B.

II. The main call to `NCMakeGB`

(3) Run `NCMakeGB` with the input B together with *start* for at most n iterations. In this `NCMakeGB` run, S-polynomials between two elements of the partial GB of A are not computed. Let C denote this partial GB.

III. Shrinking the partial GB

(4) Shrink C using the RemoveRedundantProtected operation. Call this shrunken set D.

(5) Let D_k be the set of polynomials in D which do not involve any unknowns. Let $D_u = D \backslash D_k$. Let E_u be a set of the normal forms of the elements of D_u with respect to D_k. Let $E = D_k \cup E_u$.

(6) Let F be the union of E and the user selects. Let G be a reduced form of F (see the beginning of §18.5).

(7) Shrink G by SmallBasisByCategory using iteration parameters $n+1$ and $n+2$. Call this shrunken set H.

IV. Attempt Decompose

(8) Construct the collected forms of the polynomials in H.

V. Displaying the results

(9) For elements of H, if the polynomial's only collected form is trivial, then display the rule corresponding to the polynomial, otherwise display the collected form of the polynomial. This is the step in which the "spreadsheets" of the results of this paper are constructed.

VI. Return a three tuple to the user for future use

(10) Return the triple (A, H_0, H_1) to the user where A is from item 1 above, H_0 is the set of polynomials in H which are digested and H_1 is the set of polynomials in H which are undigested.

18.5.2 NCProcess2 command

The input to NCProcess2 is a set of starting equations *start*, a number of iterations n for SmallBasis and a collection of user selects.

The steps taken by NCProcess2 are:

I. Shrinking the input equations

(1) Shrink *start* using the RemoveRedundantProtected operation. Call this shrunken set D.

(2) Let D_k be the set of polynomials in D which do not involve any unknowns. Let $D_u = D \backslash D_k$. Let E_u be a set of the normal forms of the elements of D_u with respect to D_k. Let $E = D_k \cup E_u$.

(3) Let F be the union of E and the user selects. Let G be a reduced form of F. (see the beginning of §18.5).

(4) Shrink G by SmallBasis. Set H equal to the result of the shrinking.

II. The "Attempt Decompose" "Displaying the results" and "Return a three tuple to the user for future use" as in §18.5.1.

18.6 NCProcess1 and NCProcess2: The technical descriptions

Below, *aList* is a list of polynomials, n is a positive integer, *filename* is a character string and rules is a sequence of zero or more Mathematica rules which correspond to NCProcess options.

```
NCProcess1[aList_,n_,filename_,rules___Rule]:=
   NCProcess[aList,n,filename,SBByCat->True,RR->True,rules];

NCProcess2[aList_,n_,filename_,rules___Rule]:=
   NCProcess[aList,n,filename,SB->True,RR->True,rules];
which is the same as
   NCProcess[aList,n,n+1,n+1,n+2,filename,SB->True,RR->True,rules];
```

For an understanding of "`n,n+1,n+1,n+2`", see the footnote in §18.2 Arguments.

Part V

NCGB: FOR THE MORE ADVANCED USER

Chapter 19

NCProcess: The Concepts

We turn to a much more adventurous pursuit which is in a primitive stage. This is a highly computer assisted method for discovering certain types of theorems.

At the beginning of "discovering" a theorem, an engineering or math problem is often presented as a large system of matrix or operator equations. The point of the method is to isolate and to minimize what the user must do by running algorithms heavily. Often when viewing the output of the algorithms, one can see what additional hypothesis should be added to produce a useful theorem and what the relevant matrix quantities are.

Many theorems in engineering systems, matrix and operator theory amount to giving hypotheses under which it is possible to solve large collections of equations. (It is *not* our goal to reprove already proven theorems, but rather to develop technique which will be useful for discovering new theorems.)

Rather than use the word "algorithm," we call our method a *strategy* since it allows for modest human intervention. We are under the impression that many theorems might be derivable in this way. A detailed description of a strategy is given in [**HS**]. Prestrategies are a particular type of strategy which are easier to explain. This chapter will describe the ideas behind a prestrategy.

In this chapter, unlike Chapter 12, we will not be providing descriptions of how to use commands. Our goal is just to mention the main ideas. These ideas are described in detail in [**HS**]. The example in Chapter 17 is a very good illustration of these ideas. The description of how NCProcess is called is given in Chapter 18.

We use the abbreviations *GB* and *GBA* to refer to Gröbner Basis and Gröbner Basis Algorithm respectively. We begin by describing a facet of noncommutative GB's which we have not yet described. GB's are very effective for eliminating, or solving for, variables.

19.1 NCProcess: Input and Output

The commands which we use to implement a prestrategy are called NCProcess1 and NCProcess2. NCProcess1 and NCProcess2 are variants on a more general command called NCProcess. A person can make use of NCProcess1 and NCProcess2 without knowing any of the NCProcess options.

The NCProcess commands are based upon a GBA and will be described in §18.2. GBA's are very effective at eliminating or solving for variables. A person can use this practical approach to performing computations and proving theorems *without knowing anything about* GBA's. Indeed, this chapter is a self-contained description of our method.

The **input** to NCProcess command one needs:

I1. A list of knowns.

I2. A list of unknowns (together with an order which gives you priorities for eliminating them).

I3. A collection of equations in these knowns and unknowns. [1]

I4. A number of iterations.

The knowns (I1) are set using the *SetKnowns* command, The unknowns (I2) are set using the *SetUnknowns* command, For example, $SetKnowns[A, B, C]$ sets A, B and C known and $SetUnknowns[x, y, z]$ sets x, y and z unknown. Also, in this case, the algorithms sets the highest prioirity on eliminating z, then y and then x. Some readers might recall this is exactly the information needed as input to NCMakeGB.

The **output** of the NCProcess commands is a list of expressions which are **mathematically equivalent to the equations which are input** (in step I3). That is, the output equations and input equations have exactly the same set of solutions as the input equations. When using NCProcess1, this equivelent list hopefully has solved for some unknowns. The output is presented to the user as

O1. Unknowns which have been solved for and equations which yield these unknowns.

O2. Equations involving no unknowns.

O3. Equations selected or created by the user. [2] For example, in the context of S1 below, one would want to select the equation E_{17}. There are also times during a strategy when one wants to introduce new variables and equations. This is illustrated in Chapter 17.

O4. Equations involving only one unknown.

O5. Equations involving only 2 unknowns. etc.

We say that an equation which is in the output of an NCProcess command is *digested* if it occurs in items O1, O2 or O3 and is *undigested* otherwise. Often, in practice, the digested polynomial equations are those which are well understood.

---

[1] In future sections we will refer to two collections of equations: the relation mentioned above as well as a set of **user selected** relations.

[2] These do not exist in the first run. A user-selected equation is a polynomial equation which the user has selected. The algorithm described in §19.3.4 treats these equations as "digested." This, for example, implies that they are given the highest priority in eliminating other equations when NCProcess runs. For example, equations which one knows can be solved by Matlab can be selected.

19.2 Elimination

Since we will not always let the *GBA* algorithm run until it finds a Gröbner Basis, we will often be dealing with sets which are not Gröbner Basis, but rather an intermediate result. We call such sets of relations *partial GB's*.

Commutative Gröbner Basis Algorithms can be used to systematically eliminate variables from a collection (e.g., $\{p_j(x_1,\ldots,x_n)=0 : 1\le j\le k_1\}$) of polynomial equations so as to put it in triangular form. One specifies an order on the variables ($x_1<x_2<x_3<\ldots<x_n$) [3] which corresponds to ones priorities in eliminating them. Here a GBA will try hardest to eliminate x_n and try the least to eliminate x_1. The output from it is a list of equations in a "canonical form" which is triangular: [4]

$$\begin{aligned} q_1(x_1)&=0 && (19.1)\\ q_2(x_1,x_2)&=0 && (19.2)\\ q_3(x_1,x_2)&=0 && (19.3)\\ q_4(x_1,x_2,x_3)&=0 && (19.4)\\ &\ldots && (19.5)\\ q_{k_2}(x_1,\ldots,x_n)&=0\,. && (19.6)\end{aligned}$$

Here the polynomials $\{q_j : 1\le j\le k_1\}$ generate the same ideal that the polynomials $\{p_j : 1\le j\le k_2\}$ do. Therefore, the set of solutions to the collection the polynomial equations $\{p_j=0 : 1\le j\le k_1\}$ equals the set of solutions to the collection of polynomial equations $\{q_j=0 : 1\le j\le k_2\}$. This canonical form greatly simplifies the task of solving the collection of polynomial equations by facilitating backsolving for x_j in terms of $x_1,\ldots,x_{j-1}$. The effect of the ordering is to specify that variables high in the order will be eliminated while variables low in the order will not be eliminated.

In the noncommutative case, again a GB for a collection of polynomial equations is a collection of noncommuting polynomial equations in triangular form (see [**HS**]). There are some difficulties which don't occur in the commutative case. For example, a GB can be infinite in the noncommutative case. However, we present software here based on the noncommutative GBA which might prove to be extremely valuable in some situations.

19.3 What is a prestrategy?

We wish to stress that one does not need to *know* a theorem in order to *discover* it using the techniques in this paper. Any method which assumes that all of the hypotheses can be stated algebraically and that all of the hypotheses are known at the beginning of the computation will be of limited practical use. For example, since the Gröbner Basis algorithm only discovers polynomial equations which are algebraically true and not those which require analysis or topology, the use of this algorithm alone has a limited use.

---

[3] From this ordering on indeterinates one induces a order on polynomials which is different than we used before. There we used a graded order here we use a strict lexicographic order.

[4] There need not be $\le$ n equations in this list and there need not be any equation in just one variable.

Insights gained from analysis during a computer session could be added as (algebraic) hypotheses while the session is in progress. Decisions can take a variety of forms and can involve recognizing a Riccati equation, recognizing that a particular square matrix is onto and so invertible, recognizing that a particular theorem now applies to the problem, etc. The user would then have to record and justify these decisions independently of the computer run. [5] While a strategy allows for human intervention, the intervention must follow certain rigid rules for the computer session to be considered a strategy.

19.3.1 Prestrategy

The idea of a *prestrategy* is :

S0. Set $C' = \{\}$ (see footnote in §19.1 on I3.)

S1. Run NCProcess1 which creates a display of the output (see O1-O5 in §18.2) and look at the list of equations involving only one unknown (say a particular equation E_{17} contains only x_3).

S2. The user must now make a decision about equations in x_3 (e.g., E_{17} is a Riccati equation so I shall not try to simplify it, but leave it for Matlab). Now the user declares the unknown x_3 to be known.

S3. Either do the "End game" (see §19.3.2) or Go to S1.

The above listing is, in fact, a statement of a *1-prestrategy*. Sometimes one needs a *2-prestrategy* in that the key is equations in 2 unknowns.

The point is to isolate and to minimize what the user must do. This is the crux of a prestrategy.

19.3.2 When to stop

The prestrategy described above is a loop and we now discuss when to exit the loop.

The digested equations (those in items O1, O2 and O3) often contain the necessary conditions of the desired theorem and the main flow of the proof of the converse. If the starting polynomial equations follow as algebraic consequences of the digested equations, then we should exit the above loop. The starting equations, say $\{p_1 = 0, \ldots, p_{k_1} = 0\}$, follow as algebraic consequences of the digested equations, say $\{q_1 = 0, \ldots, q_{k_2} = 0\}$, if and only if the Gröbner Basis generated by $\{q_1, \ldots, q_{k_2}\}$ reduces (in a standard way) the polynomial p_j to 0 for $1 \leq j \leq k_1$. Checking whether or not this happens is a purely mechanical process.

When one exits the above loop, one is presented with the question of how to finish off the proof of the theorem. We shall call the steps required to go from a final spreadsheet to the actual theorem the "end game." We shall describe some "end game" technique in §19.3.4. We shall illustrate the "end game" in §17.4 and §20.4. As we shall see, typically the first step is to run NCProcess2 whose output is a very small set of equations.

---

[5] See Appendix 41 for an example of this.

19.3.3 Redundant Equations

We mentioned earlier that NCProcess uses the Gröbner Basis algorithm. This GBA is implemented via the command NCMakeGB. If NCProcess consisted of a call to the GBA and the formatted output (§18.2) alone, then NCProcess would not be a powerful enough tool to generate solutions to engineering or math problems. This is because it would generate too many equations. It is our hope that the equations which it generates contain all of the equations essential to solution of whatever problem you are treating. For the problems we have considered, this has been our experience. On the other hand, it contains equations derived from these plus equations derived from those derived from these as well as precursor equations which are no longer relevant. That is, a GB contains a few jewels and lots of garbage. In technical language a GB is almost never a small basis for an ideal and what a human seeks in discovering a theorem is a small basis for an ideal. [6] Thus we have algorithms and substantial software for finding small (or smallest) sets of equations associated to a problem. The process of running GBA followed by an algorithm for finding small sets of equations is what constitutes NCProcess.

19.3.4 Summary of a Prestrategy

We have just given the basic ideas. As a prestrategy proceeds, more and more equations are digested by the user and more and more unknowns become knowns. Thus we ultimately have two classes of knowns: original knowns $\mathcal{K}_0$ and user designated knowns $\mathcal{K}_U$. Often a theorem can be produced directly from the output by taking as hypotheses the existence of knowns $\mathcal{K}_U \cup \mathcal{K}_0$ which are solutions to the equations involving only knowns.

Assume that we have found these solutions. To prove the theorem, that is to construct solutions to the original equations, we must solve the remaining equations. Fortunately, the digested equations often are in a block triangular form which is amenable to backsolving. This is one of the benefits of "digesting" the equations.

An example, makes all of this more clear.

19.4 A strategy

A strategy is like a prestrategy except in addition the user can make (and the program prompts) certain changes of variables. The nature of these changes of variables is explained in [HS99] and sketched in Chapter 35. The NCProcess output prompts for changes of variables if `NCCOV` $\rightarrow$ `True` by placing parentheses in carefully selected places. The experimental commands described in §35 actually automate this change of variable business to some extent.

---

[6] The use of the word "small" rather than minimal is intentional. See §12 of [**HS**].

Chapter 20

Another Example: Solving the H^{∞} Control Problem

In this section we give a more extensive example of a strategy. For more information of strategies, see [**HS**]. The command `NCCollectOnVariables` is an extremely useful command which "collects" knowns and products of knowns out of expressions. For example, suppose that A and B are knowns and X,Y and Z are unknowns. The collected form of

$$X **A **B **Z + Y **A **B **Z + A **X + A **Y$$

is $(X + Y) **A **B **Z + A **(X + Y)$. This is discussed more in Appendix 22.

The demonstration in this section will repeatedly call the **NCProcess** commands with the option `NCCV->True` (which is the default). This displays formulas in the spreadsheet in an informative way. Whenever a "collected" form of a polynomial is found, the **NCProcess** command displays it in the collected form rather than as a rule.

A basic problem in systems engineering is to make a given system dissipative by designing a feedback law. We now give a demonstration of how one discovers the algebraic part of the solution to this problem. The following section is reasonably self-contained.

20.1 Problem statement

Let H_{xx}, H_{xz}, H_{zx} and H_{zz} be defined as follows.

$$\begin{aligned} H_{xx} = E_{11}\,A + A^T\,E_{11} + C_1^T\,C_1 + E_{12}^T\,b\,C_2 + C_2^T\,b^T\,E_{12}^T \\ +E_{11}\,B_1\,b^T\,E_{12}^T + E_{11}\,B_1\,B_1^T\,E_{11} + \\ E_{12}\,b\,b^T\,E_{12}^T + E_{12}\,b\,B_1^T\,E_{11} \end{aligned}$$

$$\begin{aligned} H_{xz} = E_{21}\,A + \frac{a^T\,(E_{21} + E_{12}^T)}{2} + c^T\,C_1 + E_{22}\,b\,C_2 + c^T\,B_2^T\,E_{11}^T + \\ \frac{E_{21}\,B_1\,b^T\,(E_{21} + E_{12}^T)}{2} + E_{21}\,B_1\,B_1^T\,E_{11}^T + \end{aligned}$$

$$\frac{E_{22}\,b\,b^T\,(E_{21}+E_{12}^T)}{2} + E_{22}\,b\,B_1^T\,E_{11}^T$$

$$\begin{aligned} H_{zx} = A^T\,E_{21}^T + C_1^T\,c + \frac{(E_{12}+E_{21}^T)\,a}{2} + E_{11}\,B_2\,c + C_2^T\,b^T\,E_{22}^T + \\ E_{11}\,B_1\,b^T\,E_{22}^T + E_{11}\,B_1\,B_1^T\,E_{21}^T + \\ \frac{(E_{12}+E_{21}^T)\,b\,b^T\,E_{22}^T}{2} + \frac{(E_{12}+E_{21}^T)\,b\,B1^T\,E_{21}^T}{2} \end{aligned}$$

$$\begin{aligned} H_{zz} = E_{22}\,a + a^T\,E_{22}^T + c^T\,c + E_{21}\,B_2\,c + c^T\,B_2^T\,E_{21}^T + \\ E_{21}\,B_1\,b^T\,E_{22}^T + E_{21}\,B_1\,B_1^T\,E_{21}^T + \\ E_{22}\,b\,b^T\,E_{22}^T + E_{22}\,b\,B_1^T\,E_{21}^T \end{aligned}$$

The math problem we address is:

(HGRAIL) *Let A, B_1, B_2, C_1, C_2 be matrices of compatible size be given. Solve $H_{xx}=0$, $H_{xz}=0$, $H_{zx}=0$, and $H_{zz}=0$ for a, b, c and for E_{11}, E_{12}, E_{21} and E_{22}. When can they be solved? If these equations can be solved, find formulas for the solution.*

20.2 The key relations: executable form

The first step is to assemble all of the key relations in executable form:

The properties of E_{ij}

We make the strong assumption that each E_{ij} is invertible. While this turns out to be valid, making it at this point is cheating. Ironically we recommend strongly that the user make heavy invertibility assumptions at the outset of a session. Later after the main ideas have been discovered the user can selectively relax them and thereby obtain more general results.

Also, the 2 x 2 matrix (E_{ij}) $1 \leq i,j \leq 2$ is symmetric.

Creating an input file

It is a good idea to create an input file for the strategy session before starting Mathematica. There are several preliminary steps which may have to be done several times before the desired results are obtained. Setting the monomial order, setting variables noncommutative and defining the starting equations can all be done beforehand. This way, if the user wants to try the same run with a slightly different ordering, he only needs to edit the input file and load it again.

In this case, the input file is called **cntrl**. Notice that we are using **Tp**[ ] to denote transpose and **Inv**[] to denote inverse, which is inconsistent with the NCAlgebra package which uses **tp**[ ] and **inv**[ ] respectively. This is done intentionally to ensure that actual transposes and inverses are not taken within the GBA. The relations can be easily converted later if it is necessary to do mathematical operations on them.

This is the file **"cntrl"**:

```
SetNonCommutative[E11,E22,E12,E21,Inv[E11],Inv[E22],Inv[E12],Inv[E21],
                  Tp[Inv[E11]],Tp[Inv[E22]],Tp[Inv[E12]],Tp[Inv[E21]],
                  Tp[E11],Tp[E22],Tp[E12],Tp[E21]];

(* These relations imply that the Eij  generate a symmetric quadratic form. *)
transE = {
Tp[E21]-E12,
Tp[E11]-E11,
Tp[E22]-E22,
Tp[E12]-E21,
Tp[Inv[E21]]-Inv[E12],
Tp[Inv[E11]]-Inv[E11],
Tp[Inv[E22]]-Inv[E22],
Tp[Inv[E12]]-Inv[E21]};

(* These relations assume that everything is invertible *)
inverseE = {
E11**Inv[E11] -1,
Inv[E11]**E11 -1,
E12**Inv[E12] -1,
Inv[E12]**E12 -1,
E21**Inv[E21] -1,
Inv[E21]**E21 -1,
E22**Inv[E22] -1,
Inv[E22]**E22 -1,
Tp[E11]**Tp[Inv[E11]] -1,
Tp[Inv[E11]]**Tp[E11] -1,
Tp[E12]**Tp[Inv[E12]] -1,
Tp[Inv[E12]]**Tp[E12] -1,
Tp[E21]**Tp[Inv[E21]] -1,
Tp[Inv[E21]]**Tp[E21] -1,
Tp[E22]**Tp[Inv[E22]] -1,
Tp[Inv[E22]]**Tp[E22] -1};

SetNonCommutative[A,Tp[A],B1,Tp[B1],B2,Tp[B2],
                  C1,Tp[C1],C2,Tp[C2],
                  b,Tp[b],c,Tp[c],a,Tp[a]];

(* These are the Hamiltonian equations *)
Hxx=E11 ** A + Tp[A] ** Tp[E11] + Tp[C1] ** C1 +
  Tp[C2] ** Tp[b] ** (E21 + Tp[E12])/2 + (E12 + Tp[E21]) ** b ** C2/2 +
  E11 ** B1 ** Tp[b] ** (E21 + Tp[E12])/2 + E11 ** B1 ** Tp[B1] ** Tp[E11] +
  (E12 + Tp[E21]) ** b ** Tp[b] ** (E21 + Tp[E12])/4 +
  (E12 + Tp[E21]) ** b ** Tp[B1] ** Tp[E11]/2;

Hxz=E21 ** A + Tp[a] ** (E21 + Tp[E12])/2 + Tp[c] ** C1 + E22 ** b ** C2 +
  Tp[c] ** Tp[B2] ** Tp[E11] + E21 ** B1 ** Tp[b]** (E21 + Tp[E12])/2 +
  E21 ** B1 ** Tp[B1] ** Tp[E11] + E22 ** b ** Tp[b] ** (E21 + Tp[E12])/2 +
  E22 ** b ** Tp[B1] ** Tp[E11];
```

```
Hzx=Tp[A] ** Tp[E21] + Tp[C1] ** c + (E12 + Tp[E21]) ** a/2 + E11 ** B2 ** c +
  Tp[C2] ** Tp[b] ** Tp[E22] + E11 ** B1 ** Tp[b] ** Tp[E22] +
  E11 ** B1 ** Tp[B1] ** Tp[E21] +
  (E12 + Tp[E21]) ** b ** Tp[b] ** Tp[E22]/2 +
  (E12 + Tp[E21]) ** b ** Tp[B1] ** Tp[E21]/2;


Hzz=E22 ** a + Tp[a] ** Tp[E22] + Tp[c] ** c + E21 ** B2 ** c +
  Tp[c] ** Tp[B2] ** Tp[E21] + E21 ** B1 ** Tp[b] ** Tp[E22] +
  E21 ** B1 ** Tp[B1] ** Tp[E21] + E22 ** b ** Tp[b] ** Tp[E22] +
  E22 ** b ** Tp[B1] ** Tp[E21];

Hameq = {Hxx,Hxz,Hzx,Hzz};

(* Set the knowns and the order of the unknowns *)
SetKnowns[A,Tp[A],B1,Tp[B1],B2,Tp[B2],C1,Tp[C1],C2,Tp[C2]];
SetUnknowns[E12,Tp[E12],E21,Tp[E21],E22,Tp[E22],E11,Tp[E11],
            Inv[E12],Tp[Inv[E12]],Inv[E21],Tp[Inv[E21]],
             Inv[E22],Tp[Inv[E22]],Inv[E11],Tp[Inv[E11]],
            b,Tp[b],c,Tp[c],a,Tp[a]];

startingrels = Union[transE, inverseE, Hameq];

result1 = NCProcess1[startingrels,2,"cntrlans1"];
```

Now when we load this file, we will be ready to begin the strategy session.

20.3 Solving $(HGRAIL)$ using NCProcess

20.3.1 Step 1

```
In[1]:= Get["NCGB.m"];
In[2]:= Get["cntrl"];
In[3]:= result1
```

We can ignore the Mathematica output **Out[3]** of the `NCProcess1` command for now. What is important is that the spreadsheet which `NCProcess1` produces is in the file **"cntrlans1.dvi"**. There is no need to record all of it here, since the only work which we must do is on the undigested relations.

When the file **"cntrlans1.dvi'** is displayed, the undigested relations are:

SOME RELATIONS WHICH APPEAR BELOW
MAY BE UNDIGESTED

THE FOLLOWING VARIABLES HAVE NOT BEEN SOLVED FOR:
$\{b, c, E_{11}, E_{12}, E_{22}, E_{11}^{-1}, E_{12}^{-1}, E_{22}^{-1}, b^T, c^T, E_{12}^T, E_{12}^{-1T}\}$

1.3 The expressions with unknown variables $\{E_{11}^{-1}, E_{11}\}$
and knowns $\{\}$

$E_{11}\, E_{11}^{-1} \rightarrow 1$

$E_{11}^{-1}\, E_{11} \rightarrow 1$

1.3 The expressions with unknown variables $\{E_{12}^{-1}, E_{12}\}$

and knowns $\{\}$

$E_{12}\,E_{12}^{-1} \to 1$

$E_{12}^{-1}\,E_{12} \to 1$

1.3 The expressions with unknown variables $\{E_{22}^{-1}, E_{22}\}$
and knowns $\{\}$

$E_{22}\,E_{22}^{-1} \to 1$

$E_{22}^{-1}\,E_{22} \to 1$

1.5 The expressions with unknown variables $\{E_{12}^{-1T}, E_{12}^{T}\}$
and knowns $\{\}$

$E_{12}^{T}\,E_{12}^{-1T} \to 1$

$E_{12}^{-1T}\,E_{12}^{T} \to 1$

3.8 The expressions with unknown variables $\{b^T, b, E_{12}^{-1T}, E_{12}^{-1}, E_{11}\}$
and knowns $\{A, B_1, C_1, C_2, A^T, B_1^T, C_1^T, C_2^T\}$

$b\,b^T + b\,C_2\,E_{12}^{-1T} + E_{12}^{-1}\,C_2^T\,b^T + E_{12}^{-1}\,E_{11}\,A\,E_{12}^{-1T} + E_{12}^{-1}\,E_{11}\,B_1\,b^T + E_{12}^{-1}\,A^T\,E_{11}\,E_{12}^{-1T} + E_{12}^{-1}\,C_1^T\,C_1\,E_{12}^{-1T} + (b + E_{12}^{-1}\,E_{11}\,B_1)\,B_1^T\,E_{11}E_{12}^{-1T} \;=\; 0$

4.9 The expressions with unknown variables $\{c^T, c, E_{12}^{-1T}, E_{12}^{-1}, E_{11}, E_{22}, E_{12}^T, E_{12}\}$
and knowns $\{A, B_1, B_2, C_1, A^T, B_1^T, B_2^T, C_1^T\}$

$c^T\,c \,+\, c^T\,B_2^T\,(E_{12} \,-\, E_{11}\,E_{12}^{-1T}\,E_{22}) \,-\, (E_{22}\,E_{12}^{-1}\,E_{11} \,-\, E_{12}^T)\,B_2 c \,-\, E_{22}\,E_{12}^{-1}\,A^T\,(E_{12} \,-\, E_{11}\,E_{12}^{-1T}\,E_{22}) \,-\, E_{22}\,E_{12}^{-1}\,C_1^T\,(c \,-\, C_1\,E_{12}^{-1T}\,E_{22}) \,-\, c^T\,C_1\,E_{12}^{-1T}\,E_{22} \,+\, (E_{22}\,E_{12}^{-1}\,E_{11} \,-\, E_{12}^T)\,AE_{12}^{-1T}\,E_{22} \,-\, (E_{22}\,E_{12}^{-1}\,E_{11} - E_{12}^T)\,B_1B_1^T\,(E_{12} - E_{11}\,E_{12}^{-1T}\,E_{22}) \;=\; 0$

20.3.2 Step 2: The user attacks

As we can easily see from the spreadsheet above, there are only two nontrivial relations left undigested by the `NCProcess1` command. The user can ignore the rest of the spreadsheet for now. Since the leading terms of the last two polynomials above are $b\,b^T$ and $c\,c^T$, and the fact that the two equations are decoupled (i.e. the $b\,b^T$ equation does not depend on c or C^T and the $c\,c^T$ equation does not depend on b or b^T) further iterations of an NCProcess command would probably not help. At this point, we need to be more clever.

We begin by assigning variables to the polynomials that we are interested in. We can see from **Out[3]** that the polynomial involving b and $Tp[b]$ is the first element of the third list in the **result1**, and the c rule is the last element of that list. These relations are in the form of rules which need to be converted to polynomials before we can continue. This is done with the command `RuleToPoly`. The next step is to convert the **Tp**[ ] in these rules to **tp**[ ], which will be recognized as transpose by NCAlgebra. Here is how this is done:

```
In[4]:= bpoly = RuleToPoly[result1[[3,1]]]

Out[4]:= b ** Tp[b] + b ** C2 ** Inv[E21] + Inv[E12] ** Tp[C2] ** Tp[b] +
>    b ** Tp[B1] ** E11 ** Inv[E21] + Inv[E12] ** E11 ** A ** Inv[E21] +
>    Inv[E12] ** E11 ** B1 ** Tp[b] + Inv[E12] ** Tp[A] ** E11 ** Inv[E21] +
>    Inv[E12] ** Tp[C1] ** C1 ** Inv[E21] + Inv[E12] ** E11 ** B1 **
```

```
>    Tp[B1] ** E11 ** Inv[E21]

In[5]:= cpoly = RuleToPoly[result1[[3,-1]]]

Out[5]:= Tp[c] ** c + E21 ** B2 ** c + Tp[c] ** Tp[B2] ** E12 -
>    E21 ** A ** Inv[E21] ** E22 + E21 ** B1 ** Tp[B1] ** E12 -
>    E22 ** Inv[E12] ** Tp[A] ** E12 - E22 ** Inv[E12] ** Tp[C1] ** c -
>    Tp[c] ** C1 ** Inv[E21] ** E22 - E22 ** Inv[E12] ** E11 ** B2 ** c -
>    Tp[c] ** Tp[B2] ** E11 ** Inv[E21] ** E22 -
>    E21 ** B1 ** Tp[B1] ** E11 ** Inv[E21] ** E22 +
>    E22 ** Inv[E12] ** E11 ** A ** Inv[E21] ** E22 -
>    E22 ** Inv[E12] ** E11 ** B1 ** Tp[B1] ** E12 +
>    E22 ** Inv[E12] ** Tp[A] ** E11 ** Inv[E21] ** E22 +
>    E22 ** Inv[E12] ** Tp[C1] ** C1 ** Inv[E21] ** E22 +
>    E22 ** Inv[E12] ** E11 ** B1 ** Tp[B1] ** E11 ** Inv[E21] ** E22

In[6]:= bpoly = bpoly /. Tp->tp;
In[7]:= cpoly = cpoly /. Tp->tp;
```

Observe that the polynomials Out[4] is quadratic in b. We could complete the square and get to put the polynomial in the form

$$(b+\mu)(b^T+\mu^T)+\nu$$

where μ and ν are expressions involving C_2, C_2^T, B_1, B_1^T, A, A^T, E_{21}^{-1}, E_{12}^{-1} and E_{11}. Since there are many unknowns in the problem, there is probably excess freedom. Let us investigate what happens when we take $b+\mu=0$. NCAlgebra is very good with quadratics so this is easy to execute, but since this is not a general NCAlgebra tutorial we shall not describe how this is done, but just write down the answer.

```
Out[8]:=      -iE12 ** tp[C2]   tp[iE21] ** tp[C2]
          b -> --------------- - ------------------ -
                      2                  2

    iE12 ** E11 ** B1   tp[iE21] ** tp[E11] ** B1
>   ----------------- - -------------------------
           2                        2
```

We can also complete the square for the expression in c and put that expression in the form

$$(c+\lambda)(c^T+\lambda^T)+\gamma\,.$$

We also assume that $c+\lambda=0$.

```
Out[9]:=      -tp[B2] ** E12   tp[B2] ** tp[E21]   C1 ** iE21 ** E22
         c -> -------------- - ----------------- + ----------------- +
                    2                  2                   2

      C1 ** tp[iE12] ** tp[E22]   tp[B2] ** E11 ** iE21 ** E22
>     ------------------------- + ---------------------------- +
                  2                            2

      tp[B2] ** tp[E11] ** tp[iE12] ** tp[E22]
>     ----------------------------------------
                         2
```

Next we do some petty bookkeeping to get transposes of the above two rules. Once we have solved for b and c, we can then take the transpose of each of these rules to solve for $tp[b]$ and $tp[c]$. In our strategy session, these are simply two additional unknowns which we can now eliminate.

```
In[10]:= bpoly = RuleToPoly[brule];
In[11]:= cpoly = RuleToPoly[crule];
In[12]:= newpolys = {bpoly,cpoly,tp[bpoly],tp[cpoly]};
In[13]:= newrules = PolyToRule[newpolys];
In[14]:= newrules = newrules /. tp->Tp;
In[15]:= newrules = newrules /. PolyToRule[transE]
```

```
Out[15]:= {b -> -iE12 ** Tp[C2] + -iE12 ** E11 ** B1,
   Tp[b] -> -C2 ** iE21 + -Tp[B1] ** E11 ** iE21,
   c -> -Tp[B2] ** E12 + C1 ** iE21 ** E22 + Tp[B2] ** E11 ** iE21 ** E22,
   Tp[c] -> -E21 ** B2 + E22 ** iE12 ** Tp[C1] + E22 ** iE12 ** E11 ** B2}
```

`In[14]` takes these four rules and replaces **tp** with **Tp**. `In[15]` simplifies these equations by making the substitutions for the transposes of E which we have been using. Now we have four additional polynomials which can be added to the input for the next call to `NCProcess1`.

20.3.3 Step 3

The starting relations for this step will be the output from the first `NCProcess1` call which was **result1**, as well as the four new equations that we have just derived. Just as we did in the first step, we will create a file to be read in to the Mma session.

This is the file **"cntrl2"**.

```
digested=RuleToPoly[result1[[2]]];
undigested=RuleToPoly[result1[[3]]];
relations=Join[digested,newpolys,undigested];
result2=NCProcess1[relations,2,"cntrlans2",
                   DegreeCap->6,DegreeSumCap->9];
```

Now, if we do not like the results, we can change the DegreeCap options or the iteration count and simply read the file again, without typing the entire sequence of commands again. Then in the Mathematica session, we simply type

```
In[16]:= Get["cntrl2"];
```

Once again we go directly to the file which `NCProcess1` created. There is no need to record all of it, since at this stage we shall be concerned only with the undigested relations.

When the file **"cntrlans2.dvi"** is displayed, the undigested relations are:

SOME RELATIONS WHICH APPEAR BELOW
MAY BE UNDIGESTED

THE FOLLOWING VARIABLES HAVE NOT BEEN SOLVED FOR:
$\{E_{11}, E_{12}, E_{22}, E_{11}^{-1}, E_{12}^{-1}, E_{22}^{-1}, E_{12}^{T}, E_{12}^{-1T}\}$

1.0 The expressions with unknown variables $\{E_{11}\}$
and knowns $\{A, B_1, C_1, C_2, A^T, B_1^T, C_1^T, C_2^T\}$

$E_{11}\, B_1\, C_2 \to E_{11}\, A + A^T\, E_{11} + C_1^T\, C_1 - C_2^T\, C_2 - C_2^T\, B_1^T\, E_{11}$

1.3 The expressions with unknown variables $\{E_{11}^{-1}, E_{11}\}$
and knowns $\{\}$

$E_{11}\, E_{11}^{-1} \to 1$

$E_{11}^{-1}\, E_{11} \to 1$

1.3 The expressions with unknown variables $\{E_{12}^{-1}, E_{12}\}$
and knowns $\{\}$

$E_{12}\, E_{12}^{-1} \to 1$

$E_{12}^{-1}\, E_{12} \to 1$

1.3 The expressions with unknown variables $\{E_{22}^{-1}, E_{22}\}$
and knowns $\{\}$

$E_{22}\, E_{22}^{-1} \to 1$

$E_{22}^{-1}\, E_{22} \to 1$

1.5 The expressions with unknown variables $\{E_{12}^{-1T}, E_{12}^T\}$
and knowns $\{\}$

$E_{12}^T\, E_{12}^{-1T} \to 1$

$E_{12}^{-1T}\, E_{12}^T \to 1$

3.8 The expressions with unknown variables $\{E_{12}^{-1T}, E_{12}^{-1}, E_{11}, E_{22}, E_{12}^T, E_{12}\}$
and knowns $\{A, B_1, B_2, C_1, A^T, B_1^T, B_2^T, C_1^T\}$

$E_{22}\, E_{12}^{-1}\, A^T\, (E_{12} - E_{11}\, E_{12}^{-1T}\, E_{22}) - (E_{22}\, E_{12}^{-1}\, E_{11} - E_{12}^T)\, A E_{12}^{-1T}\, E_{22} + (E_{22}\, E_{12}^{-1}\, E_{11} - E_{12}^T)\, B_1 B_1^T\, (E_{12} - E_{11}\, E_{12}^{-1T}\, E_{22}) - (E_{22}\, E_{12}^{-1}\, E_{11} - E_{12}^T)\, B_2 B_2^T\, (E_{12} - E_{11}\, E_{12}^{-1T}\, E_{22}) - E_{22}\, E_{12}^{-1}\, C_1^T\, B_2^T\, (E_{12} - E_{11}\, E_{12}^{-1T}\, E_{22}) + (E_{22}\, E_{12}^{-1}\, E_{11} - E_{12}^T)\, B_2 C_1\, E_{12}^{-1T}\, E_{22} = 0$

Once again, we have two equations worth looking at.

