NCALGEBRA

Version 1.0
(Mathematica 1.2 and 2.0 compatible)

J. William Helton
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 contact ncalg@osiris.ucsd.edu or MathSource@wri.com. The program was written by the authors and by Mark Stankus, David Hurst, Daniel Lamm, Orlando Merino, Robert Obarr, and Henry Pfister. Simplification commands are based on theory being 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.

Contents

Table of Contents 7

Preface 8

I The Basics of NCAlgebra 9

1 The Most Basic Commands — A short tutorial 10

2 Getting Started 18

3 Examples 20

- 3.1 Simplifying a Complicated Expression 20
- 3.2 Vector Differentiation 26

4 Strategies 30

- 4.1 Edit 30
- 4.2 Expanding 31
- 4.3 Simplifying Expressions 31
- 4.4 Conventions 33

5 NC Commands 34

- 5.1 Manipulating an expression 34
  - 5.1.1 ExpandNonCommutativeMultiply[expr] 34
  - 5.1.2 NCCollect[expr, aListOfVariables] 34
  - 5.1.3 NCStrongCollect[expr, aListOfVariables] 35
  - 5.1.4 NCCollectSymmetric[expr] 35
  - 5.1.5 NCTermsOfDegree[expr,aListOfVariables,indices] 35
  - 5.1.6 NCBackward[expr] 35

5.1.7 NCForward[expr] 36
5.1.8 Substitute[expr,aListOfRules,(Optional On)] 36
5.1.9 SubstituteSymmetric[expr, aListOfRules, (optional On)] . . 36
5.1.10 SubstituteSingleReplace[expr, aListOfRules, (optional On)] . 37
5.1.11 SubstituteAll[expr, aListOfRules, (optional On)] 37
5.1.12 Transform[expr,aListOfRules] 37
5.1.13 NCMono[expr] 37
5.2 Manipulating an Expression — less useful commands 38
5.2.1 NCTermArray[expr,aList,anArray] 38
5.2.2 NCReconstructFromTermArray[anArray] 39
5.2.3 NCCompose[aVerySpecialList] 40
5.2.4 NCDecompose 40
5.2.5 NCCombine[expr] 40
5.3 Simplification 40
5.3.1 NCSimplifyRational[expr], NCSimplify1Rational[expr], and NCSimplify2Rational[expr] 41
5.3.2 NCSimplify1Rational[expr] 42
5.3.3 NCSimplify2Rational[expr] 42
5.3.4 NCSimplify0Rational[expr] 42
5.4 Vector Differentiation 43
5.4.1 DirectionalD[expr, aVariable, h] 43
5.4.2 Gradient[expr, aVariable] 43
5.4.3 CriticalPoint[expr, aVariable] 43
5.5 Block Matrix Multiplication 44
5.5.1 MatMult[x, y, . . .] 44
5.5.2 tpMat[u] 44
5.5.3 NCMToMatMult[expr] 44
5.5.4 TimesToNCM[expr] 45
5.5.5 Partitioning a Matrix into a Block Matrix 45
5.5.6 invMat2[mat] 46
5.5.7 GaussElimination[X] 47
5.5.8 Cascade[P, K] 47
5.5.9 Chain[P] 47

5.5.10 Redheffer[P] 47
5.5.11 DilationHalmos[x] 47
5.6 Complex Analysis 48
5.6.1 A tutorial 48
5.6.2 ComplexRules 50
5.6.3 ComplexCoordinates[expr] 51
5.6.4 ComplexD[expr, aVariable] 51
5.7 Setting symbols to commute or not commute 52
5.7.1 SetNonCommutative[a, b, c, ...] 52
5.7.2 SetCommutative[a, b, c, ...] 52
5.7.3 CommuteEverything[expr] 52
5.7.4 SetCommutingOperators[b,c] 52
5.7.5 LeftQ[expr] 53
5.7.6 CommutativeQ[x] 53
5.7.7 CommutativeAllQ[expr] 53
5.8 Setting Properties of an element in an algebra 53
5.8.1 inv[x] 53
5.8.2 invL[x] 54
5.8.3 invR[x] 54
5.8.4 invQ[x] 54
5.8.5 ExpandQ[inv] 54
5.8.6 ExpandQ[tp] 55
5.8.7 OverrideInverse 55
5.8.8 SetInv[a, b, c, ...] 55
5.8.9 SetSelfAdjoint[Symbols] 55
5.8.10 SelfAdjointQ[aSymbol] 56
5.8.11 SetIsometry[Symbols] 56
5.8.12 IsometryQ[aSymbol] 56
5.8.13 SetCoIsometry[Symbols] 56
5.8.14 CoIsometryQ[aSymbol] 57
5.8.15 SetUnitary[Symbols] 57
5.8.16 UnitaryQ[aSymbol] 57
5.8.17 SetProjection[Symbols] 57

5.8.18 ProjectionQ[Symbol] 58
5.8.19 SetSignature[Symbols] 58
5.8.20 SignatureQ[Symbol] 58
5.9 Setting Properties of functions on an algebra 58
5.9.1 SetSesquilinear[Functions] 58
5.9.2 SetSesq[Functions] 59
5.9.3 SesquilinearQ[aFunction] 59
5.9.4 SetBilinear[Functions] 59
5.9.5 BilinearQ[aFunction] 59
5.9.6 SetLinear[Functions] 59
5.9.7 LinearQ[aFunction] 60
5.9.8 SetConjugateLinear[Functions] 60
5.9.9 ConjugateLinearQ[aFunction] 60
5.9.10 SetIdempotent[Functions] 60
5.9.11 IdempotentQ[aFunction] 61
5.9.12 SetCommutingFunctions[Functions] 61
5.9.13 SetNonCommutativeMultiplyAntihomomorphism[Functions] 61
5.9.14 NCAntihomo[Functions] 62
5.10 Utilities 62
5.10.1 SaveRules[expression, 'optional tag → "message"'] 62
5.10.2 SaveRulesQ[] 62
5.10.3 FunctionOnRules[Rules, Function1, Function2, (optional On)] 63

II Output Formats 64

6 Text Oriented Output Commands 65
6.0.4 SetOutput[optionlist,. . .] 65
6.0.5 NCSave[aString, expr] 66
6.0.6 CEEP 66

7 Interfacing with TeX 67
7.1 Some Examples 68

8 List of TeX Commands **71**

8.0.1 LongExpressionToTeXFile[aString_, Expressions___] 71

8.0.2 LookAtMatrix[aMatrix] 71

8.0.3 OpenTeXFile[aString] and OpenTeXFile[] 72

8.0.4 CloseTeXFile[aString] and CloseTeXFile[] 72

8.0.5 ExpressionToTeXFile[aString, expr] and ExpressionToTeXFile[expr] 73

8.0.6 StartParagraph[aString] and StartParagraph[] 73

8.0.7 CommentToTeXFile[aString, comment] and CommentToTeXFile[comment] 73

8.0.8 InLineExpressionToTeXFile[aString, expr] and InLineExpressionToTeXFile[expr] 74

8.0.9 EqualityToTeXFile[aString, expr1, expr2] and EqualityToTeXFile[expr1, expr2] 74

8.0.10 AddSplitsTeX[thing] 75

8.0.11 SplittingTeXHeads 75

8.0.12 AddSplitsTeXText[”Plus”] = ”+”; 75

8.0.13 TeXTheFile[aString] TeXTheFile[] 76

9 The Command DoTeX – How To Output an Expression in TeX (Unix only) **77**

9.0.14 DoTeX 77

10 Modifications to the TeXForm Command **80**

III Appendices **82**

A NCAlgebra INSTALLATION **83**

B UNIX DEPENDENCIES **87**

C RUNNING SYSTEMS **88**

D GETTING NCALGEBRA THROUGH ANONYMOUS FTP **89**

E WELCOME OTHER DEVELOPERS 92

F THE MOST BASIC COMMANDS — TEAR OFF SECTION 94

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 INVOLUTION. 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.

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 SYStems.doc which is included and see the appendix on running SYSTEMS.

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

Also coming soon will be noncommutative Gröbner basis features in NCAlgebra.

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 Strategy chapter.)

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 useful to appropriate to use in every situation." See also APPENDIX F.

We now 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

The Basics of NCAlgebra

Chapter 1

The Most Basic Commands — A short tutorial

”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 uses constantly are built-in. The multiplicative identity is denoted Id in the program. At the present time, Id is set to 1. An element A may have an inverse, which will be denoted by inv[A], or it may have a left or right inverse, denoted invL[A] and invR[A] respectively.

Each example is independent of the rest and assumes that you are executing it as the first command after you load in our packages. For this reason, every time we want to show a feature of a typical command, like the SetLinear command we explicitly invoke the SetLinear command (even though once one says SetLinear[f], f will remain linear for the rest of your Mathematica session until changed or redefined).

At present, single-letter lower case variables are non-commutative by default and all others are commutative by default.

When the lower case variables are non-commutative then we get the following:

```
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[a];
In[12]:= a ** b
Out[12]= a 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]= tp[x] ** a+b) ** x + z

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]:= f = 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[f]
Out[48]= 1

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

NCSR is the alias for NCSimplifyRational.

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

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

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

Note: Linear equations in one unknown only.

```
In[56]:= SetIsometry[s];
In[57]:= IsometryQ[s]
Out[57]= True

In[58]:= SetIsometry[s];
In[59]:= aj[s] ** s
Out[59]= 1
```

Remember the line In[58] is redundant and unnecessary. Nevertheless, it is restated for the sake of clarity. Normally, In[58] would be left out, because s is still defined as in line In[56].

```
In[60]:= SetIsometry[s];
In[61]:= tp[s] ** s
Out[61]= 1

In[62]:= SetSelfAdjoint[t];
In[63]:= tp[t]
Out[63]= t

In[64]:= SetSelfAdjoint[t];
In[65]:= aj[t]
Out[65]= t

In[66]:= SetProjection[p];
In[67]:= p ** p
Out[67]= p

In[68]:= SetCoIsometry[s];
In[69]:= s ** tp[s]
Out[69]= 1

In[70]:= SetCoIsometry[w];
In[71]:= w** aj[w]
Out[71]= 1

In[72]:= SetInv[x];
In[73]:= invL[x]
Out[73]= inv[x]

In[74]:= SetInv[x];
In[75]:= invR[x]
Out[75]= inv[x]

In[76]:= SetLinear[f];
In[77]:= f[x + y]
Out[77]= f[x] + f[y]

In[78]:= SetLinear[f];
In[79]:= f[(2 + 3 I) x ]
Out[79]= 2 f[x] + 3 I f[x]

In[80]:= SetConjugateLinear[q];
In[81]:= q[x + y]
Out[81]= q[x] + f[y]

In[82]:= SetConjugateLinear[g];
In[83]:= g[(2 + 3 I) x ]
```

```
Out[83]= 2 g[x] - 3 I g[x]

In[84]:= SetBilinear[h];
In[85]:= h[x + y, w]
Out[85]= h[x, w] + h[y, w]

In[86]:= SetBilinear[f];
In[87]:= f[w, x + y]
Out[87]= f[w, x] + f[w, y]

In[88]:= SetBilinear[f];
In[89]:= f[(2 + 3 I) x, w ]
Out[89]= 2 f[x, w] + 3 I f[x, w]

In[90]:= SetBilinear[f];
In[91]:= f[w,(2 + 3 I) x]
Out[91]= 2 f[w, x] + 3 I f[w, x]

In[92]:= SetSesquilinear[f];
In[93]:= f[x + y, w]
Out[93]= f[x, w] + f[y, w]

In[94]:= SetSesquilinear[f];
In[95]:= f[w, x + y]
Out[95]= f[w, x] + f[w, y]

In[96]:= SetSesquilinear[f];
In[97]:= f[(2 + 3 I) x, w]
Out[97]= 2 f[x, w] + 3 I f[x, w]

In[98]:= SetSesquilinear[f];
In[99]:= f[w,(2 + 3 I) x]
Out[99]= 2 f[w, x] - 3 I f[w, x]

In[100]:= SetIdempotent[f];
In[101]:= f[f[w]]
Out[101]= w

In[102]:= SetCommutingFunctions[f, g]; LeftQ[f, g] = True;
In[103]:= g[f[x]]
Out[103]= f[g[x]]

In[104]:= SetCommutingFunctions[f, g]; LeftQ[f, g] = True;
In[105]:= f[g[x]]
Out[105]= f[g[x]]

In[106]:= SetCommutingFunctions[f, g]; LeftQ[f, g] = False;
In[107]:= g[f[x]]
Out[107]= g[f[x]]
```

```
In[108]:= SetCommutingFunctions[f, g]; LeftQ[f, g] = False;
In[109]:= f[g[x]]
Out[109]= g[f[x]]

In[110]:= mat = {{a + tp[b],aj[c]},{inv[d],e}}
Out[100]= {{a + tp[b], aj[c]}, {inv[d], e}}

In[111]:= LookAtMatrix[mat]
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 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). This window contains the following formula in its display.

