Computer Assistance for "Discovering" Formulas in System Engineering and Operator Theory*

J. William Helton and Mark Stankus

Department of Mathematics, University of California, San Diego, California

The objective of this paper is two-fold. First we present a methodology for using a combination of computer assistance and human intervention to discover highly algebraic theorems in operator, matrix, and linear systems engineering theory. Since the methodology allows limited human intervention, it is slightly less rigid than an algorithm. We call it a strategy. The second objective is to illustrate the methodology by deriving four theorems. The presentation of the methodology is carried out in three steps. The first step is introducing an abstraction of the methodology which we call an idealized strategy. This abstraction facilitates a high level discussion of the ideas involved. Idealized strategies cannot be implemented on a computer. The second and third steps introduce approximations of these abstractions which we call prestrategy and strategy, respectively. A strategy is more general than a prestrategy and, in fact, every prestrategy is a strategy. The above mentioned approximations are implemented on a computer. We stress that, since there is a computer implementation, the reader can *use* these techniques to attack their own algebra problems. Thus the paper might be of both practical and theoretical interest to analysts, engineers, and algebraists. Now we give the idea of a prestrategy. A prestrategy relies almost entirely on two commands which we call NCProcess1 and NCProcess2. These two commands are sufficiently powerful so that, in many cases, when one applies them repeatedly to a complicated collection of equations, they transform the collection of equations into an equivalent but substantially simpler collection of equations. A loose description of a prestrategy applied to a list of equations is:

(1) Declare which variables are known and which are unknown. At the beginning of a prestrategy, the order in which the equations are listed is not important, since NCProcess1 and NCProcess2 will reorder them so that the simplest ones appear first.

(2) Apply NCProcess1 to the equations; the output is a set of equations, usually some in fewer unknowns than before, carefully partitioned based upon which unknowns they contain.

(3) The user must select "important equations," especially any which solve for an unknown, say x. (When an equation is declared to be important or a

* This work was sponsored by the Air Force Office for Scientific Research and by the National Science Foundation.

variable is switched from being an unknown to being a known, then the way in which NCProcess1 and NCProcess2 reorder the equations is modified.)

(4) Switch x to being known rather than unknown. Go to (2) above or stop.

When this procedure stops, it hopefully gives the "canonical" necessary conditions for the original equations to have a solution. As a final step we run NCProcess2 which aggressively eliminates redundant equations and partitions the output equations in a way which facilitates proving that the necessary conditions are also sufficient. Many classical theorems in analysis can be viewed in terms of solving a collection of equations. We have found that this procedure actually discovers the classic theorem in a modest collection of classic cases involving factorization of engineering systems and matrix completion problems. One might regard the question of which classical theorems in analysis can be proven with a strategy as an analog of classical Euclidean geometry where a major question was what can be constructed with a compass and ruler. Here the goal is to determine which theorems in systems and operator theory could be discovered by repeatedly applying NCProcess1 and NCProcess2 (or their successors) and the (human) selection of equations which are important. The major practical challenge addressed here is finding operations which, when implemented in software, present the user with crucial algebraic information about his problem while not overwhelming him with too much redundant information. This paper consists of two parts. A description of strategies, a high-level description of the algorithms, a description of the applications to operator, matrix, and linear system engineering theory, and a description of how one would use a strategy to "discover" four different theorems are presented in the first part of the paper. Thus, one who seeks a conventional viewpoint for this rather unconventional paper might think of this as providing a unified proof of four different theorems. The theorems were selected for their diverse proofs and because they are widely known (so that many readers should be familiar with at least one of them). The NCProcess commands use noncommutative Gröbner Basis algorithms which have emerged in the last decade, together with algorithms for removing redundant equations and a method for assisting a mathematician in writing a (noncommutative) polynomial as a composition of polynomials. The reader needs to know *nothing* about Gröbner Basis to understand the first part of this paper. Descriptions involving the theory of Gröbner Basis appear in the second part of the paper.

Contents.

I. Strategies and applications

I. STRATEGIES AND APPLICATIONS

1. INTRODUCTION

The goal in this paper is to describe and illustrate a highly computer assisted method for discovering certain types of theorems. Many theorems in operator theory and engineering systems theory amount to giving hypotheses under which it is possible to solve large collections of equations. At the beginning of "discovering" a theorem, the problem is to determine the nature of solutions to a large collection of equations. Discovering a theorem involves exploring the consequences of these equations until a simpler equivalent set of equations is found.

The equations mentioned above take the form of a polynomial,[1] with matrices or operators substituted for the variables, set equal to zero.

Vast experience with the commuting case of the manipulation of collections of equations suggests that using a noncommutative Gröbner Basis algorithm is an essential tool for analyzing systems of equations.

We wish to stress that one does not need to *know* a theorem in order to "discover" it using the techniques in this paper. Furthermore, any method which assumes that all of the hypotheses are known at the beginning of the computation or that all of the hypotheses can be stated algebraically will be of limited practical use. For example, since the Gröbner Basis algorithm only discovers polynomial equations which are algebraically true and not those which require analysis or topology, the use of this algorithm alone has a limited use. Insights gained from analysis during a computer session could be added as (algebraic) hypotheses while the session is in progress. Decisions can take a variety of forms and can involve recognizing a Riccati equation, recognizing that a particular square matrix is onto and so invertible, recognizing that a particular theorem now applies to the problem, etc. The user would then have to record and justify these decisions independently of the computer run.[2] While a strategy allows for human intervention, the intervention must follow certain rigid rules for the computer session to be considered a strategy.

One thing to bear in mind when reading the examples in this paper is that they present not just proofs but also *the methods for discovering proofs*. This makes their presentation longer than the mere presentation of the slickest known proof. Also, to a specialist, some of the examples have the feel of a "back of the envelope computation."

The software which was developed for this paper is freely available. It may be downloaded from the NCAlgebra web site at http://math.ucsd.edu/~ncalg. The code is accessed through Mathematica. The bulk of the computations are performed by the C^{++} code. This is invisible to the user. The output is automatically displayed to the screen. The output is formatted using the program LaTeX. One would probably want to use this software in conjunction with NCAlgebra which is the main Mathematica package for doing noncommutative algebraic computations. NCAlgebra is

[1] Informally, a noncommutative polynomial in the indeterminates $x_1, ..., x_n$ over the field K is a polynomial, but one removes the assumption that the indeterminates commute (e.g., $x_1 x_2 - x_2 x_1$ is *not* the zero polynomial). Formally, a noncommutative polynomial in the indeterminates $x_1, ..., x_n$ over the field K is a formal sum of elements of formal products ct where $c \in K$ and t is a member of the free semigroup generated by $\{x_1, ..., x_n\}$.

[2] See Appendix B for an example of this.

also available from the web site http://math.ucsd.edu/$\sim$ncalg.[3] This extensive software makes these techniques available for exploration of a new and unusual type of question in operator theory or systems engineering: which classical theorems in operator theory or systems theory can be "discovered" using a strategy? Of course we hope new theorems will be discovered, but the best place to evaluate these ideas is on the classics.