The first polynomial equation is a (Riccati-Lyapunov) equation in E_{11}. Numerical methods for solving Riccati equations are common. For this reason assuming that a Riccati equation has a solution is a socially acceptable necessary condition throughout control engineering. Thus we can consider E_{11} to be known.

We notice that the same products of unknowns appear over and over. It is likely that we can factor or group this equation in such a way that we can understand it a little better.

20.3.4 Step 4

We start by grabbing the relation which we want to explore. Although the spreadsheet above shows the equation in factored form, it is returned to Mathematica in expanded form. In this case, the relation we are interested in is the thirteenth element in the third list of **result2**.

```
In[18] := equ2 = result2[[3,13]]

Out[18]= E22 ** Inv[E12] ** Tp[A] ** E12 +
>    Tp[E12] ** A ** Tp[Inv[E12]] ** E22 - Tp[E12] ** B1 ** Tp[B1] ** E12 +
>    Tp[E12] ** B2 ** Tp[B2] ** E12 -
>    E22 ** Inv[E12] ** Tp[C1] ** Tp[B2] ** E12 -
>    Tp[E12] ** B2 ** C1 ** Tp[Inv[E12]] ** E22 -
>    E22 ** Inv[E12] ** E11 ** A ** Tp[Inv[E12]] ** E22 +
>    E22 ** Inv[E12] ** E11 ** B1 ** Tp[B1] ** E12 -
>    E22 ** Inv[E12] ** E11 ** B2 ** Tp[B2] ** E12 -
>    E22 ** Inv[E12] ** Tp[A] ** E11 ** Tp[Inv[E12]] ** E22 +
>    Tp[E12] ** B1 ** Tp[B1] ** E11 ** Tp[Inv[E12]] ** E22 -
>    Tp[E12] ** B2 ** Tp[B2] ** E11 ** Tp[Inv[E12]] ** E22 +
>    E22 ** Inv[E12] ** E11 ** B2 ** C1 ** Tp[Inv[E12]] ** E22 +
>    E22 ** Inv[E12] ** Tp[C1] ** Tp[B2] ** E11 ** Tp[Inv[E12]] ** E22 -
>    E22 ** Inv[E12] ** E11 ** B1 ** Tp[B1] ** E11 ** Tp[Inv[E12]] ** E22 +
>    E22 ** Inv[E12] ** E11 ** B2 ** Tp[B2] ** E11 ** Tp[Inv[E12]] ** E22
```

Now we can see from the factored form in the spreadsheet that this equation is symmetric. It would not take an experienced mathematician long to realize that by multiplying this equation on the left by $E_{12}\,E_{22}^{-1}$ and on the right by $E_{22}^{-1}\,E_{21}$, we will have an equation in one unknown.

```
In[19]:= equ3 = NCExpand[E12**Inv[E22]**equ2**Inv[E22]**E21];
```

Inspection of **equ3** shows that the following valid substitution would be helpful.

```
In[20]:= equ4 = Transform[equ3,
       {Inv[E22]**E22->1,E12**Inv[E12]->1,E22**Inv[E22]->1,Inv[E21]**E21->1}];
```

We now obtain the collected form of **equ4**.

```
In[21]:= equ5 = NCCollectOnVariables[equ4]

Out[21]:= -(E11 - E12 ** Inv[E22] ** E21) ** A -
>   Tp[A] ** (E11 - E12 ** Inv[E22] ** E21) +
>   (E11 - E12 ** Inv[E22] ** E21) ** B2 ** C1 +
>   Tp[C1] ** Tp[B2] ** (E11 - E12 ** Inv[E22] ** E21) -
>   (E11 - E12 ** Inv[E22] ** E21) ** B1 ** Tp[B1] ** (E11 - E12 ** Inv[EE22] ** E2
>   (E11 - E12 ** Inv[E22] ** E21) ** B2 ** Tp[B2] ** (E11 - E12 ** Inv[E22] ** E21
```

Now we can replace $E_{11} - E_{12}\,E_{22}^{-1}\,E_{21}$ with a new variable X.

```
In[22]:= Transform[equ5,(E11 - E12 ** Inv[E22] ** E21)->X]

Out[22]:= -X ** A -  Tp[A] ** X +  X ** B2 ** C1 + Tp[C1] ** Tp[B2] ** X -
>   X ** B1 ** Tp[B1] ** X + X ** B2 ** Tp[B2] ** X
```

Observe that this is an equation in the one unknown X. Of course, the only other undigested equation was in the one unknown E_{11} and the previous spreadsheet featured an equation in the single unknown b (and its transpose) and an equation in the single unknown c (and its transpose). Thus we have solved (HGRAIL) with a symmetrized liberalized 2-strategy (see [**HS**]).

20.4 End Game

Now let us compare what we have found to the well known solution of $(HGRAIL)$. In that theory there are two Riccati equations due to Doyle, Glover, Khargonekar and Francis. These are the DGKF X and Y equations. One can read off that the E_{11} equation which we found is the DGKF equation for Y^{-1}, while the Riccati equation which we just analyzed is the DGKF X equation.

Indeed what we have proved is that if $(HGRAIL)$ has a solution with E_{ij} invertible and if b and c are given by formulas **Out[8]** *and* **Out[9]** *in §20.3.2, then*

(1) the DGKF X and Y^{-1} equations must have a solution

(2) X and Y are self-adjoint

(3) $Y^{-1} - X$ is invertible

Now we turn to the converse. The straightforward converse of the above italicized statement would be: If items (1), (2) and (3) above hold, then $(HGRAIL)$ has a solution with E_{ij} invertible and b and c are given by formulas **Out[8]** and **Out[9]** in §20.3.2. There is no reason to believe (and it is not the case) that b and c *must* be given by the formulas **Out[8]** and **Out[9]** in §20.3.2. These two formulas came about in §20.3.2 and were motivated by "excess freedom" in the problem. The converse which we will attempt to prove is:

Proposed Converse 20.1 *If items (1), (2) and (3) above hold, then $(HGRAIL)$ has a solution with E_{ij} invertible.*

To obtain this proposed converse, we need a complete spreadsheet corresponding to the last stages of our analysis. The complete spreadsheet is:

THE ORDER IS NOW THE FOLLOWING:
$A < A^T < B_1 < B_1^T < B_2 < B_2^T < C_1 < C_1^T < C_2 < C_2^T < X < X^{-1} < Y < Y^{-1} \ll E_{12} \ll E_{21} \ll E_{22} \ll E_{12}^T \ll E_{21}^T \ll E_{22}^T \ll E_{11} \ll E_{11}^T \ll E_{11}^{-1} \ll E_{11}^{-1T} \ll E_{12}^{-1} \ll E_{21}^{-1} \ll E_{22}^{-1} \ll E_{12}^{-1T} \ll E_{21}^{-1T} \ll E_{22}^{-1T} \ll b \ll b^T \ll c \ll c^T \ll a \ll a^T$

YOUR SESSION HAS DIGESTED
THE FOLLOWING RELATIONS

THE FOLLOWING VARIABLES HAVE BEEN SOLVED FOR:
$\{a, b, c, E_{11}, E_{11}^{-1}, a^T, b^T, c^T, E_{11}^T, E_{12}^T, E_{21}^T, E_{22}^T, E_{11}^{-1T}, E_{12}^{-1T}, E_{21}^{-1T}, E_{22}^{-1T}\}$
The corresponding rules are the following:

$a \rightarrow -E_{12}^{-1}\, A^T\, E_{12} + E_{12}^{-1}\, C_1^T\, B_2^T\, E_{12} + E_{12}^{-1}\, C_2^T\, B_1^T\, E_{12} + E_{12}^{-1}\, E_{11}\, B_2\, B_2^T\, E_{12} - E_{12}^{-1}\, C_1^T\, C_1\, E_{21}^{-1}\, E_{22} - E_{12}^{-1}\, E_{11}\, B_2\, C_1\, E_{21}^{-1}\, E_{22} - E_{12}^{-1}\, C_1^T\, B_2^T\, E_{11}\, E_{21}^{-1}\, E_{22} - E_{12}^{-1}\, E_{11}\, B_2\, B_2^T\, E_{11}\, E_{21}^{-1}\, E_{22}$

$b \rightarrow -E_{12}^{-1}\, C_2^T - E_{12}^{-1}\, E_{11}\, B_1$

$c \rightarrow -B_2^T\, E_{12} + C_1\, E_{21}^{-1}\, E_{22} + B_2^T\, E_{11}\, E_{21}^{-1}\, E_{22}$

$E_{11} \rightarrow Y^{-1}$

$E_{11}^{-1} \rightarrow Y$

$a^T \rightarrow -E_{21}\,A\,E_{21}^{-1} + E_{21}\,B_1\,C_2\,E_{21}^{-1} + E_{21}\,B_2\,C_1\,E_{21}^{-1} + E_{21}\,B_2\,B_2^T\,E_{11}\,E_{21}^{-1} - E_{22}\,E_{12}^{-1}\,C_1^T\,C_1\,E_{21}^{-1} - E_{22}\,E_{12}^{-1}\,E_{11}\,B_2\,C_1\,E_{21}^{-1} - E_{22}\,E_{12}^{-1}\,C_1^T\,B_2^T\,E_{11}\,E_{21}^{-1} - E_{22}\,E_{12}^{-1}\,E_{11}\,B_2\,B_2^T\,E_{11}\,E_{21}^{-1}$

$b^T \rightarrow -C_2\,E_{21}^{-1} - B_1^T\,E_{11}\,E_{21}^{-1}$

$c^T \rightarrow -E_{21}\,B_2 + E_{22}\,E_{12}^{-1}\,C_1^T + E_{22}\,E_{12}^{-1}\,E_{11}\,B_2$

$E_{11}^T \rightarrow E_{11}$ $E_{12}^T \rightarrow E_{21}$ $E_{21}^T \rightarrow E_{12}$ $E_{22}^T \rightarrow E_{22}$

$E_{11}^{-1T} \rightarrow E_{11}^{-1}$ $E_{12}^{-1T} \rightarrow E_{21}^{-1}$ $E_{21}^{-1T} \rightarrow E_{12}^{-1}$ $E_{22}^{-1T} \rightarrow E_{22}^{-1}$

The expressions with unknown variables $\{\}$
and knowns $\{A, B_1, B_2, C_1, C_2, X, Y, X^{-1}, Y^{-1}, A^T, B_1^T, B_2^T, C_1^T, C_2^T\}$

$X\,X^{-1} \rightarrow 1$

$Y\,Y^{-1} \rightarrow 1$

$X^{-1}\,X \rightarrow 1$

$Y^{-1}\,Y \rightarrow 1$

$Y^{-1}\,B_1\,C_2 \rightarrow Y^{-1}\,A + A^T\,Y^{-1} + C_1^T\,C_1 - C_2^T\,C_2 - C_2^T\,B_1^T\,Y^{-1}$

$X\,B_2\,B_2^T\,X \rightarrow X\,A + A^T\,X - X\,B_2\,C_1 - C_1^T\,B_2^T\,X + X\,B_1\,B_1^T\,X$

USER CREATIONS APPEAR BELOW

$E_{11}^{-1} \rightarrow Y$

$E_{12}\,E_{22}^{-1}\,E_{21} \rightarrow E_{11} - X$

SOME RELATIONS WHICH APPEAR BELOW
MAY BE UNDIGESTED

THE FOLLOWING VARIABLES HAVE NOT BEEN SOLVED FOR:
$\{E_{12}, E_{21}, E_{22}, E_{12}^{-1}, E_{21}^{-1}, E_{22}^{-1}\}$

1.3 The expressions with unknown variables $\{E_{12}^{-1}, E_{12}\}$
and knowns $\{\}$

$E_{12}\,E_{12}^{-1} \rightarrow 1$

$E_{12}^{-1}\,E_{12} \rightarrow 1$

1.3 The expressions with unknown variables $\{E_{21}^{-1}, E_{21}\}$
and knowns $\{\}$

$E_{21}\,E_{21}^{-1} \rightarrow 1$

$E_{21}^{-1}\,E_{21} \rightarrow 1$

1.3 The expressions with unknown variables $\{E_{22}^{-1}, E_{22}\}$
and knowns $\{\}$

$E_{22}\,E_{22}^{-1} \rightarrow 1$

$E_{22}^{-1}\,E_{22} \rightarrow 1$

4.3 The expressions with unknown variables $\{E_{22}^{-1}, E_{11}, E_{21}, E_{12}\}$
and knowns $\{X\}$

$\Uparrow\ E_{12}\,E_{22}^{-1}\,E_{21} \rightarrow E_{11} - X$

In the spreadsheet, we use conventional X, Y^{-1} notation rather than "discovered" notation so that our arguments will be familiar to experts in the field of control theory.

Now we use the above spreadsheet to verify the proposed converse. To do this, we assume that matrices A, B_1, B_2, C_1, C_2, X and Y exist, that X and Y are invertible, that X and Y are self-adjoint, that $Y^{-1} - X$ is invertible and that the DGKF X and Y^{-1} equations hold. That is, the two following polynomial equations hold.

$Y^{-1}\, B_1\, C_2 = Y^{-1}\, A + A^T\, Y^{-1} + C_1^T\, C_1 - C_2^T\, C_2 - C_2^T\, B_1^T\, Y^{-1}$

$X\, B_2\, B_2^T\, X = X\, A + A^T\, X - X\, B_2\, C_1 - C_1^T\, B_2^T\, X + X\, B_1\, B_1^T\, X$

We wish to assign values for the matrices E_{12}, E_{21}, E_{22}, E_{11}, a, b and c such that each of the equations on the above spreadsheet hold. If we can do this, then each of the equations from the starting polynomial equations from §20.1 will hold and the proposed converse will follow.

(1) Note that all of the equations in the {}-Category of the above spreadsheet hold since X and Y solve the $DGKF$ equations and are both invertible.

(2) Set E_{11} equal to the inverse of Y. This assignment is dictated by the user selects. Note that $E_{11} = E_{11}^T$ follows since Y is self-adjoint.

(3) Let E_{12} and E_{21} be any invertible matrices such that $E_{12}^T = E_{21}$. For example, one could choose E_{12} and E_{21} to both be the identity matrix.

(4) Note that there is there is a user select $E_{12}\, E_{22}^{-1}\, E_{21} = E_{11} - X$ and that E_{12}, E_{21} are invertible. Since $Y^{-1} - X$ is invertible and $E_{11} = Y^{-1}$, $E_{11} - X$ are invertible. Therefore, we set $E_{22} = E_{12}^{-1}(E_{11} - X)^{-1}E_{21}^{-1}$. Since $E_{12}^T = E_{21}$, $E_{11}^T = E_{11}$ and $X^T = X$, it follows that E_{22} is invertible and self-adjoint.

(5) Since E_{ij} has been set for $i, j = 1, 2$, we can set a, b and c according to their formulas at the top of the spreadsheet .

With the assignments of E_{12}, E_{21}, E_{22}, E_{11}, a, b and c as above, it is easy to verify by inspection that every polynomial equation on the spreadsheet above holds.

We have proven the proposed converse and, therefore, have proven the following approximation to the classical [**DGKF**] theorem.

Theorem 20.2 *If (HGRAIL) has a solution with invertible E_{ij} and b and c are given by the formulas* **Out[8]** *and* **Out[9]** *in §20.3.2, then the DGKF X and Y^{-1} equations have solutions X and Y which are symmetric matrices with X,Y^{-1} and $Y^{-1} - X$ invertible. The DGKF X and Y^{-1} equations have solutions X and Y which are symmetric matrices with X,Y^{-1} and $Y^{-1} - X$ invertible, then (HGRAIL) has a solution with invertible E_{ij}.*

Note that we obtained this result with an equation in the one unknown X and an equation with the one unknown $E_{11} = Y^{-1}$. From the strategy point of view, the first spreadsheet featured an equation in the single unknown b (and its transpose) and an equation in the single unknown c (and its transpose) and so is the most complicated. For example, the polynomial **Out[4]** in §20.3.2 decomposes as

$$p = q_1^T q_1 + q_2 \tag{20.3}$$

where $q_1 = b + E_{12}^{-1}\, C_2^T + E_{12}^{-1}\, E_{11}\, B_1$ and q_2 is a symmetric polynomial which does not involve b. This forces us to say that the proof of the necessary side of Theorem 20.2 was done with a 2-strategy.

A more aggressive way of selecting knowns and unknowns allows us to obtain this same result with a symmetrized 1-strategy. In particular, one would set a, b and c to be the only unknowns to obtain a first spreadsheet. The first spreadsheet contains key equations like (20.3), which is a symmetric 1-decomposition, because q_2 does not contain a, b or c. Once we have solved for a, b and c, we turn to the next spreadsheet by declaring the variables involving E_{ij} (e.g., E_{11}, E_{11}^{-1}, $\ldots$) to be unknown. At this point, the computer run is the same as Steps 2, 3 and 4 above.

Part VI

NCGB: LISTS OF COMMANDS AND OTHER DETAILS

Chapter 21

Ordering on variables and monomials

As was mentioned above (Section 13.4.1), one needs to declare a monomial order before making a Gröbner Basis. There are various monomial orders which can be used when computing Gröbner Basis. The most common are called lexicographic and graded lexicographic orders. In the previous section, we used only graded lexicographic orders. See Section 21.1 for a discussion of lexicographic orders.

We will be considering lexicographic, graded lexicographic and multi-graded lexicographic orders. Lexicographic and multi-graded lexicographic orders are examples of elimination orderings. An elimination ordering is an ordering which is used for solving for some of the variables in terms of others.

We now discuss each of these types of orders.

21.1 Lex Order: The simplest elimination order

To impose lexicographic order $a << b << x << y$ on a, b, x and y, one types

```
In[14]:=SetMonomialOrder[a,b,x,y];
```

This order is useful for attempting to solve for y in terms of a, b and x, since the highest priority of the GB algorithm is to produce polynomials which do not contain y. If producing high order polynomials is a consequence of this fanaticism so be it. Unlike graded orders, lex orders pay little attention to the degree of terms. Likewise its second highest priority is to eliminate x.

Once this order is set, one can use all of the commands in the preceeding section in exactly the same form.

We now give a simple example how one can solve for y given that a,b,x and y satisfy the equations:

$$-b\,x + x\,y\,a + x\,b\,a\,a = 0$$

$$x\,a - 1 = 0$$

$$a\,x - 1 = 0\,.$$

```
In[15]:= NCMakeGB[{-b ** x + x ** y ** a + x ** b ** a ** a,
x**a-1,a**x-1},4];
Out[15]= {-1 + a ** x, -1 + x ** a, y + b ** a - a ** b ** x ** x}
```

If the polynomials above are converted to replacement rules, then a simple glance at the results allows one to see that y has been solved for.

```
In[16]:= PolyToRule[%]
Out[16]= {a ** x -> 1, x ** a -> 1, y -> -b ** a + a ** b ** x ** x}
```

Now, we change the order to

```
In[20]:=SetMonomialOrder[y,x,b,a];
```

and do the same *NCMakeGB* as above:

```
In[21]:= NCMakeGB[{-b ** x + x ** y ** a + x ** b ** a ** a,
x**a-1,a**x-1},4];
In[22]:= PolyToRule[%];
In[23]:= ColumnForm[%];
Out[23]= a ** x -> 1
         x ** a -> 1
         x ** b ** a -> -x ** y + b ** x ** x
         b ** a ** a -> -y ** a + a ** b ** x
         b ** x ** x ** x -> x ** b + x ** y ** x
         a ** b ** x ** x -> y + b ** a
         x ** b ** b ** a ->
         >   -x ** b ** y - x ** y ** b ** x ** x + b ** x ** x ** b ** x ** x
         b ** a ** b ** a ->
         >   -y ** y - b ** a ** y - y ** b ** a + a ** b ** x ** b ** x ** x
         x ** b ** b ** b ** a ->
         >   -x ** b ** b ** y - x ** b ** y ** b ** x ** x -
         >    x ** y ** b ** x ** x ** b ** x ** x +
         >    b ** x ** x ** b ** x ** x ** b ** x ** x
         b ** a ** b ** b ** a ->
         >   -y ** b ** y - b ** a ** b ** y - y ** b ** b ** a -
         >    y ** y ** b ** x ** x - b ** a ** y ** b ** x ** x +
         >    a ** b ** x ** b ** x ** x ** b ** x ** x
```

In this case, it turns out that it produced the rule $a * *b * *x * *x \rightarrow y + b * *a$ which shows that the order is not set up to solve for y in terms of the other variables in the sense that y is not on the left hand side of this rule (but a human could easily solve for y using this rule). Also the algorithm created a number of other relations which involved y. If one uses the lex order $a << b << y << x$, the *NCMakeGB* call above generates 12 polynomials of high total degree which do not solve for y.

See [**CoxLittleOShea**].

21.2 Graded lex ordering: A non-elimination order

This is the ordering which was used in all demos appearing before this section. It puts high degree monomials high in the order. Thus it tries to decrease the total degree of expressions.

21.3 Multigraded lex ordering : A variety of elimination orders

There are other useful monomial orders which one can use other than graded lex and lex. Another type of order is what we call multigraded lex and is a mixture of graded lex and lex order. This multigraded order is set using *SetMonomialOrder*, *SetKnowns* and *SetUnknowns* which are described in Section 21.4. As an example, suppose that we execute the following commands:

```
SetMonomialOrder[{A,B,C},{a,b,c},{d,e,f}];
```

We use the notation

$$A < B < C << a < b < c << d < e < f\,,$$

to denote this order.

For an intuitive idea of why multigraded lex is helpful, we think of A, B and C as corresponding to variables in some engineering problem which represent quantities which are known and a, b, c, d, e and f to be unknown[1]. The fact that d, e and f are in the top level indicates that we are very interested in solving for d, e and f in terms of A, B, C, a, b and c, but are not willing to solve for b in terms of expressions involving either d, e or f.

For example,

(1) $d > a**a**A**b$

(2) $d**a**A**b > a$

(3) $e**d > d**e$

(4) $b**a > a**b$

(5) $a**b**b > b**a$

(6) $a > A**B**A**B**A**B$

This order induces an order on monomials in the following way. One does the following steps in determining whether a monomial m is greater in the order than a monomial n or not.

(1) First, compute the total degree of m with respect to only the variables d, e and f.

(2) Second, compute the total degree of n with respect to only the variables d, e and f.

(3) If the number from item (2) is smaller than the number from item (1), then m is smaller than n. If the number from item (2) is bigger than the number from item (1), then m is bigger than n. If the numbers from items (1) and (2) are equal, then proceed to the next item.

---

[1] If one wants to speak *very* loosely, then we would say that a, b and c are unknown and d, e and f are "very unknown."

(4) First, compute the total degree of m with respect to only the variables a, b and c.

(5) Second, compute the total degree of n with respect to only the variables a, b and c.

(6) If the number from item (5) is smaller than the number from item (4), then m is smaller than n. If the number from item (5) is bigger than the number from item (4), then m is bigger than n. If the numbers from items (4) and (5) are equal, then proceed to the next item.

(7) First, compute the total degree of m with respect to only the variables A, B and C.

(8) Second, compute the total degree of n with respect to only the variables A, B and C.

(9) If the number from item (8) is smaller than the number from item (7), then m is smaller than n. If the number from item (8) is bigger than the number from item (7), then m is bigger than n. If the numbers from items (7) and (8) are equal, then proceed to the next item.

(10) At this point, say that m is smaller than n if and only if m is smaller than n with respect to the graded lex order $A < B < C < a < b < c < d < e < f$

For more information on multigraded lex orders, consult [**HSStrat**].

21.4 The list of commands

21.4.1 SetMonomialOrder[aListOfListsOfIndeterminates, . . .]

Aliases: **None**

Description: *SetMonomialOrder[a,b,c,. . .]* sets the graded lex order $a < b < c < \ldots$ with $a < b < c < \cdots$. If one uses a list of variables rather than a single variable as one of the arguments, then multigraded lex order is used. It is synonomous with *SetMonomialOrder[{a,b,c,. . . }]*. Pure lex order $a << b << c << \ldots$ on these variables is set by *SetMonomialOrder[{ {a}, {b}, {c },. . . }]*.

Arguments: A multigraded lex order $a < b << c < \ldots$ on these variables is set by *SetMonomialOrder[{ {a, b }, {c },. . . }]*. *aListOfListsOfIndeterminates* is a list of Mathematica variable or a list of Mathematica variables.

Comments / Limitations: Not available before NCAlgebra 1.2.

Equivalent to *SetMonomialOrder[{a,b }, {c , A }]* is *SetMonomialOrder[{{a,b }, {c , A } }]*. Or alternatively this is equivalent the following two commands

```
SetKnowns[a,b]
SetUnKnowns[c, A]
```

which we now describe.

21.4.2 SetUnknowns[aListOfIndeterminates]

Aliases: **None**

Description: *SetUnknowns[aListOfVariables]* records the variables in the list of variables *aListOfIndeterminates* to be corresponding to unknown quantities. This and SetUnknowns prescribe a monomial order with the knowns at the the bottom and the unknowns at the top.

Arguments: *aListOfIndeterminates* is a list of Mathematica variables.

Comments / Limitations: Not available before NCAlgebra 1.2. This is equivalent to *Do[SetMonomialOrder[aListOfVariables[[i]],i+1],* $\{i, 1, Length[aListOfVariables]\}$*]*

21.4.3 SetUnKnowns[aListOfVariables]

Aliases: **None**

Description: *SetUnKnowns[aListOfVariables]* records the variables in the list of variables *aListOfVariables* to be corresponding to unknown quantities. This and SetUnknowns prescribe a monomial order with the knowns at the the bottom and the unknowns at the top.

Arguments: *aListOfVariables* is a list of Mathematica variables.

Comments / Limitations: Not available before NCAlgebra 1.2. This is equivalent to *Do[SetMonomialOrder[aListOfVariables[[i]],i+1],* $\{i, 1, Length[aListOfVariables]\}$*]*

21.4.4 ClearMonomialOrder[]

Aliases: **None**

Description: After *ClearMonomialOrder[]* is called, there are no indeterminates which are considered ordered. The monomial order can be retrieved by using the *ReinstallOrder[]* command.

Arguments: None

Comments / Limitations: Not available before NCAlgebra 1.2

21.4.5 PrintMonomialOrder[]

Aliases: **None**

Description: *PrintMonomialOrder[]* prints the order to the screen.

Arguments: None

Comments / Limitations: See Chapter 21. Not available before NCAlgebra 1.2

21.4.6 NCAutomaticOrder[aMonomialOrder, aListOfPolynomials]

Aliases: **None**

Description: This command assists the user in specifying a monomial order. It inserts all of the indeterminants found in *aListOfPolynomials* into the monomial order. If x is an indeterminant found in *aMonomialOrder* then any indeterminant whose symbolic representation is a function of x will appear next to x. For example, NCAutomaticOrder[{{a},{b}},{ a**Inv[a]**tp[a] + tp[b]}] would set the order to be $a < tp[a] < Inv[a] \ll b < tp[b]$.

Arguments: A list of indeterminants which specifies the general order. A list of polynomials which will make up the input to the Gröbner basis command.

Comments / Limitations: If tp[Inv[a]] is found after Inv[a] NCAutomaticOrder[ ] would generate the order $a < tp[Inv[a]] < Inv[a]$. If the variable is self-adjoint (the input contains the relation $tp[Inv[a]] == Inv[a]$) we would have the rule, $Inv[a] \rightarrow tp[Inv[a]]$, when the user would probably prefer $tp[Inv[a]] \rightarrow Inv[a]$.

21.5 Fancier Order Setting Commands

The following commands were created for a project, long ago and we have not used them recently.

21.5.1 SetMonomialOrder[aListOfIndeterminants, n]

Aliases: **None**

Description: *SetMonomialOrder[aListOfIndeterminates,n]* sets the order of monomials (e.g., if *aListOfIndeterminates* is $\{a, b, c, d, f, e\}$, then the order is $a < b < c < d < f < e$) and assigns them grading level n. To obtain a graded lexicographic order, one can use $n = 1$.

Arguments: *aListofIndeterminants* is a list of indeterminates, n is a natural number

Comments / Limitations: See Chapter 21. Not available before NCAlgebra 1.2

21.5.2 ClearMonomialOrderN[n]

Aliases: **None**

Description: *ClearMonomialOrderN[n]* clears the order having level n. This command is equivalent to clearing $SetMonomialOrder[\{\}, n]$

Arguments: n is an integer or blank

Comments / Limitations: Not available before NCAlgebra 1.2.

21.5.3 ClearMonomialOrderAll[]

Aliases: **None**

Description: *ClearMonomialOrderAll[]* effectively executes *ClearMonomialOrderN[n]* for all positive integers n.

Arguments: None

Comments / Limitations: Not available before NCAlgebra 1.2.

21.5.4 WhatIsMultiplicityOfGrading[]

Aliases: **None**

Description: *WhatIsMultiplicityOfGrading[]* returns a positive integer which is the multiplicity of the grading. If the value is 1, then one is using graded lexicographical order.

Arguments: None

Comments / Limitations: See Chapter 21. Not available before NCAlgebra 1.2

21.5.5 WhatIsSetOfIndeterminants[n]

Aliases: **None**

Description: WhatIsSetOfIndeterminants[n] gives the n-th set of indeterminates.

Arguments: None

Comments / Limitations: See Chapter 21. Not available before NCAlgebra 1.2

Chapter 22

More NCProcess Options

This appendix summarizes NCProcess options that were not described in Chapter 19.

22.1 Creating small generating sets: RR→ *True*, RRByCat→ *True*, SB→ *False*, SBByCat→ *True*

Section 19.3.3 discusses why the NCProcess commands used algorithms for creating small sets X from a partial GB Y such that the ideal generated by X equals the ideal generated by Y. The options of the NCProcess commands are

$$RR, RRByCat, SB, SBByCat\,.$$

These actually abbreviate longer names. For example, *RR* abbreviates *RemoveRedundant*. See §22.5. Any combination of options will leave the output of NCProcess a generating set for the ideal generated by the input relations. The more (redundant) relations you try to remove, the slower *NCProcess* runs. The options indicate which algorithms will be used to remove the redundant relations. *RemoveRedundant*, for instance, runs very quickly, but does not do a very thorough job.

A complete list of options together with indications of their speed can be found at the end of the chapter. If two overlapping options are set to True, like RR and RRByCat, then only one of them is actually run. The amount of time that it takes to run *NCProcess* depends on which options are set.

The defaults for *NCProcess* are: SBByCat and RR are set to True while SmallBasis and SBFlatOrder are set to False.

SBFlatOrder→ *False*

If *SBFlatOrder* → *True* the small basis algorithm will be performed under the length lexicographic monomial order. Relations which have a leading term with only one indeterminate, also known as *singletons*, are not included in the input to the small basis algorithm since the user typically does not want them removed. The singletons do however appear in the output of NCProcess when *SBFlatOrder* → *True*. The default is *SBFlatOrder* → *False*, however *SBFlatOrder* → *True* is probably significantly faster.

DegreeCapSB$\rightarrow$ $aNumber1$ **and DegreeSumCapSB**$\rightarrow$ $aNumber2$

Beginners should skip this part of Appendix 22 and rely on the *NCProcess* defaults. The reducing algorithms *SmallBasis* and *SmallBasisByCategory* call *NCMakeGB* (§12.3). These calls to *NCMakeGB* can be very time consuming and so it can be beneficial to direct *NCMakeGB*. The *NCProcess* options *DegreeCapSB* and *DegreeSumCapSB* are used when *NCProcess* calls *SmallBasis* or *SmallBasisByCategory*. Thus DegreeCaps are valuable in limiting the time they consume. In particular,

```
NCProcess[polys, iters, "filename", DegreeCapSB -> j1,
                          DegreeSumCapSB -> j2, OtherOptions]
```

produces the call

```
NCMakeGB[polys, iters, DegreeCap -> j1, DegreeSumCap -> j2, OtherOptions]
```

inside *SmallBasis* or *SmallBasisByCategory* (§More appears on this in the long document describing the commands *SmallBasis* and *SmallBasisByCategory*.

The defaults **at the moment** inside of *NCProcess* are

$$DegreeCapSB \rightarrow -1$$

$$DegreeSumCapSB \rightarrow -1$$

22.2 NCCollectOnVars

An extremely important option for NCProcess is NCCollectOnVars. In addition to being an option, it can be called as a stand-alone command.

22.2.1 NCCollectOnVars[aListOfExpressions, aListOfVariables]

Aliases: **NCCV**

Description: This is a command which “collects” certain products of each variables in aListOfVariables, thus it returns a ”collected polynomial”. In NCProcess, when this is used via NCCV → True, the variables are typically chosen to be knowns. The command `NCCollectOnVars` itself has the option LeftsAndRights → True, which prints out a list containing two lists

$$\{\{l_1, \ldots, l_t\}, \{r_1, \ldots, r_t\}\}$$

Here l_1 is the left side of term 1 of ”collected polynomial”, and r_1 is the right side of term 1 of ”collected polynomial”, etc. An example is

$$NCCollectOnVars[X**A**B**Z + Y**A**B**Z + A**X + A**Y, \{A, B\}] = \\ (X+Y)**A**B**Z + A**(X+Y)\,.$$

with LeftsAndRights → False. When LeftsAndRights → True, the output is

$$\{\{X+Y,1\},\{1,(X+Y)\}\}$$

More detail is in 22.2.

Arguments: aListOfExpressions is a list of expressions. aListOfVariables is a list of variables.

Comments / Limitations: Not available before NCAlgebra1.2

In the example

$$NCCollectOnVars[X**A**B**Z+Y**A**B**Z+A**X+A**Y,\{A,B\}] = \\ (X+Y)**A**B**Z+A**(X+Y)\,.$$

note that if we are in an environment where A and B have been set known and X,Y and Z are unknowns, then one may omit the list of variables $\{A,B\}$ in the call above. In other words,

```
In[2]:= SetKnowns[A,B];
In[3]:= SetUnknowns[X,Y,Z];
In[4]:= NCCollectOnVars[X**A**B**Z + Y**A**B**Z + A**X+A**Y,{A,B}]

Out[4] := (X+Y)**A**B**Z + A**(X+Y)
```

Another example is if A,$Tp[A]$, $B1$, $Tp[B1]$ $B2$, $Tp[B2]$ $C1$, $Tp[C1]$ $C2$ and $Tp[C2]$ are knowns and all other variables are unknowns, then:

```
In[2] := long = E21 ** A ** iE21 ** E22 - E21 ** B1 ** Tp[B1] ** E12 +
>    E21 ** B2 ** Tp[B2] ** E12 + E22 ** iE12 ** Tp[A] ** E12 -
>    E21 ** B2 ** C1 ** iE21 ** E22 -
>    E22 ** iE12 ** Tp[C1] ** Tp[B2] ** E12 +
>    E21 ** B1 ** Tp[B1] ** E11 ** iE21 ** E22 -
>    E21 ** B2 ** Tp[B2] ** E11 ** iE21 ** E22 -
>    E22 ** iE12 ** E11 ** A ** iE21 ** E22 +
>    E22 ** iE12 ** E11 ** B1 ** Tp[B1] ** E12 -
>    E22 ** iE12 ** E11 ** B2 ** Tp[B2] ** E12 -
>    E22 ** iE12 ** Tp[A] ** E11 ** iE21 ** E22 +
>    E22 ** iE12 ** E11 ** B2 ** C1 ** iE21 ** E22 +
>    E22 ** iE12 ** Tp[C1] ** Tp[B2] ** E11 ** iE21 ** E22 -
>    E22 ** iE12 ** E11 ** B1 ** Tp[B1] ** E11 ** iE21 ** E22 +
>    E22 ** iE12 ** E11 ** B2 ** Tp[B2] ** E11 ** iE21 ** E22

In[3] := NCCollectOnVariables[long]

Out[3]:= E22 ** iE12 ** Tp[A] ** (E12 - E11 ** iE21 ** E22) +
>    (E21 - E22 ** iE12 ** E11) ** A ** iE21 ** E22 -
>    (E21 - E22 ** iE12 ** E11) ** B1 ** Tp[B1] **
>    (E12 - E11 ** iE21 ** E22) +
>    (E21 - E22 ** iE12 ** E11) ** B2 ** Tp[B2] **
>    (E12 - E11 ** iE21 ** E22) -
>    E22 ** iE12 ** Tp[C1] ** Tp[B2] ** (E12 - E11 ** iE21 ** E22) -
>    (E21 - E22 ** iE12 ** E11) ** B2 ** C1 ** iE21 ** E22
```

The NCCollectOnVars option LeftAndRight $\rightarrow True$

This prints out a list containing two lists

$$\{\{l_1, \ldots, l_t\}, \{r_1, \ldots, r_t\}\}$$

Here l_1 is the left side of term 1 of ”collected polynomial”, and r_1 is the right side of term 1 of ”collected polynomial”, etc. In the example above the list of left sides is

```
 E22 ** iE12,
>    (E21 - E22 ** iE12 ** E11),
>    -(E21 - E22 ** iE12 ** E11),
>    (E12 - E11 ** iE21 ** E22),
>    (E21 - E22 ** iE12 ** E11),
>    (E12 - E11 ** iE21 ** E22),
>    -E22 ** iE12,
>    -(E21 - E22 ** iE12 ** E11)
```

and the list of right sides is

```
{
 (E12 - E11 ** iE21 ** E22), iE21 ** E22,
-(E12 - E11 ** iE21 ** E22),  (E12 - E11 ** iE21 ** E22),
 -  (E12 - E11 ** iE21 ** E22), - iE21 ** E22 }
```

The NCProcess option NCCV $\rightarrow True$

If the $NCProcess$ option $NCCV \rightarrow True$ is set $NCProcess$ will perform the sort of collection described in Section 22.2. Specifically, NCCollectOnVars[$OutputofGroebnerBasisAlgorithms$, $KnownIndeterminates$] is called before the result is output to LaTeX. This has been shown to be useful in the discovery of “motivated unknowns” as discussed in [HS].