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

The LaTeXfile generated is shown below.

```
In[112]:= !!file.tex
\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.

$$\begin{gathered} a(1) + a(2) + a(3) + a(4) + a(5) + a(6) + a(7) + a(8) + \\ a(9) + a(10) + a(11) + a(12) + a(13) + a(14) + a(15) \\ +a(16) + a(17) + a(18) + a(19) + a(20) + a(21) + \\ a(22) + a(23) + a(24) + a(25) + a(26) + a(27) + \end{gathered}$$

$$
\begin{gathered}
a(28) + a(29) + a(30) + a(31) + a(32) + a(33) + \\
a(34) + a(35) + a(36) + a(37) + a(38) + a(39) + \\
a(40) + a(41) + a(42) + a(43) + a(44) + a(45) + \\
a(46) + a(47) + a(48) + a(49) + a(50)
\end{gathered}
$$

```
In[114]:= SetOutput[ all -> True ];
In[115]:= inv[ tp[k] ]
             T -1
Out[115]= (k)

In[116]:= SetOutput[ all -> True ];
In[117]:= rt[x - inv[y + aj[z]]] + tp[x]
            T              * -1 1/2
Out[117]=  x  + (x - (y + z )  )

In[118]:= SetOutput[ all -> True, inv -> False ];
In[119]:= inv[1 + tp[1-x]]


                   T
Out[119]= inv[2 - x ]
```

See tp for transformation rules.

```
In[120]:= SetOutput[ all -> False, tp -> True, aj -> True ];
In[121]:= aj[ tp[k] ]
            T *
Out[121]= (k)

In[122]:= SetOutput[ all -> False, tp -> True, aj -> True ];
In[123]:=aj[ tp[k] ]
            * T
Out[123]= (k )
In[124]:= SetOutput[ all -> True ];
In[125]:= (p^x) ** tp[q]
             x     T
Out[125]= (p ) . (q )
In[126]:= SetOutput[ all -> False, dot -> True];
In[127]:= x ** tp[q] ** inv[1-x]
Out[127]= x . tp[q] . inv[1-x]
```

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

Chapter 2

Getting Started

First you must install NCAlgebra on your computer. How to do it is discussed in APPENDIX A.

This section contains instructions for getting started using NCAlgebra 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. The details for setting up a directory to use NCAlgebra are given in APPENDIX A. It is assumed in this section that such preparations have been made.

Steps for getting started:

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

(2) Move into the directory where NCAlgebra resides.

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

Steps (1) through (3) will vary from one computer system to another. If you are an inexperienced user, ask your system manager or a user with some experience with Mathematica to assist you with these. After executing step (3), you will see expressions of the form

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

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

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

The following are the initial NCAlgebra commands:

(4) At the input prompt, type:

```
<<CEEP
```

Note: This step is not essential. Mathematica will respond with:

```
TYPE FILE NAME WHERE YOU WANT THE SESSION SAVED ?
```

(5) Type a file name such as:

```
MYSESSION
```

(6) Then type:

```
<<NCAlgebra.m
```

You are now ready to begin a Mathematica session using NCAlgebra.

The double arrows "<<" above mean that Mathematica should bring in the contents of the file specified (see also "Get"). The "CEEP" file tells Mathematica that you want to record the functions you use during the session for later use or examination. CEEP records your session – two different ways into two different files – the first file (e.g. MYSESSION in the above example) 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

```
<<filename.ex
```

inside Mathematica, where "filename" is the name of the file that you entered in step (5) above. 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.

You may exit from the Mathematica program (and thus also stop using the NCAlgebra functions) by typing

```
Quit
```

at any input prompt(In[..]:= Quit). Note that the capital "Q" followed by lowercase "uit" is important. Now you are ready to do a simple example.

Chapter 3

Examples

Now we do a couple of examples. The first example illustrates basic techniques. The second example is more specialized. Many more examples can be found in the file SYSDOC.doc.

3.1 Simplifying a Complicated Expression

The next example is one most people would be advised to work through.
Note that the output of In[2] below will change based upon which version of NCAlgebra you have. We have not changed this output each time we have a new release. If the example does not work in any way, let us know.

```
In[1]:= <<CEEP

TYPE FILE NAME WHERE YOU WANT THE SESSION SAVED ?  mathsession

In[2]:= <<NCAlgebra.m

NCMultiplication.m loaded
NC1SetCommands.m loaded
NCInverses.m loaded
NCTransposes.m loaded
NCAdjoints.m loaded
NCRoots.m loaded
NC2SetCommands.m loaded
NCSetRule.m loaded
NCPInverses.m loaded
NCCollect.m loaded
NCComplex.m loaded
NCCo.m loaded
NCDoTeX.m loaded
NCSchur.m loaded
NCSubstitute.m loaded
NCMono.m loaded
NCSolve.m loaded
```

```
NCMatMult.m loaded

Setting stubs for NCNagyFoias files.
Done setting stubs for NCNagyFoias files.

NCTools.m loaded
NC2SimplifyRational.m loaded
NC1SimplifyRational.m loaded
NC0SimplifyRational.m loaded
NCSimplifyRational.m loaded
NCDiff.m loaded
NCAlias.m loaded

Setting stubs for Groebner basis files.
Done setting stubs for Groebner basis files.

SYSTEMS not loaded
```

There may be a few changes to what is documented above.

```
In[3]:= <<NCMessyFunction.m
```

NCMessyFunction.m contains several grubby functions. The variable coexz was defined in the file NCMessyFunction.m. Also, a lot of variables are defined to be noncommutative in that file. As an exercise let us simplify the expression named coexz.

```
In[5]:= coexz

Out[5]= -tp[C1] ** C1 - XX ** B2 ** inv[d12] ** C1 -
   tp[C1] ** inv[tp[d12]] ** tp[B2] ** XX -
   XX ** inv[1 - YY ** XX] ** B1 ** inv[d21] ** C2 -
   XX ** inv[1 - YY ** XX] ** B1 ** tp[B1] ** XX +
   inv[YY] ** inv[1 - YY ** XX] ** B1 ** inv[d21] ** C2 +
   inv[YY] ** inv[1 - YY ** XX] ** B1 ** tp[B1] ** XX -
   XX ** B2 ** inv[d12] ** inv[tp[d12]] ** tp[B2] ** XX -
   XX ** inv[1 - YY ** XX] ** YY ** tp[C2] **
   inv[tp[d21]] ** inv[d21] ** C2 -
   XX ** inv[1 - YY ** XX] ** YY ** tp[C2] **
   inv[tp[d21]] ** tp[B1] ** XX +
   inv[YY] ** inv[1 - YY ** XX] ** YY ** tp[C2] ** inv[tp[d21]] **
   inv[d21] ** C2 + inv[YY] ** inv[1 - YY ** XX] ** YY ** tp[C2] **
   inv[tp[d21]] ** tp[B1] ** XX
```

```
In[6]:= NCInverseForward[ coexz ]
```

applies the basic rule

$$XX * * inv[1 - YY * * XX] \rightarrow inv[1 - XX * * YY] * * YY$$

```
Out[6]= -tp[C1] ** C1 - XX ** B2 ** inv[d12] ** C1 -
   tp[C1] ** inv[tp[d12]] ** tp[B2] ** XX -
   inv[1 - XX ** YY] ** XX ** B1 ** inv[d21] ** C2 -
   inv[1 - XX ** YY] ** XX ** B1 ** tp[B1] ** XX +
   inv[1 - XX ** YY] ** inv[YY] ** B1 ** inv[d21] ** C2 +
   inv[1 - XX ** YY] ** inv[YY] ** B1 ** tp[B1] ** XX +
   inv[1 - XX ** YY] ** tp[C2] ** inv[tp[d21]] ** inv[d21] ** C2 +
   inv[1 - XX ** YY] ** tp[C2] ** inv[tp[d21]] ** tp[B1] ** XX -
   XX ** B2 ** inv[d12] ** inv[tp[d12]] ** tp[B2] ** XX -
   inv[1 - XX ** YY] ** XX ** YY ** tp[C2] ** inv[tp[d21]] **
   inv[d21] ** C2 - inv[1 - XX ** YY] ** XX ** YY ** tp[C2] **
   inv[tp[d21]] ** tp[B1] ** XX

In[7]:= NCCollect[ %, B1 ** inv[d21] ** C2 ]
```

Taylor expansion of 6 in the variable $B1 * *inv[d21] * *C2$.

```
TermApplyRules has to loop 1 times. Please be patient.
Finished iteration number 1

Out[7]= -tp[C1] ** C1 - XX ** B2 ** inv[d12] ** C1 -
   (inv[1 - XX ** YY] ** XX - inv[1 - XX ** YY] ** inv[YY]) ** B1 **
   inv[d21] ** C2 - tp[C1] ** inv[tp[d12]] ** tp[B2] ** XX -
   inv[1 - XX ** YY] ** XX ** B1 ** tp[B1] ** XX +
   inv[1 - XX ** YY] ** inv[YY] ** B1 ** tp[B1] ** XX +
   inv[1 - XX ** YY] ** tp[C2] ** inv[tp[d21]] ** inv[d21] ** C2 +
   inv[1 - XX ** YY] ** tp[C2] ** inv[tp[d21]] ** tp[B1] ** XX -
   XX ** B2 ** inv[d12] ** inv[tp[d12]] ** tp[B2] ** XX -
   inv[1 - XX ** YY] ** XX ** YY ** tp[C2] **
   inv[tp[d21]] ** inv[d21] ** C2 -
   inv[1 - XX ** YY] ** XX ** YY ** tp[C2] ** inv[tp[d21]] **
   tp[B1] ** XX

In[8]:= NCC[ %, B1 ** tp[B1] ** XX ]
```

NCC is an alias for NCCollect.