1.1. *Background on Polynomial Equations and Ideals*

When doing analysis, there is often an algebraic component to the calculations. This algebraic component involves a collection of polynomial equations in a finite number of variables which are to be evaluated at elements of some algebra $\mathscr{A}$—the algebra might be $n \times n$ matrices, bounded linear transformations on a complex separable Hilbert space or a variety of other algebras.[4] This paper concerns algorithms which assist in transforming the collection of equations into a more suitable form.

In this paper, we will blur the difference between the polynomial equation $p = q$ and the polynomial $p - q$. An algebraist would say that p is a relation and an analyst would say that $p = 0$ is a polynomial equation. The difference in language does not lead to problems.

1.1.1. *A Few Polynomial Equations*

We set the stage by listing a (very) few polynomial equations which commonly occur in analysis.

(1) A matrix T is an isometry if and only if $T^*T - 1 = 0$, that is, if (T, T^*) is a zero of a (noncommutative) polynomial.

(2) X satisfies a Riccati equation $AX + XA^* + XRX + Q = 0$ if and only if (A, A^*, X, R, Q) is a zero of a (noncommutative) polynomial.

(3) A matrix T is an coisometry if and only if $TT^* - 1 = 0$, that is, if (T, T^*) is a zero of a (noncommutative) polynomial.

[3] The C^{++} code compiles with the free compiler **g++** (version 2.6.3 or higher) from the Free Software Foundation. The C^{++} code is about 15,000 lines and compiles to a binary of about 1.5 megabytes on a Sun Workstation. We have compiled it with g++ version 2.6.3 on a Sun Workstation running Sun OS 4.1.3 and with g++ version 2.7.2 on a PC running Linux. The minimal Mathematica code needed is about 0.30 megabytes, while the Mathematica code for NCAlgebra is about 0.75 megabytes.

[4] One can often consider problems which do not seem to involve a single algebra. An example is to consider computations involving $m \times m$, $m \times n$, $n \times m$ and $n \times n$ matrices. Even though the union of these four sets of matrices is not an algebra, one can create an algebra in which one can embed these four algebras. Without going into the details, this can be done using a path algebra where $\mathbf{R}^m$ and $\mathbf{R}^n$ are vertices and, for example, an $m \times n$ matrix is a path from the $\mathbf{R}^m$ vertex to the $\mathbf{R}^n$ vertex, c.f. [FaFeGr].

(4) A matrix A is invertible if and only if there exists a matrix B such that $AB-1=0$ and $BA-1=0$, that is, if (A, B) is a common zero of two (noncommutative) polynomials.

(5) A bounded linear transformation V of a complex Hilbert space is a partial isometry if and only if $V^*VV^*=V^*$, that is (V, V^*) is a zero of a (noncommutative) polynomial.

(6) A bounded linear transformation T of a complex Hilbert space has a pair of complementary invariant subspaces if and only if there exist operators P_1 and P_2 such that $P_1^2=P_1$, $P_1=P_1^*$, $P_2^2=P_2$, $P_2=P_2^*$, $P_1+P_2=1$, $P_1TP_1=TP_1$ and $P_2TP_2=TP_2$, that is, $(P_1, P_1^*, P_2, P_2^*, T)$ is a common zero of seven (noncommutative) polynomials.

(7) Using a polar decomposition of a bounded linear transformation of a complex Hilbert space T is equivalent to introducing bounded linear transformations P and V such that $T=PV$, $V^*VV^*=V^*$, $P=P^*$ and $P\geqslant 0$, that is, (T, P, P^*, V, V^*) is a common zero of three (noncommutative) polynomials and one additional constraint holds. Using the idea of a strategy (which is to be given in this paper), one can use such additional constraints to one's advantage during an algebraic computation.

(8) (V_1, V_2) is a pair of commuting isometries if and only if V_1 and V_2 are isometries ($V_1^*V_1=1$, $V_2^*V_2=1$) and V_1 and V_2 commute ($V_1V_2=V_2V_1$). The pair (V_1, V_2) lifts to a pair of commuting unitaries if and only if there exists U_1, U_2, L_1 and L_2 such that L_1 and L_2 are an isometries ($L_1^*L_1=1$, $L_2^*L_2=1$), U_1 and U_2 are unitaries ($U_1^*U_1=1$, $U_1U_1^*=1$, $U_2^*U_2=1$, $U_2U_2^*=1$) and U_1 and U_2 commute ($U_1U_2=U_2U_1$) and L_j intertwines V_j and U_j for $j=1, 2$ ($U_1L_1=L_1V_1$ and $U_2L_2=L_2V_2$). Therefore, the pair (V_1, V_2) lifts to a pair of commuting unitaries if and only if $(V_1, V_1^*, V_2, V_2^*, U_1, U_1^*, U_2, U_2^*, L_1, L_1^*, L_2, L_2^*)$ is a common zero of 12 (noncommutative) polynomials.

We can summarize the above examples by saying that a number of properties of matrices and operators (bounded linear transformations of complex Hilbert space) are equivalent to the statement that a tuple of matrices (or tuple of operators) is a common zero of a set of polynomial equations. We now discuss the traditional connection between common zeros and ideals.

1.1.2. *Solutions of Polynomial Equations and Ideals*

Let $K[x_1, ..., x_n]$ denote the set of (noncommutative) polynomials in the indeterminates $x_1, x_2, ..., x_n$ over a field K. $K[x_1, ..., x_n]$ is a noncommutative algebra. Let $p_1, ..., p_m\in K[x_1, ..., x_n]$, C be a set of polynomials $\{p_1, ..., p_m\}$ and $r=(r_1, ..., r_n)$ with each $r_j\in\mathscr{A}$. If r is a common zero of $p_1, ..., p_m$, then r is a zero of every polynomial p which lies in the smallest ideal $\mathscr{I}_C$ generated by C. Furthermore, if $\{q_1, ..., q_\ell\}$ is a generating set for the ideal $\mathscr{I}_C$, then r is a common zero of the p's if and only if it is a common zero of the q's. These ideas are learned by most mathematics undergraduates in the case that the indeterminates x_j lie in a commutative algebra and they also hold when the x_j lie in a noncommutative algebra.

The practical value of doing this algebraic manipulation is that instead of trying to solve for the common zeros of $p_1, ..., p_m$ directly, it could be advantageous to find a different set $C' = \{q_1, ..., q_\ell\}$ of polynomials which generates the same ideal as $\{p_1, ..., p_m\}$ and *then* solve for the common zeros on the q's. For example, C' might consist of decoupled polynomial equations which could be solved numerically (say using Matlab) while the original set C produces numerically intractable problems.

The reader who is interested in seeing examples at this point might decide to skip directly to Section 3.