22.3 Turning screen output off

In the notebook environment you will definitely want to turn a lot of NCProcess screen output off. Too much output drowns the notebook. In fact we are so sure of this that suppressing diagnostic output is the default. Turning it on or off or on is done with the Options

PrintScreenOutput → True
PrintScreenOutput → False

The default is PrintScreenOutput → False

also see the command

NCGBMmaDiagnostics[**True**]
NCGBMmaDiagnostics[**False**]

which turns on (and off) more diagnostics.

22.4 Output Options

There are several options for NCProcess which determine the appearance of the output. These are actually options for the RegularOutput command which is called by NCProcess. Additional options for RegularOutput can be handed to RegularOutput with the NCProcess option AdditionalRegularOutputOptions → aList, where aList is a list of rules. The following options can be used directly as if they were NCProcess options. These three options are independent of one another. All three files can be created with one NCProcess call.

22.4.1 Turning screen output off: PrintScreenOutput$\rightarrow False$

In the notebook environment you will definitely want to turn a lot of NCProcess screen output off. Too much output drowns the notebook. Turning it off or on is done with the Option

PrintScreenOutput → False
PrintScreenOutput → True

The default is PrintScreenOutput → False

22.4.2 TeX$\rightarrow True$

This option gives spreadsheets a LaTeX appearance. The default is $TeX \rightarrow True$.

There is a bug in the program to create spreadsheets. In the list of variables that have been solved for, the letter i is sometimes replaced by a number. The TeX file is absolutely correct. The bug only appears in a list of variables, never in an equation.

22.4.3 ASCII$\rightarrow False$

This option creates the spreadsheet in ascii text form. This is a lot more difficult to read, but it can be useful for copying and pasting relations from the spreadsheet into a Mathematica session. The default is $ASCII \rightarrow False$.

22.4.4 NCGBFastRegularOutput$\rightarrow False$

If $NCGBFastRegularOutput \rightarrow True$ processing of equations done by NCProcess will be done by the C++ kernel rather than Mathematica. This is much faster, but not quite as pretty as with Mathematica. The default is $NCGBFastRegularOutput \rightarrow False$. We hope this option is functional as of August 1999.

22.4.5 NCShortFormulas$\rightarrow -1$

See Section 18.3.4.

22.5 NCProcess Summary Table

NCProcess[*aListOfPolynomials*, *iter*, “*Filename*”, *Options*]

UserSelect → {}	This list of polynomials will appear in the *UserSelects* category in the file “*Filename*”.
Deselect → {}	Polynomials in this list will not be used to do something or other.
MainUnknowns→ {}	Indeterminates in this list will cause the output of NCProcess to only contain equations containing these indeterminates or functions of them.
RR → *True*	Setting RR → *True* will cause the algorithm to use RemoveRedundant to reduce partial Gröbner bases. RemoveRedundant is very fast.
RRByCat → *False*	Specifies whether or not to use RemoveRedundantByCategory. This is slower than RemoveRedundant, but it is still fast.
SB → *False*	Specifies whether or not to use SmallBasis. This can be very slow.
SBByCat → *False*	Specifies whether or not to use SmallBasisByCategory, which is a slower but more particular form of SmallBasis. This can be slow.
SBFlatOrder → *False*	Specifies whether or not to use NCFlatSmallBasis which takes the length lexicographic monomial order for the small basis algorithm. This might be faster than the above two options.
DegreeCap → -1	Sets the DegreeCap for NCMakeGB within NCProcess. The DegreeCap for SmallBasis is set to one higher. -1 means there is no cap.
DegreeSumCap → -1	Sets the DegreeSumCap for NCMakeGB within NCProcess. The DegreeSumCap for SmallBasis is set to one higher.
NCCV→ *True*	Turns on NCCollectOnVariables, which collects on the knowns.

NCProcess options continued

NCGBNag $\rightarrow True$	Setting NCGBNag $\rightarrow$ $True$ will cause the current partial GB to be called into Mathematica, output to the screen and saved to a file after each iteration.
NCShortFormulas$\rightarrow -1$	Sets a maximum length for the expressions output by NCProcess. Longer relations will simply be eliminated. Eg. NCShortFormulas $\rightarrow$ 200. -1 means no expressions will be eliminated.
TeX$\rightarrow True$	Produces a TeX Spreadsheet, which pops up on the screen, if you are setup properly.
NCGBFastRegularOutput$\rightarrow False$	Specifies whether or not to do processing of equations by the C++ kernel rather than Mathematica. This is much faster, but not quite as pretty as with Mathematica.
NCKnownIndeterminant$\rightarrow \{\}$	An indeterminant in this list will only change the NCProcess display of knowns and unknowns. Indeterminants lower in the order than the "group" containing the specified indeterminant will be regarded as known for purposes of sorting by categories in the NCProcess display. (Not implemented as of July 19, 1999)
DisplayOptions$\rightarrow \{\}$	False turns off the beginning part of the spreadsheet which displays the input to NCProcess
NCGBDebug$\rightarrow False$	Creates lots of files that an NCProcess specialist can use to asses problems with the Mma paprts of NCProcess.
PrintScreenOutput$\rightarrow False$	Suppresses some NCProcess diagnostic output.

Chapter 23

Commands for Making and Using GB's

This section contains a glossary of commands available for generating and using non-commutative Gröbner Basis.

23.1 Simplification

Before using these commands, a monomial order must be specified. See the section after this for information on how to set the monomial order.

23.1.1 NCSimplifyAll[expressions, startRelations, iterations]

Aliases: **GroebnerSimplify, NCGroebnerSimplify**

Description: *NCSimplifyAll[expressions, startRelations, iterations]* calls *NCMakeGB[startRelations iterations]* and uses the result to reduce the expressions in *expressions*. This is useful when the form the Gröbner Basis is irrelevant and is only used to simplify some set of predetermined equations.

Arguments: *exprs* can be either an expression or a list of expressions. *startRelations* is a list of polynomials. *iterations* is a positive integer.

Comments / Limitations: Not available before NCAlgebra 1.2

23.1.2 NCSimplifyRationalX1[expressions, startRelations, iterations]

Aliases: **None**

Description: *NCSimplifyRationalX1[expressions, startRelations, iterations]* is (essentially) equivalent to
NCSimplifyAll[expressions, Join[startRelations, inverses], iterations]
where *inverses* is a list of polynomials. *inverses* consists of the defining relations of all inverses (elements with *inv*[...]) in *expressions* and *startRelations*. *NCSimplifyRationalX1* simplifies *expressions* with *startRelations* together with some additional innocuous relations.
NCSimplifyRationalX1 is different from *NCSimplifyAll* with the following adjustment: first all instances of *inv* are converted to *Inv* before calling *NCSimplifyAll*, and are converted back afterwards. This overrides NCAlgebra's automatic handling of inverses. For example, the command
$NCSimplifyRationalX1[\{inv[x]**inv[1-x]+y\},\{z**inv[y]-a\},10]$
equals
$NCSimplifyAll[\{Inv[x]**Inv[1-x]+y\},\{z**Inv[y]-a,y**Inv[y]-1,Inv[y]**y-1,Inv[x]**x-1,x**Inv[x]-1,(1-x)**Inv[1-x]-1,Inv[1-x]**(1-x)-1\},10]$.
This call to *NCSimplifyAll* returns *Inv[x] + Inv[1-x] + y*. The call to *NCSimplifyRationalX1* returns *inv[x] + inv[1-x] + y*.

Arguments: *expressions* is a list of polynomials. *startRelations* is a list of polynomials. *iterations* is a natural number.

Comments / Limitations: Not available before NCAlgebra 1.2.

23.2 Making a Gröbner Basis and various options (with their defaults)

Before using any of these commands, you must set a monomial order. See Chapter 21.

23.2.1 NCMakeGB[aListOfPolynomials, iterations]

Aliases: **NCMakeRules, MorasAlgorithm**

Description: The GB algorithm proceeds through *at most iterations* iterations until a Gröbner basis is found for the list of polynomials *aListofPolynomials* with respect to the order imposed by *SetMonomialOrder*, see §21.5.1. NCMakeGB calls a C++ program and while the C++ program is running, it prints intermediate lists of rules to the screen. When the C++ program finishes, it is either because it has run *iterations* number of iterations or it has found a Gröbner Basis. If it has found a Gröbner Basis, it prints a message saying so. One may also use the command *FinishedComputingBasisQ[]* in §23.2.5. The Mathematica output of NCMakeGB is a list of polynomials.

The following computes the Gröbner basis for the ideal generated by $x^2 - a$ and $x^3 - b$.

$$NCMakeGB[\{x^2 - a, x^3 - b\}, 20]$$

For options which can be used inside *NCMakeGB*, see *ReturnRelationsToMma*, *SupressCOutput??*, *SupressAllCOutput??*, *UserSelect* and *Deselect -DISABLED.*

Arguments: *aListofPolynomials* is a list of polynomials. *iterations* is a natural number.

Comments / Limitations: Not available before NCAlgebra 1.2

23.2.2 UserSelect→ {}(Distinguishing important relations)

Aliases: **None**

Description: UserSelect is an option for *NCMakeGB* and for *Spreadsheet.* If *UserSelect*→*aListOfPol* is given as an option, then the main effect is on the display produced by RegularOuput throughout the rest of the session. *UserSelect* is cumulative and more and more polynomials fall into the selected category as the user puts them in. The command *RegularOutput* automatically scans for selected polynomials to display in the collection of relations labelled USER SELECTS. It finds them only after *NCMakeGB* has been run with the user selects (since *RegularOutput* searches for the selected polynomials by there number. Henceforth, *aListOfPolynomials* is displayed as a special category. The default value is {}.

Arguments: None

Comments / Limitations: Not available before NCAlgebra 1.2

23.2.3 ClearUserSelect[]

Aliases: **None**

Description: Clears the UserSelected polynomials, that is, it makes the USER SELECTED empty

Arguments: None

Comments / Limitations: Not available before NCAlgebra 1.2

23.2.4 Deselect→ {} (DISABLED)

Aliases: **None**

Description: *Deselect* is an option for *NCMakeGB*. If *DeSelectPoly→aListOfPolynomials* is given as an option for *NCMakeGB*, then certain S-polynomials will not be computed. Specifically, no S-polynomial will be formed between two deselected polynomials. *Deselect* is NOT cumulative. The default value is {}.

Arguments: None

Comments / Limitations: Not available before NCAlgebra 1.2. Bug: Only the deselected polynomials which are also members of the first argument for *NCMakeGB* are deselected.

23.2.5 FinishedComputingBasisQ[] - Untested in 1999

Aliases: **None**

Description: *FinishedComputingBasisQ[]* returns 0 or 1. If it returns 1, then NCMakeRules stopped because it obtained a Gröbner Basis. If it returns 0, then *NCMakeRules* stopped because it ran the maximal number of iterations and did not find a Gröbner Basis.

Arguments: None

Comments / Limitations: Not available before NCAlgebra 1.2

.

23.2.6 WhatIsPartialGB[]

Aliases: **None**

Description: Since we do not require GB algorithm to run until it obtains a Gröbner Basis, the output of the algorithm need not be a Gröbner Basis. We call the output of the algorithm a *partial basis*. *WhatIsPartialGB[]* returns a list of all of the elements in the partial groebner basis. See *PartialBasis[n]*.

Arguments: None

Comments / Limitations: Not available before NCAlgebra 1.2. See also §33.1.3

23.2.7 NCGBSetIntegerOverflow[False]

Aliases: **None**

Description: An MakeNCGB and NCProcess command which if set True permits large integers to be handled carelessly by C++. This is dangerous since it causes mistakes when you see large integers (at about 2 billion watch out). The code runs faster with this setting, but still for saftey sake the default is *NCGBSetIntegerOverflow[False].*

Arguments: None

Comments / Limitations:

23.2.8 PartialBasis[aNumber] - Untested in 1999

Aliases: **None**

Description: *PartialBasis[aNumber]* (where *aNumber* is a positive integer) returns a character string. If this character string is input to the built-in Mathematica command *ToExpression*, then the result is the *aNumber*-th element of the partial basis data base.

Arguments: *aNumber* is a natural number.

Comments / Limitations: Not available before NCAlgebra 1.2. See also 33.1.3. **Note:** *aNumber* does **not** correspond to the numbers generated from the *WhatIsHistory* command.

23.2.9 IterationNumber[aList] or IterationNumber[aNumber] - UNTESTED 1999

Aliases: **None**

Description: It might be interesting to find out which relations are generated during which iteration of the algorithm. The command $IterationNumber$ returns these numbers. The most common applications of this command will probably be $IterationNumber[WhatAreGBNumbers[]]$ and $IterationNumber[WhatAreNumbers[]]$.

Arguments: *aList* is a list of integers. $aNumber$ is an integer

Comments / Limitations: Not available before NCAlgebra 1.2.

23.2.10 CleanUp

Aliases: **None**

Description: CleanUp → 1

Arguments: NCProcess and NCMakeGB Option which controls the reduction step inside Mora's GB algorithm. When set to 1 the new S- polynomials reduce the current set of relations. When CleanUp → 0 they do not.

Comments / Limitations:

23.2.11 SetCleanUpBasis[n] - Untested in 1999

Aliases: **None**

Description: If n is nonzero (best to use n=1), *SetCleanUpBasis [n]* indicates that the program used to implement NCCleanUpRules should be applied after each iteration of the GB algorithm. *SetCleanUpBasis[0]* indicates that such additional processing should not be done.

Arguments: n is an integer

Comments / Limitations: Not available before NCAlgebra 1.2

23.2.12 CleanUpBasisQ[] - Untested in 1999

Aliases: **None**

Description: *CleanUpBasisQ[]* returns the argument of the SetCleanUpBasis command the last time it was used.

Arguments: None

Comments / Limitations: Not available before NCAlgebra 1.2. This allows one to see which "cleaning" option is in force.

23.2.13 History Off

NCGB has the ability to track the history of GB production. This is used when the RemoveRedundant Options are turned on in NCProcess. Since much storage is consumed by this history it is often advantageous to turn it off. If you are just running NCMakeGB and you want to turn it off do this.

The command

SetRecordHistory[False]

turns off the history collection, except that it replaces useless rules which the code produced by the one dummy rule

InvalidHistoryRule → 0

The command

SetRecordHistory[True]

turns on the history collection.

If you happen to bring an InvalidHistoryRule → 0 back into Mma, either

a) InvalidRuleHistory is not a member of the order (as expected) and an error message is printed and then is reformatted as 1 → 0 OR

b) InvalidRuleHistory is (artificially) a member of the order and one gets InvalidHistoryRule → 0.

23.2.14 Correspondence to sections 'Simplification' and 'Reduction'

This subsection is a brief note to bring out the conceptual connection between making a Gröbner Basis and the *simplification* commands and the Reduction commands. Making a Gröbner Basis is the first step in the *simplification* commands presented above. Making a Gröbner Basis is also the main step in the the command *SmallBasis* in §24.1.1 and *Spreadsheet* in §18.2.1.

23.2.15 Setting Monomial Orders- See Chapter 21

23.2.16 ReinstateOrder[]

Aliases: **None**

Description: After a call to *SetGlobalPtr[]*, *ClearMonomialOrde r[]* or *ClearMonomialOrderN[n]*, *ReinstateOrder[]* is used to restore the previously established order of indeterminates. The exception is *C learOrder[].* After this function is called, the order cannot be restored.

Arguments: None

Comments / Limitations: Not available before NCAlgebra 1.2.

23.3 Reduction

23.3.1 Reduction[aListOfPolynomials, aListOfRules]

Aliases: **None**

Description: *Reduction[aListOfPolynomials, aListOfRules]* computes the Normal Forms of *aListOfPolynomials* with respect to the rules aListOfRules.

Arguments: *aListOfPolynomials* is a list of polynomials. *aListOfRules* is a list of rules. That is the rules are applied repeatedly to the polynomials in the list until no further reduction occurs

Comments / Limitations: Not available before NCAlgebra 1.2

23.3.2 PolyToRule[aPolynomial]

Aliases: **PolToRule,ToGBRule,ToGBRules**

Description: *PolyToRule[aPolynomial]* returns the replacement rule corresponding to *aPolynomial*, assuming the order of monomials has been set.

Arguments: *aPolynomial* is a polynomial

Comments / Limitations: Not available before NCAlgebra 1.2

23.3.3 RuleToPoly[aRule]

Aliases: **None**

Description: *RuletoPoly[aRule]* returns *aPolynomial* corresponding to the replacement rule *aRule*.

Arguments: *aPolynomial* is a polynomial

Comments / Limitations: Not available before NCAlgebra 1.2

Chapter 24

Commands for Making Small Bases for Ideals: Small Basis, Shrink Basis

A Gröbner Basis can be infinite. Even when a Gröbner Basis is finite, it can be very large and therefore difficult for a person to digest. One often finds that there are many relations which are generated which do not enhance our understanding of the mathematics. In many cases we want a basis for an ideal which is minimal (i.e., has smallest cardinality) or which is minimal as a subset of a given basis. We, therefore, find it helpful to take a list of rules which are generated by the GB algorithm and make them smaller. Consider the following example.

Example 24.1 *A GB generated by the set* $\{PTP - TP, P^2 - P\}$ *is the set* $\{PT^nP - T^nP : n \geq 1\} \cup \{P^2 - P\}$ *regardless of the term order used. No smaller GB exists.*

Here just two relations generate infinitely many. One way to view this example is that the computer discovers that if a subspace (represented by $\mathrm{ran}(P)$ in the computation) is invariant for a linear transformation T, then it is invariant for T^n for every $n \geq 1$.

The GB algorithms tries to generate this infinite set of relations and at any time has generated a finite subset of them. When we are trying to discover a theorem or elegant formulas, often these relations having higher powers are irrelevant and clutter the spreadsheet which merely serves to confuse the user.

This introduces the next topic which is shrinking a set of relations to eliminate redundancy. Our desire is to take the generated basis and to remove mathematical redundancy from the generating set without destroying the information which was gained while running the GB algorithm.

We have two lines of attack on this problem. One will be described in this chapter, another will be described in Chapter 34.

24.1 Brute Force: Shrinking

Our second line of attack can be slower. It finds a subset of the set of relations by applying the GB machinery. For example, if one knows that p_n is a member of the ideal generated by $\{p_1, \ldots, p_{n-1}\}$, then the set $\{p_1, \ldots, p_n\}$ can be shrunk to $\{p_1, \ldots, p_{n-1}\}$. Determining whether or not p_n is a member of the ideal generated by $\{p_1, \ldots, p_{n-1}\}$ can be partially decided by running the GB algorithm for several iterations.

24.1.1 SmallBasis[aListOfPolynomials, anotherListOfPolynomials, iter]

Aliases: **None**

Description: $SmallBasis[aListOfPolynomials, , iter]$ computes and returns a list $aList$ which is a subset of $aListOfPolynomials$ such that the ideal generated by *aList* equals the ideal generated by *aListOfPolynomials* . *anotherListOfPolynomials* adds the feature that when one includes it in the call it just joins to the answer above without being changed. In other words, *SmallBasis* computes and returns a list *aList* which is a subset of $aListOfPolynomials$ such that the ideal generated by *Join[aList, anotherListOfPolynomials]* equals the ideal generated by *Join[aListOfPolynomials, anotherListOfPolynomials]*. This is done via calls to $NCMakeGB$. $NCMakeGB$ iterates at most iter times.
Here is how it works: Let LP denote $\{p_1, p_2, \ldots, p_u\}$. First *SmallBasis[LP,{},iter]* generates a partial GB for p_1, p_2, and reduces $p_3, p_4, \ldots, p_u$ with this GB. If the result is 0, then *SmallBasis* stops and returns $\{p_1, p_2\}$. If not *SmallBasis* generates $GB(p_1, p_2, p_3)$ and reduces the remaining polynomials. Etc. Finally, *SmallBasis* returns $\{p_1, p_2, \ldots, p_k\}$ with the property that the partial Göbner Basis $GB(p_1, \ldots, p_k)$ reduces $\{p_1, p_2, \ldots, p_n\}$ to zero. Clearly, the result is very dependent on the order if the entries lie in LP. The integer iter determines the number of iterations used to produce the partial GB's.
SmallBasis[LP1,LP2,iter] has the same functionality as *Complement[SmallBasis[Join[LP2,LP1], {}, iter], LP2]*, (i.e., *SmallBasis[Join[LP2,LP1], {}, iter]* minus the set *LP2*) but is much faster.

Arguments: $aListOfPolynomials$ and $anotherListOfPolynomials$ are lists of polynomials. *iter* is a natural number.

Comments / Limitations:

WARNING: This command calls *NCMakeGB* which effects the output of the *WhatIsHistory* command which in turn can effect the use of the command RemoveRedundant and its variants (subsections 34.1.2, 34.1.3, 34.1.4 and 34.1.5) or the use of the command *SmallBasis* and its variants (subsections 24.1.1 and 24.1.2).

24.1.2 SmallBasisByCategory[aListOfPolynomials, iter]

Aliases: **None**

Description: $SmallBasisByForCategory$ returns a list $aList$ which is a sorted subset of $aListOfPolynomials$ such that the ideal generated by $Join[aList, anotherListOfPolynomial$ equals the ideal generated by $Join[aListOfPolynomials, anotherListOfPolynomials]$. This is done via calls $NCMakeGB$.

Arguments: $aListOfPolynomials$ is a list of polynomials. $iter$ is a natural number.

Comments / Limitations: Not available before NCAlgebra 1.2

24.1.3 ShrinkOutput[aListOfPolynomials,fileName]

Aliases: **None**

Description: *ShrinkOutput* can take a very long time to run on large or complicated sets of polynomials. *ShrinkOutput[aListOfPolynomials,fileName]* takes the list of polynomials in *aListOfPolynomials* and puts a subset of *aListOfPolynomials* into a file via *RegularOutput* (subsection 26.3.1) This smaller subset is generated by the command *SmallBasis* (subsection 24.1.1)

Arguments: *aListOfPolynomials* is a list of polynomials. *fileName* is a character string

Comments / Limitations: Not available before NCAlgebra 1.2

24.2 Brute Force: Many shrinks

24.2.1 ShrinkBasis[aListOfPolynomials,iterations]

Aliases: **None**

Description: *ShrinkBasis* can take a long time to run and returns a $\{L_1, L_2, \ldots, L_r\}$ of lists of polynomials. Each list L_j of polynomials is contained in *aListOfPolynomials*. Each list L_j is a generating set for the ideal generated by *aListOfPolynomials* and the computer makes a limited attempt to show that any proper subset of L_j does not generate the ideal. Moreover, the union of the L_j's is *aListOfPolynomials*.

ShrinkBasis depends upon the monomial order which is set, because it calls *NCMakeGB* where it iterates at most *iterations* times. If all of the runs of *NCMakeGB* generate Gröbner Bases in *iterations* iterations, then no proper subset of an L_j is a generating set for the ideal generated by *aListOfPolynomials*.

The command works by brute force. It selects a polynomial from *aListOfPolynomials* and computes a partial GB for the remaining polynomials (with *iterations* iterations). It uses these to eliminate the original polynomial.

Arguments: *aListOfPolynomials* is a list of polynomials. *fileName* is a string.

Comments / Limitations: Not available before NCAlgebra 1.2

24.3 First Example

For example, after loading the files *NCGB.m*, *SmallBasis.m* (§24.1.1) we can execute the commands to compute a subset of a Gröbner Basis for the set of relations $\{p**p - p, p**a**p - a**p\}$:

```
In[2]:= SetKnowns[a,p]
In[3]:= {p**p-p,p**a**p - a**p}
Out[3]= {-p + p ** p, -a ** p + p ** a ** p}
In[4]:= NCMakeGB[%,4]

Out[4]= {-p + p ** p, -a ** p + p ** a ** p, -a ** a ** p + p ** a ** a ** p,
>    -a ** a ** a ** p + p ** a ** a ** a ** p,
>    -a ** a ** a ** a ** p + p ** a ** a ** a ** a ** p,
>    -a ** a ** a ** a ** a ** p + p ** a ** a ** a ** a ** a ** p,
>    -a ** a ** a ** a ** a ** a ** p + p ** a ** a ** a ** a ** a ** a ** p,
>    -a ** a ** a ** a ** a ** a ** a ** p +
>     p ** a ** a ** a ** a ** a ** a ** a ** p,
>    -a ** a ** a ** a ** a ** a ** a ** a ** p +
>     p ** a ** a ** a ** a ** a ** a ** a ** a ** p,
>    -a ** a ** a ** a ** a ** a ** a ** a ** a ** p +
>     p ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p,
>    -a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p +
>     p ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p,
>    -a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p +
>     p ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p,
>    -a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p +
>     p ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p,
>    -a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p +
>     p ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p\
>     , -a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a **
>       a ** p + p ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a **
>      a ** a ** a ** p, -a ** a ** a ** a ** a ** a ** a ** a ** a ** a **
>       a ** a ** a ** a ** a ** p +
>     p ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a **
>      a ** a ** p, -a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a **
>       a ** a ** a ** a ** a ** p +
>     p ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a **
>      a ** a ** a ** p}
```

The command `SmallBasis` takes this (or any) set of relations and shrinks it down to a smaller set of relations which generate the same ideal. One must have a monomial order set because *SmallBasis* (§24.1.1) calls *NCMakeGB*. `SmallBasis` returns a subset of the original set of relations. In the example below the `SmallBasis` command shows that the ideal generated by the set `Out[4]` equals the ideal generated by $\{-p+p**p, -a**p+p**a**p\}$. [1]

```
In[5]:= SmallBasis[%4,{},4];
Out[5]= {-p + p ** p, -a ** p + p ** a ** p}
```

24.4 Second Example

As a second example, after loading the files *NCGB.m*, *SmallBasis.m* and *RemoveRedundent.m*, we can execute the following commands to compute a Gröbner Basis for the set of relations $\{x^2-a, x^3-b\}$:

```
In[3]:= SetKnowns[a,b,x]

In[4]:= NCMakeGB[{x**x-a,x**x**x-b},10]
Out[4]= {-a + x ** x, -b + a ** x, -b + x ** a, -a ** a + b ** x,
>    -a ** b + b ** a, -a ** a + x ** b, -b ** b + a ** a ** a}
```

Now, one might want to find a smaller generating set for the ideal specified above. The following command does that and took 9 seconds using the C++ version of the code.

```
In[5]:= SmallBasis[%4,{},3]
Out[5]= {a ** x -> b, x ** x -> a}
```

24.5 Smaller Bases and the Spreadsheet command

Here is a session which does roughly what the spreadsheet command does. For more detail see Chapter 18

```
In[11]:=NCMakeGB[FAC,2];
In[11]:=SmallBasisByCategory[
RemoveRedundant[%, HISTORY see Section \ref{}??] ];
```

The next command tries to see if any of the undigested relations can be made simpler using the digested relations

```
In[12]:=NCSimplifyAll[%11,  DigestedRelations[%11] ];
```

Finally we output the result to the file “SecondOutputForDemo”.

```
In[13]:=RegularOutput[%12, "SecondOutputForDemo"];
```

Now we return to the strategy demos of Chapter 18.

Inserting the command `RemoveRedundant`, see Section 34, inside of small basis may change the answer, but it yields the same answer as `SmallBasis` would have given with one particular order on the set of relations given as input. All this assumes that the number of iterations is large. Inserting `RemoveRedundant` saves much computer time.

---

[1] It takes 113 seconds. The present implementation of the code involves alot of interaction with Mathematica. We expect future versions of the code to be faster.

24.6 How Small Basis commands relate to the similar NCProcess Options

The small basis and small basis by cawtegory options inside NCProcess are constructed from the small basis command described in this chapter.

Chapter 25

Help in Typing Relations .

25.0.1 NCMakeRelations[aSpecialList, aSpecialList, . . .]

Aliases: **None**

Description: This command allows the user to easily generate certain lists of polynomials which can be used as input to NCMakeRules. We begin with some examples and then explain the command in the third paragraph of this subsection. Suppose that $z1$, $z2$, $z3$ and $z4$ have been set to be noncommutative (if not, just type *SetNonCommutative[z1,z2,z3,z4]*). If one types

```
<< NCMakeRelations.m
NCMakeRelations[{Inv, a, b, c},
{SelfAdjoint, x, y, {{z1, z2}, {z3, z4}}}]
```

then this function returns the list

$$\begin{aligned}\{a**Inv[a] == 1, Inv[a]**a == 1, b**Inv[b] == 1, Inv[b]**b == 1,\\ c**Inv[c] == 1, Inv[c]**c == 1, -x + aj[x] == 0, -y + aj[y] == 0,\\ -z1 + aj[z1] == 0, -z2 + aj[z3] == 0,\\ -z3 + aj[z2] == 0, -z4 + aj[z4] == 0\}\end{aligned}$$

NCMakeRelations takes any number of arguments and each of its arguments is a List. For each list, the first word describes the type of relations and the other elements can be either simple expressions (like x, y, $1 - y ** x$, $1 - x ** y$, etc.) or square matrices with symbolic entries (such as in the example above) or any mixture of these two types of data. We now list the allowed types of relations. Some of the relations are stated in terms of the hereditary functional calculus of Jim Agler (see HereditaryCalculus.m).

[1] Isometry (i.e., T such that $T^*T - 1 = 0$)

[2] CoIsometry (i.e., T such that $TT^* - 1 = 0$)

[3] SelfAdjoint (i.e., T such that $T^* - T = 0$)

[4] Projection (i.e., T such that $T^2 - T = 0$)

[5] InvL (i.e., T such that $InvL[T] * *T = 1$)

[6] InvR (i.e., T such that $T * *InvR[T] = 1$)

[7] Inv (i.e., T such that $Inv[T] * *T = 1$ and $T * *Inv[T] = 1$)

[8] Rt (i.e., T such that $Rt[T]^2 = T$)

[9] Pinv (i.e., T such that $T * *Pinv[T] * *T = T$, $Pinv[T] * *T * *Pinv[T] = Pinv[T]$, $tp[T * *Pinv[T]] = T * *Pinv[T]$, $tp[Pinv[T] * *T] = Pinv[T] * *T$)

[10] PerpL (i.e., T such that $PerpL[T] * *T = 0$)

[11] PerpR (i.e., T such that $T * *PerpR[T] = 0$)

[12] PerpL2 (i.e., T such that $1 - T * *Pinv[T] = PerpL2[T]$ and $tp[PerpL2[T]] = PerpL2[T]$)

[13] PerpR2 (i.e., T such that $1 - Pinv[T] * *T = PerpR2[T]$ and $tp[PerpR2[T]] = PerpR2[T]$)

[14] Isometry[aInteger] (where aInteger is a natural number) (i.e., T such that $(yx - 1)^{aInteger}(T) = 0$)

[15] Symmetry[aInteger] (where aInteger is a natural number) (i.e., T such that $(y - x)^{aInteger}(T) = 0$)

[16] Isosymmetry (i.e., T such that $(yx - 1)(y - x)(T) = 0$ or equivalently $T^{*2}T - T^{*}T^2 - T^{*} + T = 0$). These extremely important operators were the subject of Mark Stankus incredible thesis.

Arguments: aSpecialList is a list whose first element is one of the allowed types above and the rest of whose elements are either simple expressions or square matrices.

Comments / Limitations: Not available before NCAlgebra 1.2

25.1 Output notation for pseudoinverse and perp's

$$
\begin{array}{lll}
Pinv[x] & \longrightarrow \; x^{+} & xx^{+}x = x, \quad x^{+}xx^{+} = x^{+}, \\
 & & (x^{+}x)^{T} = x^{+}x, \quad (xx^{+})^{T} = xx^{+} \\
PerpL[x] & \longrightarrow \; x^{\llcorner} & x^{\llcorner}x = 0 \\
PerpR[x] & \longrightarrow \; x^{\lrcorner} & xx^{\lrcorner} = 0 \\
PerpL2[x] & \longrightarrow \; x^{\llcorner\!\llcorner} & x^{\llcorner\!\llcorner} = 1 - xx^{+}, \quad (x^{\llcorner\!\llcorner})^{T} = x^{\llcorner\!\llcorner} \\
PerpR2[x] & \longrightarrow \; x^{\lrcorner\!\lrcorner} & x^{\lrcorner\!\lrcorner} = 1 - x^{+}x, \quad (x^{\lrcorner\!\lrcorner})^{T} = x^{\lrcorner\!\lrcorner}
\end{array}
$$

25.1.1 NCAddTranspose[aListOfExpressions]

Aliases:

Description: One can save time when working in an algebra with transposes or adjoints by using the command NCAddTranspose[ ]. These commands "symmetrize" a set of relations by applying tp[ ] to the relations and returning a list with the new expressions appended to the old ones. This saves the user the trouble of typing both $a = b$ and $tp[a] = tp[b]$.

Arguments: *aListOfExpressions* is a list of expressions

Comments / Limitations:

25.1.2 NCAddAdjoint[aListOfExpressions]

Aliases:

Description: One can save time when working in an algebra with transposes or adjoints by using the command NCAddAdjoint[ ]. These commands "symmetrize" a set of relations by applying aj[ ] to the relations and returning a list with the new expressions appended to the old ones. This saves the user the trouble of typing both $a = b$ and $aj[a] = aj[b]$.

Arguments: *aListOfExpressions* is a list of expressions

Comments / Limitations:

25.1.3 Pulling important equations into your session from an *NCProcess* output - See *GetCategories* in §26.0.5.

25.1.4 Help in typing Monomial Orders - See NCAutomaticOrder Section 21.4.6

Chapter 26

Retrieving Categories and Regular Output

Crucial to our approach to "semi-automatic" discovery of formulas and theorems is the sorting of relations according to which unknowns appear in them — in other words, sorting lists of polynomials into categories. *RegularOutput* does this sorting and presents categories of relations (by storing them in file). Actually the core of *RegularOutput* is a command *CreateCategories* which creates categories and gives them a name. These categories can be summoned into your session using the *GetCategory* command.

26.0.5 GetCategory[aListOfVariables, NCPAns]

Aliases: **None**

Description: $GetCategory[aListOfVariables, aName]$ retrieves the polynomials in the category stored in $aName$ corresponding to the list of variables in the list $aListOfVariables$. See Section 17.3 for an example. To be recognized immediately after an NCProcess run $aListOfVariables$ must equal a list of unknowns which corresponds to a category in that NCProcess run. The NCProcess stores all category information in NCPAns. The next NCProcess starts by clearing NCPAns and writes the category information it produces in NCPAns.

Arguments: $aListOfVariables$ is a list of variables. $aName$ is a symbol which has not been used previously.

Comments / Limitations: Not available before NCAlgebra 1.2. See CreateCategories.

Another way to summon categories is by "name".

26.0.6 GetCategory[aCharString,NCPAns]

Aliases: **None**

Description:

(1) If *aCharString* is "Undigested", then all of the undigested relations are returned.

(2) If *aCharString* is "Digested", then all of the digested relations are returned.

(3) If *aCharString* is "Unknowns", then all of the undigested relations are returned.

(4) If *aCharString* is "Knowns", then all of the relations involving just knowns are returned.

(5) If *aCharString* is "digestedLabels", then {{}} is returned.

(6) If *aCharString* is "digestedRules", then all of the digested relations are returned.

(7) If *aCharString* is "userSelectsRules", then all of the user selected relations are returned.

(8) If *aCharString* is "userSelectsLabels", then all of the user selected labels are returned.

(9) If *aCharString* is "knownsLabels", then {{}} is returned.

(10) If *aCharString* is "knownsRules", then all of the relations involving just knowns are returned.

(11) If *aCharString* is "singleRules", then all of the relations which solve directly for unknowns are returned.

(12) If *aCharString* is "singleVars", then all of the unknowns which have been solved for directly are returned.

Otherwise, *NCPAns[aCharString]* is returned.

Arguments: *aCharString* is a character string. *NCPAns* is a symbol.

Comments / Limitations:

26.0.7 Clear[NCPAns]

Aliases: **None**

Description: Clear[aName] clears whatever info is stored in aName.

Arguments: Clear[NCPAns] clears NCPAns which might matter to you if you run shaorty of memory.