```
TermApplyRules has to loop 1 times. Please be patient.
Finished iteration number 1

Out[8]= -tp[C1] ** C1 - XX ** B2 ** inv[d12] ** C1 -
   (inv[1 - XX ** YY] ** XX - inv[1 - XX ** YY] ** inv[YY]) ** B1 **
   inv[d21] ** C2 - (inv[1 - XX ** YY] ** XX -
   inv[1 - XX ** YY] ** inv[YY]) ** B1 ** tp[B1] ** XX -
   tp[C1] ** inv[tp[d12]] ** tp[B2] ** XX +
   inv[1 - XX ** YY] ** tp[C2] ** inv[tp[d21]] ** inv[d21] ** C2 +
   inv[1 - XX ** YY] ** tp[C2] ** inv[tp[d21]] ** tp[B1] ** XX -
   XX ** B2 ** inv[d12] ** inv[tp[d12]] ** tp[B2] ** XX -
   inv[1 - XX ** YY] ** XX ** YY ** tp[C2] ** inv[tp[d21]] **
   inv[d21] ** C2 - inv[1 - XX ** YY] ** XX ** YY ** tp[C2] **
   inv[tp[d21]] ** tp[B1] ** XX
```

```
In[9]:= Sub[%, inv[1-XX**YY]**XX - inv[1-XX**YY]**inv[YY] -> -inv[YY]]
```

Here we are using a strategy that favors collecting pairs of terms that have similar leftmost inv[1-XX**YY]'s and rightmost non-inv[1-XX**YY] parts. And now, we are substituting the identity:

$$inv[1-a**b]**b - inv[1-a**b]**inv[b] == -inv[b]$$

```
Out[9]= -tp[C1] ** C1 - XX ** B2 ** inv[d12] ** C1 +
   inv[YY] ** B1 ** inv[d21] ** C2 + inv[YY] ** B1 ** tp[B1] ** XX -
   tp[C1] ** inv[tp[d12]] ** tp[B2] ** XX +
   inv[1 - XX ** YY] ** tp[C2] ** inv[tp[d21]] ** inv[d21] ** C2 +
   inv[1 - XX ** YY] ** tp[C2] ** inv[tp[d21]] ** tp[B1] ** XX -
   XX ** B2 ** inv[d12] ** inv[tp[d12]] ** tp[B2] ** XX -
   inv[1 - XX ** YY] ** XX ** YY ** tp[C2] ** inv[tp[d21]] **
   inv[d21] ** C2 - inv[1 - XX ** YY] ** XX ** YY ** tp[C2] **
   inv[tp[d21]] ** tp[B1] ** XX
```

```
In[10]:=SetOutput[dot -> True]
```

This does nothing but beautify the screen output.

```
In[11]:= NCInverseBackward[ %9 ]
```

NCInverseBackward will cause the inv[1-XX**YY] in the last two terms to hide behind the XX**YY, so that when we collect on inv[1-XX**YY] we will be extracting inv[1-XX**YY] and (1-XX**YY) at the same time.

```
Out[11]= -tp[C1] . C1 - XX . B2 . inv[d12] . C1 +
   inv[YY] . B1 . inv[d21] . C2 + inv[YY] . B1 . tp[B1] . XX -
   tp[C1] . inv[tp[d12]] . tp[B2] . XX +
   inv[1 - XX . YY] . tp[C2] . inv[tp[d21]] . inv[d21] . C2 +
   inv[1 - XX . YY] . tp[C2] . inv[tp[d21]] . tp[B1] . XX -
   XX . B2 . inv[d12] . inv[tp[d12]] . tp[B2] . XX -
   XX . YY . inv[1 - XX . YY] . tp[C2] . inv[tp[d21]] . inv[d21] .
   C2 - XX . YY . inv[1 - XX . YY] . tp[C2] . inv[tp[d21]] . tp[B1] .
   XX
```

```
In[12]:= NCC[ %, inv[1-XX**YY] ]

TermApplyRules has to loop 2 times. Please be patient.
Finished iteration number 1
Finished iteration number 2

Out[12]= -tp[C1] . C1 - XX . B2 . inv[d12] . C1 +
   inv[YY] . B1 . inv[d21] . C2 + inv[YY] . B1 . tp[B1] . XX -
   tp[C1] . inv[tp[d12]] . tp[B2] . XX +
   tp[C2] . inv[tp[d21]] . inv[d21] . C2 +
   tp[C2] . inv[tp[d21]] . tp[B1] . XX -
   XX . B2 . inv[d12] . inv[tp[d12]] . tp[B2] . XX
```

We are FINISHED simplifying according to some tastes, because all inv[1-XX**YY] terms are gone. There is ANOTHER WAY which works on some things but not others.

```
In[13]:= coexz

Out[13]= -tp[C1] . C1 - XX . B2 . inv[d12] . C1 -
   tp[C1] . inv[tp[d12]] . tp[B2] . XX -
   XX . inv[1 - YY . XX] . B1 . inv[d21] . C2 -
   XX . inv[1 - YY . XX] . B1 . tp[B1] . XX +
   inv[YY] . inv[1 - YY . XX] . B1 . inv[d21] . C2 +
   inv[YY] . inv[1 - YY . XX] . B1 . tp[B1] . XX -
   XX . B2 . inv[d12] . inv[tp[d12]] . tp[B2] . XX -
   XX . inv[1 - YY . XX] . YY . tp[C2] . inv[tp[d21]] . inv[d21] .
   C2 - XX . inv[1 - YY . XX] . YY . tp[C2] . inv[tp[d21]] . tp[B1] .
   XX + inv[YY] . inv[1 - YY . XX] . YY . tp[C2] . inv[tp[d21]] .
   inv[d21] . C2 + inv[YY] . inv[1 - YY . XX] . YY . tp[C2] .
   inv[tp[d21]] . tp[B1] . XX

In[14]:= NCSimplifyRational[ coexz ]

Out[14]= -tp[C1] . C1 - XX . B2 . inv[d12] . C1 +
   inv[YY] . B1 . inv[d21] . C2 + inv[YY] . B1 . tp[B1] . XX -
   tp[C1] . inv[tp[d12]] . tp[B2] . XX +
   tp[C2] . inv[tp[d21]] . inv[d21] . C2 +
   tp[C2] . inv[tp[d21]] . tp[B1] . XX -
   XX . B2 . inv[d12] . inv[tp[d12]] . tp[B2] . XX
```

WHAMMO! We got the answer Out[14] in one shot. The function NCSimplifyRational is based on automatically applying a carefully chosen collection of rules like NCBackward and In[9] repeatedly. Rules can be added so it should get better and better (that is, work well on many expressions.) Now we turn to condensing the expression. It is a good exercise.

```
In[15]:= NCC[ %14, tp[C2] ** inv[tp[d21]]]

TermApplyRules has to loop 1 times. Please be patient.
Finished iteration number 1

Out[15]= -tp[C1] . C1 + tp[C2] . inv[tp[d21]] .
   (inv[d21] . C2 + tp[B1] . XX) - XX . B2 . inv[d12] . C1 +
   inv[YY] . B1 . inv[d21] . C2 + inv[YY] . B1 . tp[B1] . XX -
   tp[C1] . inv[tp[d12]] . tp[B2] . XX -
   XX . B2 . inv[d12] . inv[tp[d12]] . tp[B2] . XX

In[16]:= NCC[ %, inv[YY] ** B1 ]

TermApplyRules has to loop 1 times. Please be patient.
Finished iteration number 1

Out[16]= -tp[C1] . C1 + inv[YY] . B1 . (inv[d21] . C2 + tp[B1] . XX) +
```

```
   tp[C2] . inv[tp[d21]] . (inv[d21] . C2 + tp[B1] . XX) -
   XX . B2 . inv[d12] . C1 - tp[C1] . inv[tp[d12]] . tp[B2] . XX -
   XX . B2 . inv[d12] . inv[tp[d12]] . tp[B2] . XX

In[17]:= NCC[ %, inv[tp[d12]] ]

TermApplyRules has to loop 2 times. Please be patient.
Finished iteration number 1
Finished iteration number 2

Out[17]= -tp[C1] . C1 + inv[YY] . B1 . (inv[d21] . C2 + tp[B1] . XX) +
   tp[C2] . inv[tp[d21]] . (inv[d21] . C2 + tp[B1] . XX) -
   XX . B2 . inv[d12] . C1 -
   (XX . B2 . inv[d12] + tp[C1]) . inv[tp[d12]] . tp[B2] . XX

In[18]:= NCC[ %, C1 ]

TermApplyRules has to loop 2 times. Please be patient.
Finished iteration number 1
Finished iteration number 2

Out[18]= -(XX . B2 . inv[d12] + tp[C1]) . C1 +
   inv[YY] . B1 . (inv[d21] . C2 + tp[B1] . XX) +
   tp[C2] . inv[tp[d21]] . (inv[d21] . C2 + tp[B1] . XX) -
   (XX . B2 . inv[d12] + tp[C1]) . inv[tp[d12]] . tp[B2] . XX

In[19]:= factor = %[[ 1,2,1 ]]
```

This is standard Mathematica, picking a part of an expression.

```
Out[19]= XX . B2 . inv[d12] + tp[C1]

In[20]:= NCC[ %%, factor ]

TermApplyRules has to loop 2 times. Please be patient.
Finished iteration number 1
Finished iteration number 2

Out[20]= -(XX . B2 . inv[d12] + tp[C1]) .
   (C1 + inv[tp[d12]] . tp[B2] . XX) +
   inv[YY] . B1 . (inv[d21] . C2 + tp[B1] . XX) +
   tp[C2] . inv[tp[d21]] . (inv[d21] . C2 + tp[B1] . XX)

In[21]:= factor = %[[ 2,3 ]]

Out[21]= inv[d21] . C2 + tp[B1] . XX

In[22]:= NCC[ %%, factor ]

TermApplyRules has to loop 2 times. Please be patient.
Finished iteration number 1
```

```
Finished iteration number 2

Out[22]= (inv[YY] . B1 + tp[C2] . inv[tp[d21]]) .
   (inv[d21] . C2 + tp[B1] . XX) -
   (XX . B2 . inv[d12] + tp[C1]) . (C1 + inv[tp[d12]] . tp[B2] . XX)
```

This is about as good as I can do.

3.2 Vector Differentiation

We now do a demo where we optimize a function denoted sHWU in its variables W and U. We do this by setting the gradients of the sHWU in W, U to 0 and solving for 0.

BEWARE YOU CAN NOT WORK THROUGH THIS DEMO YOURSELF UNLESS YOU HAVE INSTALLED OUR SYSTEMS PACKAGE.

DO NOT GET NCAlgebra.m BEFORE LOADING SYStems.m since SYStems.m calls NCAlgebra.m

```
In[3]:= <<SYStems.m
```

Load in the function 'sHWU' and a rule 'rue' associated with it. Ignore the printout.

```
In[4]:= sHWU

Out[4]= -tp[W] ** W + (tp[U] ** tp[D12[x]] + tp[C1[x]]) **
  (C1[x] + D12[x] ** U) + ((tp[U] ** tp[B2[x]] + tp[W] ** tp[B1[x]] +
         tp[A[x]]) ** GEx[x, z] +
      tp[GEx[x, z]] ** (A[x] + B1[x] ** W + B2[x] ** U))/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

In[5]:= DirD[%,W,h]
```

Use directional derivative to calculate the gradient of sHWU in W(safer than Grad). Next we shall set to 0 and solve for W.

```
Out[5]= -(tp[W] ** h + tp[h] ** W) +
   (tp[h] ** tp[B1[x]] ** GEx[x, z] + tp[GEx[x, z]] ** B1[x] ** h)/2 +
   (tp[h] ** tp[D21[x]] ** tp[b[z]] ** GEz[x, z] +
      tp[GEz[x, z]] ** b[z] ** D21[x] ** h)/2

In[6]:= NCCSym[NCE[%],h]
```

Collect on h and tp[h].

```
"TermApplyRules has to loop "2" times. Please be patient."
"Finished iteration number "1
"Finished iteration number "2
```

```
Out[6]= (tp[GEx[x, z]] ** B1[x]/2 + tp[GEz[x, z]] ** b[z] ** D21[x]/2 -
      tp[W]) ** h - tp[h] ** (W - tp[B1[x]] ** GEx[x, z]/2 -
      tp[D21[x]] ** tp[b[z]] ** GEz[x, z]/2)

In[7]:= %//.tp[h] -> 0
```

Need to keep only half of symmetry when setting expression equal to 0.