1.2. *A Highly Idealized Picture*

While what we do in the paper is symbolic computation and centers on computer experiments, perhaps the simplest description of the ideas is in terms of an idealized problem in noncommuting algebra. We first consider two extremely powerful hypothetical operations and then show in Section 1.2.2 how they are combined in what we call an idealized strategy.

1.2.1. *Basic "Idealized" Operations*

Suppose that we are in a context where there are knowns $\{a_1, ..., a_r\}$ and unknowns $\{x_1, ..., x_s\}$.

The first operation which we consider is called *Categorize*. There are two key ideas behind the functioning of Categorize. Firstly, it finds equations not involving any unknowns, equations involving one unknown, equations involving two unknowns, etc. We will give an example below which shows how this can be beneficial. Secondly, Categorize applies to polynomials which are members of $\mathscr{I}_{C \cup C'}$, but which are not members of $\mathscr{I}_{C'}$. More precisely, Categorize associates to two collections of polynomials C and C' (in $K[a_1, ..., a_r, x_1, ..., x_s]$), the collection $\{P_j\}_{j \geqslant 0}$ of subsets of $\mathscr{I}_{C \cup C'}$. Each P_j consists of polynomials which depend on exactly j unknowns and which lie in $\mathscr{I}_{C \cup C'} \backslash \mathscr{I}_{C'}$, that is, which are members of $\mathscr{I}_{C \cup C'}$ but which are not members of $\mathscr{I}_{C'}$. Categorize is not implementable on a computer, because subsumed in the Categorize command is the ability to eliminate unknowns (whenever algebraically possible) from equations in the original set of polynomial equations. In fact, the paper [TMora] shows that there does not exist a computer algorithm which can determine whether or not P_1 is the empty set.

For an example of how Categorize can be useful, suppose that C is a collection of polynomial equations in knowns a_j and unknowns x_k. If it could be shown algebraically that the Riccati equation $x_1 a_1 a_2 x_1 + a_1 x_1 + x_1 a_3 + a_4 = 0$ follows from C, then this Riccati equation would be a member of P_1. Knowing that x_1 satisfies a Riccati equation can be of great value since Riccati equation can be quickly solved numerically. In this example, C' is the empty set.

A slightly more complicated example would be if it could be shown algebraically that an expression, such as $x_1 x_2 + x_3$, solved a Riccati equation, e.g., if

$$(x_1 x_2 + x_3)\, a_1 a_2 (x_1 x_2 + x_3) + a_3 (x_1 x_2 + x_3) + a_5 a_6 = 0 \tag{1.1}$$

follows from a collection of polynomial equations C. The left hand side of (1.1) would depend on three unknowns x_1, x_2, and x_3 and, therefore, would be a member of P_3, not P_1. It is natural, however, to view (1.1) as an equation in one new unknown y and to rewrite the left hand side of (1.1) as the composition

$$k(a_1, ..., a_6, q(a_1, ..., a_6, x_1, x_2, x_3))$$

where $q(a_1, ..., a_6, x_1, x_2, x_3) = x_1 x_2 + x_3$ and $k(a_1, ..., a_6, y) = y a_1 a_2 y + a_3 y + a_5 a_6$. In this example, we would call y a *motivated unknown*. The second of our two idealized operations is called *Decompose* and associates to a polynomial p all non-trivial maximal compositions.[5] *Decompose*, therefore, produces motivated unknowns. Decompose will not be used, or discussed further, until Section 5.

1.2.2. *Idealized Strategies*

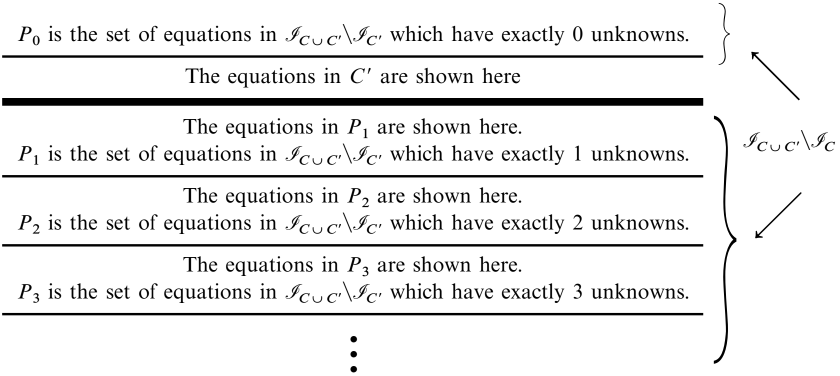
We now describe what we mean by an idealized strategy. An idealized strategy uses the Categorize and Decompose operations. While an idealized strategy cannot be implemented on a computer, approximations of it can be implemented and the use of these approximations is the core of our paper.

An idealized strategy is an iterative procedure which would invoke Categorize on the pair of sets of polynomial equations C and C' (C' is initially empty), allow creation of motivated unknowns, allow for unknowns to be redeclared as knowns and allow for human intervention. Human intervention would consist of selecting particular equations which

[5] If p and k are polynomials and k is a polynomial in one unknows (say y), then we say polynomial p is a composition of k and q if we can obtain p by replacing every occurrence of y with q, that is, $p(a_1, ..., a_r, x_1, ..., x_s) = k(a_1, ..., a_r, q(a_1, ..., a_r, x_1, ..., x_s))$. If p is a composition of k and q, then the composition is *trivial* if and only if k has the form $k(a_1, ..., a_r, y) = cy + d$ for some scalars c and d. The decomposition is called maximal if k has no non-trivial decomposition.

are nice in some way (e.g., this is a Riccati equation), adding them to the collection C' and possibly selecting a motivated unknown y as described above. This procedure would terminate when the ideal generated by $P_0 \cup C'$ equals the ideal generated by $C \cup C'$. This, of course, is equivalent to the condition that P_j is a subset of $\mathscr{I}_{P_0 \cup C'}$ for each $j \geqslant 1$. In other words, an idealized strategy terminates when all of the polynomial equations which can be derived algebraically and involve unknowns can be derived algebraically from the equations in C' together with the equations which do not involve unknowns.

If the reader would indulge us, we now discuss a more visual way of thinking about idealized strategies. For ease of exposition, the discussion in this paragraph will ignore the possibility of introducing motivated unknowns, the redeclaration of unknowns as knowns or human intervention. For each pair of collections of polynomials C and C', suppose that one has an idealized display:



P_0 is the set of equations in $\mathscr{I}_{C \cup C'} \backslash \mathscr{I}_{C'}$ which have exactly 0 unknowns.

The equations in C' are shown here

The equations in P_1 are shown here.
P_1 is the set of equations in $\mathscr{I}_{C \cup C'} \backslash \mathscr{I}_{C'}$ which have exactly 1 unknowns.

The equations in P_2 are shown here.
P_2 is the set of equations in $\mathscr{I}_{C \cup C'} \backslash \mathscr{I}_{C'}$ which have exactly 2 unknowns.