Comments / Limitations:

26.1 Example

Suppose we are in a NCProcess session which produced the output on page 156 and we really like the category whose unknowns are n_1 and m_1. To bring that category into our session so you can manipulate these equations with Mathematica use the command

```
In[17]:=  GetCategories[ { n1,m1 } , NCPAns ]

Out[17]=  {n1**m1 - 1}

In[18]:=  GetCategories[ "singleVars", NCPAns ]

Out[18]=  {a,b,c,e,f,g}
```

26.2 Creating Categories

This is the command which is used inside NCProcess to instruct the $C++$ code to make categories. It takes aName to be NCPAns.

Before using this commands one must use *SetMonomialOrder*.

26.2.1 CreateCategories[aListOfPolynomials, aName]

Aliases: **None**

Description: *CreateCategories[aListOfPolynomials,aName]* sorts *aListOfPolynomials* into categories. Each category and its label is stored in the array associated to the symbol $aName$ and it is accessible by $aName[label]$ using the *GetCategory* command. $aListOfPolynomials$ is a list of polynomials. $aName$ is a symbol which has not been used previously.

Arguments: *aListOfPolynomials* is a list of polynomials. *aName* is a Mathematica symbol.

Comments / Limitations: Not available before NCAlgebra 1.2. See GetCategory.

This is the command which is used inside NCProcess to instruct the $C++$ code to make categories. It takes aName to be NCPAns.

26.3 RegularOutput[aListOfPolynomials, "fileName"]

26.3.1 RegularOutput[aListOfPolynomials, "fileName"]

Aliases: **None**

Description: *RegularOutput[aListOfPolynomials, "fileName"]* takes the list of polynomials in *aListOfPolynomials* and organizes this list in a certain way, and places the organized information in a file called "fileName". It is the heart of the NCProcess display. The organization performed is based on the monomial order which was specified using *SetMonomialOrder*. Think of every variable in aList1 declared, via *SetMonomialOrder[aList1,1]*, to be lowest in the ordering as a known. The unknowns are the remaining variables denoted *var* which are defined, via *SetMonomialOrder[aList,n]* where *aList* is a list of variables and *var* in a member of this list and $n \geq 2$. For *RegularOutput* the only thing that counts is unknowns. *RegularOutput* sorts polynomials in *aListOfPolynomials* into *categories consisting of polynomials having the same unknowns.*

Arguments: *"fileName"* is a character string. *aListOfPolynomials* is a list of polynomials.

Comments / Limitations: Not available before NCAlgebra 1.2

26.4 How to Really Change Regular Output

Important to research is how one sorts the output of a GB algorithm. Serious changes might prompt you to edit the command itself. We now give a very few tips. The only files we have modified in the last few years are

```
OutputArrayForTeX.m
OutputSingleCategory.m
TypesOfStrategyOutput.m*
NCProcess.m
NCCollectOnVariables.m
NCMakeRelations.m
oCategories.m
```

The top three of which could be useful for those who wish to make serious changes to the output of the GB algorithm.

Chapter 27

The Dimension of the Solution Set of a Set of Polynomial Equations

Eric C. Rowell

Given a system of noncommuting polynomials S, one may wish to have some sense of how large the solution set is, i.e. the degrees of freedom. There are a number of possible approaches (none of which is perfect) to answering this question, and we outline one of them here. This method is useful and interesting of its own right, and is perhaps more valuable as an example in which the Groebner Basis algorithm can be used by pure algebraists.

27.1 The Commuting Case

If the polynomials S are members of a commutative affine polynomial (finite number of variables) ring, then there is a very classical way of computing the dimension of the solution set. One uses the *Krull dimension* or *trancendance degree* which are the same for affine commutative algebras. Daniel Lichtblau at Mathematica has written code that will do this.

27.2 Noncommutative Case: Gelfand-Kirillov dimension

Let $R = k < x_1, ..., x_n >$, the polynomial ring over k in n noncommuting variables. The *standard filtration* on R is the sequence of vector subspaces of R, R_t, consisting of all polynomials of degree t or less. If I is an ideal of R, set $T = R/I$. Then the standard filtration T_t on T is the sequence $\{R_t/(I \cap R_t)\}$. If we define $H_T(t) = \dim_k(T_t)$, the Hilbert series of T corresponding to this filtration is

$$\sum_{t \geq 0} H_T(t) z^t$$

where z is an indeterminate. The *standard gradation* on R is the sequence $R^{(s)}$ of vector subspaces of R consisting of polynomials that are homogeneous of degree s. This has the nice property that

$$R = \bigoplus_{s \geq 0} R^{(s)}.$$

Now T may not have a gradation, but in the case that I is a homogeneous ideal then $T^{(s)}$ is defined similarly as in the filtration case. Note that $T_t = \bigcup_{0 \leq s \leq t} T^{(s)}$. Setting $H^T(s) = \dim_k T^{(s)}$, the Hilbert series corresponding to this gradation is

$$\sum_{s \geq 0} H^T(s) z^s.$$

By the note above we have that

$$H_T(t) = \sum_{0 \leq s \leq t} H^T(s).$$

Usually when we refer to the Hilbert series of an algebra we must be specific about the gradation or filtration we are using as there are many besides the standard ones. However, here we take the convention that if an algebra is graded we take the Hilbert series to mean the one with respect to the standard gradation, otherwise we are using the standard filtration. This should cause no confusion, as we are dealing strictly with finitely generated polynomial algebras.

The following dimension has been around since the 1960s and has become a main tool in noncommutative algebra. It was first introduced by I.M. Gelfand and A.A. Kirillov in a paper in 1966 describing work on enveloping algebras of Lie algebras. References for background and proofs of theorems are at the end of this chapter.

Definition 27.1 *The Gelfand-Kirillov dimension of T is defined to be*

$$\limsup_{t \to \infty} \frac{\log(H_T(t))}{\log(t)}.$$

We need a little bit more notation. Let A be an affine k-algebra, that is, an algebra finitely generated over a field k. Then $A = k[V]$ where $V = Span_k\{1, x_1, \ldots, x_m\}$ for some $\{x_j\} \subset A$. Define

$$V^n = Span_k\{x_{j_1} x_{j_2} \cdots x_{j_s} : 1 \leq j_i \leq m, \sum_i j_i \leq n\}$$

so that $k, V, V^2, \ldots, V^n, \ldots$ gives the standard filtration of A. Note also that $H_A(t) = \dim_k V^t$.

A quick fact about GK dimension: It is known that there exist algebras of GK dimension r for any real number $r \in \{0\} \cup \{1\} \cup [2, \infty)$, and moreover no other number is attained.

There are several reasons that the *Gelfand-Kirillov* (hereafter abreviated *GK*) dimension is an apropriate measure of degrees of freedom. The following four standard theorems tell us that the GK dimension behaves much like the transcendance degree from commutative algebra. They follow easily from the definition, and a little combinitorics. Proofs can be found in the references at the end of the chapter.

Theorem 27.2 *If A is a commutative algebra, then*

$$GK(A) = Krull(A).$$

Theorem 27.3 *If $B \subset A$ is a subalgebra, then*

$$GK(B) \leq GK(A).$$

Theorem 27.4 *If C is a homomorphic image of A, then*

$$GK(C) \leq GK(A).$$

Theorem 27.5 *Let A be an affine algebra, with GK dimension r. Then $GK(A[x]) = r+1$, where x is a new central indeterminate.*

The following theorem shows that the GK dimension is well-behaved with respect to quotient algebras.

Theorem 27.6 *Let A be an affine algebra and I an ideal of A that contains a left (or right) non-zero-divisor, $c \in A$. Then*

$$GK(A/I) + 1 \leq GK(A).$$

Proof. *Using the notation above, suppose c is a degree l non-zero-divisor. Let C_n be the vector space complement of*
$V^n \cap I$ in V^n, So $C_n \simeq V^n/(V^n \cap I)$. Note that the sequence $\{V^n/(V^n \cap I)\}_{n=1}^{\infty}$ gives a filtration of A/I. Now $C_n \cap \langle c \rangle = \{0\}$, as $C_n \cap I = \{0\}$. So $C_n \oplus C_n c \oplus \cdots \oplus C_n c^n \subset V^{n(l+1)}$ and the $\oplus$ are valid as $ac^i = bc^j, i < j$ implies $c^i(a - bc^{j-i}) = 0$ and c a left non-zero-divisor then gives $a - bc^{j-i} = 0$ so $a \in \langle c \rangle \cap C_n$ which implies $a = 0$. So

$$\dim_k(V^{n(l+1)}) \geq n \cdot \dim_k(C_n) = \dim_k(V^n/(V^n \cap I)).$$

Thus

$$GK(A/I) + 1 = \limsup_{n\to\infty} \frac{\log(\dim_k(C_n))}{\log(n)} + 1 =$$
$$\limsup_{n\to\infty} \frac{\log(\dim_k(C_n))}{\log(n)} + \frac{\log(n)}{\log(n)} = \limsup_{n\to\infty} \frac{\log(n \cdot \dim_k(C_n))}{\log(n)} \leq$$
$$\limsup_{n\to\infty} \frac{\log(\dim_k(V^{n(l+1)}))}{\log(n)} \leq \limsup_{n\to\infty} \frac{\log(\dim_k(V^n))}{\log(n)} = GK(A).$$

The computation of the GK dimension is in general difficult. Since the sequence whose limit is the GK dimension converges very slowly, the only hope is to compute some of the coefficients of the Hilbert series and then guess a generating formula for them. Then the GK dimension can be computed by taking the limit of this sequence. This may sound a bit *ad hoc*, but it has been implemented for several interesting algebras, an example of which follows.

The algebra we consider is the algebra on two variables generated by the relations that say that any two degree 2 monomials commute with each other. The commands used here are explained in the section on Commands.

We use the input file as follows:

```
<<NCHilbertCoefficient.m;

SNC[x,y];

SetMonomialOrder[{x,y}];

rels = {x**x**y**y - y**y**x**x, x**x**x**y - x**y**x**x,
    -x**y**y**y + y**y**x**y, x**x**y**x - y**x**x**x,
    -y**x**y**y + y**y**y**x, x**y**y**x - y**x**x**y};

NCHilbertCoefficient[18,rels,3,ExpressionForm->Homogeneous];
```

The call to `NCMakeGB` finishes after only 2 iterations, so we know that the coefficients are being computed using a full Groebner basis, hence they are exact. The output is the following:

```
{2, 4, 8, 10, 12, 16, 20, 25, 30, 36, 42, 49, 56, 64, 72,
81, 90, 100}
```

After staring at this sequence for a while, one sees that every other term is a square and the intermediate terms are products of sucessive integers after the 6th term. A formula then is $[n][n+1]$ where $[a]$ is the greatest integer less than or equal to a. This is then clearly asymtotic to a quadratic polynomial. This sequence came from a gradation, so to get the sequence coming from the filtration we simply take the partial sums. This in turn will be asymtotic to a cubic polynomial in n, which we will call $P(n)$. So the GK dimension of the quotient of the free polynomial algebra on two variables by the ideal generated by `rels` is:

$$\limsup_{n\to\infty} \frac{\log(P(n))}{\log(n)} = 3.$$

For further information on the function `NCHilbertCoefficient` used above, see the documentation section.

27.3 References

1. Nastasescu, Constantin; van Oystaeyen, Freddy. Dimesions of Ring Theory.

2. McConnell, J.C.; Robson, J.C. Noncommutative Noetherian rings.

3. Krause, G.R.; Lenagan, T.H. Growth of algebras and Gelfand-Kirillov dimension.

27.4 Commands

The commands and algorithms were done by Eric Rowell with help from Dell Kronewitter and Bill Helton.

27.4.1 NCHilbertCoefficient[integer1, aListOfExpressions, integer2, anOption]

Aliases: **none**

Description: NCHilbertCoefficient[integer1,aListOfExpressions,integer2, anOption] attempts to compute the first *integer*1 coefficients of the Hilbert series for the algebra generated by the relations in `aListOfExpressions`. There are four possible calls to this function, here expressed in order of longest time used to least.

- The default (no fifth arguement) is for algebras that are nonhomogeneous. This will compute a (possibly partial) Groebner Basis out to *integer*2 iterations of Mora's algorithm with respect to the ambient order, convert this basis to rules, and procede to compute the specified number of Hilbert series coefficients. Unless the partial Groebner basis computed contains all the polynomials that will ever appear in the (possibly infinite) full Groebner basis up to degree *integer*1, the dimensions of T_t computed will only be upper bounds.
- $ExpressionForm \rightarrow Homogeneous$. This is only valid for homogeneous ideals. This does as above, only the resulting dimensions are for the standard gradation, and takes much less time. In theory, there should be no problem with the dimensions being inaccurate provided enough iterations are used. There is an algorithm for homogeneous problems that *will* return all the polynomials of a specified degree and less.
- $ExpressionForm \rightarrow partialGBHomogeneous$. This is an option that will avoid the Groebner basis computation and simply convert the relations in `aListOfExpressions` to rules and use them to compute the Hilbert coefficients. This is useful particularly when one has already gone to the trouble of computing a (partial) Groebner basis. This is only coded for homogeneous ideals. The iteration number *integer*2 is ignored (although it must be there) so one may as well set it to 0.
- $ExpressionForm \rightarrow HomogeneousBinomial$. This is a very specific option for ideals whose generators are the difference of two monic monomials (i.e. of the form: $xyxz - yxzx$). This is essentially the same as the homogeneous version above, only faster.

Arguments: integer1, aListOfExpressions, integer2, anOption

Comments / Limitations: The order is alway the ambient order. Make certain that your order is only length lexicographic as this will save time. There is no reason to use any order other than length lexicographic for Hilbert series computations that the author of this code can think of. Currently the with the default version of this function the ambient order will be cleared during the computation, as there is a new variable introduced that is later removed. For now, just remember to reset the order before proceding.

27.4.2 NCX1VectorDimension[alist]

Aliases: **none**

Description: NCX1VectorDimension computes the dimension of the span of a set of polynomials as a vector space over the ground field.

Arguments: alist

Comments / Limitations: none

Chapter 28

Commands which are not supported

Here we list commands which we started to develop but did not pursue to the point of thorough testing. If we pursue them we may change their calling sequences, etc.

BlockPartition

This probably does not work reliably.

NCEliminate[ExpressionsList,varsList,opts]

This crudely generalizes the Mma command Eliminate to the noncommuting case. You should have set a monomial order before running it. If not, there is an option *UseNewOrder* → *False* which tells NCEliminate to use its (stupid) default ordering.

28.1 A Mathematica Groebner Basis Package Without C++

The files containing the word Old in their names, many residing in the directory "OldMmaGB", contain a primative version of NCGB but **all in Mma; no C++.** That was our first version.

Probably it still works but we have not tried it in a long time. It is slow, we remember that much. Since there is no document, about the best chance you have at deciphering it is to cruise thru the usage statements and function definitions.

28.2 NCXWholeProcess[polys, orderList, fileName, grobIters]

This *very* experimental function makes repeated calls to NCProcess[ ], changing the order at each iteration in an attempt to triangularize the set of relations, *polys*.

Let us walk through the first iteration of NCXWholeProcess. It begins with the order, *orderList*, and calls NCProcess[$polys$,$grobIters$,$fileName$-1] creating a LaTeXfile, $fileName$-1.tex.

The output of NCProcess will (hopefully) have polynomials in one unknown. These unknowns will then be regarded as "determined". These determined unknowns are moved to the bottom of the unknowns in the order.

A polynomial of the form

$$unknownIndeterminate \rightarrow polynomialsInKnownIndeterminates$$

is known as a singleton. Determined unknowns associated with singletons are moved to the top of the order and the other determined unknowns are moved to the bottom of the unknowns. That is, the new order is of the form

$$Knowns < determinedKnowns \ll otherUnknowns \ll singletonUnknowns.$$

We have finished the first iteration of NCXWholeProcess. The second iteration goes as follows.

After clearing the old order and setting the new order, NCProcess[*polys*,*grobIters*,"*fileName*-2", *UserSelects* → *determinedPolys*] is called creating a file, *fileName*-2.tex. Here *determinedPolys* are all polynomials in 0 or 1 unknowns and *polys* denotes the same set of polynomials that we input in the first iteration. The process is continued, changing the order and calling NCProcess repeatedly.

If no relations in one unknown are discovered at some iteration the currently unknown indeterminates are cyclically permuted. If all cyclic permutations fail to produce any new relations in one unknown the function NCXWholeProcess terminates stating that the problem was not solved. Alternatively, if all unknown variables become determined the function NCXWholeProcess terminates stating that the problem has been solved.

This function is in a very primitive stage and the particulars of it will likely change.

Chapter 29

Getting NCALGEBRA and NCGB

The easiest way to get NCAlgebra and NCGB is through the NCAlgebra homepage. It is also available by ftp. NCGB also requires compiling. We have compiled versions of NCGB available for a Microsoft Windows or a Solaris 1.1.3 or higher UNIX operating system. If your operating system does not fall into one of these categories you must get gnu C++ and compile it yourself. See the Web version of the NCGBDOCUMENT for instructions on compilation. As of November 1999 we consider windows operation of NCGB experimental and you should consider yourself α-testers. NCAlgebra works fine under windows and has for many years.

The goal of this section is to tell you how to get the files and directories which constitute NCAlgebra and NCGB.

29.1 Getting NCAlgebra and NCGB off the web

The NCAlgebra homepage address is

`http://math.ucsd.edu/~ncalg`

It can also be easily found by searching the web for NCAlgebra.

In the first line of text, there is a hyperlink attached to the word `downloaded` (i.e.,the word `downloaded` is underlined). Click on the word `downloaded` and you will move to a world-wide-web page with several choices as to what you might load. Click on the appropriate hyperlinks. (Some versions of Netscape require you to shift-click to download a file.)

29.2 Getting NCAlgebra and NCGB through anonymous ftp

It is possible to get NCAlgebra through anonymous ftp. Here is the standard description of how to get NCAlgebra through anonymous ftp.

```
(****************************************************************************)

                                   NCALGEBRA and NCGB

                                      Version 2.1



(****************************************************************************)
```

Thanks you for your interest in NCAlgebra and NCGB. This message contains the information necessary for you to use anonymous ftp to transfer the files from our site to yours. The ONLY thing which you have to do to get the NCAlgebra or NCGB package is to follow the following sample terminal session.

NCAlgebra (with or without) NCGB is at osiris.ucsd.edu, in the pub/ncalg directory. Below is a record of an actual ftp session. What the user types is underlined. Quoted expressions describe what should be typed. (e.g., where you see "Any thing will do,but please type your e-mail address.", you may type anything you want). Ignore all timing data shown below.

BELOW IS A SAMPLE SESSION. IF YOU FOLLOW IT, THEN YOU WILL BE ABLE TO GET A COPY OF THE NCAlgebra or NCGB PACKAGE.

YOU CAN HAVE YOUR LOCAL COMPUTER "GURU" FOLLOW THE BELOW INSTRUCTIONS BELOW AND INSTALL NCAlgebra or NCGB AS AN OFFICIAL MATHEMATICA PACKAGE.
ONCE THIS IS DONE, EVERYONE USING THE COMPUTER MAY USE NCAlgebra or NCGB.

```
% ftp osiris.ucsd.edu  or
% ftp 132.239.145.6
  -------------------

Connected to 132.239.145.6.

220 osiris FTP server (SunOS 4.0) ready.


Name (132.239.145.6:joejones): anonymous
                               ---------

331 Guest login ok, send ident as password.

Password: "Anything will do,but please type your e-mail address."
          -------------------------------------------------------

230 Guest login ok, access restrictions apply.
```

```
ftp> binary
     ------

200 Type set to I.


ftp> cd pub/ncalg
   ---------------

250 CWD command successful.

ftp> ls
     --
```

At this point, you will see a list of filenames. Some of them should [1] look like `NCGB.date.tar.Z`, `NCGB.date.tar.gz` and `NCGB.date.zip` . The ".Z" indicates the the file was compressed using the UNIX utility compress and the ".gz" indicates that the file was compressed using the GNU utility gzip. The ".tar" indicates that the tar program was used (tar is a UNIX standard is available for other platforms such as PCs). The ".zip" is a zipped file and is appropriate for PCs. One can choose either of the three versions. Obviously, you should pick the most recent version of the code and the one which is most appropriate to your computer system. To be concrete in the following material, we assume that you are interested in the file "NCGB.date.tar.Z".

```
ftp> get NCGB.date.tar.Z
     -----------------------

ftp> bye
     ---
221 Goodbye.
```

Now you want to read **ONE** of the sections 29.2.1, 29.2.2 or 29.2.3.

29.2.1 The ".Z" file

The next thing to do is uncompress the file. To do this, type

```
% uncompress NCGB.2.14.96.tar.Z
```

The next thing to do is to create a directory for the files and 'tar -xf' the file by typing:

```
% mkdir NCGB
% cd NCGB
% mv ../NCGB.2.14.96.tar ./
% tar -xf NCGB.2.14.96.tar
```

Proceed to 29.2.4.

---

[1] Here `date` represents the month,date and year that the file was put on anonymous ftp (for example, 2.14.96 means that the the file was put on anonymous ftp on Febuary 14,1996.

29.2.2 The ".gz" file

The next thing to do is uncompress the file. To do this, type

```
% gunzip NCGB.2.14.96.tar.gz
```

The next thing to do is to create a directory for the files and 'tar -xf' the file by typing:

```
% mkdir NCGB
% cd NCGB
% mv ../NCGB.2.14.96.tar ./
% tar -xf NCGB.2.14.96.tar
```

Proceed to 29.2.4.

29.2.3 The ".zip" file

The next thing to do is uncompress the file. To do this, type

```
% unzip NCGB.2.14.96.tar.zip
```

Proceed to 29.2.4.

29.2.4 Look at the document

So that we are able to inform them that new versions of the code are available, etc., PLEASE send us a email message so that we know that you are using the program.

The documentation for NCAlgebra is contained in the file
NC/DOCUMENTATION/NCDOCUMENT.dvi or .ps or .html

Maybe we shall have a PDF file soon too. Printing out this file to a printer is the best first step. (This can be done via a variety of programs including dvi2ps, pageview and ghostview.)

Also there are documents

NC/DOCUMENTATION/NCGBDOCUMENT.dvi or .ps
NC/DOCUMENTATION/NCOLDDOC.dvi or .ps
NC/DOCUMENTATION/SYSDOC.dvi or .ps

Also there are demos.

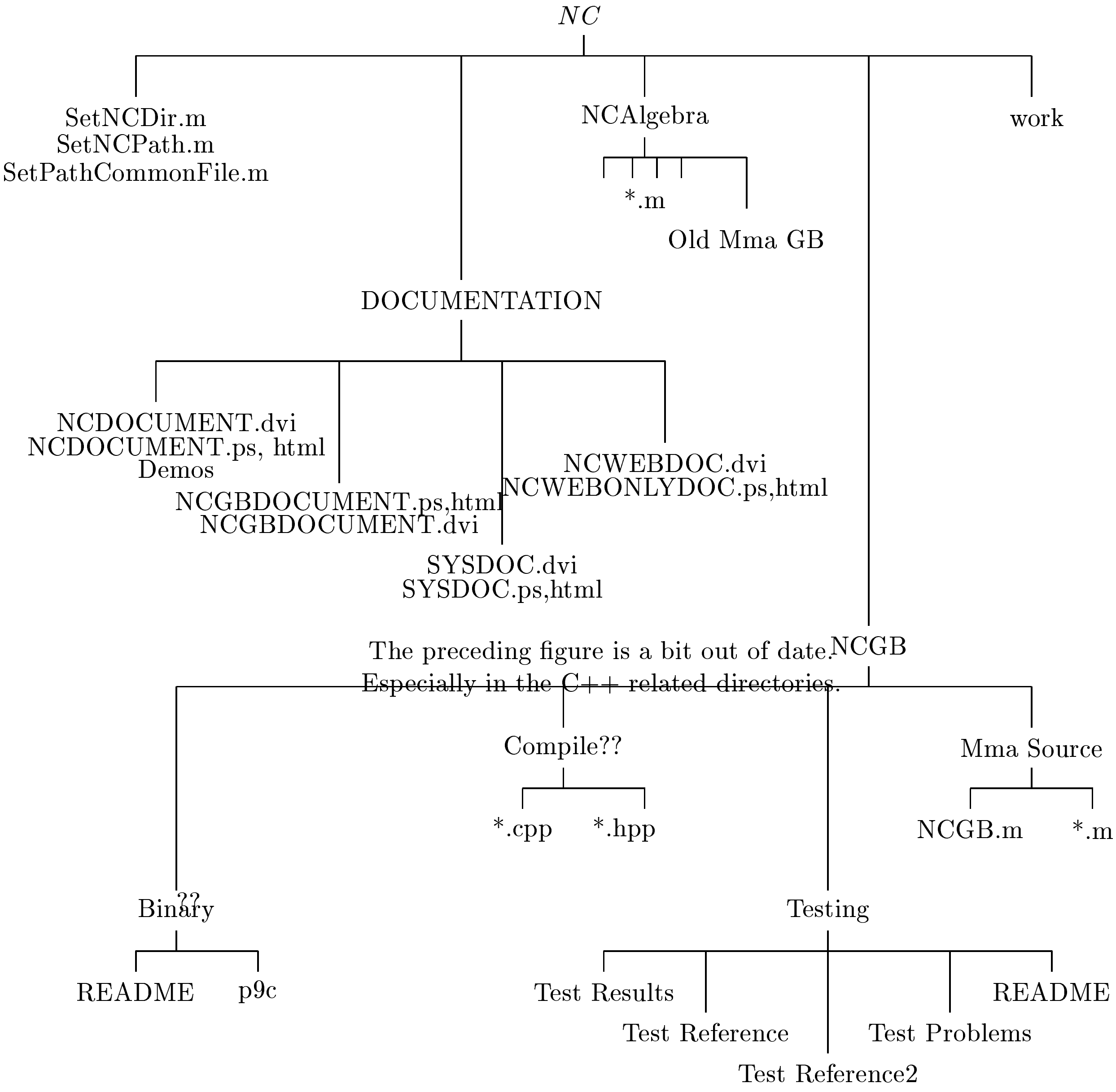
```
               Email any questions, comments or requests to

                          ncalg@ucsd.edu
```

29.3 The last step

If all you want to do is use NCAlgebra you are now ready. Enjoy!

For the NCGB experience under any operating system other than Solaris , Linux, or Windows more installation is required. Compiling the C++ part of the code is next. This is explained on the WWW NCAlgebra homepage NCDOCUMENT.

29.4 The NC directory structure

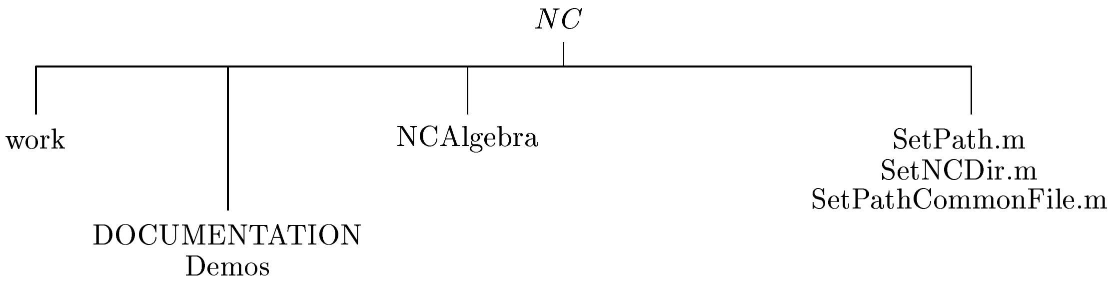


The preceding figure is a bit out of date.
Especially in the C++ related directories.

29.5 Directory structure of NCAlgebra alone

This adds a little more detail to Section 29.

(1) To run NCAlgebra alone you only need part of the directory system above. For NCAlgebra alone it is



If you have the whole huge sets of directories and are desperate for space you can delete all but this.

(2) The directory NCAlgebra should contain many files, for example,

```
NCAlgebra.m              CEEP
Lots of files starting with NC.
Lots of files ending with Extra.
Files related to TeX.
```

The most reliable way to find out the names of the NC-files actually used for computation is to consult the file NCAlgebra.m. It automatically loads them.

The files which have the suffix “Extra.” were small programs that are not fully tested and some of which are small specialized functions which may not be (are probably not) valuable to others. We include them just in case they may be valuable to someone and so that they can give examples of how someone may extend the code for their own purposes.

Chapter 30

Testing your version of NCGB

The goal is to test TWO versions of NCGB against each other to see if they produce exactly the same answer. Or one can test a version of NCGB against output which has been stored for reference. All files involved are in the subdirectory *Testing* of NCGB.
There are four directories inside *Testing*.

TestProblems
TestReference
TestReference2
TestResults

30.1 Beginners

We do not trust our testing programs for NCGB much. They need work. The NCAlgebra test program is excellent.

Basic Tests

There is an old testing program. We do not support or upgrade it but find it quite useful. It is **GBTEST** and the use of it is different as that described for **NCGBTEST** above. First you load the file GBTest.m . You run it by typing GBTEST[1,48] to run tests 1 thru 48, or for those with less patience GBTEST[12,28] to run tests 12 thru 28, etc.

Yet another class of specialized tests is run simply by loading the file NCGBXTEST99. These tests follow the format of the NCAlgebra tests NCTEST exactly. After you load the file

```
<<NCGBXTEST99
```

you hope to get True, True, printing to the screen. The tests in NCGBXTEST99 test programs which run under NCGB and were developed in 1999. They give NCGB a pretty good work out.

Someday we plan to integrate our testing procedures. As you see we now have a mixed bag of tests.

Fancy test program- DOES NOT WORK

THE COMPAREING STEP DOES NOT WORK in 2001.

When you download NCGB from the net, the directories *TestProblems* and *TestReference* contain many files. Those in *TestProblems* supply mathematical exercises to the testing programs. Those in *TestReference* are the result of running NCGB in ideal circumstances.

To see if the version of NCGB you downloaded is complete and functioning load **NCGBTEST**, If you do not have write access to NC/NCGB/Testing you may have to change the variable `$NC$TestResults$`. TestingEnvironment.m defines the default depository for Test Result files. This is described in greater detail in Section 30.3.1.

Enter Mathematica
≪ NCGBTEST

Go get a cup of coffee or two cups of coffee or a good night's sleep. When you come back look at the resulting screen output by typing

??test.

You should get

test #00 → True
test #01 → True
test #02 → True
etc.

This tests your version of NCGB against answers obtained from the version we have. If they do not agree there is a problem (possibly minor). One must be very careful with NCGBTEST. If paths are set wrong, then it will find no files for either the reference answers or the new code you are testing and since these behaviors are identical NCGBTEST will return True, True, etc. which is of course False, False, False!

30.2 Turning On Screen Output

There is a command which acts globally to turn output on and off:

NCGBMmaDiagnostics[True]
NCGBMmaDiagnostics[False]

also the NCProcess option

PrintSreenOutput → True
PrintSreenOutput → False

turns on (and off) more diagnostics only inside NCProcess. Default on both of these is False.

30.3 More Testing for Developers - DOES NOT WORK 2001

This section is for heavier testing than above, say of a new version of NCGB.[1]

30.3.1 Setting the Testing Environment

The file **TestingEnvironment.m** has four commands that may require editing. This file basically defines the directories where the testing is being done, where the sample problems are stored, where the results from the test are outputed and where the trusted reference answers are kept.

The command **NCTestPrefix$** defines where the testing is being done. Its default value is **/NC/NCGB/Testing/**, and should not be monkeyed with unless you have created your own testing directory.

NCTestInput$ defines where the test problems are kept. If you have some test problems of your own, you may test them by giving the path to them as the value. The default is

---

[1] You may want to type **NCDevelopers$=False. Note the default is NCDevelopers$=True. This unsets an environment variable which means you get the fancy version of our testing programs. We have forgotten what this does.**

NCTestInput$ = "StringJoin[NCTestPrefix$,"TestProblems/"]

If you do several tests, you can keep the results in separate directories by giving **NCTestResults$** the path as its value.

The default setting is

NCTestResults$ = "StringJoin[NCTestPrefix$,"TestResults/"]

NCTestReferences$ defines the directory in which the trusted answers to the test problems are stored. When you run the test, the files in *TestResults* will be compared to these reference files. Also if you are creating your own reference files, you may want to redefine the path to avoid overwriting the answer we so helpfully provide. The default is

NCTestReferences$ = "StringJoin[NCTestPrefix$,"TestReferences/"]

Remember, before doing anything, make sure the **TestingEnvironment.m** file has the right definitions to avoid sending files in all directions.

Creating Test Results

To create a set of test outputs stay in the *Testing* directory. Make sure you have lots of test problem files in the *TestProblems* directory; say c01.data.m thru c85.data.m. Make sure the *TestReference* directory is loaded with files generated by applying a version of NCGB that you trust to the problems in *TestProblems*. This should be the case because the package NCGB is shipped with such a collection of sample answers.

Edit 2 lines in the file

init.TestResults.m

to put in the path to the version of NCGB that you want to test.

Copy the file

init.TestReferences.m

to the file init.m

Run Mathematica.

Load NCGBTestCreate.m

Run

NCGBTestCreate[Integer1, Integer2]

Exit Mma

Here the arguments Integer must be a non-negative integer, satisfying Integer1 ≤ Integer2 ≤ 85 This creates a set of files of answers obtained from NCProcess labelled

cN.out.tex

where N is an integer between Integer1 and Integer2. They are stored in the directory *TestResults*.

If you want to test a program besides NCProcess you can use

NCGBTestCreate[Integer1,Integer2,FunctionName]

Label whichever program you want to test with FunctionName. For example, the name NCMakeGBFunction has already been assigned to NCMakeGB, so running

NCGBTestCreate[Integer1,Integer2,NCMakeGBFunction]

will create files for testing NCMakeGB.

Comparing to Reference Code

While in the testing directory

Run

NCGBTestCompare[Integer, Integer] (* for *.tex files only *)

or to test programs other than NCProcess

NCGBTestCompare[Integer, Integer, suffix]

(* for example, suffix is "GB" only — remember the quotes *)

Go get a cup of coffee. When you come back look at the resulting screen output.
You should get

test #01 → True
test #02 → True
etc.

The command *NCGBTestCompare* has created answer files and compared them to the old reference files.

Creating Reference Answers

Maybe you do not like the reference results we provided to you (some people are not very grateful). To create a new set of reference outputs stay in the *Testing* directory. Make sure you have lots of test problem files in the *TestProblems* directory; say c01.data.m thru c78.data.m.

Edit 2 lines in the file

init.TestReference.m

to put in the path to the version of NCGB that you want as a reference.

Copy the file

init.TestReferences.m to the file **init.m**

Run **Mathematica**
Load **NCGBTestCreate.m**
Run

NCGBTestCreate[Integer1, Integer2]

This is beginning to look familiar. From here on proceed as you did in creating test files. This creates a collection of files in the directory *TestReference.*

Index

Chapter 31

References

[BGK] H. Bart, I. Gohberg and M. A. Kaashoek, *Minimal factorization of matrix and operator functions*, Birkhäuser, 1979.

[BW] T. Becker, V. Weispfenning, *Gröbner Basis: A Computational Approach to Commutative Algebra*, Springer-Verlag, Graduate Texts in Mathematics, v. 41, 1993.

[CLS] D. Cox, J. Little, D. O'Shea, *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*, Springer-Verlag, Undergraduate Texts in Mathematics, 1992.

[DGKF] J. C. Doyle, K. Glover, P. P. Khargonekar and B. A. Francis, "State–space solutions to standard H_2 and H_∞ control problems," IEEE Trans. Auto. Control 34 (1989), 831–847.

[FMora] F. Mora, "Gröbner Bases for Non-commutative Polynomial Rings," Lecture Notes in Computer Science, number 229 (1986) pp 353-362.

[G] Ed Green, "An introduction to noncommutative Gröbner bases", Computational algebra, Lecture Notes in Pure and Appl. Math., 151 (1993), pp. 167–190.

[GHK] E.L. Green and L.S. Heath and B.J. Keller, "Opal: A system for computing noncommutative Gröbner bases", Eighth International Conference on Rewriting Techniques and Applications (RTA-97), LNCS# 1232, Springer-Verlag, 1997, pp 331-334.

[HS] J. W. Helton and M. Stankus, "Computer Assistance for Discovering Formulas in System Engineering and Operator Theory", Journal of Functional Analysis 161 (1999), pp. 289—368.

[HSW] J. W. Helton, M. Stankus and J. J. Wavrik, "Computer simplification of formulas in linear systems theory," IEEE Transactions on Automatic Control 43 (1998), pp. 302—314.

[HW] J. W. Helton and J. J. Wavrik "Rules for Computer Simplification of the formulas in operator model theory and linear systems," Operator Theory: Advances and Applications 73 (1994), pp. 325—354.