```
Out[7]= (tp[GEx[x, z]] ** B1[x]/2 + tp[GEz[x, z]] ** b[z] ** D21[x]/2 -
     tp[W]) ** h

In[8]:= %//.h -> 1
```

Set h to 1 to pull off the operator (gradient).

```
Out[8]= tp[GEx[x, z]] ** B1[x]/2 + tp[GEz[x, z]] ** b[z] ** D21[x]/2 - tp[W]

In[9]:= tp[%]

Out[9]= -W + tp[B1[x]] ** GEx[x, z]/2 + tp[D21[x]] ** tp[b[z]] ** GEz[x, z]/2

In[10]:= NCSolve[%9==0,W]

NCCollect::notsum: Expression is not a sum of terms.

Out[10]= {W -> tp[B1[x]] ** GEx[x, z]/2 +
     tp[D21[x]] ** tp[b[z]] ** GEz[x, z]/2}
```

Here is the answer, the stuff above is all diagnostics.

```
In[11]:= sHU = NCE[Sub[sHWU, ruCRW]]

Out[11]= 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[U] ** tp[B2[x]] ** GEx[x, z]/2 +
   tp[U] ** tp[D12[x]] ** C1[x] + tp[C1[x]] ** D12[x] ** U +
   tp[C2[x]] ** tp[b[z]] ** GEz[x, z]/2 +
   tp[GEx[x, z]] ** B2[x] ** U/2 +
   tp[GEz[x, z]] ** b[z] ** C2[x]/2 +
   tp[U] ** tp[D12[x]] ** D12[x] ** U +
   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] ** D21[x] ** tp[D21[x]] **
   tp[b[z]] ** GEz[x, z]/4
```

The W problem is solved.

```
In[12]:= DirD[%,U,h]
```

Now we want critical U, using the same approach.

```
Out[12]= tp[h] ** tp[B2[x]] ** GEx[x, z]/2 + tp[h] ** tp[D12[x]] ** C1[x] +
   tp[C1[x]] ** D12[x] ** h + tp[GEx[x, z]] ** B2[x] ** h/2 +
   tp[U] ** tp[D12[x]] ** D12[x] ** h + tp[h] ** tp[D12[x]] ** D12[x] ** U

In[13]:= NCCSym[%,h]
"TermApplyRules has to loop "2" times. Please be patient."
"Finished iteration number "1
"Finished iteration number "2

Out[13]= (tp[C1[x]] ** D12[x] + tp[GEx[x, z]] ** B2[x]/2 +
      tp[U] ** tp[D12[x]] ** D12[x]) ** h +
   tp[h] ** (tp[B2[x]] ** GEx[x, z]/2 + tp[D12[x]] ** C1[x] +
      tp[D12[x]] ** D12[x] ** U)

In[14]:= %[[1]]+tp[%[[2]]]
```

This time use a trick to keep the whole expression, first part + transpose[second].

```
Out[14]= 2*(tp[C1[x]] ** D12[x] + tp[GEx[x, z]] ** B2[x]/2 +
      tp[U] ** tp[D12[x]] ** D12[x]) ** h

In[15]:= NCE[%]

Out[15]= 2*tp[C1[x]] ** D12[x] ** h + tp[GEx[x, z]] ** B2[x] ** h +
   2*tp[U] ** tp[D12[x]] ** D12[x] ** h

In[16]:= %//.h -> 1
```

Again pull off the operator.

```
Out[16]= 2*tp[C1[x]] ** D12[x] + tp[GEx[x, z]] ** B2[x] +
   2*tp[U] ** tp[D12[x]] ** D12[x]

In[17]:= rue
```

We shorten the expression with a change of notation before setting to 0.

```
Out[17]= {tp[D12[x_]] ** D12[x_] :> e1[x],
                       D21[x_] ** tp[D21[x_]] :> e2[x]}

In[18]:= Sub[%%,rue]

Out[18]= 2*e1[x] ** U + tp[B2[x]] ** GEx[x, z] +
                                    2*tp[D12[x]] ** C1[x]

In[19]:= ruCritU=NCSolve[%==0,U]

NCCollect::notsum: Expression is not a sum of terms.

Out[19]= {U -> -inv[e1[x]] ** tp[B2[x]] ** GEx[x, z]/2 -
```

```
     inv[e1[x]] ** tp[D12[x]] ** C1[x]}

In[20]:= sH = sHU//.ruCritU
```

Here is the answer:

```
Out[20]= 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] ** inv[e1[x]] -
      tp[GEx[x, z]] ** B2[x] ** inv[e1[x]]/2) **
  tp[B2[x]] ** GEx[x, z]/2 +
  (-tp[C1[x]] ** D12[x] ** inv[e1[x]] -
  tp[GEx[x, z]] ** B2[x] ** inv[e1[x]]/2) ** tp[D12[x]] ** C1[x] +
  tp[C1[x]] ** D12[x] ** (-inv[e1[x]] ** tp[B2[x]] ** GEx[x, z]/2 -
      inv[e1[x]] ** tp[D12[x]] ** C1[x]) +
  tp[C2[x]] ** tp[b[z]] ** GEz[x, z]/2 +
  tp[GEx[x, z]] ** B2[x] ** (-inv[e1[x]] ** tp[B2[x]] ** GEx[x, z]/2 -
       inv[e1[x]] ** tp[D12[x]] ** C1[x])/2 +
  tp[GEz[x, z]] ** b[z] ** C2[x]/2 +
  (-tp[C1[x]] ** D12[x] ** inv[e1[x]] -
  tp[GEx[x, z]] ** B2[x] ** inv[e1[x]]/2) ** tp[D12[x]] ** D12[x] **
  (-inv[e1[x]] ** tp[B2[x]] ** GEx[x, z]/2 -
      inv[e1[x]] ** 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] ** D21[x] ** tp[D21[x]] **
  tp[b[z]] ** GEz[x, z]/4
```

Chapter 4

Strategies

4.1 Edit

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 editor or whatever is set up. Then exiting the file throws your edited expression into the Mathematica Out[102]. 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.

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

4.2 Expanding

NCExpand is the most common command applied to expressions. Often you must do it to achieve the most obvious cancellations.

4.3 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 and see which terms cancel.

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_]:=Sub[expr, MyRules];
```

One of the tricker 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 letters and do not in some equations replace them with something else. Clearly using blanks is much more powerful. The trick is how many. For example, x___ is the best choice here. APPENDIX E discusses this.

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

This translates to the rule

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

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

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.
```

ORDERING ON POLYS:

```
We use the one imposed by Mathematica.
```

REMEMBER HIGHER ORDER EXPRESSIONS MOVE RIGHT.

4.4 Conventions

The NCAlgebra files which are called by NCAlgebra.m start with NC. This makes moving them easier; cp NC* someplace/. Many operations on expressions start with NC .

Aliases are all caps like NCC or NCE. 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 5

NC Commands

5.1 Manipulating an expression

5.1.1 ExpandNonCommutativeMultiply[expr]

Aliases: **NCE,NCExpand**

Description: ExpandNonCommutativeMultiply[expr] expands out NonCommutativeMultiply's in expr. That is, it distributes NonCommutativeMultiply's over + (Plus).

Arguments: expr is an algebraic expression.

Comments \ Limitations: None

5.1.2 NCCollect[expr, aListOfVariables]

Aliases: **NCC**

Description: Collects terms of expression expr according to the elements of aListOfVariables and attempts to combine them using rulesCollect. It is weaker than the NCStrongCollect in that first-order and second-order terms are not collected together. It basically is NCDecompose→ NCStrongCollect→NCCompose.

Arguments: expr is an algebraic expression. aListOfVariables is a list of variables (symbols) of indefinite length.

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.

5.1.3 NCStrongCollect[expr, aListOfVariables]

Aliases: **NCSC**

Description: Collects terms of expression expr according to the elements of aListOfVariables and attempts to combine them using rulesCollect. In the noncommutative case the Taylor expansion and so the collect function is not uniquely specified. This collect function often collects too much and while correct is 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 (symbols) of indefinite length.

Comments \ Limitations: Not documented yet.

5.1.4 NCCollectSymmetric[expr]

Aliases: **NCCSym**

Description: None

Arguments: expr is an algebraic expression.

Comments \ Limitations: None

5.1.5 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, $NCTermsOfDegreeh[x**y**x+x**x**y+x**x+x**w, \{x,y\}, indices]$ 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

5.1.6 NCBackward[expr]

Aliases: **NCB**

Description: apply the rules

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

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

Arguments: expr is an algebraic expression.

Comments \ Limitations: None

5.1.7 NCForward[expr]

Aliases: **NCF**

Description: apply the rules

$$B**inv[Id-AB] \rightarrow inv[Id-BA]**B$$

$$inv[B]**inv[Id-BA] \rightarrow inv[Id-BA]**inv[A]$$

Arguments: expr is an algebraic expression.

Comments \ Limitations: None

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

Aliases: **None**

Description: Replaces one symbol or sub-expression in the expression by another expression as specified by the rule. (See Wolfram 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-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 //.

5.1.9 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

5.1.10 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 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 /.

5.1.11 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

5.1.12 Transform[expr,aListOfRules]

Aliases: **None**

Description: A more efficient version of Substitute.

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

Comments \ Limitations: None.

5.1.13 NCMono[expr]

Aliases: **None**

Description: NCMono 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, $NCMono[2x^2 + 5x^4]$ evaluates to $2x**x + 5x**x**x**x$ and $NCMono[(x^2)**z**x]$ 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 $x = x**x**x**x**x**x**x**x**x**x**x**x**y**x**x$, 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 NCMono 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 NCMono.

5.2 Manipulating an Expression — less useful commands

5.2.1 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

Out[22]= 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
```

5.2.2 NCReconstructFromTermArray[anArray]

Aliases: **None**

Description: $NCTermArray[expr, aList, anArray]; newexpr = NCReconstructFromTermArr$
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
```

5.2.3 NCCompose[aVerySpecialList]

Aliases: **NCCom compose**

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.**

5.2.4 NCDecompose

Aliases: **NCDec decompose**

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.**

5.2.5 NCCombine[expr]

Aliases: **None**

Description: Groups the terms in 'expr" with the same inv[] together.

Arguments: expr is an algebraic expression

Comments \ Limitations: Called withing NCCollect. **The average user would never use this.**

5.3 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 rules we are implementing come from papers of Helton Stankus and Wavrik.

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,NCS0R. 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 $inv[S]**inv[1 - \frac{(a**b)}{S}]$ 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.

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

Aliases: **NCSR NCS1R NCS2R**

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 specialized class of functions.

5.3.2 NCSimplify1Rational[expr]

Aliases: **NCS1R**

Description: 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 $inv[S]**inv[1 - \frac{(a**b)}{S}]$ 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 paper **[HW]**,**[HSW]**, these rules implement a superset of the union of the Gröbner basis for EB and the Gröbner basis for RESOL.

5.3.3 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.

5.3.4 NCSimplify0Rational[expr]

Aliases: **NCS0R**

Description: You do not use this one much. Ignore. First apply the function 'NCCombine' to the expression input. 'NCCombine' will group the terms with the same inv[] together. Then this function(NCSimplifyRational) will look for the pattern inv[] in the expression. Everything in inv[] will be replaced by a variable. All the variables in the 'everything' will be collected into a list and will be expressed in terms of the variables used to replace them. Then an ExpandNonCommutativeMultiply is done. The 'NCCombine' function is applied again. Finally, the original expressions are substituted back into the inv[]s.

Arguments: expr is an algebraic expression.

Comments \ Limitations: None

5.4 Vector Differentiation

5.4.1 DirectionalD[expr, aVariable, h]

Aliases: **DirD**

Description: Takes the Directional Derivative of expression 'expr' w.r.t. 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.

5.4.2 Gradient[expr, aVariable]

Aliases: **Grad**

Description: Takes the Gradient of expression 'expr' w.r.t. the variable 'aVariable'. Quite useful for computations with quadratic Hamiltonians in H^∞ control.

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.

5.4.3 CriticalPoint[expr, aVariable]

Aliases: **Crit, Cri**

Description: 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 Gradient and NCSolve functions.

5.5 Block Matrix Multiplication

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.

5.5.1 MatMult[x, y, ...]

Aliases: **MM**

Description: Multiplies matrices. The Mathematica code executed 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]]].

5.5.2 tpMat[u]

Aliases: **tpM**

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

Arguments: u is a block m×n matrix

Comments \ Limitations: None

5.5.3 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

5.5.4 TimesToNCM[expr]

Aliases: **TTNCM**

Description: The Mathematica code executed is expr/.Times → NonCommutative-Multiply

Arguments: expr is an algebraic expression.

Comments \ Limitations: It changes commutative multiplication (Times) to Non-Commutative multiplication.

5.5.5 Partitioning a Matrix into a Block Matrix

We have developed the file NCBlockPartition.m to do matrix partitioning. The easiest way to explain how it works is through an example. If one has a 3×3 matrix

$$\begin{bmatrix} 3 & 6 & 9 \\ 1 & 2 & 4 \\ 5 & 9 & 1 \end{bmatrix}$$

then one might say that it equals

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

where

A is the 1×1 matrix containing the entry 3
B is the 1×2 matrix containing the entries 6 and 9
C is the 2×1 matrix containing the entries 1 and 5
D is the 2×2 matrix containing the entries 2, 4, 9, and 1. One would do the above using the command