The equations in P_3 are shown here.
P_3 is the set of equations in $\mathscr{I}_{C \cup C'} \backslash \mathscr{I}_{C'}$ which have exactly 3 unknowns.

$\vdots$

$\mathscr{I}_{C \cup C'} \backslash \mathscr{I}_{C}$

When one begins using an idealized strategy, C would be the collections of equations coming from the problem at hand and C' would be initially empty. During each iteration of a strategy, one would choose a number of polynomials from the ideal $\mathscr{I}_{C \cup C'}$ generated by $C \cup C'$, (that is, below the dark line) and place them in C' (that is, above the dark line). The idealized display would then adjust itself so that what appears in P_0 above the line and what appears below the line corresponds to the new value of C'. When the equations below the dark line are in the ideal generated by the equations above the dark line, the idealized strategy is complete. If $C'_0 = \{\}$ and, for $1 \leqslant k \leqslant \ell$, C'_k is produced from C and C'_{k-1} using one step of an idealized strategy and each equation which is in C'_k but not in C'_{k-1} has at most 1 (motivated) unknown, then we say that C'_ℓ is derivable from C by an idealized strategy.

A question about strategies is

Which classical results can be derived using an idealized strategy?

This question gives an abstract statement of what we address in this paper.

One intends, of course, to move beyond this question to the derivation of new theorems. However, in the early stages of the subject, we think that the most urgent task is to understand classical theorems from this viewpoint. Evaluating these new and unusual techniques on a new and not well understood problem gives less of an idea of their strength than evaluating them on a well understood problem. Typically, if we derive a new theorem using symbol manipulation, then one can go back and produce a proof by hand, using ideas gotten from the symbolic manipulation.

1.2.3. *New Derivations of Classical Theorems: Idealized Picture*

We shall see in this paper that the algebra components of four theorems can be "discovered" using a strategy:

(1) The minimal factorization of a system due to Bart–Gohberg–Kaashoek and van Dooren.

(2) The Doyle–Glover–Khargonekar–Francis theorem of H^∞ control.

(3) A matrix completion theorem due to W. W. Barrett, C. R. Johnson, M. E. Lundquist, and H. Woerderman.

(4) A matrix completion theorem due to Steve Parrot.

We can interpret this from a traditional viewpoint as the following theorem.

THEOREM 1.2. *The key formulas in Theorems* 1, 3, 4 *above can be derived with an idealized strategy.*

We use the name 1-Decompose to also refer to the Decompose operation, we use the name 1-motivated unknown to refer to motivated unknown and we use the name idealized 1-strategy to refer to an idealized strategy. For the 1-Decompose operation, the key was to find a polynomial k in the knowns and one unknown and a polynomial q such that when q is substituted for the one unknown of k, one obtains p. Likewise, we could consider ℓ-decompositions which would consist of finding a polynomial k in ℓ unknowns and ℓ polynomials $q_1, ..., q_\ell$ such that when q_j is substituted for the jth unknown for $1 \leqslant j \leqslant \ell$, one obtains p. An ℓ-Decompose operation would then be an operation which found all j-motivated unknowns for $1 \leqslant j \leqslant \ell$ and an idealized ℓ-strategy would allow the use of the ℓ-Decompose operation.

A variant on 1-Decompose which we shall use frequently is called *symmetric* 1-*Decompose*. This applies in an algebra with involution, $w \rightarrow w^*$ for all w, for example, a matrix algebra with adjoints or transposes.

Symmetric 1-Decompose applied to p yields a 2-decomposition of p as $p(a_1, ..., a_r, x_1, ..., x_s) = k(a_1, ..., a_r, q(a_1, ..., a_r, x_1, ..., x_s), q(a_1, ..., a_r, x_1, ..., x_s)^*)$ where the second polynomial $q(a_1, ..., a_r, x_1, ..., x_s)^*$ is the adjoint of the first.[6]

We were not able to derive the key formulas of the theorem of item 2 above with a 1-strategy, but they can be derived with a symmetrized 2-strategy.[7] The use of a symmetrized 2-strategy forces human intervention, but it is small because the 2-decomposition required is easy to recognize from the output of the NCProcess1 command. (Note that it originally took 5 years to discover the theorem).

There are less conservative ways of choosing knowns and unknowns under which the derivation would be classed as a symmetrical 1-strategy.

1.3. *A Practical Picture*

As mentioned before, an idealized strategy cannot be implemented on a computer. Section 1.3.1 gives a very brief introduction to the command which will be used to approximate to an idealized strategy.

1.3.1. *NCProcess Commands*

The idealized operations Categorize and Decompose cannot be implemented on a computer. There are two difficulties with implementing an approximation to the Categorize operation. The first difficulty is that a computer cannot generate the ideal $\mathscr{I}_C$. The second difficulty is that if a human is presented with a large subset[8] of the ideal $\mathscr{I}_C$, then looking for "interesting" polynomials to place in C' can be overwhelming.

The NCProcess commands approximate the Categorize and Decompose operations and address the two difficulties mentioned above. This paper studies two such commands, NCProcess1 and NCProcess2. NCProcess1 functions as follows:

(1) NCProcess1 takes as input a set C of polynomial equations. Some of these equations may be marked "important." These important equations are in the set C' mentioned in Section 1.2.2.

[6] The precise algebraic meaning of $q(a_1, ..., a_r, x_1, ..., x_s)^*$ is given in Appendix A. For example, one must assume that there is a notion of conjugation for the coefficients of q. The meaning of this notation will be clear from context whenever it is used in this paper. See also Section 1.5.

[7] There are less conservative ways of choosing knowns and unknowns under which the derivation would be classed as a symmetrized 1-strategy.

[8] A canonical choice of a large subset would be a Gröbner Basis or a subset of a Gröbner Basis which could be computed in a reasonable amount of time.

(2) NCProcess1 takes the set C and computes a different set of polynomial equations by running the noncommuting Gröbner Basis algorithm. The Gröbner Basis Algorithm hopefully eliminates unknowns.

(3) NCProcess1 then attempts to find smaller subsets of the set created in item 2 which generate the same ideal. The way in which NCProcess1 find smaller subsets is described in Section 12 and in [NCGBDoc],

(4) NCProcess1 then takes the small generating set from item 3 and "factors" each polynomial in a way which suggests possible decompositions to the users.

The NCProcess command is described further in Section 2.2.

Item 4 above is the best approximation which we know to the Decompose operation. The notion of factoring mentioned in item 4 is different from the standard one and is explained in Section 5.

As a final step we run NCProcess2 which aggressively eliminates redundant equations and partitions the output equations in a way which facilitates proving that the necessary conditions are also sufficient. In terms of the list above, NCProcess2 carries out items 3 and 4, but not item 2. NCProcess2 uses a more aggressive algorithm for item 3 than NCProcess1 uses.