[NCA] J.W. Helton, R.L. Miller and M. Stankus, "NCAlgebra: A Mathematica Package for Doing Non Commuting Algebra," available from http://math.ucsd.edu/˜ncalg

[NCGBDoc] J.W. Helton and M. Stankus, "NonCommutative Gröbner Basis Package," available from http://math.ucsd.edu/˜ncalg

[TMora] T. Mora, "An introduction to commutative and noncommutative Gröbner Bases," Theoretical Computer Science, Nov 7,1994, vol. 134 N1:131-173.

Part VII

DEMOS - ONLY ON THE WEB

Chapter 32

Singular Perturbation Demo

Part VIII

DETAILS AND OLD OR NEW COMMANDS -ONLY ON THE WEB

Chapter 33

History of the Production of a GB and Playing By Numbers

This chapter contains two advanced topics whose relationship is more technical than conceptual.

33.1 Play By Numbers

The Mathematica code and the C++ code both attach numbers to the polynomials which occur in the running of the GB algorithm. While these numbers are not externally visible at least in the commands described so far, they can be accessed by the user and are quite useful. One feature is that they can save typing time for the user who chooses to select or deselect relations in later runs of *NCMakeGB*. Also, if one is running the C++ version of the code, this will save considerable computer time because the C++ version of the code only has to send a number rather than the full polynomial to the Mathematica session. This time can be very significant when the polynomial has a large number of terms. See also section 33.2. Recall the Option *ReturnRelationsToMma* → False for NCMakeGB stops the partial GB calculated by the C++ program from transferring the answer back to Mathematica. This is the typical prelude to a "play by numbers " session.

Recall the option *ReturnRelationsToMma*→ *False* for *NCMakeGB* stops the results from the *NCMakeGB* command from being returned to Mathematica. This is typically the first step in "playing by numbers".

33.1.1 WhatAreGBNumbers[]

Aliases: **None**

Description: *WhatAreGBNumbers[]* returns a list of numbers. These numbers correspond to the elements of *WhatIsHistory* which determine what the ending relations are. If one computes *Map[(#[[2]])&, WhatIsHistory[WhatAreGBNumbers[]]]* then one gets the same result as *WhatIsPartialGB[]*

Arguments: None

Comments / Limitations: Not available before NCAlgebra 1.2

33.1.2 WhatAreNumbers[]

Aliases: **None**

Description: *WhatAreNumbers[]* returns a list of numbers. These numbers correspond to all of the polynomials which were used and retained inside the C++ code. One needs this command to make sense of the *WhatIsHistory* output. No argument is included because it can be applied only to the previous GB run.

Arguments: None

Comments / Limitations: Not available before NCAlgebra 1.2

33.1.3 WhatIsPartialGB[aListOfIntegers]

Aliases: **None**

Description: *WhatIsPartialGB[aListOfIntegers]* returns the polynomials corresponding to the entries in *aListOfIntegers*. The command *WhatIsPartialGB[]* is equivalent to *WhatIsPartialGB[WhatAreGBNumbers[]]*.

Arguments: *aListOfIntegers* is a list of natural numbers.

Comments / Limitations: Not available before NCAlgebra 1.2. The list of integers does not have to be a subset of the integers in *WhatAreGBNumbers[]*, it can also be a subset of the list of integers in *WhatAreNumbers[]*. In plain english this says that $WhatIsPartialGB$ can retrieve any polynomial which has occurred in the course of running the last call to *NCMakeGB* (not just the output of *NCMakeGB*).

33.1.4 NumbersFromHistory[aPolynomial,history]

Aliases: **None**

Description: *NumbersFromHistory[aPolynomial,history]* returns all numbers n such that one of the elements of the list *history* has the form $\{n, aPolynomial, whatever, whatever2\}$ where we do not care what the value of whatever or whatever2 is. Usually this function will return only a list with one integer in it.

Arguments: *aPolynomial* is a polynomial. *history* is the output from the command *WhatIsHistory* (subsection 33.2.1).

Comments / Limitations:

33.2 History of the production of a GB

33.2.1 WhatIsHistory[aListOfIntegers]

Aliases: **None**

Description: Applies to any List of Integers subset of the output of WhatAreNumbers[]. *WhatIsHistory[WhatAreNumbers[]]* lists the tree which produced the elements of the previous GB run. Also possible is *WhatIsHistory[2,6,7]* which will produce only relation numbers 2, 6, 7 and describe their immediate ancestors. It is handy if you already know that you are interested in 2, 6, 7.

Arguments: *aListOfIntegers* is a list of natural numbers.

Comments / Limitations: WARNING: The output to the command *WhatIsHistory* changes after every call to *NCMakeGB*. The command *NCMakeGB* is called during the run of the *NCProcess* commands (subsections 18.2.1) and the run of any of the variants of the *SmallBasis* command (subsections 24.1.1 and 24.1.2).

33.2.2 WhatIsKludgeHistory[aListOfIntegers]

Aliases: **None**

Description: *WhatIsKludgeHistory[aListOfIntegers]* returns the same output as *WhatIsHistory[aListOfIntegers]* except that second entry which is a polynomial in *WhatIsKludgeHistory* is much shorter (and is not a relation in the ideal) than *WhatIsHistory.* The point is to save time. Kludge History is generated inside of C++ and transferring it back into Mathematica takes much less time than the full history because transferring long polynomials into Mathematica takes a long time. Every polynomial in KludgeHistory lies in the same category as the polynomial it replaces in history. Thus, *RemoveRedundentUseNumbers* applies to *KludgeHistory* returns the same relation numbers as *RemoveRedundentUseNumbers* applied to history (if one uses the option).???

Arguments: None

Comments / Limitations: WARNING: The output to the command *WhatIsHistory* changes after every call to *NCMakeGB*. The command *NCMakeGB* is called during the run of the *Spreadsheet* commands (subsections 18.2.1) and the run of any of the variants of the *SmallBasis* command (subsections 24.1.1 and 24.1.2).

33.2.3 More on the History of how NCMakeGB produced its answer

We now continue with the demo from Subsection

```
In[14]:= WhatAreNumbers[]
Out[14]= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

In[15]:= ColumnForm[ WhatIsHistory[Out[14]] ]
Out[15]= {1, x ** x -> a, {0, 0}, {}}
         {2, x ** y -> a, {0, 0}, {}}
         {3, x ** y -> b, {0, 0}, {}}
         {4, x ** x ** x -> b, {0, 0}, {}}
         {5, x ** a -> a ** x, {1, 1}, {}}
         {6, a ** y -> a ** x, {1, 2}, {5}}
         {7, x ** b -> a ** x, {1, 3}, {6}}
         {8, a ** x -> b, {1, 4}, {}}
         {9, a ** a -> b, {1, 4}, {1, 7, 8}}
         {10, b ** x -> b, {4, 1}, {1, 9}}
         {11, b -> a, {2, 3}, {}}
```

We now describe what the above output means. Notice that each line is a number followed by a replacement rule followed by a pair of numbers followed by a list of zero or more numbers. The tuples mean
{relation number, relation, the 2 parents of thr relation, which rules relations were applied to the S-polynomial}

For example, we say that $x**x \to a$ is the first replacement rule, $x**y \to a$ is the second replacement rule,etc. The third entry is the two parents of rule There are two cases:

(Case: the third component is the pair $\{0, 0\}$) When the pair $\{0, 0\}$ appears as the third component of list, the relations was provided as input to the NCMakeGB. In this case, the fourth list is always empty.

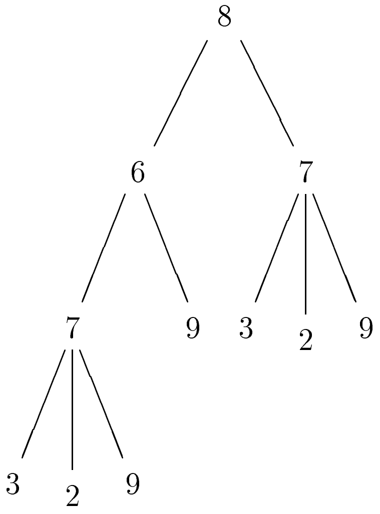
(Case: the third component is the pair $\{a, b\}$ and a and b are positive) When the pair $\{a, b\}$ appears as the third component of list, then the first step in constructing this rule comes from taking the S-polynomial of the ath and bth replacement rules. The last list of numbers indicates which replacement rules were applied to that S-polynomial in order to produce the relation.

(Case: the third component is the pair $\{-a, -a\}$ and a is positive) The relation was generated using CleanUpBasis using the ath relation. The last list of numbers indicates which replacement rules were applied to this ath replacement rule to generate the present rule.

Note that one can use history to do two things. The first thing is to see how relations are derived. The second is to find a smaller subset of the relations mentioned which will generate the same ideal. Note that in the above example, $x**a - a**x$ lies in the ideal generated by $x**x - a$ can be seen by referring to only the above history. Also, note that one can see that $a**y - a**x$ lies in the ideal generated by $\{x**x - a$, $x**y - a$ and $x**a - a**x\}$. This will prove very helpful in finding a subset of a particular set of a generated relations R which generates the same ideal as R. (See Section 34).

33.2.4 The DAG associated with a History

Consider the following tree on the nodes $\{2, 3, 6, 7, 8, 9\}$.



Let us suppose that this tree represents the history corresponding to $\{p_2, p_3, p_6, p_7, p_8, p_9\}$. That is, p_8 is generated as an s-polynomial from p_6 and p_7, p_6 is generated as an S-polynomial from p_7 and p_9 and p_7 is generated as a reduced S-polynomial. The polynomial generated is from p_3 and p_9 and a reduction step using p_2 was used. In the notation of the history output, this would take the form

```
{
{2,p_2,{0,0},{}},
{3,p_3,{0,0},{}},
{6,p_6,{7,9},{}},
{7,p_7,{3,9},{2}},
{8,p_8,{6,7},{}},
{9,p_9,{0,0},{}},
}
```

One sees from the picture that

(1) $p_8 \in \langle p_6, p_7 \rangle$

(2) $p_6 \in \langle p_7, p_9 \rangle$

(3) $p_6 \in \langle p_3, p_2, p_9 \rangle$

(4) $p_7 \in \langle p_3, p_2, p_9 \rangle$

(5) $p_8 \in \langle p_6, p_3, p_2, p_9 \rangle$

(6) $p_8 \in \langle p_7, p_9 \rangle$

(7) $p_8 \in \langle p_7, p_3, p_2, p_9 \rangle$

(8) $p_8 \in \langle p_3, p_2, p_9 \rangle$

Chapter 34

Commands for Making Small Bases for Ideals: Remove Redundant

A Gröbner Basis can be infinite. Even when a Gröbner Basis is finite, it can be very large and therefore difficult for a person to digest. One often finds that there are many relations which are generated which do not enhance our understanding of the mathematics. In many cases we want a basis for an ideal which is minimal (i.e., has smallest cardinality) or which is minimal as a subset of a given basis. We, therefore, find it helpful to take a list of rules which are generated by the GB algorithm and make them smaller. Consider the following example.

Example 34.1 *A GB generated by the set* $\{PTP - TP, P^2 - P\}$ *is the set* $\{PT^nP - T^nP \ : n \geq 1\} \cup \{P^2 - P\}$ *regardless of the term order used. No smaller GB exists.*

Here just two relations generate infinitely many. One way to view this example is that the computer discovers that if a subspace (represented by $\mathrm{ran}(P)$ in the computation) is invariant for a linear transformation T, then it is invariant for T^n for every $n \geq 1$.

The GB algorithms tries to generate this infinite set of relations and at any time has generated a finite subset of them. When we are trying to discover a theorem or elegant formulas, often these relations having higher powers are irrelevant and clutter the spreadsheet which merely serves to confuse the user.

This introduces the next topic which is shrinking a set of relations to eliminate redundancy. Our desire is to take the generated basis and to remove mathematical redundancy from the generating set without destroying the information which was gained while running the GB algorithm.

We have two lines of attack on this problem. In this chapter we describe one which is embodied in the command *RemoveRedundant*. In Chapter 24 we describe another approach.

34.1 Removing excess relations

34.1.1 Introduction

Our first line of attack works only when the set of relations under consideration has been generated by *NCMakeGB* and then only either

(1) The history was recorded right after using *NCMakeGB* via the *WhatIsHistory* command. OR

(2) There have been no calls to *NCMakeGB* (either directly or indirectly) since the set of relations was generated.

This approach is very fast.

If one has created a set of relations G by using *NCMakeGB*, then one can use history (gotten by the *WhatIsHistory* command — see 33.2.1 recorded during the run of *NCMakeGB* to find a smaller subset of G which generates the same basis. The essential idea is to take the history from the previous run of *NCMakeGB* and use it to construct a directed acyclic graph (DAG) representing the information. Then a fast combinatorial DAG search can be done.

Note that the *WhatIsHistory* command only records one way in which a certain relation is generated. For example, suppose that p_5 is an S-polynomial of p_1 and p_2 and p_5 is not reduced. It may also be the case the p_5 is an S-polynomial of p_3 and p_4 and p_5 is not reduced. If the program computed p_5 by taking the S-ppolynomials of p_1 and p_2 first, then the fact that p_5 is an S-polynomial of p_3 and p_4 will not be recorded.

34.1.2 RemoveRedundant[]

Aliases: **None**

Description: *RemoveRedundant[]* calls *RemoveRedundant[aListOfPolynomials,history]* with the default correct choices for *aListOfPolynomials* and *history.*

Arguments: None

Comments / Limitations: WARNING: The execution of this command depends upon the last *NCMakeGB* run which was executed. Some commands call *NCMakeGB* as they run such as *SpreadSheet* (subsection 18.2.1) and *SmallBasis* 24.1.1.

34.1.3 RemoveRedundant[aListOfPolynomials,history]

Aliases: **None**

Description: *RemoveRedundant*[*aListOfPolynomials*, *history*] generates a subset P of *aListOfPolynomials* which generates the same ideal as *aListOfPolynomials*. We conjecture that this is a smallest such set whose descendents in the history DAG contain *aListOfPolynomials*. We conjecture that this property charachterizes P uniquely. This command is faster while maybe not as powerful at reducing as *ShrinkBasis* since this command relies on the history of how the GB was generated and it just extracts key ancestors. That is why the list history is included to tell the routine how the various polynomials were generated. History must have been recorded using the *WhatIsHistory* command from the run of GB algorithm which produced *aListOfPolynomials*.

Arguments: *history* is an output from the *WhatIsHistory* command. *aListOfPolynomials* is a list of polynomials.

Comments / Limitations: WARNING: The history one gets from the *WhatIsHistory* command comes from the last run of the *NCMakeGB* command. The *NCMakeGB* command is called by several other commands such as *NCProcess* (subsection 18.2.1 and *SmallBasis* and its variants 24.1.1.

34.1.4 RemoveRedundentByCategory[]

Aliases: **None**

Description: *RemoveRedundentByCategory[]* calls *RemoveRedundentByCategory[aListOfPolynomials, history]* with the default correct choices for *aListOfPolynomials* and *history.*

Arguments: None

Comments / Limitations: It is wise to execute *WhatIsHistory* to set *history* right after you call *NCMakeGB*. WARNING: This command executes on the last *NCMakeGB* run which was executed. Some commands call *NCMakeGB* as they run, like *SpreadSheet* (subsection 18.2.1)

34.1.5 RemoveRedundentByCategory[aListOfPolynomials, history]

Aliases: **None**

Description: *RemoveRedundentByCategory[aListOfPolynomials,history]* generates a subset P of *aListOfPolynomials* which generates the same ideal as *aListOfPolynomials.* It is best thought of in the context of *RegularOutput* or a spreadsheet. Recall?? the digested relations are the polynomials involving no unknowns, single variable expressions, together with user selected relations. Let the digested relations from the list *aListOfPolynomials* be called D. Suppose we denote each category of undigested polynomials by X_j where j runs from 1 to the number of these categories. Then *RemoveRedundentByCategory[aListOfPolynomials, history]* is the union over j of *RemoveRedundent[Union[D,* X_j*], history].*

Arguments: *aListOfPolynomials* is a list of polynomials. *history* is a list from the command *WhatIsHistory.*

Comments / Limitations: It is wise to execute *WhatIsHistory* to set *history* right after you call *NCMakeGB*. WARNING: The history one gets from the *WhatIsHistory* command comes from the last run of the *NCMakeGB* command. The *NCMakeGB* command is called by several other commands such as *NCProcess* (subsection 18.2.1).

34.2 Discussion of RemoveRedundent command

RemoveRedundent requires a history and a list of polynomials. *NCMakeGB* records part of what it is doing during its computation and that can be used to determine some facts about ideal membership. This history can be though of as a directed acyclic graph (abbreviated as dag)[1]. The the list of polynomials is a subset V of all of the nodes and the output of *RemoveRedundent* is another set R of nodes v in V. Let $\surd$ denote a connected path which runs from v backward in time until it reaches a leaf. Now $\mathcal{P}$ ends at v in the set V and may as we go backwards along it in time leave V then come back etc. Let $v(\mathcal{P})$ denote the earliest node of $\mathcal{P}$ in V. It belongs to R. Indeed, $R = \{v(\mathcal{P})$ for all maximal connected forward flowing paths ending in $V\}$. For example, if some path $\mathcal{P}$ ends in v and the node immediately before v is not in V, then v is in R. Now we give more formal statements and about how the *RemoveRedundent* algorithm works.

Let $p_1, \ldots, p_n \in K[x_1, \ldots, x_n]$. Let T be a tree on the nodes $\{1, \ldots, n\}$. We say that the tree represents the development of $p_1, \ldots, p_n$ if it is the case that for every $j = 1, \ldots, n$ either

(1) j is a leaf of the tree OR

(2) there exists a S-polynomial s coming from p_a and p_b and $k_1, \ldots, k_r \in \{1, \ldots, n\}$ such that

  (a) $\{a, b, k_1, k_2, \ldots, k_r\}$ is the set of children of j in the tree T, and

  (b) there exist polynomials $q_1, \ldots, q_r$ such that $s \rightarrow^{p_{k_1}} q_1 \rightarrow^{p_{k_2}} \cdots \rightarrow^{p_{k_r}} q_{k_r} = p_j$.

  OR

(3) there exist a and $\{k_1, \ldots, k_r\}$ such that $\{a, k_1, \ldots, k_r\}$ are the children of j and there exist polynomials $q_1, \ldots, q_r$ such that p_a reduces to q_1 by applying the rule associated to the polynomial p_{k_1}, q_1 reduces to q_2 by applying the rule associated to the polynomial p_{k_2}, …, q_{r-1} reduces to q_r by applying the rule associated to the polynomial p_{k_r} such that $p_a \rightarrow_{p_{k_1}} q_1 \rightarrow^{p_{k_2}} \cdots \rightarrow_{p_{k_r}} q_r = p_j$.

Now suppose that T is a tree on $\{1, \ldots, n\}$ and it represents the development of $p_1, \ldots, p_n$. If j is the root of the tree and $k_1, \ldots, k_r$ are the leafs of T, then p_j lies in the ideal generated by $p_{k_1}, \ldots, p_{k_r}$. In fact, p_a lies in the ideal generated by $p_{k_1}, \ldots, p_{k_r}$ for every a such that both $1 \leq a \leq n$.

Now, if a tree T represents the development of $p_1, \ldots, p_n$ and one chooses a subset $\{k_1, \ldots, k_r\} \in \{1, \ldots, n\}$, then one can consider the largest subgraph of T for which each node k_ℓ has no edge leading from it. Each connected component of this subgraph will be a tree. Let T_o be such a connected component of T and $\{\ell_1, \ldots, \ell_w\}$ be the nodes of T_o. It is clear that the tree T_o represents the development of $p_{\ell_1}, \ldots, p_{\ell_w}$.

This is the key to the function RemoveRedundent. This function takes a tree T which represents the development of $p_1, \ldots, p_n$ and a list of nodes $\{\ell_1, \ldots, \ell_w\}$. The algorithm proceeds as follows.

```
Let $result = \{\ell_1, \ldots, \ell_w\}$;
Let $unchanged = False$;
```

---

[1] A common and canonical example of a dag is a tree

```
while(unchanged)
    unchanged = False;
    temp = result;
    while($unchanged$ is False and $temp$ is not the empty set)
        Pick $\ell \in temp$;
        Let $temp = temp \setminus \{\ell\}$;
        More here ? MARK whats this mean
    end while
end while
```

34.3 Examples

This shrinking is done using the commands `RemoveRedundent` and `SmallBasis`.

34.4 First Example

For example, after loading the files *NCGB.m*, *SmallBasis3.m* (§24.1.1) and *RemoveRedundent.m* (§34.1.2), we can execute the commands to compute a subset of a Gröbner Basis for the set of relations $\{p**p - p, p**a**p - a**p\}$:

```
In[2]:= SetKnowns[a,p]
In[3]:= {p**p-p,p**a**p - a**p}
Out[3]= {-p + p ** p, -a ** p + p ** a ** p}
In[4]:= NCMakeGB[%,4]

Out[4]= {-p + p ** p, -a ** p + p ** a ** p, -a ** a ** p + p ** a ** a ** p,
>    -a ** a ** a ** p + p ** a ** a ** a ** p,
>    -a ** a ** a ** a ** p + p ** a ** a ** a ** a ** p,
>    -a ** a ** a ** a ** a ** p + p ** a ** a ** a ** a ** a ** p,
>    -a ** a ** a ** a ** a ** a ** p + p ** a ** a ** a ** a ** a ** a ** p,
>    -a ** a ** a ** a ** a ** a ** a ** p +
>     p ** a ** a ** a ** a ** a ** a ** a ** p,
>    -a ** a ** a ** a ** a ** a ** a ** a ** p +
>     p ** a ** a ** a ** a ** a ** a ** a ** a ** p,
>    -a ** a ** a ** a ** a ** a ** a ** a ** a ** p +
>     p ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p,
>    -a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p +
>     p ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p,
>    -a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p +
>     p ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p,
>    -a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p +
>     p ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p,
>    -a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p +
>     p ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p\
>     , -a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a **
>       a ** p + p ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a **
```

```
>      a ** a ** a ** p, -a ** a ** a ** a ** a ** a ** a ** a ** a ** a **
>       a ** a ** a ** a ** a ** p +
>     p ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a **
>      a ** a ** p, -a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a **
>       a ** a ** a ** a ** a ** p +
>     p ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a **
>      a ** a ** a ** p}
```

The command `SmallBasis` takes this (or any) set of relations and shrinks it down to a smaller set of relations which generate the same ideal. One must have a monomial order set because *SmallBasis* (§24.1.1) calls *NCMakeGB*. `SmallBasis` returns a subset of the original set of relations. In the example below the `SmallBasis` command shows that the ideal generated by the set `Out[4]` equals the ideal generated by $\{-p + p**p, -a**p + p**a**p\}$. [2]

```
In[5]:= SmallBasis[%4,{},4];
Out[5]= {-p + p ** p, -a ** p + p ** a ** p}
```

Unfortunately in larger examples, `SmallBasis` is very slow which prompts the development of a more specialized command `RemoveRedundant`. This can be run only after `NCMakeGB` has been run because it uses the history of how the GB was produced. This history is equivalent to a tree which tells what relation came from what other relations `RemoveRedundant` uses only tree search methods so it is faster than `SmallBasis`.

As one sees below the information required for `RemoveRedundant` is a subset of the last GB produced in your session.

Before calling `RemoveRedundent`, one must acquire the history of the last GB produced in your section. This takes 2 commands which we now illustrate and which we explain afterward.

```
In[5]:= WhatAreNumbers[]
Out[5]= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17}
In[6]:= WhatIsHistory[%]

Out[6]= {{1, p ** p -> p, {0, 0}, {}},
>    {2, p ** a ** p -> a ** p, {0, 0}, {}},
>    {3, p ** a ** a ** p -> a ** a ** p, {2, 2}, {2}},
>    {4, p ** a ** a ** a ** p -> a ** a ** a ** p, {3, 2}, {2}},
>    {5, p ** a ** a ** a ** a ** p -> a ** a ** a ** a ** p, {3, 3}, {3}},
>    {6, p ** a ** a ** a ** a ** a ** p -> a ** a ** a ** a ** a ** p,
>     {5, 2}, {2}}, {7, p ** a ** a ** a ** a ** a ** a ** p ->
>      a ** a ** a ** a ** a ** a ** p, {5, 3}, {3}},
>    {8, p ** a ** a ** a ** a ** a ** a ** a ** p ->
>      a ** a ** a ** a ** a ** a ** a ** p, {5, 4}, {4}},
>    {9, p ** a ** a ** a ** a ** a ** a ** a ** a ** p ->
>      a ** a ** a ** a ** a ** a ** a ** a ** p, {5, 5}, {5}},
>    {10, p ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p ->
>      a ** a ** a ** a ** a ** a ** a ** a ** a ** p, {9, 2}, {2}},
>    {11, p ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p ->
```

---

[2] It takes 113 seconds. The present implementation of the code involves alot of interaction with Mathematica. We expect future versions of the code to be faster.

```
>      a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p, {9, 3}, {3}},
>    {12, p ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p ->
>      a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p, {9, 4}, {4}}\
>     , {13, p ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a **
>       p -> a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p,
>     {9, 5}, {5}}, {14, p ** a ** a ** a ** a ** a ** a ** a ** a ** a **
>       a ** a ** a ** a ** p ->
>      a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p,
>     {9, 6}, {6}}, {15, p ** a ** a ** a ** a ** a ** a ** a ** a ** a **
>       a ** a ** a ** a ** a ** p ->
>      a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** p
>      , {9, 7}, {7}}, {16, p ** a ** a ** a ** a ** a ** a ** a ** a ** a **
>       a ** a ** a ** a ** a ** a ** p ->
>      a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a **
>       a ** p, {9, 8}, {8}}, {17,
>     p ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a **
>       a ** a ** a ** p ->
>      a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a ** a **
>       a ** a ** p, {9, 9}, {9}}}
```

The call to `RemoveRedundent` is:

```
In[7]:= RemoveRedundent[Out[4],Out[6]];
Out[7]= {-p + p ** p, -a ** p + p ** a ** p}
```

The first command recalls the number associated to all of the relations which occurred during the previous run of Mora's algorithm. The second command gives the ancestory and other information related to relations 1, 2, ... One could have used any list of numbers (between 1 and 17) as the argument to the `WhatIsHistory` command and obtained only the history of those relations.

34.5 Second Example

As a second example, after loading the files *NCGB.m*, *SmallBasis3.m* and *RemoveRedundent.m*, we can execute the following commands to compute a Gröbner Basis for the set of relations $\{x^2 - a, x^3 - b\}$:

```
In[3]:= SetKnowns[a,b,x]

In[4]:= NCMakeGB[{x**x-a,x**x**x-b},10]
Out[4]= {-a + x ** x, -b + a ** x, -b + x ** a, -a ** a + b ** x,
>    -a ** b + b ** a, -a ** a + x ** b, -b ** b + a ** a ** a}
```

Now, one might want to find a smaller generating set for the ideal specified above. The following command does that and took 9 seconds using the C++ version of the code.

```
In[5]:= SmallBasis[%4,{},3]
Out[5]= {a ** x -> b, x ** x -> a}
```

Alternatively, one can use the command `RemoveRedundent` to generate the same result in 2 seconds. There are two steps which need to be performed before calling `RemoveRedundent`.

```
In[6]:= WhatAreNumbers[]
Out[6]= {1, 2, 3, 4, 5, 6, 7}
In[7]:= WhatIsHistory[%]

Out[7]= {{1, a ** x -> b, {0, 0}, {}}, {2, x ** x -> a, {0, 0}, {}},
>    {3, b ** x -> a ** a, {1, 2}, {}}, {4, x ** a -> b, {2, 2}, {1}},
>    {5, x ** b -> a ** a, {4, 1}, {3}}, {6, b ** a -> a ** b, {3, 2}, {1}},
>    {7, a ** a ** a -> b ** b, {3, 4}, {}}}
```

The call to RemoveRedundent is:

```
In[8]:= RemoveRedundent[%4,Out[12]]
Out[8]= {a ** x -> b, x ** x -> a}
```

34.6 Smaller Bases and the Spreadsheet command

Here is a session which does roughly what the spreadsheet command does. For more detail see Chapter 18

```
In[11]:=NCMakeGB[FAC,2];
In[11]:=SmallBasisByCategory[RemoveRedundant[%] ];
```

The next command tries to see if any of the undigested relations can be made simpler using the digested relations

```
In[12]:=NCSimplifyAll[%11,  DigestedRelations[%11] ];
```

Finally we output the result to the file “SecondOutputForDemo”??.

```
In[13]:=RegularOutput[%12, "SecondOutputForDemo"];
```

Now we return to the strategy demos of Chapter 18.

Inserting the command `RemoveRedundant` inside of small basis may change the answer, but it yields the same answer as `SmallBasis` would have given with one particular order on the set of relations given as input. All this assumes that the number of iterations is large. Inserting `RemoveRedundant` saves much computer time.

Chapter 35

NCXFindChangeOfVariables: The Long Description

The main purpose of `NCXFindChangeOfVariables` is to take some list of polynomials, probably the result running `NCProcess` on some (possibly large) problem and to find a motivated unknown which simplifies the problem. A motivated unknown M is a polynomial which, when substituted into some polynomial which motivates it, produces a polynomial in only one unknown, namely M. Furthermore, we want M to be nontrivial in the sense that given a polynomial P in knowns and unknowns, $M \neq aP + b$ where a and b are numbers.

`NCXFindChangeOfVariables` does not accomplish this, but it often finds solutions. The method is to find a number of candidates for motivated unknowns and then try them one by one until we find one which, in fact, works. Trying the candidates involves running a Grobner basis algorithm (`NCProcess`) on each possibility. To make things more efficient, there are a number of ways we may try to eliminate candidates or reorder them so that we find the motivated unknown faster.

35.1 Details of the Algorithm

35.1.1 Preparation

The first step is to put `motUnknown` and `Tp[motUnknown]` in the order. If the variable `motUnknown` is already in the order, then nothing is done. Otherwise, the two variables are put in the order in a graded piece just after the knowns. For example, if the old order is $a < b < c \ll x < x^T \ll y$ then the new order would be $a < b < c \ll motUnknown < motUnknown^T \ll x < x^T \ll y$.

The default options are also set at this stage.

35.1.2 Collect and extract

The next step is to go through the polynomials and to collect on knowns and on monomials consisting only of knowns. The NCAlgebra command `NCCollectOnVariables` looks for knowns and collects terms around them. For instance, given $xyax + zax$, `NCCollectOnVariables` would return $(xy + z)ax$. The first set of candidates for motivated unknowns is to extract what is collected to either side of the known. The program now keeps a list of pairs {P, C} where P is the polynomial on which `NCCollectOnVariables` was called and C is one of the things collected to either side of the knowns. The resulting list has all such pairs related to the given list of polynomials.

Note that if nothing can be collected, no entries are returned and our algorithm cannot find a motivated unknown.

35.1.3 Eliminate candidates which are too small

The next step is to look at the number of unknowns in the candidates for motivated unknowns and to try to eliminate some without running a Grobner basis algorithm. This step counts the number of unknowns in the candidate (C in the pair described above) and compares it to the number of unknowns in the polynomial that motivated it (P above). Since the idea is that the candidate C will reduce P to a function of one variable, we can eliminate the pair if C has less unknowns than P. This is exactly what this step does.

It also eliminates pairs where C is just one variable.

35.1.4 Eliminate purely numerical terms from candidates - Default is Off

Here we eliminate purely numerical terms from the candidates for motivated unknowns. Thus if our candidates starts as $xy + 1$ we instead take as our candidate xy. I'm not sure if this step is still necessary, but in the past there were difficulties matching when the candidate had a numerical term.

To turn off this step, redefine NCXKillConstantTerms to be the identity function, i.e. NCXKillConstantTerms[list_] := list.

35.1.5 Sort list of candidates by number of terms

Now we sort the candidates by their length, where by length we mean the number of terms in the polynomial. It generally turns out that the smallest polynomials are more likely to work, so by sorting in such a way that the polynomials with the least number of terms come first, we will probably find the motivated unknown (if one exists) earlier than if we had a random order.

35.1.6 Multiply through by monomials - Default is off

Sometimes it turns out that we need to find the motivated unknown, we actually need to multiply the polynomial P by some monomial on the left and/or right. Then this new polynomial will admit a motivated unknown. This step adds new pairs {P', C'} where P' is a LPR where P is from the original list and L and R are monomials. L and R are dtermined in the following way. Given a pair {P, C} from the original list we find all prefixes L of the leading term of C and all suffixes R of the leading term of C. Prefixes of a monomial M are monomials on the left of M and suffixes are monomials on the right. Thus the monomial xyz has prefixes x, xy, and xyz and has suffixes z, yz, and xyz. We add to our list pairs {LP, C}, {PR, C}, and {LPR, C}.

These candidate pairs are added at the end of the list so that they are tried after the candidates without multiplying through by monomials.

This option needs to be hcanged so that lists can be multiplied through solely on the left or solely on the right.

In order to turn this on, give the option `MultiplyByMonomials -> True`.

35.1.7 Run the Grobner basis algorithm

We now can step through out list and try each to see if the candidate is, in fact, a good motivated unknown. Given a pair {P, C} we run `NCProcess` on the union of the P, polynomials with 2 terms (these we shall think of as important polynomials since they include relations defining inverses and symmetry), and the rules $C \to motUnknown$ and $C^T \to motUnknown^T$. If it finds a motivated unknown that works (i.e. eliminates all other unknowns), then it stops and returns the pair {P,C}.

35.1.8 Options

Most of these steps can be eliminated by setting the appropriate option. See manual for details in setting options.

35.2 Finding Coefficients of Variables in a Polynomial

35.2.1 NCCoefficientList[Expression, aListOfIndeterminants]

Aliases: **None**

Description: This generalizes the Mathematica command `CoefficientList[Expression, aListOfIndeterminants]` to noncommutative algebras. There are many legitimate generalizations to the noncommuting case and we picked one here.The user can experiment to see if it is what he wants.

Arguments:

Comments / Limitations:

35.3 Main Change Of Variables Command

The main command is `NCXFindChangeOfVariables`. The general purpose of these commands is to produce 1-strategies from a given list of relations. That is, we would like to find a motivated unknown that eliminates all other unknowns from some equation in a nontrivial way. By nontrivial, we mean that the motivated unknown is not the entire given expression E or $aE + b$ where a and b are numbers.

35.3.1 NCXFindChangeOfVariables[aListofPolynomials, anInteger, aString, Options]

Aliases: **None**

Description: This command needs the monomial order to already be set. It then uses the ambient order in NCGB. `NCXFindChangeOfVariables[ aListOfPolynomials, anInteger, aString, Options]` takes a list of relations `aListOfPolynomials`, finds the candidates for motivated unknowns and then tries each one in NCProcess until it finds that all unknowns except the candidate have been eliminated (and hence would make a good motivated unknown). If it finds a motivated unknown (which absorbs all other unknowns) then it returns a list {`E, M`} where `M` is the motivated unknown and `E` is the expression which motivated it. Otherwise it returns False. It can also be made to return a list of outputs from the calls to `NCProcess`.

Arguments: aListofPolynomials is a list of polynomials, aString is a string for the beginning of the tex files produced by `NCProcess`, and anInteger is the number of iterations for `NCProcess`. The options are:

- $IncludeTranspose \to False$: This option adds the transpose of the candidate to the set of relations. The default (False) is not to add the transpose relation, while setting it to True will not add the transpose relation.
- $AllRelations \to False$: This option determines whether the Grobner Basis is computed using only the relation that motivated the candidate together with the candidate or if it uses all of the given relations plus the candidate. The default (False) only uses the polynomial that motivated the unknown plus all relations of length 2 (these we will consider "important" relations and include relations defining inverses and symmetry) while setting it to True uses all of the relations.
- $CountVariables \to True$: This option determines whether or not `NCXPossibleChangeOfVariables[ ]` eliminates the candidates which do not contail all of the unknowns present in the polynomial that motivated it (and thus the candidate cannot reduce that polynomial to a polynomial in one variable). The default (True) does eliminate these possibilities while setting it to False does not do this elimination.
- $MultiplyByMonomials \to False$: This option determines whether or not `NCXMultiplyByMonomials` is called, so that if no candidate works, it tries to multiply though by monomials on the left and/or right. The default (False) does no multiplying, while setting it to True tries multiplying through by monomials.
- $SortByTermLength \to True$: This option decides whether or not to sort the results of `NCXPossibleChangeOfVariables` by the length (number of terms) in the candidate

(shortest to longest). The default (True) does sort it so that it tries the shortest candidates first (since in practice the longer ones don't tend to work). Setting it to False does not do the sorting step.

- $NCProcessOptions \rightarrow \{SBByCat \rightarrow False, RR \rightarrow False\}$: This allows one to set additional options when `NCProcess` is run. The default is $\{SBByCat \rightarrow False, RR \rightarrow False\}$. A typical setting might be $NCProcessOptions- > \{SBByCat \rightarrow False, NCCV \rightarrow True\}$. list of the outputs from `NCProcess`.
- $StopIfFound \rightarrow True$: This option determines whether the program stops if a motivated unknown is found. The default (True) stops if a motivated unknown is found and returns only this pair. Setting this option to False runs all possibilities in `NCProcess` and returns all of the results (whether a motivated unknown is found or not).