```
PartitionMatrix[{{3,6,9}, {1,2,4}, {5,9,1}}, 1, 1]
```

(Note: the second parameter is the number of rows of the upper left hand and the third parameter is the number of columns of the upper left hand block. That is, A is a 1×1 matrix.)

and the output would have the form

```
{{$1,$2},{$3,$4}}
```

and would have the consequence of typing ?ValueQ and ?BlockQ

```
ValueOf[$1] = {{3}}
ValueOf[$2] = {{6,9}}
ValueOf[$3] = {{1},{5}}
ValueOf[$4] = {{2,4},{9,1}}

BlockQ[$1] = True
BlockQ[$2] = True
BlockQ[$3] = True
BlockQ[$4] = True
```

One can reconstruct the original matrix by executing

```
FormMatrix[{{$1,$2},{$3,$4}}]
```

If one does not want a '$", then one can use a different 'prefix" by adding a fourth parameter to the call to PartitionMatrix. The numbers following the prefix are generated by the 'Unique" command. We hope that their uglyness is not offensive.

We use the ValueOf function to associate the value of the block to the block without actually assigning it. If it were assigned immediately, then the assigned value would be dragged around instead of the variable representing the entry. If one is partitioning a 100×100 matrix into a 2×2 block matrix, then delaying the assignment makes following the calculation much simpler.

The BlockQ command tells the user if the variable represents a block matrix and is assigned within PartitionMatrix via the SetBlock command. At this time, BlockQ is never used. The intention is to provide it at this point for future expansion.

Lastly, note that

```
FormMatrix[PartitionMatrix[x, m, n]]
```

will give back x (as long as m and n are less than the dimensions of the matrix x, of course).

5.5.6 invMat2[mat]

Aliases: **None**

Description: This is the formula for the inverse of a 2×2 block matrix.

Arguments: mat is a 2×2 block matrix.

Comments \ Limitations: It assumes invertibility of everything in sight.

5.5.7 GaussElimination[X]

Aliases: **GauE**

Description: This is the formula for 2×2 block gauss elimination. It returns 3 block 2×2 matrices which are the block LDU decomposition of X.

Arguments: X is a block 2×2 matrix.

Comments \ Limitations: If X is the block 2×2 matrix of the form $\{\{a, b\}, \{c, d\}\}$, then we assume that a is invertible. If you want to assume that d is invertible, type $GaussElimination[X, btm]$. $GaussElimination[X]$ is the same as $GaussElimination[X, top]$. Note the Schur complement for X can be extracted from the diagonal matrix D, so we did not make it a separate function.

5.5.8 Cascade[P, K]

Aliases: **None**

Description: Cascade 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.

5.5.9 Chain[P]

Aliases: **None**

Description: The chain matrix arising from P as is found is systems engineering.

Arguments: P is a block 2×2 matrix.

Comments \ Limitations: Assumes appropriate matrices are invertible.

5.5.10 Redheffer[P]

Aliases: **None**

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

Arguments: P is a block 2×2 matrix.

Comments \ Limitations: It assumes the invertiblity of everything in site.

5.5.11 DilationHalmos[x]

Aliases: **None**

Description: Halmos dilation

Arguments: x is a symbol.

Comments \ Limitations: It has the property ut = tpMat[u].

5.6 Complex Analysis

5.6.1 A tutorial

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

- •ComplexRules, a transformation rules
- •ComplexCoordinates, a function that applies rules to expr.
- •ComplexD[], a complex differentiation function.

This package is for handling complex algebra and differentiation. 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 rule 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.

5.6.2 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.

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

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

5.6.4 ComplexD[expr, aVariable]

Aliases: **None**

Description: ComplexD[expr, aVariable] calculates the derivative of the complex "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.

5.7 Setting symbols to commute or not commute

5.7.1 SetNonCommutative[a, b, c, ...]

Aliases: **SNC SetNC**

Description: sets all the symbols a, b, c, ... to be non-commutative.

Arguments: Symbols

Comments \ Limitations: None

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

Aliases: **SC SetC**

Description: sets all the symbols a, b, c, ... to be commutative.

Arguments: Symbols

Comments \ Limitations: None

5.7.3 CommuteEverything[expr]

Aliases: **CE**

Description: Changes NonCommutativeMultiply to Times.

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 unaffected, so you can get back to working on it arned with new ideas.

5.7.4 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).

5.7.5 LeftQ[expr]

Aliases: **None**

Description: See SetCommutingFunctions and SetCommutingOperators.

Arguments: expr is an algebraic expression.

Comments \ Limitations: None

5.7.6 CommutativeQ[x]

Aliases: **CQ**

Description: CommutativeQ[x] is True if x is commutative (the default), and False if x is non-commutative.

Arguments: x is a symbol.

Comments \ Limitations: None

5.7.7 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

5.8 Setting Properties of an element in an algebra

5.8.1 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.

5.8.2 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.

5.8.3 invR[x]

Aliases: **invR**

Description: 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.

5.8.4 invQ[x]

Aliases: **None**

Description: invQ[m] = True makes invR[m]=invL[m]

Arguments: x is an expression.

Comments \ Limitations: None

5.8.5 ExpandQ[inv]

Aliases: **None**

Description: If set to True, an inverse of a product will be expanded to a product of inverses. If set to False, then a product of inverses will be rewritten to be a inverse of a product.

Arguments: inv

Comments \ Limitations: None

5.8.6 ExpandQ[tp]

Aliases: **None**

Description: If set to True, a transpose of a product will be expanded to a product of transposes. If set to False, then a product of transposes will be rewritten to be a transpose of a product.

Arguments: tp

Comments \ Limitations: None

5.8.7 OverrideInverse

Aliases: **OI**

Description: This is a variable which is either True or False.

Arguments: If 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

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

Aliases: **None**

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

Arguments: Symbols

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.

5.8.9 SetSelfAdjoint[Symbols]

Aliases: **SSA SetSA**

Description: SetSelfAdjoint[a, b, ...] will set a, b, ... to be self-adjoint. If set 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

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.

5.8.10 SelfAdjointQ[aSymbol]

Aliases: **SAQ**

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

Arguments: aSymbol is an algebraic expression

Comments \ Limitations: None

5.8.11 SetIsometry[Symbols]

Aliases: **SI SetI**

Description: SetIsometry[a, b, ...] 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.

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.

5.8.12 IsometryQ[aSymbol]

Aliases: **IQ**

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

Arguments: aSymbol is an algebraic expression.

Comments \ Limitations: None

5.8.13 SetCoIsometry[Symbols]

Aliases: **SCI SetCI**

Description: SetCoIsometry[a, b, ...] will set a, b, ... to be co-isometries. If set 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.

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.

5.8.14 CoIsometryQ[aSymbol]

Aliases: **CIQ**

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

Arguments: aSymbol is an algebraic expression.

Comments \ Limitations: None

5.8.15 SetUnitary[Symbols]

Aliases: **SU SetU**

Description: SetUnitary[a, b, ...] 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.

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.

5.8.16 UnitaryQ[aSymbol]

Aliases: **UQ**

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 an algebraic expression.

Comments \ Limitations: None

5.8.17 SetProjection[Symbols]

Aliases: **SP SetP**

Description: SetProjection[a, b, ...] will set a, b, ... to be projections. If set 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.

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.

5.8.18 ProjectionQ[Symbol]

Aliases: **PQ**

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

Arguments: Symbol is an algebraic expression.

Comments \ Limitations: None

5.8.19 SetSignature[Symbols]

Aliases: **SSIG SetSIG**

Description: If set as well as SelfAdjoint, the rule a ** a := -1 will be automatically applied. See SetSelfAdjoint and SignatureQ.

Arguments: Symbols is one or more symbols.

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.

5.8.20 SignatureQ[Symbol]

Aliases: **SIGQ**

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

Arguments: Symbol is an algebraic expression.

Comments \ Limitations: None

5.9 Setting Properties of functions on an algebra

5.9.1 SetSesquilinear[Functions]

Aliases: **SetSesq**

Description: SetSesquilinear[a, b, c, ...] 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.

Comments \ Limitations: None

5.9.2 SetSesq[Functions]

Aliases: **None**

Description: Synonym for SetSesquilinear.

Arguments: Functions is one or more symbols.

Comments \ Limitations: None

5.9.3 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

5.9.4 SetBilinear[Functions]

Aliases: **None**

Description: SetBilinear[a, b, c, ...] 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.

Comments \ Limitations: None

5.9.5 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

5.9.6 SetLinear[Functions]

Aliases: **None**

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

Arguments: Functions is one or more symbols.

Comments \ Limitations: None

5.9.7 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

5.9.8 SetConjugateLinear[Functions]

Aliases: **None**

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

Arguments: Functions is one or more symbols.

Comments \ Limitations: None

5.9.9 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

5.9.10 SetIdempotent[Functions]

Aliases: **None**

Description: SetIdempotent[b, c, d,...] 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.

Comments \ Limitations: None

5.9.11 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

5.9.12 SetCommutingFunctions[Functions]

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: Functions is one or more symbols.

Comments \ Limitations: None

5.9.13 SetNonCommutativeMultiplyAntihomomorphism[Functions]

Aliases: **NCAntihomo**

Description: SetNonCommutativeMultiplyAntihomomorphism[b, c, d, . . .] 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.

Comments \ Limitations: None

5.9.14 NCAntihomo[Functions]

Aliases: **None**

Description: Synonym for SetNonCommutativeMultiplyAntihomomorhism.

Arguments: Functions is one or more symbols.

Comments \ Limitations: None

5.10 Utilities

5.10.1 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 NCSubstitute 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 NCSubstitute commands, but any NCSubstitute 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 → "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.

5.10.2 SaveRulesQ[]

Aliases: **SaveRQ**

Description: Indicates the status of the continuous recording feature of the NCSubstitute 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].

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

Aliases: **FORules**

Description: Maps function1 onto the Lhs and function2 onto the Rhs 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, or user-defined function.

Comments \ Limitations: None

Part II

Output Formats

Chapter 6

Text Oriented Output Commands

6.0.4 SetOutput[optionlist,. . .]

Aliases: **None**

Description: SetOutput displays non-commutative 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,

- opt → 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 '**'
- rt → True, rt[X] is displayed as $X^{\frac{1}{2}}$
- rt → False, rt[X] is displayed as rt[x]
- 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: You can turn on or off options individually at any time. The nesting order of the final display is determined first by any NC-Algebra-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 $(x**y)^2$.

6.0.5 NCSave[aString, expr]

Aliases: **None**

Description: saves an expression to a file and all of the environmental settings for the variables in the expressions expr. sometimes gives cancellations.

Arguments: expr is a mathematical expression. aString is a string containing the name of the file to be outputted too.

Comments \ Limitations: Do not use with the commands TeXForm or MatrixForm, . . . which give weird output.

6.0.6 CEEP

Aliases: **None**

Description: After one types

$$<< CEEP,$$

one can record what occurs on the screen into a file. The CEEP program asks you for a name of a file where you want to deposit the rest of your session. It also creates a second file which stores just the commands which you type in.

Arguments: None

Comments \ Limitations: None

Chapter 7

Interfacing with TeX

In Mathematica, if expr is a Mathematica expression, then TeXForm[expr] yields a string which, when given to TeX with a minimal amount of surrounding TeX , will allow you to generate a "dvi" file which can be outputted to a printer or, in some cases, shown on a terminal screen. This allows one to decipher a Mathematica expression visually. In the NCAlgebra package we have "expanded" this capacity in the following ways:

(1) The output conventions for the new "heads" we have generated are processed correctly. For example, TeXForm[tp[x]] will give "x^T".

(2) Automation of the process of putting the surrounding TeXcode mentioned above and the TeX code generated from a number of expressions to a file. The file is ready to be LaTeXed. This allows one to be able to get started without knowing much TeX.

(3) Allowing one to TeX an expression "on the fly" and put it up in its own window.

(4) Adding some conventions which seem better suited to TeX(e.g., processing 1,2,3,4 so that it will look like a matrix, rather than a list of lists.)

The relevant files are (1) NCTeXForm.m, (2,3) NCDoTeX.m and (4) TeXForm.Extra.

In general, the ".Extra" suffix means that we are not willing yet willing to support maintance on this code. This code is TeXForm.Extra is stable – it is given the "*.Extra" suffix because it supports TeXForm of "*.Extra" files. See the file Extra.doc for more on Extra.TeXForm.

Note: The code has problems with complicated expressions such as $tp[aj[x]]$. When encountering such problems, it is necessary to go into the TeXfile and see what went wrong. In the above example, the output is