The output from the NCProcess commands is displayed along the lines of the Categorize operation: NCProcess displays all of the output equations in one unknown together, two unknowns together, etc. For a more detailed description of the output, see Section 2.2.3.

1.3.2. *New Derivations of Classical Theorems: Practical Picture*

A strategy is a practical approximation to an idealized strategy and is based on replacing the Categorize and Decompose commands with NCProcess1. The idea of a strategy was described in the abstract, while the precise definition is given in Section 5.1.

Theorem 1.2 follows from the stronger theorem:

THEOREM 1.3. *The key formulas in Theorems* 1, 3, 4 *can be derived with a* 1-*strategy*.

As an example of how Theorem 1.3 is stronger than Theorem 1.2, we mention that in defining an idealized strategy, we have often said that *there exist* polynomials in the ideal or the user selects certain polynomials from *the ideal*. In practice, our implemented operations for automatically reducing the size of generating sets for polynomial ideals remove many redundant polynomial equations, and the user can select from this reduced generating set. For an example, see Section 3.4. The user has little work to do in making a selection. In the same spirit as Theorem 1.3, the theorem of item 2 of Section 1.2.3 can be derived using a symmetrized 2-strategy.

Note in practice a user can get great benefit from our strategy software when he thinks up a clever (unmotivated) unknown, but we are unable to formalize or analyze this type of intervention, so it plays no role in this paper.

1.4. *Algorithms and Their Properties*

Finding different sets of noncommutative polynomials which generate the same ideal is one of the main tasks underlying this paper. The two implementable operations we find critical to this paper are the Gröbner Basis Algorithm and the "shrinking" of a basis for a polynomial ideal to a smaller generating set. These operations are encapsulated in the NCProcess commands.[9]

In Part I we give enough of an explanation of the theory behind the NCProcess commands (Sections 1.3.1, 2.2, and 2.3) for a practitioner, analyst or engineer to understand the results given in Part I. Part II of the paper gives details and properties of the algorithms of this paper and completes the description of the theory behind the NCProcess commands.

1.4.1. *Obtaining Smaller Bases*; *Shrinking*

Often the Gröbner Basis algorithm gives a generating set for a polynomial ideal which is large enough so that looking for "interesting" polynomials can be overwhelming (see Section 1.3.1); therefore, it is necessary to find a smaller generating set. It is our belief and experience that most highly algebraic mathematics theorems amount to giving a *small generating set* for an ideal. This is consistent with the esthetic that one wants simple hypotheses. Also with respect to the use of the theorem for numerics, if one tries to solve redundant equations, then small errors in data and roundoff make the equations contradictory. One reason that one does not seek a *minimal* (in cardinality) generating set for the ideal is that sometimes one does not want to eliminate certain equations involving knowns (see Section 12.7). Algorithms for finding small generating sets for ideals is the topic of Part II in Section 12.

Section 12 discusses the theory behind the algorithms which we use for finding small generating subsets[11] of a given set. These algorithms also

[9] This is available for free. See the NCAlgebra web site http://math.ucsd.edu/$\sim$ncalg.

[10] Deleted in proof.

[11] *Small* rather than *minimal* because finding minimal generating subsets is unsolvable by Lemma 12.2.

pertain to finding minimal generating sets although we do not typically employ them for that purpose.

1.4.2. *Noncommutative Elimination Theory*

Those already knowledgeable about computer algebra know that an important classical property of commutative Gröbner Basis concerns elimination ideals. If G is a Gröbner Basis with respect to an elimination order (see Definition 11.2), I is the ideal generated by G and I_j is the jth elimination ideal, then $G \cap I_j$ is a Gröbner Basis for I_j. In Section 11, we generalize this to the noncommutative case. This result is crucial to assuring that the Gröbner Basis algorithm puts the collection of polynomial equations into a triangular form. Pure lex and, more generally, multigraded lex are examples of such elimination orders.

1.5. *A Practical Notation*

Notice that, in the above text, we always considered polynomials in indeterminates $a_1, ..., a_r, x_1, ..., x_s$. We will be considering matrices or operators with names such as T, U, T^*, T^{-1}, M^T, etc. If we continue to use a's and x's for the indeterminates, then, in practice, one would be required to keep track of some not-so-easily-rememberable bijection between the indeterminates and matrices such as the bijection σ define by setting $\sigma(x_1) = T$, $\sigma(x_2) = U$, $\sigma(x_3) = T^*$, etc. Instead we have chosen to use the same name (or a similar name) for the indeterminate and the corresponding matrix.

1.6. *Computational Concerns*

The output from any of the NCProcess commands takes the form of either a LATEX file or a text file. In some cases, we have slightly reformatted the LATEX output so that it takes up less space on the printed page. This change involved displaying two or three equations on one line rather than presenting one equation on each line.

Since this paper discusses mathematics and does not explain how to use the code, it is not easy to see that, for the computer experiments presented in this paper, there is very little which needs to be typed. For example, when performing the computations in Section 2.2.2, one would *not* type in the 5 equations of (2.3). One instead would type in the 2×2 matrix and tell the computer that it is an isometry and that U is unitary.

2. PRESTRATEGY

We feel that it is helpful to begin by introducing a procedure which is simpler than a strategy, a procedure which we call a prestrategy. Strategies will be discussed in Section 5.

In Section 1.2.2, an idealized strategy was discussed. It involves the use of the idealized operations Categorize and Decompose. A prestrategy will be based on the NCProcess1 and NCProcess2 commands which implement part of the Categorize operation and will not involve the Decompose operation at all. The NCProcess commands are based on a Gröbner Basis algorithm and much of its power is devoted to eliminating variables for equations—a topic which we now discuss briefly.

2.1. *Elimination*

The theory of elimination is well known for the case of commutative Gröbner Basis Algorithms (cf., [CLS], Ch. 3) and we show in Part II in Section 11 that it extends in a strong form to the noncommutative case.

Commutative Gröbner Basis Algorithms (GBA) can be used to systematically eliminate variables from a collection of polynomial equations (e.g., $\{p_j(x_1, ..., x_n)=0\colon 1\leqslant j\leqslant k_1\}$) so as to put it in triangular form. One specifies an order on the variables $(x_1<x_2<x_3<\cdots<x_n)$ which corresponds to one's priorities in eliminating them. Here the GBA will try hardest to eliminate x_n and try the least to eliminate x_1. The output is a list of equations in a "canonical form" which is triangular:

$$\begin{aligned} q_1(x_1)&=0\\ q_2(x_1, x_2)&=0\\ q_3(x_1, x_2)&=0\\ q_4(x_1, x_2, x_3)&=0\\ &\cdots\\ q_{k_2}(x_1, ..., x_n)&=0. \end{aligned} \tag{2.1}$$

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

$x_1, ..., x_{j-1}$. The effect of the ordering on the variables is to specify that variables high in the order will be eliminated while variables low in the order will not be eliminated.