Comments / Limitations: This procedure uses the ambient monomial order in NCGB. Furthermore, the monomial order is changed by this program. The variables `motUnknown` and `Tp[motUnknown]` are inserted in a graded piece between the current knowns and unknowns if these variables are not already present. This function runs `NCProcess` many times and therefore produces a number of tex files (actually, it produces exactly `Length[NCXMultiplyByMonomials[NCXPossibleChangeOfVariables[aListOfPolynomials, Options]]]` tex files). These files are named nameno# where name is the string given as an argument, no is added and # is a number. This function uses `NCProcess`, `NCXPossibleChangeOfVariables`, `NCXMultiplyByMonomials`.

The following command may also be useful since it gives a list of expressions, each of which is a possible new variable. It runs all steps above except the last two, that is, multiplying through by monomials and running the Grobner basis.

35.3.2 NCXPossibleChangeOfVariables[aListofPolynomials, Options]

Aliases: **None**

Description: `NCXPossibleChangeOfVariables[ aListOfPolynomials, Options]` takes a list of relations and looks for a good motivated unknown. It returns a list of pairs `{E, M}` where `E` is the expression which motivated the candidate `M` and `M` is the candidate for a motivated unknown.

Arguments: aListofPolynomials is a list of relations and options can be any of the following:

- $CountVariables \rightarrow True$: The CountVariables option counts to see if the candidate for motivated unknown has all of the variables in the expression which motivated it, and removes the entry from the list if it does not (and thus could not reduce the expression to one unknown). The default (True) is to eliminate these entries, while False will skip this step entirely.

- *RemoveNumbers* → *False*: The RemoveNumbers option decides whether or not to remove purely numerical terms from the candidates. If True, then these terms are removed. For instance, the candidate $xy+1$ becomes xy. If set to False, the candidate is not be changed.
- *SortByTermLength* → *True*: The SortByTermLength option decides whether or not to sort the results by the length (number of terms) of the candidate (with the shortest first). The default (True) does the sorting, while setting the option to False does not sort at all.

Comments / Limitations: The candidate for motivated unknown is found by first doing an `NCCollectOnVariables[ ]`, which collects around knowns, on each relation and then looking at what is found on either side of the knowns. These are the candidates for motivated unknowns. The next step is to eliminate candidates which do not contain all of the unknowns present in the expression which motivated them. This option can be turned off by *CountVariables* → *False*. The next step is to eliminate purely numerical terms from the candidates (so that you won't get an expression like $xy+1$ for a candidate, but xy instead). The final step is to sort the pairs by length (number of terms) of the motivated unknown, the shortest being first. This is done because long candidates usually do not eliminate all of the variables. This option can be turned off by *SortByTermLength* → *False*.

35.4 Less Valuable Change of Variables Commands

These commands are used by the above commands. They would not be of use to the average user.

35.4.1 NCXMultiplyByMonomials[aVerySpecialList]

Aliases: **None**

Description: `NCXMultiplyByMonomials[ aVerySpecialList]` takes a list like the one returned by `NCXPossibleChangeOfVariables` and returns the list appended (possibly) with new pairs which are multiplied through by certain monomials (prefixes and suffixes of the candidate for motivated unknown) on the left and/or right.

Arguments: `aVerySpecialList` is list of pairs of polynomials, so it looks like {{poly1, poly2}, ...,{poly3,poly4}} where poly1,..., poly4 are polynomials.

Comments / Limitations: A new pair is gotten from an old pair by looking at the candidate for motivated unknown and then multiplying by prefixes and suffixes of the candidate.

35.4.2 NCXAllPossibleChangeOfVariables[aListOfPolynomials]

Aliases: **None**

Description: `NCXAllPossibleChangeOfVariables[ aListOfPolynomials]` takes a list of polynomials and returns a list of pairs {P, C} where P is a polynomial from aListOfPolynomials and C is on the left or right side of a product of knowns inside P after P has been collected (with `NCCollectOnVariables`).

Arguments: aListOfPolynomials is a list of polynomial expressions.

Comments / Limitations: This procedrue uses the ambient order, so it must be set before use. This procedure returns a dumb set of candidates for motivated unknowns. `NCXPossibleChangeOfVari` uses it and returns a more intelligent list of candidates. Thus the average user would not find a need for this procedure.

Chapter 36

Representing Noncommutative Expressions with Commutative Ones.

36.0.3 NCXRepresent[aListOfExpressions, aListOfVariables, aListOfDims, aListOfFunctions, aListOfExtraRules]

Aliases: **none**

Description: NCXRepresent[aListOfExpressions,aListOfVariables, aListOfDims, aListOfFunctions,aListOfExraRules] replaces each occurrence of the variables in the list `aListOfVariables` with a matrix in commuting symbols of size specified in `aListOfDims` in the set of relations in the first arguement. The argument `aListOfDims` should be a list of pairs {`n1,n2`} specifying the number of rows and columns of the corresponding variable. The fourth arguement is best described by an example. If one has the variable x and also the indeterminate $F[x]$, then often one wants the $F[x]$ to be replaced by a symbolic matrix as well as x. If F appears in the fourth list, every occurrence of indeterminates $F[x], F[y]$ etc. will be replaced by matrices of the same size as x, y etc. with entries that look like $Fx11, \ldots, Fy11, \ldots$. This clearly is not appropriate for functions such as Aj and Tp denoting the adjoint and transpose of a symbol. But for other functions such as Inv this is quite necessary. The output of NCXRepresent is a matrix for each relation in `aListOfExpressions`.

The last (fifth) arguement is a sort of catch-all for unusual rules specific to the problem. The following are some favorites.

- $z \to IdentityMatrix[n]$. If some variable z is to be replaced by an identity matrix of some size then one puts this rule in `aListOfExtraRules` An important point: if constants appear in the list of relations, one MUST replace them with auxiliary variables and then use this last arguement to replace them with the appropriate multiple of the identity matrix.
- $x \to PrimeMat[\{n_1, n_2\}, k]$. This is an auxillary function built in to NCXRepresent that will replace a variables by a matrix consisting of distinct prime entries. To use this, one just puts the rule above in the list `aListOfExtraRules`. This will replace x with an $n_1 \times n_2$ matrix with prime entries beginning at the kth prime.

- Any other replacement rules may be listed in `aListOfExtraRules`, such as $tp \to Transpose$ or other hand-made rules replacing a variable by some specific matrix.

Arguments: aListOfVariables, aListOfDims, aListOfExpressions, aListOfFunctions, aListOfExtraRules

Comments / Limitations: If `aListOfExpressions` includes constant terms, one must first replace them with a variable and then use the optional list of rules to replace them with identity matrices of appropriate size. Mathematica does bad things to expressions that contain both a matrix and a constant, namely it adds the constant to each entry of the matrix. Any of the lists other than `aListOfExpressions` may be left as the empty set. A warning: if a variable appears only as the arguement in a function in `aListOfFunctions` one must still put the variable in the second arguement, and its size in the third. The reason is, its size must be specified somewhere for the function to work. `aListOfVariables` should include only those variables that are to be replaced by purely symbolic matrices, and should not appear in the left-hand side of a rule in `aListOfOptionalRules`.

Examples

```
 In[99]:= NCXRepresent[  {A**B}, { A, B}, { {2,1}, {1,1}  },{},{} ]

Out[99]:= {{{A11 B11},{A21 B11}}}
```

```
 In[100]:= NCXRepresent[  {A**tp[B]}, { A, B}, { {2,2}, {1,2} }, {tp},{} ] =

Out[100]:= {{{A11 B11+A12 B12},{A21 B11+A22 B12}}}
```

Part IX

DETAILS ON INSTALLATION AND MAINTENANCE - ONLY ON THE WEB

Chapter 37

NCAlgebra and NCGB Installation

This chapter gives more detail on some aspects of Installation and running NCGB, SYStems, OldMmaGB. These are kind of random remarks we wanted to hang onto but which are probably not too important.

37.1 Running NCAlgebra

See Chapter 2 for the basics. We have nothing to add to that here.

37.2 Running NCGB

See Chapter 2 for the basics. We remind the reader here: A UNIX session begins by

Getting in the NC directory, entering Mathematica and typing
In[**1**] :=<<SetNCPath.m

Then the file **NCGB.m** is used to load all necessary files into Mathematica. The command is:

```
In[2]:= Get["NCGB.m"]
```

There are two configurations of this file; it can load either NCAlgebra.m or shortNCAlgebra.m, depending on the user's needs. The file shortNCAlgebra.m loads a smaller segment of the complete package.

37.3 Running SYStems

First edit the SYStems.m file, whose function is to load in the raw material for your problem. Make sure it loads in what you need. The file SYStems.m is in the NCAlgebra directory. The first thing that the SYStems.m file does is to read in the NCAlgebra.m file and NCAlgebra should **NOT** be loaded either before or after getting the file SYStems.m. Thus a session begins by

Getting in the NC directory, entering Mathematica and typing
<<**SetNCPath.m**
<<**SYStems.m**

37.4 Running OldMmaGB (which we do not support)

The file OldMmaGB.m loads in old commands which we do not actively support as well as new untested ventures which may not work reliably and whose documentation, in NCOLDDOC, contains few complete english sentences.

A session begins by

Getting in the NC directory, entering Mathematica and typing

<<SetNCPath.m
<<OldMmaGB.m

Incidentally **OldMmaGB.m** is a file in the NC/NCAlgebra directory whose function is to load files found mostly in the NC/NCAlgebra/OldMmaGB directory.

37.5 Environment Settings

All variables beginning with the four characters NC are reserved for use by the NCAlgebra and NCGB development teams.

The user may set an NC variable to accomodate his/her needs. If the user does not assign a value to a NC variable, then it has no value until that variable is to be actually used. At that time, the code changes the value of the variable to a default.

37.5.1 NCAlgebra NC Variables

NCLoaded$NCAlgebra$ is used to ensure that NCAlgebra.m is not loaded twice. Every NCLoaded$*$ file will be used to avoid duplicate file Get's. This is similar to the #ifndef #define #endif scheme used in C++.

NCisCatching$ is reserved to attempt to do Mathematica exception handling via the Mathematica commands Catch and Throw. This is not implemented and is held for future development.

37.5.2 NCGB NC Variables

The NC variables which are presently part of NCGB are those that are part of NCAlgebra (see section 37.5.1) and the following variables.

```
$NC$Loaded$NCGB$
$NC$LongLoadTime$
$NC$Binary$Dir$
$NC$Binary$Name$
```

NCLoaded$NCGB$ is used to ensure that NCGB.m is not loaded twice. Every NCLoaded$*$ file will be used to avoid duplicate file Get's. This is similar to the #ifndef #define #endif scheme used in C++.

The NCLongLoadTime$ variable is True by default which means that all of NCAlgebra.m will be read in when NCGB.m is read in. If one sets it to False, **before** loading either NCAlgebra.m or NCGB.m then only part of NCAlgebra.m will be read in.

The variables NCBinaryDir and NCBinary$Name$ are used to specify which C++ executable file one is using. If one does not assign a value to either variable, then the value will be set in accordance with the variable $NCDir$ which is set in the file SetNCPath.m. The file SetNCPath.m was modified appropriately during installation.

37.6 How to set up if someone else compiled the code — UNIX

It is easy to bum off of a friend who is running NC on the same system.

37.6.1 When your system administrator installed the code

When your system administrator installed the code, he had several choices.

The first thing you should try is to start Mathematica by typing `math` and then type `<<SetNCPath.m`. If there are no error messages, then type `<<NCAlgebra.m`. Mathematica should respond with a sequence of statements indicating that various files are being loaded.

If Mathematica responds with `Get::noopen: Cannot open SetNCPath.m`, then additional steps are required as described below.

You need to determine where your system administrator installed NCAlgebra. The easiest way to do this is to ask your system administrator. If that is not easy and/or convenient, then you can type

```
cd  find / -name "NCAlgebra.m" -print
```

and look at the result. Note that this command will take some time and will access the hard drive quite a bit. If this is done on our system, then the result is the directory

```
/home/ncalg/NC/NCAlgebra/NCAlgebra.m.
```

The location of the installation in this case is `/home/ncalg/NC` .

Let us suppose that you know the location of the installation of NCAlgebra. Let us suppose that it is located in the directory `/usr/local/NC`.

Move into the directory where you want to use NCAlgebra (or create such a directory using the `mkdir` command). Create an `init.m` file with your favorite editor which contains the following lines:

```
AppendTo[$Path,"/usr/local/NC/"];
Get["SetNCPath.m"];
```

Type `math` and then type `<<NCAlgebra.m`. Then Mathematica responds with a sequence of statements indicating that files are being loaded.

37.6.2 When your friend installed the code

Suppose that your friend has installed the code and he is willing to let you use his version of the code (see (6)), then you can follow this procedure:

(1) Have your friend log into his account and change directories into the one which contains the NC code. Suppose for definitiveness that it is located in the directory `/home/JoeShmo/NC/`. In this case, you would type `cd /home/JoeShmo/NC`

(2) Type `chmod -R uog+r *`. This allows all of these files to be read by anyone. (These files are available through the web, so giving permission to read them should be OK with your friend.)

(3) Edit the file `SetNCPath.m`. Make sure that there is a command

```
$NCDir$ = "/home/JoeShmo/NC"
```

(4) Type `cd ../`

(5) Type `chmod u+rx NC`. This allows the directory to be readable by anyone. For some reason, some computer systems require that a directory be executable. I don't know why.

(6) Type `cd ..`

(7) Type `chmod u+rx JoeShmo`. This allows the directory to be readable by anyone. For some reason, some computer systems require that an directory be executable. I don't know why. Your friend may or may not be willing to take this step.

Now, follow the directions in Section 37.6.1.

37.7 Informing TEX about notebook.sty

Read this only if you have trouble with Mma's TeX conversion of notebooks. This has nothing to do with NCAlgebra, we just include it as a convenience.

Most special NCAlgebra and most special NCGB TEX definitions we use are in the file NCTeXForm.m. also TeXUtilities.m; see the section on TeX.

There is one place in the Mma's TeX setting in which explicit facts about the operating system are used. "Elusive" file notebook.sty must be in your TeX path. Here is how you find notebook.sty and get it in your path.

(1) Type in math, then type $Path and determine where your system administrator installed your copy of mathematica. When I did this, I got the output as shown below and I determined that mathematica was installed in the directory /opt/mathematica

```
{euler 190 :/home/ncalg/NC/NCAlgebra} math
Mathematica 3.0 for Solaris
Copyright 1988-96 Wolfram Research, Inc.
 -- Terminal graphics initialized --

In[1]:= $Path
```

```
Out[1]= {/home/ncalg/.Mathematica/3.0/Kernel, ., /home/ncalg,

>    /opt/mathematica/AddOns/StandardPackages,
>    /opt/mathematica/AddOns/StandardPackages/StartUp,
>    /opt/mathematica/AddOns/Applications,
>    /opt/mathematica/AddOns/ExtraPackages,
>    /opt/mathematica/SystemFiles/Graphics/Packages, /home/osiris/ncalg/NCDir}
```

(2) Now move into that directory. I typed `cd /opt/mathematica`.

(3) Now find the file notebook.sty. I did this by typing `find . -name "notebook.sty" -print` When I did this I got `./SystemFiles/IncludeFiles/notebook.sty`

(4) Now I changed my .cshrc file to add an entry to my TEXINPUTS UNIX environment variable. My .cshrc file has the line

```
setenv TEXINPUTS

".:/opt/mathematica/SystemFiles/IncludeFiles:/opt/tex/lib/texmf//"
```

in it. You may want to discuss this with your system administrator.

To run DoTeX if you really want to; UNIX only

DoTeX is old. We include this instruction for old users who cannot part with it. The file called NCTeX.process should be transfered to the bin directory before you use our TeX setting command DoTeX.

Chapter 38

Installing NCGB the Hard Way

In this section we face the task of compiling NCGB. If you are on a Solaris, Linux, or Windows system you do not have to, since we included a complied version in what you have downloaded.

In order to compile the C++ version of this package, one should obtain a copy of the GNU C++ compiler with version number at least 2.6.3.

38.1 GNU C++ Version $\geq$ 2.6.3

The freely-distributable GNU C++ compiler is available by anonymous ftp from prep.ai.mit.edu. The GNU C++ compiler is (1) freely-distributed, and (2) continually upgradable by the GNU organization. GNU is responsible for producing many high quality software programs for free, including the popular program GNU Emacs. The easiest way for you to install the compiler would be through your system administrator. You can see if you have the GNU compiler version 2.6.3 or better by typing the command 'g++ -v'. You should get something like this:

```
Reading specs from
/software/common/gnu/lib/gcc-lib/sparc-sun-sunos4.1.2/2.6.3/specs
gcc version 2.6.3
```

The version number must be 2.6.3 or higher[1]. If you do not have version 2.6.3 or better, talk to your system administrator.

38.1.1 Compiling the C++ part of NCGB

The file NC/NCGB contains a README file. This file describes how to compile the NCGB code. There are, in addition, README files in each subdirectory of NC/NCGB. In short, one wants to type `make p9c` in the NC/NCGB/Compile directory and copy and/or link (ln -s) the binary from NC/NCGB/Compile/dlink/p9c into NC/NCGB/Binary/p9c.

---

[1] We know that 2.6.0 will not work. We have not tried 2.6.1 or 2.6.2. We have tried 2.7.0 on an HP. Version 2.7.2 works on Suns

38.2 Running NCGB

This is described in the NCDOCUMENT but we repeat it here.

A session begins by

Getting in the NC directory, entering Mathematica and typing

<<**SetNCPath.m**

<< **NCGB.m**

38.2.1 Loading NCGB more quickly

When you load NCGB it automatically loads NCAlgebra. When you load NCAlgbebra you have a choice of loading many files or huge numbers of files. The default is the latter so you must sit and wait. If you get mad as hell and won't take it any more, stop. Before loading Mathematica or anything type

NCLongLoadTime$ =False

This sets an environment where the shorter version of NCAlgebra.m is loaded. Note NCLongLoadTime$ =True is the default.

Chapter 39

For the Home Team

39.1 How to make the PC version of NC

The NC.zip entire structure can be made from any person's account.

```
first use the command

"tar -cvf ~/NC.tar  *"

This will create the "tar"ed file in YOUR root directory.

Go to your root directory and make a new directory
ncgbWIN with

"mkdir NCWin"

and move NC.tar in to this directory.

"mv NC.tar NCWin/NC.tar"

change the current directory

"cd ncgbWin"

Untar the file with

"tar -xvf NC.tar"

This will recreate the NC structure.
```

run the shell script in this directory "makeWin.sh" in the ncgbWin directory. (This will make all the text files in subdirectories DOS (add a ^M))

go to NCWin/NCGB/MmaSource/" and edit the file

"NCGBPlatformSpecific.m "

Edit the file "TypesOfStrategyOutput.m" to take the "&" out of the xdvi command. Xdvi.

You will have to change the Binary Name and LaTeXCommand and dviCommand to reflect a common LaTex setup on the Windows.

go back to rootdirectory/NCWin/NCGB/Compile and remove the unnecessary object files

"/usr/bin/rm *.o"

Similarly go to rootdirectory/NCWin/NCGB/Binary and remove the UNIX executable.

"rm p9c"

return to rootdirectory/NCWin and

"rm NC.tar"

Now compress the NCAlgebra structure into a PC compatible form.

"zip -r NC.zip *"

ftp this onto your PC....

Make directory NC and decompress NC.zip in it.

Make directory NC/work and put a mathematica notebook

```
in it with the following:

Get["c:/NC/SetNCPath.m"];
Get["NCGB.m"];
ClearMonomialOrderAll[];
SetMonomialOrder[a,b];
NCProcess[{a**b**a**b + b**a**b**a}, 2 , "TestFile" ];
```

Part X

TRULY OLD MATERIAL - ONLY ON THE WEB

Chapter 40

An Old Example of Get Categories-Lots of info

This is a 1996 vintage example and we should have deleted it except that it contains a lot more info on how categories are labeled than we put in the official doucument. As an example, consider the following set of relations with the multigraded lex order $A < B < C0 < D < ACROSS < P1 < P2 << proj1 < proj2 << m1 < m2 << leftm1 < leftm2 << rightm1 < rightm2 << a < b < c < e < f < g$. This arose in the demo in Subsection 17.2. So we proceed assuming we have set the order. Suppose *eliminatedOriginal* has been set so that:

```
In[4]:= eliminatedOriginal
Out[4]= {c - C0 ** m1, g - C0 ** m2, -1 + leftm1 ** m1, leftm1 ** m2,
>    leftm2 ** m1, -1 + leftm2 ** m2, leftm1 ** A ** m2,
>    e - leftm2 ** A ** m2, b - leftm1 ** B, f - leftm2 ** B,
>    -leftm1 + rightm1, -leftm2 + rightm2, a - leftm1 ** A ** m1,
>    -leftm2 ** A ** m1 + leftm2 ** B ** C0 ** m1,
>    -1 + m1 ** leftm1 + m2 ** leftm2,
>    A ** m1 - B ** C0 ** m1 - m1 ** leftm1 ** A ** m1 +
>     m1 ** leftm1 ** B ** C0 ** m1,
>    -A ** A ** m1 + A ** B ** C0 ** m1 + B ** C0 ** A ** m1 -
>     B ** C0 ** B ** C0 ** m1 + m1 ** leftm1 ** A ** A ** m1 -
>     m1 ** leftm1 ** A ** B ** C0 ** m1 -
>     m1 ** leftm1 ** B ** C0 ** A ** m1 +
>     m1 ** leftm1 ** B ** C0 ** B ** C0 ** m1}
```

Before demonstrating the Category commands, we apply the *RegularOutput* command so the reader can see which categories occur. The following command creates a file *junk* which has a nicely formatted version of the information stored in *dummy* (see page 303).

```
In[6]:= RegularOutput[%4,"junk"]
Outputting results to the stream OutputStream["junk", 6]
Done outputting results to the stream OutputStream["junk", 6]

In[7]:= !!junk
THE ORDER IS NOW THE FOLLOWING:
A < B < C0 < D < ACROSS < P1 < P2 <<
proj1 < proj2 << m1 < m2 << leftm1 <
leftm2 << rightm1 < rightm2 << a < b < c
 < e < f < g
================================================
```

```
========= THE ALGORITHM HAS SOLVED FOR==========
================================================
THE FOLLOWING VARIABLES HAVE BEEN SOLVED FOR:
{c,g,e,b,f,rightm1,rightm2,a}

The corresponding rules are the following:
c->C0**m1
g->C0**m2
e->leftm2**A**m2
b->leftm1**B
f->leftm2**B
rightm1->leftm1
rightm2->leftm2
a->leftm1**A**m1



================================================
========= USER CREATIONS APPEAR BELOW ==========
================================================
================================================
========= UNDIGESTED RELATIONS APPEAR BELOW ====
================================================
THE FOLLOWING VARIABLES HAVE NOT BEEN SOLVED FOR:
{leftm1,leftm2,m1,m2}


The expressions with unknown variables {
leftm1,m1}
and knowns {A,B,C0}
---------------------------------------
leftm1**m1->1

m1**leftm1**B**C0**m1-> -A**m1   + B**C0
**m1 + m1**leftm1**A**m1

m1**leftm1**B**C0**B**C0**m1->A^2**m1 +
 -A**B**C0**m1   +  -B**C0**A**m1   + B
**C0**B**C0**m1 +  -m1**leftm1**A^2**m1
   + m1**leftm1**A**B**C0**m1 + m1**
leftm1**B**C0**A**m1


The expressions with unknown variables {
leftm1,m2}
and knowns {A}
---------------------------------------
leftm1**m2->0

leftm1**A**m2->0


The expressions with unknown variables {
leftm2,m1}
and knowns {A,B,C0}
---------------------------------------
leftm2**m1->0

leftm2**B**C0**m1->leftm2**A**m1


The expressions with unknown variables {
leftm2,m2}
and knowns {}
```

```
---------------------------------------
leftm2**m2->1


The expressions with unknown variables {
leftm1,leftm2,m1,m2}
and knowns {}
---------------------------------------
m2**leftm2->1 +  -m1**leftm1
```

Now we illustrate the *CreateCategories* command. One calls it on the above set of relations with the above order by typing:

```
In[5]:= CreateCategories[%,dummy]
Out[5]= dummy
```

The effect of the above command is to associate the following information to the symbol *dummy*.

```
In[6]:= ??dummy
                           <=== recall Mma does not produce an
                                output in response to ??
                                one gets a screen display (as follows)
Global‘dummy


dummy[2] = {{b, leftm1}, {c, m1}, {f, leftm2}, {g, m2}, {leftm1, m1},
   {leftm1, m2}, {leftm1, rightm1}, {leftm2, m1}, {leftm2, m2},
   {leftm2, rightm2}}

dummy[3] = {{a, leftm1, m1}, {e, leftm2, m2}}

dummy[4] = {{leftm1, leftm2, m1, m2}}

dummy["AllCategories"] =
  {{b, leftm1}, {c, m1}, {f, leftm2}, {g, m2}, {leftm1, m1}, {leftm1, m2},
   {leftm1, rightm1}, {leftm2, m1}, {leftm2, m2}, {leftm2, rightm2},
   {a, leftm1, m1}, {e, leftm2, m2}, {leftm1, leftm2, m1, m2}}

dummy["nonsingleVars"] = {leftm1, leftm2, m1, m2}

dummy["numbers"] = {2, 3, 4}

dummy["singleRules"] =
  {c -> C0 ** m1, g -> C0 ** m2, e -> leftm2 ** A ** m2, b -> leftm1 ** B,
   f -> leftm2 ** B, rightm1 -> leftm1, rightm2 -> leftm2,
   a -> leftm1 ** A ** m1}

dummy["singleVars"] = {c, g, e, b, f, rightm1, rightm2, a}

dummy[{b, leftm1}] = {b -> leftm1 ** B}

dummy[{c, m1}] = {c -> C0 ** m1}

dummy[{f, leftm2}] = {f -> leftm2 ** B}

dummy[{g, m2}] = {g -> C0 ** m2}

dummy[{leftm1, m1}] =
  {leftm1 ** m1 -> 1, m1 ** leftm1 ** B ** C0 ** m1 ->
    -A ** m1 + B ** C0 ** m1 + m1 ** leftm1 ** A ** m1,
```

```
   m1 ** leftm1 ** B ** C0 ** B ** C0 ** m1 ->
    A ** A ** m1 - A ** B ** C0 ** m1 - B ** C0 ** A ** m1 +
     B ** C0 ** B ** C0 ** m1 - m1 ** leftm1 ** A ** A ** m1 +
     m1 ** leftm1 ** A ** B ** C0 ** m1 + m1 ** leftm1 ** B ** C0 ** A ** m1}

dummy[{leftm1, m2}] = {leftm1 ** m2 -> 0, leftm1 ** A ** m2 -> 0}

dummy[{leftm1, rightm1}] = {rightm1 -> leftm1}

dummy[{leftm2, m1}] =
  {leftm2 ** m1 -> 0, leftm2 ** B ** C0 ** m1 -> leftm2 ** A ** m1}

dummy[{leftm2, m2}] = {leftm2 ** m2 -> 1}

dummy[{leftm2, rightm2}] = {rightm2 -> leftm2}

dummy[{a, leftm1, m1}] = {a -> leftm1 ** A ** m1}

dummy[{e, leftm2, m2}] = {e -> leftm2 ** A ** m2}

dummy[{leftm1, leftm2, m1, m2}] = {m2 ** leftm2 -> 1 - m1 ** leftm1}

dummy[_] := {}
```

One can pick of categories one by one in Mathematica by using the *GetCategories* command.

```
In[5]:= GetCategories[{leftm2, m1},dummy]
Out[5]= {leftm2 ** m1 -> 0, leftm2 ** B ** C0 ** m1 -> leftm2 ** A ** m1}
In[6]:= GetCategories[{},dummy]
Out[6]= {}
In[7]:= GetCategories[{leftm1,m1},dummy]
Out[7]= {leftm1 ** m1 -> 1, m1 ** leftm1 ** B ** C0 ** m1 ->
>      -A ** m1 + B ** C0 ** m1 + m1 ** leftm1 ** A ** m1,
>     m1 ** leftm1 ** B ** C0 ** B ** C0 ** m1 ->
>      A ** A ** m1 - A ** B ** C0 ** m1 - B ** C0 ** A ** m1 +
>       B ** C0 ** B ** C0 ** m1 - m1 ** leftm1 ** A ** A ** m1 +
>       m1 ** leftm1 ** A ** B ** C0 ** m1 + m1 ** leftm1 ** B ** C0 ** A ** m1
>      }
```

Chapter 41

Example of Hypothesis Selection in Discovering

Example 41.1 *Suppose you are working with matrices a, b, c etc. If an NCProcess command discovers that $ab = bc$ and, at that point, you realize that it is reasonable to assume that no eigenvalue of a is an eigenvalue of c, then b must be zero from an analytic argument.*[1] *Therefore, one can add the polynomial equation $b = 0$ to the collection of polynomial equations and continue. As research progressed, one would add more and more analytic observations at the points where it became clear what those observations were.*

The syllogism corresponding to the above type of argument would go as follow.

Fact 41.2 *If $\langle$ whatever hypothesis $\rangle$, then $ab = bc$.*

Fact 41.3 *If a, b and c are matrices, no eigenvalue of a is an eigenvalue of c and $ab = bc$, then $b = 0$.*

Fact 41.4 *If $\langle$ whatever hypothesis $\rangle$ and $b = 0$, then $\langle$ desired conclusion $\rangle$.*

Conclusion 41.5 *If $\langle$ whatever hypothesis $\rangle$, then $\langle$ desired conclusion $\rangle$.*

Fact 41.2 would be seen by using the algorithms in this paper. Fact 41.3 is an observation from analysis. Fact 41.4 would be seen by using the algorithms in this paper. Conclusion 41.5 is a tautological conclusion of the Facts 41.2, 41.3 and 41.4.

---

[1] In fact even if one realized from the beginning of the computation that a and c have no common eigenvalues, there is no way to express this hypothesis using polynomial equations. In fact, a complete analytic argument which showed that $b = 0$ would consist of an algebraic derivation of the identity $ab = bc$ followed by the use of the hypothesis that a and c have no common eigenvalues. One would not in general know in advance that $ab = bc$ and so it would be premature to add "b=0" to the collection of polynomial equations at the beginning of the computation.

Chapter 42

Possibly Obsolete Command Descriptions

42.1 NCMakeGB Options -maybe obsolete

ction:history

42.1.1 SupressCOutput→ False (less output to the screen)

Aliases: **None**

Description: *SupressCOutput* is an option of *NCMakeGB*. As one of the last arguments of *NCMakeGB* you may include either the rule *SupressCOutput→True* or *SupressCOutput→False*. If True is used, then the C++ code will decrease the amount of information it prints to the screen during the run of *NCMakeGB*. This option only effects the amount of information displayed to the screen. The default value is False. For example,*NCMakeGB[aListOfPolynomials, iterations, SupressCOutput→True]*

Arguments: None

Comments / Limitations: Not available before NCAlgebra 1.2

42.1.2 SupressAllCOutput→ False (*very* little outp ut to the screen)

Aliases: **None**

Description: *SupressAllCOutput* is just like *SupressCOutput*, but when this option is on, even more on the display to the screen will be supressed.

Arguments: None

Comments / Limitations: Not available before NCAlgebra 1.2

42.1.3 NCContinueMakeGB[iterationNumber]

Aliases: **None**

Description: This command allows you to continue running $NCMakeGB$ without resetting the starting relations. The command $NCMakeGB[aList, n, ReturnRelationsToMma->False]$ is equivalent to executing the command $NCMakeGB[aList, k, ReturnRelationsToMma->False]$ and then the command $NCContinueMakeGB[n]$ (for any $k \leq n$. The command $NCMakeGB[aList, n]$ is equivalent to executing the command $NCMakeGB[aList, k, ReturnRelationsToMma->False]$ and then the command $NCContinueMakeGB[n]$ and then $WhatIsPartialGB[]$ for any $k \leq n$.

Arguments: $iterationNumber$ is an integer.

Comments / Limitations: Not available before NCAlgebra 1.2

42.2 Special GB related commands- may not work

42.3 Starting Relations

42.3.1 SortRelations[aListOfRules]

Aliases: **None**

Description: *SortRelations[aListOfRules]* sorts the list of rules *aListOfRules*. First, the sorting is done by the number of unknowns in each rule. For rules with the same number of unknowns, the one which is considered largest is the one which has a largest left hand side (in terms of the monomial order).

Arguments: *aListOfRules* is a list of rules.

Comments / Limitations:

42.3.2 SortMonomials[aListOfVariables]

Aliases: **None**

Description: *SortMonomials[aListOfVariables]* returns a sorted list of monomials in terms of the monomial order.

Arguments: *aListOfVariables* is a list of indeterminates.

Comments / Limitations:

42.4 Changing the default options for NCMakeGB

42.4.1 ChangeNCMakeGBOptions[option,value] — need to update description

Aliases: **None**

Description: When the C++ version of the code is loaded, the statement Options[NCMakeGB] := CleanUp-¿1,ReturnRelations-¿True,SupressCOutput-¿False; is found in the code. The content of this command is that CleanUpBasis in by default on, ReturnRelations is by default true and SupressCOutput is by default false. If one wants to change these defaults, then one can use the *ChangeNCMakeGBOptions* command. For example, *ChangeNCMakeGBOptions* *¿1,ReturnRelations-¿False, SupressCOutput-¿False*} changes the default behaviour so that ReturnRelations is false by default.

Arguments: *ThreeRules* is a list containing three rules. The left hand sides of these rules must be *CleanUp*, *ReturnRelations* and *SupressCOutput*

Comments / Limitations:

Chapter 43

Generating Universal Gröbner Basis-MAY NOT WORK - untested in 1999

These commands are useful for generating a universal GB. If one starts with a GB with respect to any order this can be used to impose equivalence classes on the set of all order s. The commands below characterize these equivalence classes and select representatives from each of them. See [HSH]. this selects

43.0.2 AllOrders[aListofPolynomials, aListofIndeterminants]

Aliases: **None**

Description: **AllOrders[aListOfPolynomials, aListOfIndeterminants]** returns A list of graded lexicographical orders of the given indeterminants. Each order, represented as a list, is a member of an equivalence class of orders that produce the same leading terms in each of the polynomials in the list.

Arguments: *aListOfPolynomials* is a list of polynomials. aListofIndeterminants is a list of noncommutative indeterminants.

Comments / Limitations: Not available before NCAlgebra 1.2

43.0.3 EquivalenceClasses[aListOfPolynomials] or EquivalenceClasses[aListOfPolynomials, Simpler]

Aliases: **None**

Description: **EquivalenceClasses[aListOfPolynomials]** returns a logical expression that represents the equivalence classes of orders that produce the same leading terms in each of the polynomials in the list. If Simpler is used and it is False, no additional processing is done. If Simpler is used and it is True, the expression will be simplified as much as possible. In this case, the EquivalenceClasses command will take longer than if there were no second argument.

Arguments: *aListOfPolynomials* is a list of polynomials. *Simpler* is either True or False.

Comments / Limitations: Not available before NCAlgebra 1.2

43.0.4 UniversalBasis[aListOfPolynomials, NumberOfIterations]

Aliases: **None**

Description: **UniversalBasis[aListOfPolynomials, NumberOfIterations]** finds a universal Gröbner Basis with respect to the set of all graded lexicographical orders on the indeterminants in $aListOfPolynomials$ for the ideal generated by $aListOfPolynomials$. $NumberOfIterations$ is passed as the second argument to NCMakeRules.

Arguments: $aListOfPolynomials$ is a list of polynomials. $numberOfIterations$ is a positive integer.

Comments / Limitations: Not available before NCAlgebra 1.2

43.1 Very Technical Commands

43.1.1 GroebnerCutOffFlag[n_Integer]

Aliases: **None**

Description: Turns the "cutting off" operations either on or off.

Arguments: n is an natural number.

Comments / Limitations: Not available before NCAlgebra 1.2

43.1.2 GroebnerCutOffMin[n_Integer]

Aliases: **None**

Description: Sets the min for sum on polynomial degree cut offs.

Arguments: n is an natural number.

Comments / Limitations: Not available before NCAlgebra 1.2

43.1.3 GroebnerCutOffSum[n_Integer]

Aliases: **None**

Description: Sets the min for sum on polynomial degree cut offs.

Arguments: n is an natural number.

Comments / Limitations: Not available before NCAlgebra 1.2

Chapter 44

Commands for Producing HTML Output

This section describes commands which can be used to display formulas and expressions in HTML files. This is done by creating gif files to display formulas and expressions that can not be displayed in plain text.

44.0.4 ToHTMLString[expression]

Aliases: **HTML**

Description: Convert an expression to a Mathematica string which is a series of HTML image tags. These tags call gif images which are made from the latex version of the symbols in expression. If the proper gif does not already exist, then ToHTMLString will try to create it and put it in the directory specified by the Mathematica variable $GifDirectory. If the gif is created, then ToHTMLString will record the HTML tag in a file so that it will not have to be created again. ToHTMLString can be used to create a large assortment of symbols which can be used over and over.