```
x^{*}^{T}
```

where the output was suppose to be

```
x^{*T}
```

7.1 Some Examples

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

In[111]:= <<Extra.TeXForm;LookAtMatrix[mat]
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 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). This window contains the following formula in its display.

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

The LaTeXfile generated is shown below.

```
In[112]:= !!file.tex
\documentstyle [12pt]{article}
\begin{document}
$$
\pmatrix { a + b^T & c^{*} \cr d^{-1} & e \cr }
$$
\end{document}

In[113]:= LookAtLongExpression[Sum[a[i],{i,1,30}]];
```

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

$$a(1) + a(2) + a(3) + a(4) + a(5) + a(6) + a(7) + a(8)+$$

$$a(9) + a(10) + a(11) + a(12) + a(13) + a(14) + a(15)$$

$$+a(16) + a(17) + a(18) + a(19) + a(20) + a(21)+$$

$$a(22) + a(23) + a(24) + a(25) + a(26) + a(27)+$$

$$a(28) + a(29) + a(30)$$

The commands LongExpressionToTeXFile deposits TeX (not LaTeX) into a TeX file, but does not display anything to the screen. One can call this function repeatedly and it will append text to the file. For example,

```
In[1]:= LongExpressionToTeXFile["junk.tex",Sum[tp[a[n]],{n,1,40}]];
In[2]:= !!junk.tex
$$
\displaylines{
a(1)^T+a(2)^T+a(3)^T+a(4)^T+a(5)^T+ \cr
a(6)^T+a(7)^T+a(8)^T+a(9)^T+a(10)^T+ \cr
a(11)^T+a(12)^T+a(13)^T+a(14)^T+a(15)^T+ \cr
a(16)^T+a(17)^T+a(18)^T+a(19)^T+a(20)^T+ \cr
a(21)^T+a(22)^T+a(23)^T+a(24)^T+a(25)^T+ \cr
a(26)^T+a(27)^T+a(28)^T+a(29)^T+a(30)^T+ \cr
a(31)^T+a(32)^T+a(33)^T+a(34)^T+a(35)^T+ \cr
a(36)^T+a(37)^T+a(38)^T+a(39)^T+a(40)^T \cr
}
$$

In[2]:= <<Extra.TeXForm;
In[3]:= LongExpressionToTeXFile["junk.tex",
                                OutputAMatrix[{{a,tp[b]},{aj[c],d}}]];
In[4]:= !!junk.tex
$$
\displaylines{
a(1)^T+a(2)^T+a(3)^T+a(4)^T+a(5)^T+ \cr
a(6)^T+a(7)^T+a(8)^T+a(9)^T+a(10)^T+ \cr
a(11)^T+a(12)^T+a(13)^T+a(14)^T+a(15)^T+ \cr
a(16)^T+a(17)^T+a(18)^T+a(19)^T+a(20)^T+ \cr
a(21)^T+a(22)^T+a(23)^T+a(24)^T+a(25)^T+ \cr
a(26)^T+a(27)^T+a(28)^T+a(29)^T+a(30)^T+ \cr
a(31)^T+a(32)^T+a(33)^T+a(34)^T+a(35)^T+ \cr
a(36)^T+a(37)^T+a(38)^T+a(39)^T+a(40)^T \cr
}
$$
$$
\displaylines{
```

```
\pmatrix { a & b^T \cr c^{*} & d \cr } \cr
}
$$
```

which when displayed will look like

$$\begin{gathered}a(1)^T + a(2)^T + a(3)^T + a(4)^T + a(5)^T + \\ a(6)^T + a(7)^T + a(8)^T + a(9)^T + a(10)^T + \\ a(11)^T + a(12)^T + a(13)^T + a(14)^T + a(15)^T + \\ a(16)^T + a(17)^T + a(18)^T + a(19)^T + a(20)^T + \\ a(21)^T + a(22)^T + a(23)^T + a(24)^T + a(25)^T + \\ a(26)^T + a(27)^T + a(28)^T + a(29)^T + a(30)^T + \\ a(31)^T + a(32)^T + a(33)^T + a(34)^T + a(35)^T + \\ a(36)^T + a(37)^T + a(38)^T + a(39)^T + a(40)^T\end{gathered}$$

$$\begin{pmatrix} a & b^T \\ c^{*} & d \end{pmatrix}$$

If the above examples are enough for you to work with, then don't bother reading the next two chapters.

Chapter 8

List of TeX Commands

We give a list of TeX related commands. These commands may be used both interactively and in a Mathematica program. The function which is probably of the most use is the command LongExpressionToTeXFile. For a simpler (but more limited) approach to creating TeX files, see the description of DoTeX on page 77.

The bulk of this section should probably be put off to the second reading.

8.0.1 LongExpressionToTeXFile[aString_, Expressions___]

Aliases: **None**

Description: Output an expression to the TeXFile as a displayed equation. If the expression is long, then the expression is split between different lines. Some of the expressions may be strings.

Arguments: aString is NOT optional. Expression is one or more expressions. aString should end in ".tex". Often, one would use "file.tex".

Comments \ Limitations: See DoTeX, OpenTeXFile, CloseTeXFile, ExprToTeXFile, ExpressionToTeXFile, StartParagraphTeXFile, CommentToTeXFile, InLineExpressionToTeXFile, EqualityToTeXFile, AddSplitsTeX, TeXTheFile.

8.0.2 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

8.0.3 OpenTeXFile[aString] and OpenTeXFile[]

Aliases: **None**

Description: Opens the file "file.tex" for TeX output.

Arguments: aString is a string which contains the name of the TeXFile. aString should end in ".tex". If aString is missing,"file.tex" is used.

Comments \ Limitations: If you OpenTeXFile, then whatever was in file.tex or "aString" is erased. Therefore, if you use file.tex all of the time, then you should rename it after every CloseTeXFile[] call. See DoTeX, CloseTeXFile, ExprToTeXFile, ExpressionToTeXFile, StartParagraphTeXFile, CommentToTeXFile, InLineExpressionToTeXFile, EqualityToTeXFile, LongExpressionToTeXFile, AddSplitsTeX, TeXTheFile.

8.0.4 CloseTeXFile[aString] and CloseTeXFile[]

Aliases: **None**

Description: Closes the file "file.tex". This must be done before TeXing the file.

Arguments: aString is a string which contains the name of the TeXFile. aString should end in ".tex".If aString is missing, "file.tex" is used.

Comments \ Limitations: If you OpenTeXFile, then whatever was in file.tex or "aString" is erased. Therefore, if you use file.tex all of the time, then you should rename it after every CloseTeXFile[] call. See DoTeX, OpenTeXFile, ExprToTeXFile, ExpressionToTeXFile, StartParagraphTeXFile, CommentToTeXFile, InLineExpressionToTeXFile, EqualityToTeXFile, LongExpressionToTeXFile, AddSplitsTeX, TeXTheFile.

8.0.5 ExpressionToTeXFile[aString, expr] and ExpressionToTeXFile[expr]

Aliases: **ExprToTeXFile**

Description: Output an expression as a "displayed equation" (TeX terminology – i.e., double dollar sign mode) to a TeXFile.

Arguments: aString is a string which contains the name of the TeXFile. aString should end in ".tex".If aString is missing,"file.tex" is used.

Comments \ Limitations: One must open the TeX file before outputting anything to it (see OpenTeXFile). See DoTeX, OpenTeXFile, CloseTeXFile, StartParagraphTeXFile, CommentToTeXFile, InLineExpressionToTeXFile, EqualityToTeXFile, LongExpressionToTeXFile, AddSplitsTeX, TeXTheFile.

8.0.6 StartParagraph[aString] and StartParagraph[]

Aliases: **None**

Description: Output the TeX to start a paragraph.

Arguments: aString is a string which contains the name of the TeXFile. aString should end in ".tex".If aString is missing, "file.tex" is used.

Comments \ Limitations: See DoTeX, OpenTeXFile, CloseTeXFile, ExprToTeXFile, ExpressionToTeXFile, CommentToTeXFile, InLineExpressionToTeXFile, EqualityToTeXFile, LongExpressionToTeXFile, AddSplitsTeX, TeXTheFile.

8.0.7 CommentToTeXFile[aString, comment] and CommentToTeXFile[comment]

Aliases: **None**

Description: Output a comment to the TeX file. See StartParagraph.

Arguments: aString is a string which contains the name of the TeXFile. aString should end in ".tex".If aString is missing, "file.tex" is used. comment is a string.

Comments \ Limitations: See DoTeX, OpenTeXFile, CloseTeXFile, ExprToTeXFile, ExpressionToTeXFile, StartParagraphTeXFile, InLineExpressionToTeXFile, EqualityToTeXFile, LongExpressionToTeXFile, AddSplitsTeX, TeXTheFile.

8.0.8 InLineExpressionToTeXFile[aString, expr] and InLineExpressionToTeXFile[expr]

Aliases: **None**

Description: Output an expression as a "inline equation" (TeX terminology – i.e., single dollar sign mode) to a TeXFile.

Arguments: aString is a string which contains the name of the TeXFile. aString should end in ".tex".If aString is missing, "file.tex" is used. expr is a Mathematica expression (may be a string).

Comments \ Limitations: See DoTeX, OpenTeXFile, CloseTeXFile, ExprToTeXFile, ExpressionToTeXFile, StartParagraphTeXFile, CommentToTeXFile, EqualityToTeXFile, LongExpressionToTeXFile, AddSplitsTeX, TeXTheFile.

8.0.9 EqualityToTeXFile[aString, expr1, expr2] and EqualityToTeXFile[expr1, expr2]

Aliases: **None**

Description: Calls LongExpressionToTeXFile[aString, expr1, "=", expr2] . Will probably be removed from NCDoTeX.m in a future release.

Arguments: aString is a string which contains the name of the TeXFile. aString should end in ".tex".If aString is missing, "file.tex" is used. expr1 and expr2 are Mathematica expressions (may be strings).

Comments \ Limitations: See DoTeX, OpenTeXFile, CloseTeXFile, ExprToTeXFile, ExpressionToTeXFile, StartParagraphTeXFile, CommentToTeXFile, InLineExpressionToTeXFile, LongExpressionToTeXFile, AddSplitsTeX, TeXTheFile.

8.0.10 AddSplitsTeX[thing]

Aliases: **None**

Description: Used by LongExpressionToTeXFile

Arguments: None

Comments \ Limitations: See DoTeX, OpenTeXFile, CloseTeXFile, ExprToTeXFile, ExpressionToTeXFile, StartParagraphTeXFile, CommentToTeXFile, InLineExpressionToTeXFile, EqualityToTeXFile, LongExpressionToTeXFile, TeXTheFile.

8.0.11 SplittingTeXHeads

Aliases: **None**

Description: Used by LongExpressionToTeXFile. See AddSplitsTeXText.

Arguments: None

Comments \ Limitations: See DoTeX, OpenTeXFile, CloseTeXFile, ExprToTeXFile, ExpressionToTeXFile, StartParagraphTeXFile, CommentToTeXFile, InLineExpressionToTeXFile, EqualityToTeXFile, LongExpressionToTeXFile, AddSplitsTeX, TeXTheFile.

8.0.12 AddSplitsTeXText[”Plus”] = ”+”;

Aliases: **None**

Description: Used by LongExpressionToTeXFile. Right now an equation can be split between lines if there is an intervening “+”. One would alter AddSplitsTeXText and SplittingTeXHeads to split on something other than “+”.

Arguments: None

Comments \ Limitations: See DoTeX, OpenTeXFile, CloseTeXFile, ExprToTeXFile, ExpressionToTeXFile, StartParagraphTeXFile, CommentToTeXFile, InLineExpressionToTeXFile, EqualityToTeXFile, LongExpressionToTeXFile, AddSplitsTeX, TeXTheFile.

8.0.13 TeXTheFile[aString] TeXTheFile[]

Aliases: **None**

Description: Run latex. If no string is specified, then the NCTeX.process file in your bin will be executed. If aString is specified, then just “latexing” is done (no renaming, etc.).

Arguments: aString is a string which contains the name of the TeXFile. aString should end in “.tex”.If aString is missing, “file.tex” is used.

Comments \ Limitations: See Comments/Limitations in DoTeX. See DoTeX, OpenTeXFile, CloseTeXFile, ExprToTeXFile, ExpressionToTeXFile, StartParagraphTeXFile, CommentToTeXFile, InLineExpressionToTeXFile, EqualityToTeXFile, LongExpressionToTeXFile, AddSplitsTeX

Chapter 9

The Command DoTeX – How To Output an Expression in TeX (Unix only)

WARNING: DoTeX is not automatically installed with NCAlgebra. This requires extra steps. This is an effort of ours at producing TEXoutput which proceeded that of the commands in the Chapter 8. As a consequence, the output from it may be deficient. For example, matrices come out badly in DoTeX. DoTeX must be used interactively.

In order to install DoTeX, read the Comments/Limitations of the DoTeX command in the Chapter 8 session and look at the file DoTeX.install.

For a more flexible and more powerful set of commands for interacting with TeX which does not require any setup, see Chapter 8.

9.0.14 DoTeX

Aliases: **None**

Description: Interactive TeX outputter. The program will ask you what expressions or comments you would like to have put into a file for "LaTeXing". The program will then ask you for the name of the file which you want to put the ".tex" file and ".dvi" file in permanently. The file "file.tex", "file.aux",and "file.dvi" are used. This functions is a "front end" which calls other routines (see Comments/Limitations). For anything but the most basic TeXing, one would want to call the routines directly.

Arguments: There are no arguments. You are prompted for all relevant information.

Comments \ Limitations: You must be in a directory in which you can write files to (e.g., your own directories). In UNIX, if you do not set up your search path so that the NCTeX.process file is found, then the LaTeXing of the file will not occur. See the file DoTeX.install.

The function DoTeX makes printing an expression in TeX easy. It also creates a .dvi file. An example follows:

```
In[6]:= tp[a] ** inv[d]-l ** x

Out[6]= -l ** x + tp[a] ** inv[d]


In[7]:= DoTeX
""
"      1)  enter expression"
"      2)  enter comment"
"      3)  quit"
""
Enter Option [1-3]?  1
Enter expression ? Out[6]
"      1)  enter expression"
"      2)  enter comment"
"      3)  quit"
""
Enter Option [1-3]?  3
```

It asks for filename where you want TeX version saved. You give one. It asks if you want to print. Answer y. Out on the printer comes a beautiful TeX manifestation of equation 6. A difficulty is that long expressions run off the page. See LongExpressionToTeXFile as described in the Chapter 8. Otherwise, to fix this you must know enough TeX to insert your own formula breaks. You do this in the file .tex which DoTeX creates.

The DoTeX command repeatedly uses the Mathematica TeXForm command. We modified the TeXForm command so that ** tp etc would come out prettier.

In order to run DoTeX one has to have loaded a file into the “bin” directory. See Appendix A.

Chapter 10

Modifications to the TeXForm Command

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 through DoTeX and the file NCTeXForm.m so that

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}

A second concern involving the TeXForm command does not have as elegant a solution. The original TeXForm command does the following.

```
TeXForm[x] gives {\it x}.
```

While this may be beneficial in some circumstances, it is often undesirable since if one tries to put double dollar signs ($$) around your expression, then the italics gets in the way. To avoid this problem, one can add the following piece of code:

```
x/: Format[x, TeXForm] := "x";
```

The "x/:" tells Mathematica that it should think about this rule whenever it sees the variable x. If one omits the "x/:" then one must Unprotect[Format]; before and Protect[Format] after (and Mathematica will think about this rule every time it encounters a Format statement (which it does all the time when it does a

TeXForm command)). To use this solution, one must make the above command for each variable one uses.

It should be noted that the above solution works, but does not seem optimal. We could not think of a solution to this problem. If the reader can, please send us a copy of your code (See ADDITIONAL PACKAGES for style).

Part III

Appendices

Appendix A

NCAlgebra INSTALLATION

This is a repeat of Chapter 1 which can be used as a short version of the document.

NCAlgebra was developed on a UNIX system and it is very easy to install on one. Here we tell you how. Installation under DOS or on a MAC is similar and if you are a novice the assistance of system manager could be valuable to you. Also, if you obtain this from mathsource, then follow their directions.

(1) Create an NCAlgebra directory

(2) Copy NCAlgebra files into directory created in step 1.

The files are:

```
NCAlgebra.m              CEEP
Lots of files starting with NC.
Lots of files ending with Extra.
A file called NCTeX.process which you should transfer to the bin directory
before you use our TeX setting command DoTeX.
```

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. Here is a list of the version 0.2 files at the present time:

```
OSimplifyRational.NCTest
BlockPartition.NCTest
CEEP
CONTENTS
COVER.LETTER                 --- text --- not loaded
Collect.NCTest
ComplexRules.m
Convert.NCTest
Crit.NCTest
Date.Extra
```

```
Diff.NCTest
DoTeX.install                --- text --- not loaded
Extra.doc                    --- text --- not loaded
FTP.INSTRUCTIONS             --- text --- not loaded
FuncAssign.Extra
Hardy.Extra
MatMult.NCTest
Mono.NCTest
NC0SimplifyRational.m
NC1SetCommands.m
NC1SimplifyRational.m
NC2SetCommands.m
NC2SimplifyRational.m
NCAdjoints.m
NCAlgebra.m
NCAlias.m
NCBlockPartition.m
NCCo.m
NCCollect.m
NCComplex.m
NCDOC1Appendix.doc           --- text --- not loaded
NCDOCUMENT.doc               --- text --- not loaded
NCDiff.m                     --- text --- not loaded
NCDoTeX.m
NCDyad.Extra
NCEndPackage.m
NCHereditary.Extra
NCInverses.m
NCKernels.Extra
NCMatMult.m
NCMessyFunction.m
NCMono.m
NCMultiplication.m
NCOutput.m
NCRoots.m
NCSave.m
NCSchur.m
NCSimplifyRational.m
NCSolve.NCTest
NCSolve.m
NCSubstitute.m
NCTEST
NCTeX.process                --- text --- not loaded
NCTeXForm.m
NCTools.m
NCTransposes.m
NCUsage.m
Operator.Extra
README                       --- text --- not loaded
ROADSHOW                     --- text --- not loaded
```

```
SALESPITCH                    --- text --- not loaded
SYSDOC.doc                    --- text --- not loaded
SYSDef1PortLin.m
SYSDef2x2.m
SYSDefIA.m
SYSDefWAccess.m
SYSDefWIA.m
SYSHinfFormulas.m
SYSHinfTAC.m
SYSLinearize.m
SYSSpecialize.m
SYSTEST.m
SYStems.m
SetNumber.Extra
Shift.Extra
SimplifyRational.NCTest
Substitute.NCTest
TeXForm.Extra
Transposes.NCTest
more.Extra
testSYS
```

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.

```
(3)  Edit the file NCAlgebra.m to replace the System‘path ="" with
     whatever path takes you to the NC* files you just loaded in
     your machine.
```

If you are running NCAlgebra is the same directory which has the code, then step 3 may be ignored.

```
(4)  Run mathematica and execute the command

                  <<NCAlgebra.m

     and then the command

                  <<NCTEST

     The program should run for a while and indicate
     that all of the tests worked.

(4)  Peculiar to our package NCAlgebra is one line in NCAlgebra.m
```

```
          $$OperatingSystem="UNIX"
      You should change it to
          $$OperatingSystem="DOS"
      if you are running on a ''PC" and to
          $$OperatingSystem="MAC"
      if you are runinng on a macintosh.
```

Actually, this step does not really matter much. If the setting is incorrect, then you will occasionally get a harmless error message about attempting to determine the date.

```
(5)   Note: bookkeeping is easier if you create a working directory
      of the NCAlgebra directory, say WORKDIR, where you compute and
      save your sessions. To make this easy put NCAlgebra.m and SYStems.m
      in WORKDIR and edit NCAlgebra.m so that the left had side of the
      assignment ''System'path=whatever" gets you to NCAlgebra.