In the noncommutative case, Gröbner Basis algorithms exist ([FMora]). One can define lexicographic and lexicographic-like term orders (see Section 7 and Appendix C). Again, a Gröbner Basis for a collection of polynomial equations is a collection of noncommuting polynomial equations in triangular form (see Theorem 11.3). There are some difficulties which don't occur in the commutative case. For example, a Gröbner Basis can be infinite in the noncommutative case (see Section 12 for a way to counteract this effect). However, we believe that noncommutative Gröbner Basis may prove to be extremely useful, see [HW] or [HSW], for applications to simplification of complicated expressions. As we shall see, the present paper concerns a different type of application, that of eliminating unknowns from collections of equations, which is the main function of the NCProcess commands.

In this paper, Gröbner Bases are computed using an algorithm in [TMora]. See also Section 10 for further discussion. We use the abbreviations *GB* and *GBA* to refer to Gröbner Basis and Gröbner Basis Algorithm respectively. Since we will not always let the *GBA* run until it finds a Gröbner Basis, we will often be dealing with sets which are not Gröbner Bases, but rather intermediate results. We call such sets of polynomial equations *partial GB*s.

2.2. *NCProcess*

As we mentioned before, the operation Categorize described in Section 1.2 is a mathematical abstraction and is an idealization of what can be done on a practical level. We now discuss the NCProcess commands which have been implemented and fulfill part of the goals of the Categorize operation.

While the NCProcess commands make heavy use of the GBA, a person can use it *without knowing anything about* GBAs.

2.2.1. *The Input and Output of NCProcess*

The *input* to the NCProcess commands is:

I1. A list of knowns.

I2. A list of unknowns (together with an order which gives you priorities for eliminating them, see Section 7).

I3. Collections C and C' of equations in these knowns and unknowns.[12]

The *output* of the NCProcess commands is a list of expressions which are mathematically equivalent to $C \cup C'$. When using NCProcess1, this equivalent list hopefully has solved for some unknowns. The output is presented to the user as

O1. Unknowns which have been solved for and equations which yield these unknowns.

O2. Equations involving no unknowns.

O3. Equations selected or created by the user.[13] An example is given in the context of S1 and S2 below (Section 2.3) There are also times during a strategy or prestrategy when one wants to introduce new variables and equations. This is illustrated in Section 3.

O4. Equations involving only one unknown.

O5. Equations involving only 2 unknowns, etc.

We say that an equation which is in the output of an NCProcess command is *digested* if it occurs in items O1, O2, or O3 and is *undigested* otherwise. Often, in practice, the digested polynomial equations are those which are well understood.

We now turn to an example. In Section 2.2.4, we compare the NCProcess commands to the Categorize operation which was described in Section 1.2.

2.2.2. *A simple Example of NCProcess*1

A simple example of the use of NCProcess1 is to classify $m \times n$ matrices[14] x such that

$$\begin{bmatrix} U & x \\ 0 & W \end{bmatrix} \tag{2.2}$$

[12] C' consists of equations which are selected or created by the user. In the first run of an NCProcess command, the set C' is empty. More details about selecting equations are found in Section 2.3. See also item O3 below.

[13] In the first run of an NCProcess command, there are no equations which are selected or created by the user. A user-selected equation is a polynomial equation which the user has selected. When an equation is selected, the algorithms described in Section 12 treat these equations as "digested." The selected equation is now given the highest priority in eliminating other equations when NCProcess runs. For example, equations which one knows can be solved by Matlab can be selected. These equations of this item (O3) are the equations of C' from item I3 above. More details about selecting equations are found in Section 2.3.

[14] This session will also classify bounded linear transformations x. This is slightly less trivial, because we are not necessarily working in a finite dimensional space.

is an isometry where U is given, U is known to be unitary and W is given. Thus, we are trying to solve the following collection of equations[15] for x.

$$\begin{gathered} U^*U-1=0 \qquad UU^*-1=0 \qquad U^*x=0 \\ x^*U=0 \qquad x^*x+W^*W=1 \end{gathered} \tag{2.3}$$

Now, of course, since U^* is invertible, $x=0$. Let us see how NCProcess1 behaves on the equations of (2.3). NCProcess1 creates the following output which we call a "spreadsheet." The → appearing in the spreadsheet below may be read as an equal sign.

```
                    YOUR SESSION HAS DIGESTED
                    THE FOLLOWING RELATIONS

THE FOLLOWING VARIABLES HAVE BEEN SOLVED FOR:
{x, x*}
The corresponding rules are the following:
x → 0
x* → 0

The expressions with unknown variables {}
and knowns {U, W, U*, W*}
UU* → 1
U*U → 1
W*W → 1

                  USER CREATIONS APPEAR BELOW

              SOME RELATIONS WHICH APPEAR BELOW
                      MAY BE UNDIGESTED

THE FOLLOWING VARIABLES HAVE NOT BEEN SOLVED FOR:
{}
```

The polynomials listed in the spreadsheet above[16] generated the same ideal as those in (2.3). Therefore, any solution to (2.3) is a solution of the equations in the spreadsheet and vice-versa. Therefore, if U is unitary, then the matrix in (2.2) is an isometry if and only if x is the zero matrix and W is an isometry.

[15] Even though this article is not intended as a user manual for the software which we have developed, it is important to note that there is often very little typing required in order to input collections of equations. For example, one does *not* type in the 5 equations of (2.3), but types in the 2×2 block matrix and tells the computer that it is an isometry and that U is unitary.

[16] More precisely, the spreadsheet displays polynomial equations. We shall continue to blur the distinction between the polynomial equation $p=q$ and the polynomial $p-q$.

2.2.3. *Categories*

Suppose we are given a set V of unknown variables and a set of polynomial equations C. By the *V-category* of C we mean the collection of polynomial equations $p=q$ of C such that the set of unknowns appearing in $p-q$ is exactly V. That is, $p=q$ is in the V-category of C if and only if $p=q$ is in C and each element in V is a variable in $p-q$ and each unknown in $p-q$ belongs to V. The spreadsheet shown above has three non-empty categories: a $\{\ \}$-category which equals $\{U^*U=1,\ UU^*=1,\ W^*W=1\}$, a $\{x\}$-category which equals $\{x=0\}$ and a $\{x^*\}$-category which equals $\{x^*=0\}$. In addition to V-categories, we shall keep track of the *singleton category* which consists of equations of the form $v=p$ where v is a single variable, p is a polynomial and satisfies the technical condition that v is greater than any of the terms appearing in p with respect to the monomial order (see Section 7). In the example above, the singleton category consists of the equations $x=0$ and $x^*=0$. These equations are very useful because they say one unknown can be solved for in terms of the others variables (and therefore eliminated). Note there is a (harmless) overlap between the singleton category and V-categories.

2.2.4. *Comparison of NCProcess and Categorize*

Structurally, the NCProcess commands act like Categorize, except that Categorize would create an infinite number of sets each of which are typically infinite and NCProcess creates a finite number of finite sets. The following paragraph makes the above statement more precise.