Arguments: expression can be almost any Mathematica expression, like a symbol, list, rule, or polynomial.

Comments / Limitations: If the user does not have write access to the $GifDirectory, then the resulting gifs can not be saved. The HTMLHardWired.m file should be in the current working directory in order to save the new HTML tags so that the gifs do not need to be created again. ToHTMLString uses the following external programs to create the gifs. Latex, dvips, pstogif and giftrans.

44.0.5 MakeGif[file,expression]

Aliases: **None**

Description: Creates a gif image which is the latex version of expression. Unlike ToHTMLString, MakeGif only produces one gif file. No attempt is made to save the HTML tag in a seperate reference file.

Arguments: file is a string which will be the name of the gif file. expression can be almost any Mathematica expression, like a symbol, list, rule, or polynomial. If expression is a string, then no attempt is made to convert it to TEXform. This is good for complicated expressions if the user is familiar with TEX. MakeGif tries to display the result with xv by default. The option DisplayFunction→string will attempt to view the gif with the command string. DisplayFunction→Identity will not display the gif.

Comments / Limitations: MakeGif uses the following external programs to create the gifs. Latex, dvips, pstogif and giftrans.

44.0.6 HTML

HERE IS ANOTHER ACCOUNT OF THIS MATTERIAL
THEY SHOULD BE MERGED SOMEDAY

This options creates a directory with several files in HTML format. This was constructed before Mma 3.0 and has not been tested for compatibility with it. The spreadsheet file is called index.html, and there is a seperate file for every category. There are several complications which makes this option impractical. The indeterminates, which contain subscripts and superscripts, are stored as gif files. These gif files are created with latex, dvips, pstogif and giftrans. The command which creates them needs to know what directory they are stored in. Also, the HTML text which calls the correct gif file needs to be stored in a Mathematica file. When an indeterminate is used for the first time, the HTML text needs to be added to this file. This means that the user needs write access to this file. Most computer systems can be set up to handle these idiosyncracies, but not without some effort.

Here is a brief overview of what would need to be done. There are several gifs already created. They are in a directory called GifDirectory?? in the directory containing the Mathematica source code for NCAlgebra. If you do not have write permission, then copy this directory to someplace where you do have write permission. Also, put the file HTMLHardWired.m someplace where you have write permission. It would be best to put it in your current working directory. Adjust your Mathematica $Path variable so that this copy of HTMLHardWired.m will be loaded. When that doesn't work, send us e-mail at *ncalg@osiris.ucsd.edu*.

44.1 Using an input file

Because Mathematica is interpretive, it is tempting to perform computations by starting Mathematica and start typing. Often, however, we have found that it is often the case that it is helpful to construct an auxilary file and then to load that file. This has the benefit of allowing the user to modify the computation slightly and rerun it, as well as the additional benefit of recording the computation.

A common input file may have the form

First version of the file

```
Get["NCGB.m"];
start = {x**x-a,
         Inv[y]**y-1,y**Inv[y]-1,
         (1-x)**Inv[1-x]-1,
         Inv[1-x]**(1-x)-1
};
start = NCExpand[start];  (* NCExpand, alias NCE is explained in the NCAlgebra document. *)
Print["start:",start];
```

After the file is run, one can see whether or not the file executed correctly. It does not — we forgot the SetNonCommutative command. The new file would look like:

Second version of the file

```
Get["NCGB.m"];
SetNonCommutative[a,x,y,Inv[x],Inv[1-y]];
start = {x**x-a,
         Inv[y]**y-1,y**Inv[y]-1,
         (1-x)**Inv[1-x]-1,
         Inv[1-x]**Inv[1-x]-1
};
start = NCExpand[start];
Print["start:",start];
```

After restarting Mathematica and running this command, we find that *start* is what we expect it to be. We can then remove the `Print` statement and add more code.

Part XI

THE PACKAGE SYStems - ONLY ON THE WEB

SYStems

Version 0.1

J. William Helton [1]
Math Dept., UCSD

Michael L. Walker[1]
General Atomic Corp.
La Jolla, California 92093

This document provides an overview of the NCAlgebra tools which are provided for studying engineering systems. At present what is well developed are files for doing H^∞ control and, with a slight modification of language, differential games. This includes expressions and rules useful when using the Hamiltonian point of view. We hope others will contribute packages for doing basic systems theory or for doing research on more specialized subjects. Many topics in linear systems could EASILY be programmed into this setting.

---

[1] This work was partially supported by the AFOSR and the NSF.

Chapter 45

Preface

To what extent can we make our A, B, C, D linear systems and operator calculations easier with computer assistance? The standard symbol manipulators Mathematica, Maple, and Macsyma do not do noncommutative algebra intelligently, so it is hard to find out. Our group wrote a package NCAlgebra which runs under Mathematica and which is a reasonable beginning.

We see this package as a competitor to the yellow pad. Once you get used to it this might be considerably more effective for hand calculations of modest size. Like Mathematica the emphasis is on interaction with the program and flexibility.

I now prefer it to a yellow pad for many types of calculation and can recommend it to anyone who is good with Mathematica or alternatively is familiar with computers to the extent of doing a substantial amount of word processing. Putting a graduate student with computer talent on NCAlgebra also might work well. Our experience at UCSD is that students find it easy to learn. Indeed anyone can play with our package effortlessly, but one would have to be cautious about committing to a long term research project using NCAlgebra without being able to make additional commands on his own.

SYStems is based on NCAlgebra, a collection of "functions" for Mathematica designed to facilitate manipulation and reduction of noncommutative algebraic expressions. Specifically, it allows computer calculation in an algebra with involution. Such computations are common in many areas but our background is operator theory and engineering systems so we are aiming at research in these areas rather than at the complete treatments of the basics in these subjects.

SYSTEMS is a package which runs under NCAlgebra. It is in a primitive form with its main contents being a file for doing H^∞ control for linear and certain nonlinear systems. This originated with a paper (Aug91; by Ball, Helton and Walker [BHW]) which solves a collection of nonlinear H^∞ control problems. The formulas in that paper are executable inside NCAlgebra which greatly facilitates further research along these lines.

Chapter 46

How To Run The Systems Package

The beginner should first print out the file NCDOCUMENT.doc which explains how to use NCAlgebra. A little practice with this would help.

To run SYSTEMS, enter

```
math
```

at the UNIX prompt to execute Mathematica (or whatever command is appropriate for your non-UNIX system), followed by

```
<<SYStems.m
```

at the Mathematica

```
 In[1]:=
```

prompt.

That is all there is to it. Here SYStems.m loads NCAlgbra.m, one of the SYSTEMS DEFINITION files, plus two of the SYSTEMS UTILITY files described below.

These files are currently limited primarily to the study of H^∞ control of a certain class of nonlinear systems. There are additional files listed within comment delimiters in the SYSTEMS file which can be loaded to study slightly (but not much) different problems. This includes the option of studying so-called WIA systems in place of IA systems (see SYSHinf1 for definitions) or specializing to certain linear problems. A summary of files is given below. See the SYSTEMS file for further description.

The first time you run SYStem.s you should probably test to see that everything is in order. To do this you type:

```
math
<<SYStems.m
<<SYSTEST.m
```

while ignoreing lots of stuff which scrolls by on the screen. What counts comes at the end. You should see a list which says Test1 is True, Test2 is True, etc.

Systems Definitions

```
File                            Description
---------------------------------------------------------------
SYSDef1PortLin.m

SYSDefIA.m      Loads the system definitions and hamiltonian
                for an Input-Affine system (see EXAMPLE 5.2)

SYSDefWIA.m     Definition of a WIA (W-Input-Affine) system
                SYSTEMS file could be edited to replace loading of
                SYSDef with loading of this file.

SYSDefWAccess.m

SYSDef2x2.m     Definition of two 2 ports and the coupling laws for
                coupling them as well as hamiltonians.
```

UTILITY FILES

```
File                            Description
-----------      ---------------------------------------------------
SYSSpecialize.m  Rules for converting systems from nonlinear to
                 linear, notation changes , etc.

SYSLinearize.m   Linearize system around 0 or around the
                 diagonal x=z.  (We do not support this one)

SYSHinfFormulas.m   Formulas used in study of H-infinity control.

SYSHinf2x2Formulas.m   Formulas used in study of H-infinity control.
```

FILES CORRESPONDING TO RESEARCH ARTICLES

```
File                            Description
-----------------------------------------------------------------
SYSHinfTAC.m An executable paper on nonlinear H-infinity control
             together with rules to allow conversion to linear case.
             Primarily for study of IA systems.
```

Chapter 47

How To Contribute

Send files.m preferably as packages to

`ncalg@ucsd.edu`

Also we would like a file.doc which tells how to use it. Also we would like a SYSfileTest.m which automatically tests your program. This is not essential. See Diff.NCTest for an example of the format for these files. Usually you can just modify a copy of this file.

Chapter 48

What SYStems Does

The tools in the SYSTEMS package are designed to assist in studying various systems using the following commonplace idea (see [BHW] for more detail):

Define a finite-gain dissipative system to be a system for which

$$\int_{t0}^{t1} |out|^2 dt \leq K \int_{t0}^{t1} |W|^2 dt$$

where K is a constant, and $x(t0) = 0$. (See Figure 1.) If $K = 1$ the system will be called dissipative. In circuit theory these would be called passive. This agrees with the notion of dissipative in [W],[HM] with respect to the supply rate $|W|^2 - |out|^2$. Our notation here is like that one puts in the computer.

Define a storage or energy function on the state space to be a nonnegative function e satisfying

$$\int_{t0}^{t1} (|out|^2 - |W|^2) dt \leq e(x(t0)) - e(x(t1))$$

and $e(0) = 0$. Hill-Moylan ([HM]) showed that a system is dissipative iff an energy (storage) function (possibly extended real valued) exists. Under controllability assumptions, there exists an energy function with finite values.

We find it convenient to say that a system of the above form is e-dissipative provided that the energy Hamiltonian H defined by

$$H = out^T\, out - W^T\, W + (p\, F[x, W] + F[x, W]^T\, p^T)/2$$

is nonpositive where $p = \nabla(e(x))$. That is, $0 \geq H$ for all W and all x in the set of states reachable from 0 by the system.

Theorem 48.1 *(see [W],[HM]) Let e be a given differentiable function. Then a system is e-dissipative if and only if e is a storage function for the system. In this case, the system is dissipative.*

Using this background, we may study dissipativeness of systems by examining the non-positiveness of Hamiltonians. Replacing the dual variables p by

$$p = \nabla_x\, e$$

This substitution converts H to

$$sHW = out^T\, out - W^T\, W + (\nabla_x\, e\, F[x, W] + F[x, W]^T\, \nabla_x^T\, e)/2$$

Often one finds that

$$sHWo := \max_{W} sHW(W, x)$$

(the Hamiltonian in state space variables which has been optimized in W) is well behaved and is the first max taken in many approaches to solving the problem of checking if $sHW \leq 0$ for all x. $sHWo$ can be computed concretely for systems (see e.g. files SYSDefIA.m and SYSDefWIA.m) by taking the gradient of sHW in W and setting it to 0 to find the critical point CritW. Substitute this back into sHW to get sHWo. In our language CritW can be computed by

```
ruCritW=Crit[H,W];      (Calculate critical value for H w.r.t. W)
HWo=Sub[H,ruCritW];     (Substitute critical W back in H)
sHWo=Sub[HWo,ruxz];     (Convert to state-space coordinates)
```

since p is independent of W. Some examples are given later to illustrate.

What we have done so far is explain the very simplest case. Whenever one has a Hamiltonian which is quadratic in a variable our symbol manipulator optimizes the variable automatically. In H^∞ control one gets Hamiltonians which are very complicated expressions in many variables (e.g. compensator parameters). One must take many maxima and minima in analysing these problems and we have found SYStems very effective at such computations.

Game theory is another area which produces Hamiltonians and calls for such computations.

Rather than try to list complicated classifications of Hamiltonian types that this addresses we proceed by presenting some examples. One is the classical bounded real lemma while the other derives the famous DGKF two Riccati equation formulas for nonlinear systems (which are affine linear in the input variables).

Also there is substantial systems capability in a completely different directions. NCAlgebra contains commands for manipulating block matricies. Since such computations are the substance of many results in systems theory it is already easy to do computations in these areas with NCAlgbra. A few formulas for 2 × 2 block matricies are stored: Schur complements, block LU decomposition, inverses, and Chain to scattering formalism conversions.

Chapter 49

Sample Applications

Each of the following examples uses one of the SYSDef files as input. That SYSDef file is reproduced here for your reference. Text between comment symbols (* and *) is non-executable. In this document we have modified the actual SYSDef files by a few deletions and by transcribing many of the comments to TeX. Everything else in the SYSDef file is loaded into your session when you type:

```
<<SYSDef####.m
```

where #### represents the particular SYSDef file chosen.

49.1 Bounded Real Lemma

This example derives the bounded real lemma for a linear system. You should see the notebook **DemoBRL.ps** and have the joy of executing **DemoBRL.nb**.

49.2 Measurement Feedback H^∞ Control

We wish to analyse the dissipativity condition on 2 port systems with a one port system in feedback. The basic question is when does feedback exist which makes the full system dissipative and internally stable? This example is based on the paper [BHW]. Here is a listing of the definitions for a nonlinear system which is affine in the input. It is part of our SYS file named SYSDefIA.m - again here are TeX transcriptions:

INPUT AFFINE SYSTEM DEFINITIONS

```
(* file is SYSDefIA.m
  This file loads in basic definitions of two systems and of
the energy balance relations various connections of the
systems satisfy.


                     ------------------
                     |                |
           W  ---->--|             G1 |---->--- out1
                     |  F             |
           U  ---->--|             G2 |---->--- Y
                     |________________|
                      --------------
                      |            |
            U  ----<--| g      f   |----<--- Y
                      |____________|

                         Figure 2.

                 dx/dt  =  F[x,W,U]
                  out1  =  G1[x,W,U]
                    Y   =  G2[x,W,U]

                 dz/dt  =  f[z,Y]
                    U   =  g[z,Y]

 An Input Affine (IA) one port system is
---------------------------------------- *)
                   SetNonCommutative[f,g,a,b,c,dd,z]

                       f[z_,Y_]:=a[z]+b[z]**Y
                       g[z_,Y_]:=c[z]+dd[z]**Y

(* An Input Affine (IA) two port system is *)

                   SetNonCommutative[W,U,Y,DW,DU,DY]
                   SetNonCommutative[A,x,B1,B2,C1,C2,G1,G2]
                   SetNonCommutative[D11,D22,D12,D21]
                     D11[x_]:=0
                     D22[x_]:=0

                   F[x_,W_,U_]:=A[x]+B1[x]**W+B2[x]**U
                   G1[x_,W_,U_]:=C1[x]+D11[x]**W+D12[x]**U;
                            out1=G1[x,W,U];
                   G2[x_,W_,U_]:=C2[x]+D21[x]**W+D22[x]**U;
                            out2=G2[x,W,U];
                            G2I[x_,Y_,U_]:=inv[D21[x]]**Y-inv[D21[x]]**C2[x];
```

ENERGY BALANCE EQUATIONS

We begin with notation for analyzing the DISSIPATIVITY of the systems obtained by connecting f,g to F,G in several different ways. The energy function on the statespace is denoted by e. HWUY below is the Hamiltonian of the **two decoupled systems** where inputs are W,U, and Y.

$$HWUY = |out|^2 - |W|^2 + PP \cdot f(z,Y) + p \cdot F(x,W,U)$$

```
SetNonCommutative[F,G1,G2,f,g,p,PP];
HWUY = tp[out1]**out1-tp[W]**W+
                        (p**F[x,W,U]+tp[F[x,W,U]]**tp[p])/2+
                                       (PP**f[z,Y]+tp[f[z,Y]]**tp[PP])/2;
```

CONNECTING INPUT AND OUTPUT

If we connect the Y output of the 2 port to the f,g input then the resulting system has Hamiltonian function HWU.

$$HWU = HWUY \quad \text{with the substitution} \quad Y \to G2(x,W,U)$$

```
HWU=HWUY/.Y->G2[x,W,U];
```

If we connect the U input of the 2 port to the f,g output then the resulting system has Hamiltonian function HWY.

$$HWY = HWUY \quad \text{with the substitution} \quad U \to g(z,Y)$$

```
HWY=HWUY/.U->g[z,Y];
```

If we connect the two systems in feedback, that is tie off U and Y, then the resulting Hamiltonian function is

$$HW = HWY \quad \text{with the substitution} \quad Y \to G2(x,W,U)$$

```
HW=HWY/.Y->G2[x,W,U];
```

By definition (see Section I) the closed loop system being e-DISSIPATIVE corresponds to the energy balance function HW above being negative.

Note that the function HWUY contains both state and dual variables. Dual variables are defined by the gradient of the energy function, as follows:

$$p = \nabla_x e \quad \text{and} \quad PP = \nabla_z e$$

In the following, when we impose the IA assumptions (see (1.5) and (1.6)), we will often specialize to a plant which satisfies

$$D11(x) = D22(x) = 0,$$

$$D12(x)^T D12(x) = e1(x) > 0,$$

$$D21(x) D21(x)^T = e2(x) > 0.$$

and a compensator with d(z)=0. We now write out energy balance formulas for the plant and compensator after Figure 2 (under these assumptions) purely in terms of state space variables x and z (designated by the prefix s for state space).

PURE STATESPACE VARIABLES

STATESPACE -x,z. While the HWUY formulas mix x's and p 's , the next ones are purely on statespace variables x and z. Executable formulas for

$$p = \nabla_x e \quad \text{and} \quad PP = \nabla_z e$$

are inserted by the rule

```
SNC[GEx,GEz];
ruxz={p->tp[GEx[x,z]], PP->tp[GEz[x,z]]};
```

Here,

```
GEx[x,z], GEz[x,z]
```

stand for column vectors $\nabla_x^T e$, $\nabla_z^T e$, respectively. This unfortunate choice of notation is a holdover from earlier versions of the software.

The storage function e is homogeneous

```
    GEx[0,0]=0;
    GEz[0,0]=0;

sHWUY=HWUY/.ruxz;
sHWU=HWU/.ruxz;
sHWY=HWY/.ruxz;
sHW=HW/.ruxz;
```

RESULTS STORED FOR FAST EXECUTION

The following are redundant in that they can be computed from the above expressions.

This is the W which maximizes Hamiltonian

```
ruCritW= {W -> tp[B1[x]] ** GEx[x, z]/2 +
     tp[D21[x]] ** tp[b[z]] ** GEz[x, z]/2}

CritW[x_,z_,b_]:= tp[B1[x]] ** GEx[x, z]/2 +
                tp[D21[x]] ** tp[b] ** GEz[x, z]/2;

(*  and you get  *)

sHWo = tp[A[x]] ** GEx[x, z]/2 + tp[C1[x]] ** C1[x] + tp[GEx[x, z]] ** A[x]/2 +
  tp[GEz[x, z]] ** a[z]/2 + tp[a[z]] ** GEz[x, z]/2 +
  tp[C1[x]] ** D12[x] ** c[z] + tp[C2[x]] ** tp[b[z]] ** GEz[x, z]/2 +
  tp[GEx[x, z]] ** B2[x] ** c[z]/2 + tp[GEz[x, z]] ** b[z] ** C2[x]/2 +
  tp[c[z]] ** e1[x] ** c[z] + tp[c[z]] ** tp[B2[x]] ** GEx[x, z]/2 +
  tp[c[z]] ** tp[D12[x]] ** C1[x] +
  tp[GEx[x, z]] ** B1[x] ** tp[B1[x]] ** GEx[x, z]/4 +
  tp[GEx[x, z]] ** B1[x] ** tp[D21[x]] ** tp[b[z]] ** GEz[x, z]/4 +
  tp[GEz[x, z]] ** b[z] ** D21[x] ** tp[B1[x]] ** GEx[x, z]/4 +
  tp[GEz[x, z]] ** b[z] ** e2[x] ** tp[b[z]] ** GEz[x, z]/4


(* This is the U which minimizes Hamiltonian *)
ruCritU = {U -> -inv[e1[x]] ** tp[B2[x]] ** GEx[x, z]/2 -
     inv[e1[x]] ** tp[D12[x]] ** C1[x]};

CritU[x_,z_]:=-inv[e1[x]] ** tp[B2[x]] ** GEx[x, z]/2 -
     inv[e1[x]] ** tp[D12[x]] ** C1[x];


END OF FILE  ############################################ END OF FILE
```

49.2.1 Derivation of CritW and sHWo

Now the demo starts by taking the Hamiltonian for the closed loop system sHW and optimizing it in W to get what we denote sHWo. Soon the demo requires serious knowledge of [BHW] so if the reader has problems he is urged to read SYSHinfTAC.m a file which gives a rather complete account of the terse and unmotivated calculations here.

```
In[4]:= <<SYStems.m

NOTE: SYStems.m loads in the following files:
      NCAlgbra.m, NCAliasFunctions.m, SYSDefIA.m,
      SYSSpecialize.m, and SYSHinfFormulas.m

In[5]:= Substitute[sHW,dd[z]->0]

Out[5]= -tp[W] ** W + (tp[c[z]] ** tp[D12[x]] + tp[C1[x]]) **
    (C1[x] + D12[x] ** c[z]) + ((tp[W] ** tp[B1[x]] + tp[c[z]] ** tp[B2[x]] +
         tp[A[x]]) ** GEx[x, z] +
      tp[GEx[x, z]] ** (A[x] + B1[x] ** W + B2[x] ** c[z]))/2 +
   (tp[GEz[x, z]] ** (b[z] ** (C2[x] + D21[x] ** W) + a[z]) +
     ((tp[W] ** tp[D21[x]] + tp[C2[x]]) ** tp[b[z]] + tp[a[z]]) ** GEz[x, z])
     /2

The critical W, found by taking the gradient of % and setting it to 0,
is

In[6]:= CriticalPoint[%,W]

Out[6]= {W -> tp[B1[x]] ** GEx[x, z]/2 +
     tp[D21[x]] ** tp[b[z]] ** GEz[x, z]/2}

In[7]:= Substitute[%%,%]

Out[7]= -(tp[GEx[x, z]] ** B1[x]/2 + tp[GEz[x, z]] ** b[z] ** D21[x]/2) **
     (tp[B1[x]] ** GEx[x, z]/2 + tp[D21[x]] ** tp[b[z]] ** GEz[x, z]/2) +
   (tp[c[z]] ** tp[D12[x]] + tp[C1[x]]) ** (C1[x] + D12[x] ** c[z]) +
   (((tp[GEx[x, z]] ** B1[x]/2 + tp[GEz[x, z]] ** b[z] ** D21[x]/2) **
          tp[B1[x]] + tp[c[z]] ** tp[B2[x]] + tp[A[x]]) ** GEx[x, z] +
      tp[GEx[x, z]] ** (A[x] + B1[x] **
          (tp[B1[x]] ** GEx[x, z]/2 + tp[D21[x]] ** tp[b[z]] ** GEz[x, z]/2) \
          + B2[x] ** c[z]))/2 +
   (tp[GEz[x, z]] ** (b[z] ** (C2[x] +
            D21[x] ** (tp[B1[x]] ** GEx[x, z]/2 +
               tp[D21[x]] ** tp[b[z]] ** GEz[x, z]/2)) + a[z]) +
      (((tp[GEx[x, z]] ** B1[x]/2 + tp[GEz[x, z]] ** b[z] ** D21[x]/2) **
             tp[D21[x]] + tp[C2[x]]) ** tp[b[z]] + tp[a[z]]) ** GEz[x, z])/2


         ------Check this against the stored formula for sHWo
In[8]:= NCExpand[%-sHWo]

Out[8]= -tp[c[z]] ** e1[x] ** c[z] +
   tp[c[z]] ** tp[D12[x]] ** D12[x] ** c[z] -
   tp[GEz[x, z]] ** b[z] ** e2[x] ** tp[b[z]] ** GEz[x, z]/4 +
   tp[GEz[x, z]] ** b[z] ** D21[x] ** tp[D21[x]] ** tp[b[z]] ** GEz[x, z]/4
```

```
In[9]:= Substitute[%%,rue]

NOTE: rue = {tp[D12[x_]] ** D12[x_] :> e1[x], D21[x_] **
             tp[D21[x_]] :> e2[x]}

Out[9]= 0
```

The output sHWo is also recorded in the file SYSDefIA.m above. Note that neither sHWo nor sHWoWIA depends on a or c.

49.2.2 The MIN/MAX in U

Also useful is the optimum CritU for

$$Hopt := \min_U \max_W sHW = \min_U sHWo = \max_W \min_U sHW.$$

For WIA systems there is not an elegant formula for CritU and possibly is not unique. For IA systems, one gets the concrete formula:

$$CritU(x,z) := -e1(x)^{-1}B2(x)^T\nabla_x e(x,z)/2 - e1(x)^{-1}D12(x)^T C1(x)$$

49.2.3 Derivations of IAX, Critc, and Hopt

Now we optimize sHWo in c[z] variable and restrict to x=z to get the Doyle Glover Kargonekar Francis condition for IA systems. Henceforth we abbreviate commands by their aliases.

```
In[4]:= <<SYStems.m
In[5]:= Crit[sHWo,c[z]]

Out[5]= {c[z] ->
    -inv[e1[x]] ** tp[B2[x]] ** GEx[x, z]/2 -
     inv[e1[x]] ** tp[D12[x]] ** C1[x]}      ----------- this is ruCritU
In[6]:= Hopt = NCE[Sub[sHWo,%5]]

Out[6]= tp[A[x]] ** GEx[x, z]/2 + tp[C1[x]] ** C1[x] +
   tp[GEx[x, z]] ** A[x]/2 + tp[GEz[x, z]] ** a[z]/2 +
   tp[a[z]] ** GEz[x, z]/2 + tp[C2[x]] ** tp[b[z]] ** GEz[x, z]/2 +
   tp[GEz[x, z]] ** b[z] ** C2[x]/2 +
   tp[GEx[x, z]] ** B1[x] ** tp[B1[x]] ** GEx[x, z]/4 -
   tp[C1[x]] ** D12[x] ** inv[e1[x]] ** tp[B2[x]] ** GEx[x, z]/2 -
   tp[C1[x]] ** D12[x] ** inv[e1[x]] ** tp[D12[x]] ** C1[x] +
   tp[GEx[x, z]] ** B1[x] ** tp[D21[x]] ** tp[b[z]] ** GEz[x, z]/4 -
   tp[GEx[x, z]] ** B2[x] ** inv[e1[x]] ** tp[B2[x]] ** GEx[x, z]/4 -
   tp[GEx[x, z]] ** B2[x] ** inv[e1[x]] ** tp[D12[x]] ** C1[x]/2 +
   tp[GEz[x, z]] ** b[z] ** D21[x] ** tp[B1[x]] ** GEx[x, z]/4 +
   tp[GEz[x, z]] ** b[z] ** e2[x] ** tp[b[z]] ** GEz[x, z]/4

In[7]:= Sub[Hopt,x->z]

Out[7]= tp[A[z]] ** GEx[z, z]/2 + tp[C1[z]] ** C1[z] +
   tp[GEx[z, z]] ** A[z]/2 + tp[GEz[z, z]] ** a[z]/2 +
```

```
   tp[a[z]] ** GEz[z, z]/2 + tp[C2[z]] ** tp[b[z]] ** GEz[z, z]/2 +
   tp[GEz[z, z]] ** b[z] ** C2[z]/2 +
   tp[GEx[z, z]] ** B1[z] ** tp[B1[z]] ** GEx[z, z]/4 -
   tp[C1[z]] ** D12[z] ** inv[e1[z]] ** tp[B2[z]] ** GEx[z, z]/2 -
   tp[C1[z]] ** D12[z] ** inv[e1[z]] ** tp[D12[z]] ** C1[z] +
   tp[GEx[z, z]] ** B1[z] ** tp[D21[z]] ** tp[b[z]] ** GEz[z, z]/4 -
   tp[GEx[z, z]] ** B2[z] ** inv[e1[z]] ** tp[B2[z]] ** GEx[z, z]/4 -
   tp[GEx[z, z]] ** B2[z] ** inv[e1[z]] ** tp[D12[z]] ** C1[z]/2 +
   tp[GEz[z, z]] ** b[z] ** D21[z] ** tp[B1[z]] ** GEx[z, z]/4 +
   tp[GEz[z, z]] ** b[z] ** e2[z] ** tp[b[z]] ** GEz[z, z]/4

In[8]:= ruXXYYI

Out[8]= {GEz[x_, x_] :> 0, GEx[x_, x_] :> 2*XX[x], GEx[x_, 0] :> 2*YYI[x]}

In[9]:= Sub[%7,%]

Out[9]= tp[A[z]] ** XX[z] + tp[C1[z]] ** C1[z] + tp[XX[z]] ** A[z] +
   tp[XX[z]] ** B1[z] ** tp[B1[z]] ** XX[z] -
   tp[C1[z]] ** D12[z] ** inv[e1[z]] ** tp[B2[z]] ** XX[z] -
   tp[C1[z]] ** D12[z] ** inv[e1[z]] ** tp[D12[z]] ** C1[z] -
   tp[XX[z]] ** B2[z] ** inv[e1[z]] ** tp[B2[z]] ** XX[z] -
   tp[XX[z]] ** B2[z] ** inv[e1[z]] ** tp[D12[z]] ** C1[z]

In[10]:= NCE[%-IAX[z]]
Out[10]= 0
```

49.2.4 Derivation of IAYI

This is an advanced exercise in the use of the glossary at the end of this document file. The glossary stores the basic formulas for solutions to the IA H^∞ control problem. We use them here to familiarize the reader with the GLOSSARY.

```
In[4]:= <<SYStems.m

In[5]:= Sub[sHWo,z->0]

Out[5]= tp[A[x]] ** GEx[x, 0]/2 + tp[C1[x]] ** C1[x] +
   tp[GEx[x, 0]] ** A[x]/2 + tp[GEz[x, 0]] ** a[0]/2 +
   tp[a[0]] ** GEz[x, 0]/2 + tp[C1[x]] ** D12[x] ** c[0] +
   tp[C2[x]] ** tp[b[0]] ** GEz[x, 0]/2 + tp[GEx[x, 0]] ** B2[x] ** c[0]/2 +
   tp[GEz[x, 0]] ** b[0] ** C2[x]/2 + tp[c[0]] ** e1[x] ** c[0] +
   tp[c[0]] ** tp[B2[x]] ** GEx[x, 0]/2 + tp[c[0]] ** tp[D12[x]] ** C1[x] +
   tp[GEx[x, 0]] ** B1[x] ** tp[B1[x]] ** GEx[x, 0]/4 +
   tp[GEx[x, 0]] ** B1[x] ** tp[D21[x]] ** tp[b[0]] ** GEz[x, 0]/4 +
   tp[GEz[x, 0]] ** b[0] ** D21[x] ** tp[B1[x]] ** GEx[x, 0]/4 +
   tp[GEz[x, 0]] ** b[0] ** e2[x] ** tp[b[0]] ** GEz[x, 0]/4

In[6]:= ruhomog

Out[6]= {A[0] -> 0, C1[0] -> 0, C2[0] -> 0, a[0] -> 0, c[0] -> 0}

In[7]:= Sub[%%,ruhomog]
```

```
Out[7]= tp[A[x]] ** GEx[x, 0]/2 + tp[C1[x]] ** C1[x] +
   tp[GEx[x, 0]] ** A[x]/2 + tp[C2[x]] ** tp[b[0]] ** GEz[x, 0]/2 +
   tp[GEz[x, 0]] ** b[0] ** C2[x]/2 +
   tp[GEx[x, 0]] ** B1[x] ** tp[B1[x]] ** GEx[x, 0]/4 +
   tp[GEx[x, 0]] ** B1[x] ** tp[D21[x]] ** tp[b[0]] ** GEz[x, 0]/4 +
   tp[GEz[x, 0]] ** b[0] ** D21[x] ** tp[B1[x]] ** GEx[x, 0]/4 +
   tp[GEz[x, 0]] ** b[0] ** e2[x] ** tp[b[0]] ** GEz[x, 0]/4

In[8]:= Sub[%,ruXXYYI]

NOTE: ruXXYYI = {GEz[x_, x_] :> 0, GEx[x_, x_] :> 2 XX[x],
                 GEx[x_, 0] :> 2 YYI[x]}

Out[8]= tp[A[x]] ** YYI[x] + tp[C1[x]] ** C1[x] + tp[YYI[x]] ** A[x] +
   tp[C2[x]] ** tp[b[0]] ** GEz[x, 0]/2 + tp[GEz[x, 0]] ** b[0] ** C2[x]/2 +
   tp[YYI[x]] ** B1[x] ** tp[B1[x]] ** YYI[x] +
   tp[GEz[x, 0]] ** b[0] ** D21[x] ** tp[B1[x]] ** YYI[x]/2 +
   tp[GEz[x, 0]] ** b[0] ** e2[x] ** tp[b[0]] ** GEz[x, 0]/4 +
   tp[YYI[x]] ** B1[x] ** tp[D21[x]] ** tp[b[0]] ** GEz[x, 0]/2

In[9]:= SubSym[%,ruqb]

NOTE: ruqb = tp[b[z_]] ** GEz[x_, z_] :> q[x, z]

Out[9]= tp[A[x]] ** YYI[x] + tp[C1[x]] ** C1[x] + tp[C2[x]] ** q[x, 0]/2 +
   tp[YYI[x]] ** A[x] + tp[q[x, 0]] ** C2[x]/2 +
   tp[q[x, 0]] ** e2[x] ** q[x, 0]/4 +
   tp[YYI[x]] ** B1[x] ** tp[B1[x]] ** YYI[x] +
   tp[YYI[x]] ** B1[x] ** tp[D21[x]] ** q[x, 0]/2 +
   tp[q[x, 0]] ** D21[x] ** tp[B1[x]] ** YYI[x]/2

In[10]:= Crit[%,q[x,0]]

Out[10]= {q[x, 0] ->
    -2*inv[e2[x]] ** C2[x] - 2*inv[e2[x]] ** D21[x] ** tp[B1[x]] ** YYI[x]}

In[11]:= Sub[%%,%]

Out[11]= (-2*tp[C2[x]] ** inv[e2[x]] -
       2*tp[YYI[x]] ** B1[x] ** tp[D21[x]] ** inv[e2[x]]) ** C2[x]/2 +
   tp[A[x]] ** YYI[x] + tp[C1[x]] ** C1[x] +
   tp[C2[x]] ** (-2*inv[e2[x]] ** C2[x] -
       2*inv[e2[x]] ** D21[x] ** tp[B1[x]] ** YYI[x])/2 +
   tp[YYI[x]] ** A[x] + (-2*tp[C2[x]] ** inv[e2[x]] -
       2*tp[YYI[x]] ** B1[x] ** tp[D21[x]] ** inv[e2[x]]) ** e2[x] **
     (-2*inv[e2[x]] ** C2[x] - 2*inv[e2[x]] ** D21[x] ** tp[B1[x]] ** YYI[x])/
    4 + (-2*tp[C2[x]] ** inv[e2[x]] -
       2*tp[YYI[x]] ** B1[x] ** tp[D21[x]] ** inv[e2[x]]) ** D21[x] **
     tp[B1[x]] ** YYI[x]/2 + tp[YYI[x]] ** B1[x] ** tp[B1[x]] ** YYI[x] +
   tp[YYI[x]] ** B1[x] ** tp[D21[x]] **
     (-2*inv[e2[x]] ** C2[x] - 2*inv[e2[x]] ** D21[x] ** tp[B1[x]] ** YYI[x])/
    2

In[12]:= NCE[%]
```

```
Out[12]= tp[A[x]] ** YYI[x] + tp[C1[x]] ** C1[x] + tp[YYI[x]] ** A[x] -
   tp[C2[x]] ** inv[e2[x]] ** C2[x] +
   tp[YYI[x]] ** B1[x] ** tp[B1[x]] ** YYI[x] -
   tp[C2[x]] ** inv[e2[x]] ** D21[x] ** tp[B1[x]] ** YYI[x] -
   tp[YYI[x]] ** B1[x] ** tp[D21[x]] ** inv[e2[x]] ** C2[x] -
   tp[YYI[x]] ** B1[x] ** tp[D21[x]] ** inv[e2[x]] ** D21[x] ** tp[B1[x]] **
    YYI[x]

In[13]:= NCE[%-IAYI[x]]

Out[13]= 0
```