      For example, we use the path
             /home/osiris/helton/mathdir/NCAlgebra/
      because our computer is a SUN called osiris, the account is called
      helton, and standard Mathematica packages are stored in mathdir.
```

(6) If you want to use DoTeX (which is UNIX dependent) follow the instructions in the file DoTeX.install.

WARNING FOR DOS USERS: In DOS, the path commands are the SAME as for unix. One uses `/` and *not* `\` (which is not what the DOS user might expect). This is because the path command you are giving is to Mathematica and not to DOS.

Appendix B

UNIX DEPENDENCIES

There are several places in the code in which explicit facts about the UNIX operating system are used. Here is a list.

(1) The assignment statement in NCAlgebra.m

$$System`path = ""$$

needs to be adapted to the specific case. This was discussed in the appendix on installation.

(1') The variable $$OperatingSystem can be set to unix, dos or mac. This was discussed in the appendix on installation.

(2) In the file NCSubstitute.m, Date[] uses a pipe to the UNIX date command. If one is not using unix, mathematica outputs harmless error messages which may be ignored. This is corrected by following step (1').

(4) DoTeX uses UNIX commands and will not run elsewhere. Use the other commands for manipulating TEX, especially LongExpressionToTeXFile.

Appendix C

RUNNING SYSTEMS

In order to run SYSTEMS, you should first follow the directions for installing NCAlgebra as indicated in Appendix A. In addition to copying the file NCAlgebra.m into a WORKDIR directory (as in item 5 in Appendix A) one should copy SYStems.m. 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.

Appendix D

GETTING NCALGEBRA THROUGH ANONYMOUS FTP

It is possible to get updates of NCAlgebra through anonymous ftp. Here is the standard description of how to get NCAlgebra through anonymous ftp.

```
(****************************************************************************)

                               NCALGEBRA

                              Version 0.1


(****************************************************************************)
     Thanks you for your interest in NCAlgebra.  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 package is to follow the following sample
     terminal session.

     NCAlgebra is at osiris.ucsd.edu, in the pub/ncalg directory.  Below
     is a record of an actually 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.

     BELOW IS A SAMPLE SESSION. IF YOU FOLLOW IT, THEN YOU WILL BE
     ABLE TO GET A COPY OF THE NCALGEBRA PACKAGE.

     YOU CAN HAVE YOUR LOCAL COMPUTER "GURU" FOLLOW THE
```

INSTRUCTIONS BELOW AND INSTALL NCALGEBRA AS AN OFFICIAL MATHEMATICA PACKAGE. ONCE THIS IS DONE, EVERYONE USING THE COMPUTER MAY USE NCALGEBRA.

```
% ftp 132.239.145.6
  -------------------

Connected to 132.239.145.6.

220 osiris FTP server (SunOS 4.0) ready.


Name (132.239.145.6:mstankus): 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> prompt
     ------

Interactive mode off.

ftp> cd pub/ncalg
     -------------

250 CWD command successful.

ftp> mget *
     ------
ftp>

ftp> bye
     ---
221 Goodbye.
```

We have included a file called "ListofFilesAndSizes" in the directory so that you can compare filenames and filesizes by Kilobyte blocks.

This can be accomplished by typing the following UNIX commands,

```
ls -s -1 > MyFiles; diff MyFiles ListofFilesAndSizes
```

If what you have is identical to what we have, there should be NO output from the diff command. If there is a difference, then try ftp again. If the difference is "small", talk to a local expert.

We are interested in knowing who has NCAlgebra, 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 NCDOCUMENT. Printing out this file to a printer is the best first step.

We can email NCAlgebra, if necessary.

Email any questions, comments or requests to

ncalg@osiris.ucsd.edu

Appendix E

WELCOME OTHER DEVELOPERS

If you want to develop files or packages which run under NCAlgebra, let us know beforehand. We will let you know of progress on the program "as it happens" so that we do not reproduce each others work. Send us the result of your effort so that we may include it in the next version.

If you are developing functions which run under NCAlgebra, then we would like to have a YourFunction.NCTest file along with the function. Please stick to the format in the *.NCTest files. It is easy to imitate just by modifying a copy of one of our *.NCTest files . (Diff.NCTest is a good one to modify.)

We are constructing a document which is designed to provide a place for us to accumulate examples which have tutorial value. We have to some extent duplicated the table of contents structure of NCDOCUMENT.tex.

If you have an illustration of a command or strategy in using NCAlgebra which is instructive, please send it to ncalg@osiris.ucsd.edu for possible inclusion. Since LaTeXhas a verbatim environment, it is sufficient to e-mail the output from the Mathematica, using for example our program CEEP. We are not sure if we will be supporting notebooks, so please do not send notebooks.

We would like to comment briefly that we are working on at least two other packages which use NCAlgebra. The first is the package SYStems and the second is a collection of functions implementing a noncommutative version of the Gröbner basis algorithm. An extension of the SYStems files are included now with NCAlgebra. The SYStems programs have been used to do research in system's theory and includes files which do the computations in the paper [**Ball,Helton,Walker IEEE TAC 1991**]. Indeed, the file SYSHinfTAC.m was the early version of that paper. Read SYSDOC.doc for an explination of SYStems.The Gröbner basis code has been used to generate results for the papers [**HWS1**],[**HWS2**].

The 0.2 release of NCAlgebra has a VERY old version of the Gröbner basis code. One may get a new version of NCAlgebra, SYStems and the Gröbner basis through anonymous ftp through ncalg@osiris (see the cover of this document).

Appendix F

THE MOST BASIC COMMANDS — TEAR OFF SECTION

This is a repeat of Chapter 1 which can be used as a short version of the document.

"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 uses constantly are built-in. The multiplicative identity is denoted Id in the program. At the present time, Id is set to 1. An element A may have an inverse, which will be denoted by inv[A], or it may have a left or right inverse, denoted invL[A] and invR[A] respectively.

Each example is independent of the rest and assumes that you are executing it as the first command after you load in our packages. For this reason, every time we want to show a feature of a typical command, like the SetLinear command we explicitly invoke the SetLinear command (even though once one says SetLinear[f], f will remain linear for the rest of your Mathematica session until changed or redefined).

At present, single-letter lower case variables are non-commutative by default and all others are commutative by default.

When the lower case variables are non-commutative then we get the following:

```
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[a];
In[12]:= a ** b
Out[12]= a 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]= tp[x] ** a+b) ** x + z

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]:= f = 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[f]
Out[48]= 1

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

NCSR is the alias for NCSimplifyRational.

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

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

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

Note: Linear equations in one unknown only.

```
In[56]:= SetIsometry[s];
In[57]:= IsometryQ[s]
Out[57]= True

In[58]:= SetIsometry[s];
In[59]:= aj[s] ** s
Out[59]= 1
```

Remember the line In[58] is redundant and unnecessary. Nevertheless, it is restated for the sake of clarity. Normally, In[58] would be left out, because s is still defined as in line In[56].

```
In[60]:= SetIsometry[s];
```

```
In[61]:= tp[s] ** s
Out[61]= 1

In[62]:= SetSelfAdjoint[t];
In[63]:= tp[t]
Out[63]= t

In[64]:= SetSelfAdjoint[t];
In[65]:= aj[t]
Out[65]= t

In[66]:= SetProjection[p];
In[67]:= p ** p
Out[67]= p

In[68]:= SetCoIsometry[s];
In[69]:= s ** tp[s]
Out[69]= 1

In[70]:= SetCoIsometry[w];
In[71]:= w** aj[w]
Out[71]= 1

In[72]:= SetInv[x];
In[73]:= invL[x]
Out[73]= inv[x]

In[74]:= SetInv[x];
In[75]:= invR[x]
Out[75]= inv[x]

In[76]:= SetLinear[f];
In[77]:= f[x + y]
```

```
Out[77]= f[x] + f[y]

In[78]:= SetLinear[f];
In[79]:= f[(2 + 3 I) x ]
Out[79]= 2 f[x] + 3 I f[x]

In[80]:= SetConjugateLinear[q];
In[81]:= q[x + y]
Out[81]= q[x] + f[y]

In[82]:= SetConjugateLinear[g];
In[83]:= g[(2 + 3 I) x ]
Out[83]= 2 g[x] - 3 I g[x]

In[84]:= SetBilinear[h];
In[85]:= h[x + y, w]
Out[85]= h[x, w] + h[y, w]

In[86]:= SetBilinear[f];
In[87]:= f[w, x + y]
Out[87]= f[w, x] + f[w, y]

In[88]:= SetBilinear[f];
In[89]:= f[(2 + 3 I) x, w ]
Out[89]= 2 f[x, w] + 3 I f[x, w]

In[90]:= SetBilinear[f];
In[91]:= f[w,(2 + 3 I) x]
Out[91]= 2 f[w, x] + 3 I f[w, x]

In[92]:= SetSesquilinear[f];
In[93]:= f[x + y, w]
Out[93]= f[x, w] + f[y, w]
```

```
In[94]:= SetSesquilinear[f];
In[95]:= f[w, x + y]
Out[95]= f[w, x] + f[w, y]

In[96]:= SetSesquilinear[f];
In[97]:= f[(2 + 3 I) x, w]
Out[97]= 2 f[x, w] + 3 I f[x, w]

In[98]:= SetSesquilinear[f];
In[99]:= f[w,(2 + 3 I) x]
Out[99]= 2 f[w, x] - 3 I f[w, x]

In[100]:= SetIdempotent[f];
In[101]:= f[f[w]]
Out[101]= w

In[102]:= SetCommutingFunctions[f, g]; LeftQ[f, g] = True;
In[103]:= g[f[x]]
Out[103]= f[g[x]]

In[104]:= SetCommutingFunctions[f, g]; LeftQ[f, g] = True;
In[105]:= f[g[x]]
Out[105]= f[g[x]]

In[106]:= SetCommutingFunctions[f, g]; LeftQ[f, g] = False;
In[107]:= g[f[x]]
Out[107]= g[f[x]]

In[108]:= SetCommutingFunctions[f, g]; LeftQ[f, g] = False;
In[109]:= f[g[x]]
Out[109]= g[f[x]]
```

```
In[110]:= mat = {{a + tp[b],aj[c]},{inv[d],e}}
Out[100]= {{a + tp[b], aj[c]}, {inv[d], e}}

In[111]:= LookAtMatrix[mat]
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 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). This window contains the following formula in its display.

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

The LaTeXfile generated is shown below.

```
In[112]:= !!file.tex
\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.

$$\begin{gathered} a(1) + a(2) + a(3) + a(4) + a(5) + a(6) + a(7) + a(8) + \\ a(9) + a(10) + a(11) + a(12) + a(13) + a(14) + a(15) \\ + a(16) + a(17) + a(18) + a(19) + a(20) + a(21) + \end{gathered}$$

$$a(22) + a(23) + a(24) + a(25) + a(26) + a(27)+$$
$$a(28) + a(29) + a(30) + a(31) + a(32) + a(33)+$$
$$a(34) + a(35) + a(36) + a(37) + a(38) + a(39)+$$
$$a(40) + a(41) + a(42) + a(43) + a(44) + a(45)+$$
$$a(46) + a(47) + a(48) + a(49) + a(50)$$

```
In[114]:= SetOutput[ all -> True ];
In[115]:= inv[ tp[k] ]
             T -1
Out[115]= (k)

In[116]:= SetOutput[ all -> True ];
In[117]:= rt[x - inv[y + aj[z]]] + tp[x]
            T              * -1 1/2
Out[117]=  x  + (x - (y + z )  )

In[118]:= SetOutput[ all -> True, inv -> False ];
In[119]:= inv[1 + tp[1-x]]


                   T
Out[119]= inv[2 - x ]
```

See tp for transformation rules.

```
In[120]:= SetOutput[ all -> False, tp -> True, aj -> True ];
In[121]:= aj[ tp[k] ]
            T *
Out[121]= (k)

In[122]:= SetOutput[ all -> False, tp -> True, aj -> True ];
In[123]:=aj[ tp[k] ]
            * T
Out[123]= (k )
In[124]:= SetOutput[ all -> True ];
In[125]:= (p^x) ** tp[q]
             x     T
Out[125]= (p ) . (q )
In[126]:= SetOutput[ all -> False, dot -> True];
In[127]:= x ** tp[q] ** inv[1-x]
Out[127]= x . tp[q] . inv[1-x]
```

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