Categorize associates an infinite number of typically infinite sets[17] $\{P_j\}_{j\geqslant 0}$ to the sets C and C' (see item I3 in Section 2.2.1). The union of the P_j's is the set $\mathscr{I}_{C\cup C'}\backslash\mathscr{I}_{C'}$. Therefore, the union of $\mathscr{I}_{C'}$ and the P_j's is $\mathscr{I}_{C\cup C'}$. On the other hand, when NCProcess is given two collections of equations C and C', NCProcess computes a finite number of finite sets $\{Q_j\}_{0\leqslant j\leqslant N}$ such that the four properties appearing below hold. Properties 1, 2, and 3 show that a major difference between the Q_j's and the P_j's is that the union of $\mathscr{I}_{C'}$ and the P_j's is the ideal $\mathscr{I}_{C\cup C'}$, whereas the union of the Q_j's and C' is a *generating set* for the ideal $\mathscr{I}_{C\cup C'}$. Property 4 involves a difference in the way the equations are presented in comparison to the figure in Section 1.2.2.

(1) For all j, the elements of Q_j are equations which depend on exactly j unknowns.

[17] Either the set P_j is infinite, equals $\{0\}$ or equals the empty set.

[18] Deleted in proof.

(2) For $j \geqslant 1$, Q_j is a subset of $\mathscr{I}_{C \cup C'}$.

(3) The smallest ideal generated by the union of C' and the sets Q_j's is $\mathscr{I}_{C \cup C'}$.

(4) If there is an equation in Q_j which is in the singleton category (see Section 2.2.3), then this equation is displayed according to item O1 rather than according to items O4 and O5.

The sets Q_j above can be related to categories as described in Section 2.2.3. In a trivial way, the set Q_j is precisely the union of V-categories of D where V ranges over all sets of exactly j unknowns and where D is the union of the Q_j's.

Both NCProcess1 and NCProcess2 use algorithms which remove redundant equations (see Section 1.4) when forming the sets Q_j. Given a particular input (see Section 2.2.1), the sets Q_j which NCProcess1 creates are, in general, different from the sets Q_j which NCProcess2 creates. This is where the two differences between NCProcess1 and NCProcess2 lie. We now describe these two differences. Firstly, NCProcess1 runs a GBA before removing redundant equations, but NCProcess2 does not run a GBA before removing redundant equations. The role of NCProcess2 is to find a smaller generating set—not to find new equations which hold. Secondly, the algorithm which NCProcess2 uses for removing redundant equations is more aggressive than that of NCProcess1 and, therefore, can take a longer time to execute. NCProcess2 is used only when it is believed that all of the interesting equations for the problem under consideration have been found. NCProcess2 tries to assure that its output has the property that the kth polynomial on the spreadsheet is not in the ideal generated by the first $k-1$ polynomials.[19] The output of NCProcess2 is often smaller (and is never larger) than its input.

2.3. *Prestrategy*

The idea of a *prestrategy* is:

The input to a prestrategy is a set of equations C.

S0. Set $C' = \{\}$ (see Section 2.2.1).

S1. Run NCProcess1 which creates a display of the output (see O1–O5 in Section 2.2) and look at the list of equations involving only one unknown (say a particular equation E_{17} has exactly one unknown x_3).

[19] NCProcess2 cannot guarantee this property, since the ideal membership problem is known to be unsolvable [TMora].

S2. The user must now make a decision about equations in x_3 (e.g., E_{17} is a Riccati equation so I shall not try to simplify it, but leave it for Matlab). Now the user declares the unknown x_3 to be known. The user would also select the equation E_{17} as important. User selecting an equation corresponds to adding it to the set C'.

S3. Either do the "End game" (see Section 2.3.1) or Go to S1.

The above listing is, in fact, a statement of a 1-*prestrategy*. Sometimes one needs a 2-*prestrategy* in that the key is equations in 1 and 2 unknowns.

Another point is that the user can select certain equations which he deems important. The NCProcess commands give these priority over subsequent equations when eliminating unknowns.

The point is to isolate and to minimize what the user must do. This is the crux of a prestrategy.

2.3.1. *When to Stop; the End Game*

The prestrategy described above is a loop and we now discuss when to exit the loop.

The digested equations (those in items O1, O2 and O3) often contain the necessary conditions of the desired theorem and the main flow of the proof of the converse. If the starting polynomial equations follow as algebraic consequences of the digested equations, then we should exit the above loop. The starting equations, say $\{p_1=0, ..., p_{k_1}=0\}$, follow as algebraic consequences of the digested equations, say $\{q_1=0, ..., q_{k_2}=0\}$, if and only if the Gröbner Basis generated by $\{q_1, ..., q_{k_2}\}$ reduces (in a standard way) the polynomial p_j to 0 for $1 \leqslant j \leqslant k_1$. Checking whether or not this happens is a purely mechanical process. (See Section 10).

When one exits the above loop, one is presented with the question of how to finish of the proof of the theorem. We shall call the steps required to go from a final spreadsheet to the actual theorem the "end game." We shall describe some "end game" technique in Section 8. We shall illustrate the "end game" in Section 3.4 and Section 6.3. As we shall see, typically the first step is to run NCProcess2 whose output is a very small set of equations.

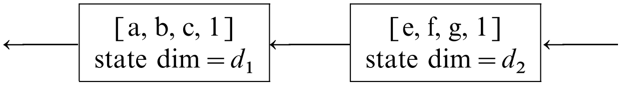
3. EXAMPLE: THE BART–GOHBERG–KAASHOEK–VAN DOOREN THEOREM

In this section, we derive a theorem due to Bart, Gohberg, Kaashoek and van Dooren. The reader can *skip the statement of this theorem*

(Section 3.1) if he wishes and go directly to the algebraic problem statement (Section 3.2).

3.1. *Background*

Definition. A factorization



of a system $[A, B, C, 1]$ is minimal if the statespace dimension of $[A, B, C, 1]$ is $d_1 + d_2$.

Theorem [BGKvD]). *Minimal factorizations of a system* $[A, B, C, 1]$ *correspond to projections* P_1 *and* P_2 *satisfying* $P_1 + P_2 = 1$,

$$AP_2 = P_2 A P_2 \qquad \textit{and} \qquad (A - BC)\, P_1 = P_1(A - BC)\, P_1. \tag{3.1}$$

We begin by giving the algebraic statement of the problem. Suppose that these factors exist. By the Youla–Tissi statespace isomorphism theorem, there is map

$$(m_1, m_2)\colon \text{Statespace of the product} \to \text{Statespace of the original} \tag{3.2}$$

which intertwines the original and the product system. Also minimality of the factoring is equivalent to the existence of a two-sided inverse $(n_1^T, n_2^T)^T$ to (m_1, m_2). These requirements combine to imply that each of the expressions of (FAC) below is zero.