49.2.5 Derivation of critical q, k, and bterm

```
In[2]:= <<SYStems.m

In[3]:= SubSym[sHWo,ruc]

NOTE: ruc = c[z_] :> -inv[e1[z]] ** (tp[B2[z]] ** XX[z] +
            tp[D12[z]] ** C1[z])

Out[3]= tp[A[x]] ** GEx[x, z]/2 + tp[C1[x]] ** C1[x] +
   tp[GEx[x, z]] ** A[x]/2 + tp[GEz[x, z]] ** a[z]/2 +
   tp[a[z]] ** GEz[x, z]/2 + tp[C2[x]] ** tp[b[z]] ** GEz[x, z]/2 +
   tp[GEz[x, z]] ** b[z] ** C2[x]/2 -
   (tp[C1[z]] ** D12[z] + tp[XX[z]] ** B2[z]) ** inv[e1[z]] ** tp[B2[x]] **
     GEx[x, z]/2 - (tp[C1[z]] ** D12[z] + tp[XX[z]] ** B2[z]) **
    inv[e1[z]] ** tp[D12[x]] ** C1[x] -
   tp[C1[x]] ** D12[x] ** inv[e1[z]] **
    (tp[B2[z]] ** XX[z] + tp[D12[z]] ** C1[z]) +
   tp[GEx[x, z]] ** B1[x] ** tp[B1[x]] ** GEx[x, z]/4 -
   tp[GEx[x, z]] ** B2[x] ** inv[e1[z]] **
     (tp[B2[z]] ** XX[z] + tp[D12[z]] ** C1[z])/2 +
   tp[GEx[x, z]] ** B1[x] ** tp[D21[x]] ** tp[b[z]] ** GEz[x, z]/4 +
   tp[GEz[x, z]] ** b[z] ** D21[x] ** tp[B1[x]] ** GEx[x, z]/4 +
   (tp[C1[z]] ** D12[z] + tp[XX[z]] ** B2[z]) ** inv[e1[z]] ** tp[D12[x]] **
    D12[x] ** inv[e1[z]] ** (tp[B2[z]] ** XX[z] + tp[D12[z]] ** C1[z]) +
   tp[GEz[x, z]] ** b[z] ** D21[x] ** tp[D21[x]] ** tp[b[z]] ** GEz[x, z]/4

In[4]:= NCE[SubSym[%,rua]]

NOTE: rua = a[z_] :> A[z] + B2[z] ** (-inv[e1[z]] **
           (tp[B2[z]] ** XX[z] + tp[D12[z]] ** C1[z]))
           - b[z] ** C2[z] + ((B1[z] - b[z] ** D21[z]) **
            tp[B1[z]]) ** XX[z]

Out[4]= tp[A[x]] ** GEx[x, z]/2 + tp[A[z]] ** GEz[x, z]/2 +
   tp[C1[x]] ** C1[x] + tp[GEx[x, z]] ** A[x]/2 + tp[GEz[x, z]] ** A[z]/2 +
   tp[C2[x]] ** tp[b[z]] ** GEz[x, z]/2 -
   tp[C2[z]] ** tp[b[z]] ** GEz[x, z]/2 + tp[GEz[x, z]] ** b[z] ** C2[x]/2 -
   tp[GEz[x, z]] ** b[z] ** C2[z]/2 +
   tp[GEx[x, z]] ** B1[x] ** tp[B1[x]] ** GEx[x, z]/4 +
```

```
   tp[GEz[x, z]] ** B1[z] ** tp[B1[z]] ** XX[z]/2 +
   tp[XX[z]] ** B1[z] ** tp[B1[z]] ** GEz[x, z]/2 -
   tp[C1[x]] ** D12[x] ** inv[e1[z]] ** tp[B2[z]] ** XX[z] -
   tp[C1[x]] ** D12[x] ** inv[e1[z]] ** tp[D12[z]] ** C1[z] -
   tp[C1[z]] ** D12[z] ** inv[e1[z]] ** tp[B2[x]] ** GEx[x, z]/2 -
   tp[C1[z]] ** D12[z] ** inv[e1[z]] ** tp[B2[z]] ** GEz[x, z]/2 -
   tp[C1[z]] ** D12[z] ** inv[e1[z]] ** tp[D12[x]] ** C1[x] +
   tp[GEx[x, z]] ** B1[x] ** tp[D21[x]] ** tp[b[z]] ** GEz[x, z]/4 -
   tp[GEx[x, z]] ** B2[x] ** inv[e1[z]] ** tp[B2[z]] ** XX[z]/2 -
   tp[GEx[x, z]] ** B2[x] ** inv[e1[z]] ** tp[D12[z]] ** C1[z]/2 -
   tp[GEz[x, z]] ** B2[z] ** inv[e1[z]] ** tp[B2[z]] ** XX[z]/2 -
   tp[GEz[x, z]] ** B2[z] ** inv[e1[z]] ** tp[D12[z]] ** C1[z]/2 +
   tp[GEz[x, z]] ** b[z] ** D21[x] ** tp[B1[x]] ** GEx[x, z]/4 -
   tp[GEz[x, z]] ** b[z] ** D21[z] ** tp[B1[z]] ** XX[z]/2 -
   tp[XX[z]] ** B1[z] ** tp[D21[z]] ** tp[b[z]] ** GEz[x, z]/2 -
   tp[XX[z]] ** B2[z] ** inv[e1[z]] ** tp[B2[x]] ** GEx[x, z]/2 -
   tp[XX[z]] ** B2[z] ** inv[e1[z]] ** tp[B2[z]] ** GEz[x, z]/2 -
   tp[XX[z]] ** B2[z] ** inv[e1[z]] ** tp[D12[x]] ** C1[x] +
   tp[GEz[x, z]] ** b[z] ** D21[x] ** tp[D21[x]] ** tp[b[z]] ** GEz[x, z]/4 +
   tp[C1[z]] ** D12[z] ** inv[e1[z]] ** tp[D12[x]] ** D12[x] ** inv[e1[z]] **
    tp[B2[z]] ** XX[z] + tp[C1[z]] ** D12[z] ** inv[e1[z]] ** tp[D12[x]] **
    D12[x] ** inv[e1[z]] ** tp[D12[z]] ** C1[z] +
   tp[XX[z]] ** B2[z] ** inv[e1[z]] ** tp[D12[x]] ** D12[x] ** inv[e1[z]] **
    tp[B2[z]] ** XX[z] + tp[XX[z]] ** B2[z] ** inv[e1[z]] ** tp[D12[x]] **
    D12[x] ** inv[e1[z]] ** tp[D12[z]] ** C1[z]

In[5]:= SubSym[%,tp[GEz[x,z]]**b[z]->q[x,z]]

Out[5]= q[x, z] ** C2[x]/2 - q[x, z] ** C2[z]/2 + tp[A[x]] ** GEx[x, z]/2 +
   tp[A[z]] ** GEz[x, z]/2 + tp[C1[x]] ** C1[x] +
   tp[C2[x]] ** tp[q[x, z]]/2 - tp[C2[z]] ** tp[q[x, z]]/2 +
   tp[GEx[x, z]] ** A[x]/2 + tp[GEz[x, z]] ** A[z]/2 +
   q[x, z] ** D21[x] ** tp[B1[x]] ** GEx[x, z]/4 +
   q[x, z] ** D21[x] ** tp[D21[x]] ** tp[q[x, z]]/4 -
   q[x, z] ** D21[z] ** tp[B1[z]] ** XX[z]/2 +
   tp[GEx[x, z]] ** B1[x] ** tp[B1[x]] ** GEx[x, z]/4 +
   tp[GEx[x, z]] ** B1[x] ** tp[D21[x]] ** tp[q[x, z]]/4 +
   tp[GEz[x, z]] ** B1[z] ** tp[B1[z]] ** XX[z]/2 +
   tp[XX[z]] ** B1[z] ** tp[B1[z]] ** GEz[x, z]/2 -
   tp[XX[z]] ** B1[z] ** tp[D21[z]] ** tp[q[x, z]]/2 -
   tp[C1[x]] ** D12[x] ** inv[e1[z]] ** tp[B2[z]] ** XX[z] -
   tp[C1[x]] ** D12[x] ** inv[e1[z]] ** tp[D12[z]] ** C1[z] -
   tp[C1[z]] ** D12[z] ** inv[e1[z]] ** tp[B2[x]] ** GEx[x, z]/2 -
   tp[C1[z]] ** D12[z] ** inv[e1[z]] ** tp[B2[z]] ** GEz[x, z]/2 -
   tp[C1[z]] ** D12[z] ** inv[e1[z]] ** tp[D12[x]] ** C1[x] -
   tp[GEx[x, z]] ** B2[x] ** inv[e1[z]] ** tp[B2[z]] ** XX[z]/2 -
   tp[GEx[x, z]] ** B2[x] ** inv[e1[z]] ** tp[D12[z]] ** C1[z]/2 -
   tp[GEz[x, z]] ** B2[z] ** inv[e1[z]] ** tp[B2[z]] ** XX[z]/2 -
   tp[GEz[x, z]] ** B2[z] ** inv[e1[z]] ** tp[D12[z]] ** C1[z]/2 -
   tp[XX[z]] ** B2[z] ** inv[e1[z]] ** tp[B2[x]] ** GEx[x, z]/2 -
   tp[XX[z]] ** B2[z] ** inv[e1[z]] ** tp[B2[z]] ** GEz[x, z]/2 -
   tp[XX[z]] ** B2[z] ** inv[e1[z]] ** tp[D12[x]] ** C1[x] +
   tp[C1[z]] ** D12[z] ** inv[e1[z]] ** tp[D12[x]] ** D12[x] ** inv[e1[z]] **
    tp[B2[z]] ** XX[z] + tp[C1[z]] ** D12[z] ** inv[e1[z]] ** tp[D12[x]] **
```

```
    D12[x] ** inv[e1[z]] ** tp[D12[z]] ** C1[z] +
   tp[XX[z]] ** B2[z] ** inv[e1[z]] ** tp[D12[x]] ** D12[x] ** inv[e1[z]] **
    tp[B2[z]] ** XX[z] + tp[XX[z]] ** B2[z] ** inv[e1[z]] ** tp[D12[x]] **
    D12[x] ** inv[e1[z]] ** tp[D12[z]] ** C1[z]

In[6]:= SubSym[%,rue]

NOTE: rue = {tp[D12[x_]] ** D12[x_] :> e1[x], D21[x_] **
             tp[D21[x_]] :> e2[x]}

Out[6]= q[x, z] ** C2[x]/2 - q[x, z] ** C2[z]/2 + tp[A[x]] ** GEx[x, z]/2 +
   tp[A[z]] ** GEz[x, z]/2 + tp[C1[x]] ** C1[x] +
   tp[C2[x]] ** tp[q[x, z]]/2 - tp[C2[z]] ** tp[q[x, z]]/2 +
   tp[GEx[x, z]] ** A[x]/2 + tp[GEz[x, z]] ** A[z]/2 +
   q[x, z] ** e2[x] ** tp[q[x, z]]/4 +
   q[x, z] ** D21[x] ** tp[B1[x]] ** GEx[x, z]/4 -
   q[x, z] ** D21[z] ** tp[B1[z]] ** XX[z]/2 +
   tp[GEx[x, z]] ** B1[x] ** tp[B1[x]] ** GEx[x, z]/4 +
   tp[GEx[x, z]] ** B1[x] ** tp[D21[x]] ** tp[q[x, z]]/4 +
   tp[GEz[x, z]] ** B1[z] ** tp[B1[z]] ** XX[z]/2 +
   tp[XX[z]] ** B1[z] ** tp[B1[z]] ** GEz[x, z]/2 -
   tp[XX[z]] ** B1[z] ** tp[D21[z]] ** tp[q[x, z]]/2 -
   tp[C1[x]] ** D12[x] ** inv[e1[z]] ** tp[B2[z]] ** XX[z] -
   tp[C1[x]] ** D12[x] ** inv[e1[z]] ** tp[D12[z]] ** C1[z] -
   tp[C1[z]] ** D12[z] ** inv[e1[z]] ** tp[B2[x]] ** GEx[x, z]/2 -
   tp[C1[z]] ** D12[z] ** inv[e1[z]] ** tp[B2[z]] ** GEz[x, z]/2 -
   tp[C1[z]] ** D12[z] ** inv[e1[z]] ** tp[D12[x]] ** C1[x] -
   tp[GEx[x, z]] ** B2[x] ** inv[e1[z]] ** tp[B2[z]] ** XX[z]/2 -
   tp[GEx[x, z]] ** B2[x] ** inv[e1[z]] ** tp[D12[z]] ** C1[z]/2 -
   tp[GEz[x, z]] ** B2[z] ** inv[e1[z]] ** tp[B2[z]] ** XX[z]/2 -
   tp[GEz[x, z]] ** B2[z] ** inv[e1[z]] ** tp[D12[z]] ** C1[z]/2 -
   tp[XX[z]] ** B2[z] ** inv[e1[z]] ** tp[B2[x]] ** GEx[x, z]/2 -
   tp[XX[z]] ** B2[z] ** inv[e1[z]] ** tp[B2[z]] ** GEz[x, z]/2 -
   tp[XX[z]] ** B2[z] ** inv[e1[z]] ** tp[D12[x]] ** C1[x] +
   tp[C1[z]] ** D12[z] ** inv[e1[z]] ** e1[x] ** inv[e1[z]] ** tp[B2[z]] **
    XX[z] + tp[C1[z]] ** D12[z] ** inv[e1[z]] ** e1[x] ** inv[e1[z]] **
    tp[D12[z]] ** C1[z] + tp[XX[z]] ** B2[z] ** inv[e1[z]] ** e1[x] **
    inv[e1[z]] ** tp[B2[z]] ** XX[z] +
   tp[XX[z]] ** B2[z] ** inv[e1[z]] ** e1[x] ** inv[e1[z]] ** tp[D12[z]] **
    C1[z]

In[7]:= Crit[%,tp[q[x,z]]]

Out[7]= {tp[q[x, z]] ->
    -2*inv[e2[x]] ** C2[x] + 2*inv[e2[x]] ** C2[z] -
     inv[e2[x]] ** D21[x] ** tp[B1[x]] ** GEx[x, z] +
     2*inv[e2[x]] ** D21[z] ** tp[B1[z]] ** XX[z]}

In[8]:= K=NCE[SubSym[%19,%20]]


Out[8]= tp[A[x]] ** GEx[x, z]/2 + tp[A[z]] ** GEz[x, z]/2 +
   tp[C1[x]] ** C1[x] + tp[GEx[x, z]] ** A[x]/2 + tp[GEz[x, z]] ** A[z]/2 -
   tp[C2[x]] ** inv[e2[x]] ** C2[x] + tp[C2[x]] ** inv[e2[x]] ** C2[z] +
```

```
   tp[C2[z]] ** inv[e2[x]] ** C2[x] - tp[C2[z]] ** inv[e2[x]] ** C2[z] +
   tp[GEx[x, z]] ** B1[x] ** tp[B1[x]] ** GEx[x, z]/4 +
   tp[GEz[x, z]] ** B1[z] ** tp[B1[z]] ** XX[z]/2 +
   tp[XX[z]] ** B1[z] ** tp[B1[z]] ** GEz[x, z]/2 -
   tp[C1[x]] ** D12[x] ** inv[e1[z]] ** tp[B2[z]] ** XX[z] -
   tp[C1[x]] ** D12[x] ** inv[e1[z]] ** tp[D12[z]] ** C1[z] -
   tp[C1[z]] ** D12[z] ** inv[e1[z]] ** tp[B2[x]] ** GEx[x, z]/2 -
   tp[C1[z]] ** D12[z] ** inv[e1[z]] ** tp[B2[z]] ** GEz[x, z]/2 -
   tp[C1[z]] ** D12[z] ** inv[e1[z]] ** tp[D12[x]] ** C1[x] -
   tp[C2[x]] ** inv[e2[x]] ** D21[x] ** tp[B1[x]] ** GEx[x, z]/2 +
   tp[C2[x]] ** inv[e2[x]] ** D21[z] ** tp[B1[z]] ** XX[z] +
   tp[C2[z]] ** inv[e2[x]] ** D21[x] ** tp[B1[x]] ** GEx[x, z]/2 -
   tp[C2[z]] ** inv[e2[x]] ** D21[z] ** tp[B1[z]] ** XX[z] -
   tp[GEx[x, z]] ** B1[x] ** tp[D21[x]] ** inv[e2[x]] ** C2[x]/2 +
   tp[GEx[x, z]] ** B1[x] ** tp[D21[x]] ** inv[e2[x]] ** C2[z]/2 -
   tp[GEx[x, z]] ** B2[x] ** inv[e1[z]] ** tp[B2[z]] ** XX[z]/2 -
   tp[GEx[x, z]] ** B2[x] ** inv[e1[z]] ** tp[D12[z]] ** C1[z]/2 -
   tp[GEz[x, z]] ** B2[z] ** inv[e1[z]] ** tp[B2[z]] ** XX[z]/2 -
   tp[GEz[x, z]] ** B2[z] ** inv[e1[z]] ** tp[D12[z]] ** C1[z]/2 +
   tp[XX[z]] ** B1[z] ** tp[D21[z]] ** inv[e2[x]] ** C2[x] -
   tp[XX[z]] ** B1[z] ** tp[D21[z]] ** inv[e2[x]] ** C2[z] -
   tp[XX[z]] ** B2[z] ** inv[e1[z]] ** tp[B2[x]] ** GEx[x, z]/2 -
   tp[XX[z]] ** B2[z] ** inv[e1[z]] ** tp[B2[z]] ** GEz[x, z]/2 -
   tp[XX[z]] ** B2[z] ** inv[e1[z]] ** tp[D12[x]] ** C1[x] +
   tp[C1[z]] ** D12[z] ** inv[e1[z]] ** e1[x] ** inv[e1[z]] ** tp[B2[z]] **
    XX[z] + tp[C1[z]] ** D12[z] ** inv[e1[z]] ** e1[x] ** inv[e1[z]] **
    tp[D12[z]] ** C1[z] - tp[GEx[x, z]] ** B1[x] ** tp[D21[x]] **
     inv[e2[x]] ** D21[x] ** tp[B1[x]] ** GEx[x, z]/4 +
   tp[GEx[x, z]] ** B1[x] ** tp[D21[x]] ** inv[e2[x]] ** D21[z] **
     tp[B1[z]] ** XX[z]/2 + tp[XX[z]] ** B1[z] ** tp[D21[z]] ** inv[e2[x]] **
     D21[x] ** tp[B1[x]] ** GEx[x, z]/2 -
   tp[XX[z]] ** B1[z] ** tp[D21[z]] ** inv[e2[x]] ** D21[z] ** tp[B1[z]] **
    XX[z] + tp[XX[z]] ** B2[z] ** inv[e1[z]] ** e1[x] ** inv[e1[z]] **
    tp[B2[z]] ** XX[z] + tp[XX[z]] ** B2[z] ** inv[e1[z]] ** e1[x] **
    inv[e1[z]] ** tp[D12[z]] ** C1[z]

In[9]:= K = NCC[NCC[K,inv[e2[x]]],inv[e1[z]]]

Out[9]= tp[A[x]] ** GEx[x, z]/2 + tp[A[z]] ** GEz[x, z]/2 +
   tp[C1[x]] ** C1[x] + tp[GEx[x, z]] ** A[x]/2 + tp[GEz[x, z]] ** A[z]/2 +
   (tp[GEx[x, z]] ** B2[x] + tp[GEz[x, z]] ** B2[z]) ** inv[e1[z]] **
    (-tp[B2[z]] ** XX[z]/2 - tp[D12[z]] ** C1[z]/2) +
   (tp[C1[z]] ** D12[z] + tp[XX[z]] ** B2[z]) ** inv[e1[z]] **
    (-tp[B2[x]] ** GEx[x, z]/2 - tp[B2[z]] ** GEz[x, z]/2 -
      tp[D12[x]] ** C1[x]) + tp[C2[x]] ** inv[e2[x]] **
    (-C2[x] + C2[z] - D21[x] ** tp[B1[x]] ** GEx[x, z]/2 +
      D21[z] ** tp[B1[z]] ** XX[z]) +
   (tp[XX[z]] ** B1[z] ** tp[D21[z]] + tp[C2[z]]) ** inv[e2[x]] **
    (C2[x] - C2[z] + D21[x] ** tp[B1[x]] ** GEx[x, z]/2 -
      D21[z] ** tp[B1[z]] ** XX[z]) +
   tp[C1[x]] ** D12[x] ** inv[e1[z]] **
    (-tp[B2[z]] ** XX[z] - tp[D12[z]] ** C1[z]) +
   tp[GEx[x, z]] ** B1[x] ** tp[B1[x]] ** GEx[x, z]/4 +
   tp[GEz[x, z]] ** B1[z] ** tp[B1[z]] ** XX[z]/2 +
```

```
   tp[XX[z]] ** B1[z] ** tp[B1[z]] ** GEz[x, z]/2 +
   (tp[C1[z]] ** D12[z] + tp[XX[z]] ** B2[z]) ** inv[e1[z]] ** e1[x] **
    inv[e1[z]] ** (tp[B2[z]] ** XX[z] + tp[D12[z]] ** C1[z]) +
   tp[GEx[x, z]] ** B1[x] ** tp[D21[x]] ** inv[e2[x]] **
    (-C2[x]/2 + C2[z]/2 - D21[x] ** tp[B1[x]] ** GEx[x, z]/4 +
      D21[z] ** tp[B1[z]] ** XX[z]/2)

In[10]:= qpart = NCE[%19-(%19//.q[x,z]->0)]

Out[10]= q[x, z] ** C2[x]/2 - q[x, z] ** C2[z]/2 +
   tp[C2[x]] ** tp[q[x, z]]/2 - tp[C2[z]] ** tp[q[x, z]]/2 +
   q[x, z] ** e2[x] ** tp[q[x, z]]/4 +
   q[x, z] ** D21[x] ** tp[B1[x]] ** GEx[x, z]/4 -
   q[x, z] ** D21[z] ** tp[B1[z]] ** XX[z]/2 +
   tp[GEx[x, z]] ** B1[x] ** tp[D21[x]] ** tp[q[x, z]]/4 -
   tp[XX[z]] ** B1[z] ** tp[D21[z]] ** tp[q[x, z]]/2

In[11]:= %//.tp[q[x,z]]->0

Out[11]= q[x, z] ** C2[x]/2 - q[x, z] ** C2[z]/2 +
   q[x, z] ** D21[x] ** tp[B1[x]] ** GEx[x, z]/4 -
   q[x, z] ** D21[z] ** tp[B1[z]] ** XX[z]/2

In[12]:= L = %//.q[x,z]->1

Out[12]= C2[x]/2 - C2[z]/2 + D21[x] ** tp[B1[x]] ** GEx[x, z]/4 -
   D21[z] ** tp[B1[z]] ** XX[z]/2

In[13]:= Q = e2[x]/4

Out[13]= e2[x]/4

In[14]:= bterm = q[x,z]+tp[L]**inv[Q]

Out[14]= 4*(tp[GEx[x, z]] ** B1[x] ** tp[D21[x]]/4 -
       tp[XX[z]] ** B1[z] ** tp[D21[z]]/2 + tp[C2[x]]/2 - tp[C2[z]]/2) **
     inv[e2[x]] + q[x, z]

In[15]:= Sub[NCE[%19-bterm**Q**tp[bterm]-K],rue]

Out[15]= 0

In[16]:= Quit
```

49.3 Specializing to less general systems

49.3.1 Specializing to linear systems

To specialize to the linear case, just apply

- rulinearsys to make the systems linear
- rulinearEB to make the energy function quadratic
- ruGE1 then ruGEXY to make the energy function solve the Hinf problem (max entropy soln.)

- **rulinearall** contains all of the rules above and is what we usually use.
(See Glossary)

49.3.2 Specializing Using The Doyle Glover Khargonekar Francis Simplifying Assumptions

A special class of IA systems are those satisfying

$$D12(x)^T C1(x) = 0 \quad \text{and} \quad B1(x) D21(x)^T = 0$$

denoted in this paper as the Doyle-Glover-Kargonekar-Francis (DGKF) simplifying assumptions (see [DGKF]). These simplify algebra substantially so are good for tutorial purposes even though they are not satisfied in actual control problems. Look for rules ruDGKF*.
(See Glossary)

49.3.3 Demo: Linear Doyle Glover Kargonekar Francis Equations

The following demo verifies IAX and IAYI are same as DGX DGYI the Doyle Glover X and inv[Y] Riccati equations in the special case of a linear system.

```
In[24]:= <<SYStems.m

In[25]:= NCE[IAX[x]//.rulinearall]

Out[25]= tp[x] ** XX ** A ** x + tp[x] ** tp[A] ** tp[x] ** XX +
   tp[x] ** tp[C1] ** C1 ** x + tp[x] ** XX ** B1 ** tp[B1] ** tp[x] ** XX -
   tp[x] ** XX ** B2 ** inv[e1] ** tp[B2] ** tp[x] ** XX -
   tp[x] ** XX ** B2 ** inv[e1] ** tp[D12] ** C1 ** x -
   tp[x] ** tp[C1] ** D12 ** inv[e1] ** tp[B2] ** tp[x] ** XX -
   tp[x] ** tp[C1] ** D12 ** inv[e1] ** tp[D12] ** C1 ** x

In[26]:= Sub[%,x->1]

Out[26]= XX ** A + tp[A] ** XX + tp[C1] ** C1 + XX ** B1 ** tp[B1] ** XX -
   XX ** B2 ** inv[e1] ** tp[B2] ** XX -
   XX ** B2 ** inv[e1] ** tp[D12] ** C1 -
   tp[C1] ** D12 ** inv[e1] ** tp[B2] ** XX -
   tp[C1] ** D12 ** inv[e1] ** tp[D12] ** C1

In[27]:= NCE[%-DGX]

Out[27]= 0

In[28]:= NCE[IAYI[x]//.rulinearall]

Out[28]= tp[x] ** inv[YY] ** A ** x + tp[x] ** tp[A] ** tp[x] ** inv[YY] +
   tp[x] ** tp[C1] ** C1 ** x - tp[x] ** tp[C2] ** inv[e2] ** C2 ** x +
   tp[x] ** inv[YY] ** B1 ** tp[B1] ** tp[x] ** inv[YY] -
   tp[x] ** inv[YY] ** B1 ** tp[D21] ** inv[e2] ** C2 ** x -
   tp[x] ** tp[C2] ** inv[e2] ** D21 ** tp[B1] ** tp[x] ** inv[YY] -
    tp[x] ** inv[YY] ** B1 ** tp[D21] ** inv[e2] ** D21 ** tp[B1] ** tp[x] **
```

```
    inv[YY]

In[29]:= Sub[%,x->1]

Out[29]= inv[YY] ** A + tp[A] ** inv[YY] + tp[C1] ** C1 -
   tp[C2] ** inv[e2] ** C2 + inv[YY] ** B1 ** tp[B1] ** inv[YY] -
   inv[YY] ** B1 ** tp[D21] ** inv[e2] ** C2 -
   tp[C2] ** inv[e2] ** D21 ** tp[B1] ** inv[YY] -
   inv[YY] ** B1 ** tp[D21] ** inv[e2] ** D21 ** tp[B1] ** inv[YY]

In[30]:= NCE[YY**%**YY-DGY]

Out[32]= 0

In[34]:= Quit
```

Chapter 50

References

[AV] B. D. O. Anderson and S. Vongpanitlerd, Network Analysis and Synthesis, Prentice Hall, Englewood Cliffs, 1973.

[BHW] J. A. Ball, J. W. Helton, and M. L. Walker, IEEE Trans Aut Contrl April 1993, originally the computer file SYSHinfTAC.m included here.

[BH5] J. A. Ball and J. W. Helton, Nonlinear H^∞ control theory for stable plants, Math. Control Signals Systems 5 (1992), pp. 233-261.

[DGKF] J. C. Doyle, K. Glover, P. P. Khargonekar and B. A. Francis, State–space solutions to standard H_2 and H_∞ control problems, IEEE Trans. Auto. Control 34 (1989), 831–847.

[HM] D. J. Hill and P. J. Moylan, "Dissipative Dynamical Systems: Basic Input-Output and State Properties", J. Franklin Inst. v.309, No.5, May 1980, pp.327-357

[HWZ] J.W. Helton, M.L. Walker, W. Zhan, Compensators which have access to the command signal, preprint

[IA] A. Isidori and A. Astolfi, Nonlinear H^∞ control via measurement feedback, J. Math. Systems, Estimation and Control 2 (1992), pp. 31-34

[IB] A. Isidori and C. I. Byrnes, "Output Regulation of Nonlinear Systems", IEEE Trans. Aut. Contr., V.35, No.2, Feb.1990, pp. 131-140

[PAJ] I. R. Petersen, B. D. O. Anderson and E. A. Jonckheere, A First Principles Solution to the Non-Singular H^∞ Control Problem, Int. J. Robust and Nonlinear Control, v.1, 1991, pp.171-185

[V2] A. J. Van der Schaft, L_2-gain analysis of nonlinear systems and nonlinear H_∞ control, IEEE Trans. Auto. Control 37 (1992), pp.770-784.

[W] J. C. Willems, Dissipative Dynamical Systems, Part I: general theory, Arch. Rat. Mech. Anal. 45 (1972), 321-351.

Chapter 51

Glossary For System Hamiltonian Calculations

This glossary contains expressions available for manipulation, functions available for evaluation, and substitution rules all prefaced by ru.

Hamiltonians

```
These formulas contain mixed state and dual variables (IA systems, sec 5.2):
HWUY  = Hamiltonian for the system in SYSDefIA.m .
HWU = HWUY/.Y->G2[x,W,U];
HWY = HWUY/.U->g[z,Y];
HW = HWY/.Y->G2[x,W,U];

These formulas contain state variables only (IA systems):
sHWUY = HWUY/.ruxz;
sHWU = HWU/.ruxz;
sHWY = HWY/.ruxz;
sHW = HW/.ruxz;

These formulas contain dual variables only (IA systems):
dHWUY = HWUY//.rudual;
dHWU = HWU//.rudual;
dHWY = HWY//.rudual;
dHW = HW//.rudual;
```

Extremizing Hamiltonians

```
ricd = Riccati from bounded real lemma for SISO systems (sec 5.1)

CritW[x_,z_,b_] = value of W which makes Grad[sHW,W]=0(both IA and WIA systems).

sHWo  = max sHW (IA systems)
            W

sHWoWIA  = max sHW (WIA systems)
               W
```

```
CritU[x_,z_] = value of U which makes Grad[sHWU,U]=0 (IA systems).

RULES FOR CRITICAL VALUES:

ruCritW = {W -> tp[B1[x]] ** tp[GEx[x, z]] +
                tp[D21[x]] ** tp[b[z]] ** tp[GEz[x, z]]};

ruCritU = {U -> -inv[e1[x]] ** tp[B2[x]] ** tp[GEx[x, z]] -
                             inv[e1[x]] ** tp[D12[x]] ** C1[x]}
```

Nonlinear Riccatis

THE DGKF RICCATIS

```
IAX[x_] = XX[x] ** (A[x] - B2[x] ** inv[e1[x]] ** tp[D12[x]] ** C1[x])+
   (-tp[C1[x]] ** D12[x] ** inv[e1[x]] ** tp[B2[x]] + tp[A[x]]) **
   tp[XX[x]] + tp[C1[x]] ** C1[x] +
    XX[x] ** (B1[x] ** tp[B1[x]] - B2[x] ** inv[e1[x]] ** tp[B2[x]]) **
    tp[XX[x]] - tp[C1[x]] ** D12[x] ** inv[e1[x]] ** tp[D12[x]] ** C1[x]

IAYI[x_] = YYI[x] ** (A[x] - B1[x] ** tp[D21[x]] ** inv[e2[x]]**C2[x]) +
(-tp[C2[x]]**inv[e2[x]]**D21[x]**tp[B1[x]] + tp[A[x]])**tp[YYI[x]] +
      tp[C1[x]] ** C1[x] + YYI[x] **
       (B1[x]** tp[B1[x]] - B1[x]**tp[D21[x]] ** inv[e2[x]] ** D21[x] **
          tp[B1[x]]) ** tp[YYI[x]] - tp[C2[x]] ** inv[e2[x]] ** C2[x]


k[x_,z_]   = minimum in b of sHWo evaluated at a* and c* (see [BHW])
                 This is the 2 variable generalization of DGKF Riccati.


Hopt[x,z,a,b]  = min max sHWU, where you can specify functions a[z],b[z]
                  U   W
```

The Central Controller

```
    ruc = c[z_]:> -inv[e1[z]]**(tp[B2[z]]**XX[z] + tp[D12[z]]**C1[z]);
    rua = a[z_]:> A[z] + B2[z]**c[z] - b[z]**C2[z] +
                        (B1[z]- b[z]**D21[z])**tp[B1[z]]**XX[z]//.ruc ;
    rublin= the rule in the linear case for the b in the central soln
               to the H-infinity control problem.
    ruaWIA = a[z_]:> AB[z,c[z]] - b[z]**C2[z] +
                        (B1[z]- b[z]**D21[z])**tp[B1[z]]**XX[z];

Plug rua and ruc into the main Hamiltonian sHWo to get
     sHWo= k[x,z]  +  tp[bterm]**bterm
        which defines bterm.
```

Special Systems

Linear Systems

Linear systems satisfying the DGKF simplifying assumption are special cases of an IA system. To take an expression in H^∞ control which has been derived for an IA system and specialize it to linear systems apply

```
SubSym[expr,rulinearall]
```

where rulinearall is a long list of rules for specializing IA systems formulas described below:

```
ruGEXY={ruGEXY1,ruGEXY2};
                      where ruGEXY1=tp[GEx[x_,0]]:>tp[x]**inv[YY];
                            ruGEXY2=tp[GEx[x_,x_]]:>tp[x]**XX;

rulinearall=Union[rulinearsys,ruGE1, ruGEXY,rulinearXY];
              where rulinearsys={A[x_]:>A**x, B1[x_]:>B1,
                                         B2[x_]:>B2,C1[x_]:>C1**x,C2[x_]:>C2**x,
                                 D21[x_]:>D21,D12[x_]:>D12,a[x_]:>a**x,
                                         b[x_]:>b,c[x_]:>c**x,dd[x_]:>dd,
                                         e1[x_]:>e1,
                                         e2[x_]:>e2,tp[e1]->e1,tp[e2]->e2};
                    rulinearXY={XX[x_]:>XX**x,YY[p_]:>p**YY,
                                                YYI[x_]:>inv[YY]**x};

                            ruinvYY=YYI->inv[YY];

Lin[expr_]:= SubSym[expr,rulinearall];
Energy Ansatzes  which  are true for linear systems:
rulinearEB = sets GEx[x,z], GEz[x,z] to be linear
ruGEz0=GEz[x_,x_]:>0


 Ansatz in [BHW] %(5.1):

ruGE1=Flatten[{ruGEz0,SubSym[{ruGEz0,ruGEx1,ruGEz1},ruXXYYI]}];
             where ruGEx1=GEx[x_,z_]:>GEx[x,x]+GEx[x-z,0]-GEx[x-z,x-z];
                   ruGEz1=GEz[x_,z_]:>-GEx[x-z,0]+GEx[x-z,x-z];
```

Homogeneous Systems:

```
ruhomog = {A[0]->0, C1[0]->0, C2[0]->0, a[0]->0, c[0]->0};
```

Doyle Glover Khargonekar Francis Simplifying Assumptions:

```
ruDGKFNL = Apply DGKF simplifying assumptions for IA systems

ruDGKFlin = Apply DGKF simplifying assumptions for linear systems

ruCRcDGKF=c[z_]:>-tp[B2[z]]**tp[XX[z]];
ruCRqDGKF=q[x_,z_]:>-tp[C2[x-z]];
```

```
ruaDGKF = a[z_]:>A[z]+B1[z]**tp[B1[z]]**tp[XX[z]] -
               B2[z]**tp[B2[z]]**tp[XX[z]] - b[z]**C2[z]+cheat[z];

ruaDGKF2=a[z_]:> AA1[z,z]-b[z]**C2[z];
```

More Rules Available For Substitution

```
CHANGE OF VARIABLES:
Convert energy Hamiltonians from dual or mixed state/dual to state
variables only:
ruxz={p->GEx[x,z], PP->GEz[x,z]};

Convert energy Hamiltonian from dual or mixed state/dual to dual variables only:
rudual={rudualx,rudualz};
                              where rudualx = x->IGE1[p,PP];
                                    ruduualy = z->IGE2[p,PP];

CHANGE OF NOTATION:
rue = {tp[D12[x_]] ** D12[x_]:> e1[x], D21[x_] ** tp[D21[x_]] :> e2[x]}
rubtoq=tp[GEz[x_,z_]]**b[z_]:>q[x,z]

ruXXYYI={ruGEz0,ruXX, ruYYI}
                      where   ruXX = GEx[x_,x_]:>2*XX[x]
                              ruYYI = GEx[x_,0]:>2*YYI[x]

ruIAXYI={ruIAX,ruIAYI};
                      where ruIAX = replace terms linear in A or tp[A]
                                           using IAX[x]=0 (see 4.4b).
                            ruIAYI= replace terms linear in A or tp[A]
                                           using IAYI[x]=0 (see 4.4c).



FOR ENERGY ANSATZ EXPERIMENTATION AS IN [BHW]:

    EnergyGuess[ruleGE_] = generate sHWo with the Ansatz on the energy function
   given by ruleGE and a, b, and c are substituted out
   according to the RECIPE.
    EnergyGuess2[ruleGE,rulea] = same but a is substituted out by the
                                 user provided rulea.

    EnergyGuessDGKF[ruleGE_] = same with DGKF simplifying assumptions
    EnergyGuess2DGKF[ruleGE,ruaDGKF] = same with DGKF simplifying assumptions
```

Index