3.2. *The Problem*

Minimal factors exist if and only if there exist $m_1, m_2, n_1, n_2, a, b, c, e, f$ and g such that the following polynomials are zero.

$$\begin{array}{cc} Am_1 - m_1 a - m_2 fc & Am_2 - m_2 e \\ B - m_1 b - m_2 f & -c + Cm_1 \\ n_1 m_1 - 1 & n_2 m_2 - 1 \\ n_1 m_2 & n_2 m_1 \\ -g + Cm_2 & m_1 n_1 + m_2 n_1 - 1 \end{array} \qquad (FAC)$$

Each of these expressions must equal 0. Here A, B and C are known.

The problem is to solve these equations. That is, we want a constructive theorem which says when and how they can be solved.

3.3. *Solution via a Prestrategy*

We now apply a prestrategy to see how one might discover the theorem of Section 3.2.

We run *NCProcess*1 for 2 iterations where the input is the equations (FAC), together with the declaration of A, B, C as knowns and the remaining variables as unknowns. The file created by *NCProcess* is a list of equations whose solution set is the same as the solution set for the FAC equations. The output is the spreadsheet appearing below. (We added the $\Leftarrow ==$ appearing below after the spreadsheet was created.) The → below can be read as an equal sign.

THE ORDER IS NOW THE FOLLOWING:
$A < B < C \ll m_1 \ll m_2 \ll n_1 \ll n_2 \ll a \ll b \ll c \ll d \ll e \ll f \ll g$

YOUR SESSION HAS DIGESTED
THE FOLLOWING RELATIONS

THE FOLLOWING VARIABLES HAVE BEEN SOLVED FOR:
$\{a, b, c, e, f, g\}$
The corresponding rules are the following:

$a \to n_1 A m_1 \qquad b \to n_1 B \qquad c \to Cm_1$
$e \to n_2 A m_2 \qquad f \to n_2 B \qquad g \to Cm_2$

USER CREATIONS APPEAR BELOW

SOME RELATIONS WHICH APPEAR BELOW
MAY BE UNDIGESTED

THE FOLLOWING VARIABLES HAVE NOT BEEN SOLVED FOR:
$\{n_1, n_2, m_1, m_2\}$

The expressions with unknown variables $\{n_1, m_1\}$
and knowns $\{A, B, C\}$

$n_1 m_1 \rightarrow 1$
$(1 - m_1 n_1) A m_1 - (1 - m_1 n_1) B C m_1 = 0 \qquad \Leftarrow ==$

The expressions with unknown variables $\{n_1, m_2\}$
and knowns $\{A\}$

$n_1 m_2 \rightarrow 0$
$n_1 A m_2 \rightarrow 0$

The expressions with unknown variables $\{n_2, m_1\}$
and knowns $\{A, B, C\}$

$n_2 m_1 \rightarrow 0$
$n_2 B C m_1 \rightarrow n_2 A m_1$

The expressions with unknown variables $\{n_2, m_2\}$
and knowns $\{\}$

$n_2 m_2 \rightarrow 1$

The expressions with unknown variables $\{n_2, n_1, m_2, m_1\}$
and knowns $\{\ \}$
$m_2 n_2 \rightarrow 1 - m_1 n_1 \qquad \Leftarrow ==$

The above "spreadsheet" indicates that the unknowns a, b, c, e, f, and g are solved for and states their values. The following are facts about the output: (1) there are no equations in 1 unknown, (2) there are 4 categories of equations in 2 unknowns and (3) there is one category of equations in 4 unknowns. A user must observe that the first equation[20] which we marked with $\Leftarrow ==$ becomes an equation in the unknown quantity $m_1 m_1$ when multiplied on the right by n_1. This motivates the creation of a new variable P defined by setting

$$P_1 = m_1 n_1. \tag{3.3}$$

The user may notice[21] at this point that the second equation marked with $\Leftarrow ==$ is an equation in only one unknown quantity $m_2 n_2$ once the above assignment has been made and P_1 is considered known. These observations lead us to "select" (see footnote corresponding to O2 in Section 2.2) the equations $m_1 n_1 - P_1$ and $m_2 n_2 - 1 + m_1 n_1$. Since we selected an equation in $m_2 n_2$ and an equation in $m_2 n_2$, it is reasonable to select the equations $n_1 m_1 - 1$, and $n_2 m_2 - 1$ because they have exactly the same unknowns.

[20] This polynomial is not written as a rule since it has a collected form as described in Section 9. This collect form can be used to assist a person in finding decompositions (Section 1.2.1). This will be described in Section 5.

[21] If the user does note notice it at this point, it will become very obvious with an additional run of *NCProcess*.

Run *NCProcess*1 again[22] with (3.3) added and P_1 declared known as well as A, B and C declared known. The output is:

THE ORDER IS NOW THE FOLLOWING:
$A < B < C < P_1 \ll m_1 \ll m_2 \ll n_1 \ll n_2 \ll a \ll b \ll c \ll e \ll f \ll g$

YOUR SESSION HAS DIGESTED
THE FOLLOWING RELATIONS

THE FOLLOWING VARIABLES HAVE BEEN SOLVED FOR:
$\{a, b, c, e, f, g\}$
The corresponding rules are the following:

$a \to n_1 A m_1 \qquad b \to n_1 B \qquad c \to C m_1$
$e \to n_2 A m_2 \qquad f \to n_2 B \qquad g \to C m_2$

The expressions with unknown variables $\{\}$
and knowns $\{A, B, C, P_1\}$

$P_1^2 \to P_1$
$-P_1 A(1 - P_1) = 0$
$AP_1 - P_1 A - (1 - P_1)\, BCP_1 = 0$

USER CREATIONS APPEAR BELOW

$m_1 m_1 \to P_1$
$n_1 m_1 \to 1$
$n_2 m_2 \to 1$
$m_2 n_2 \to 1 - m_1 n_1$

SOME RELATIONS WHICH APPEAR BELOW
MAY BE UNDIGESTED

THE FOLLOWING VARIABLES HAVE NOT BEEN SOLVED FOR:
$\{n_1, n_2, m_1, m_2\}$

The expressions with unknown variables $\{n_1, m_1\}$
and knowns $\{P_1\}$

$\Uparrow n_1 m_1 \to 1$
$\Uparrow m_1 n_1 \to P_1$

The expressions with unknown variables $\{n_2, m_2\}$
and knowns $\{\}$

$\Uparrow n_2 m_2 \to 1$

The expressions with unknown variables $\{n_2, n_1, m_2, m_1\}$
and knowns $\{\}$

$\Uparrow m_2 n_2 \to 1 + -m_1 n_1$

Note that the equations in the above display which are in the undigested section (i.e., below the lowest line of thick black lines) are repeats of those which are in the digested section (i.e., above the lowest line of thick black lines). The symbol $\Uparrow$ indicates that the polynomial equation also appears

[22] There is limit of 2 iterations.

as a user select on the spreadsheet. We relist these particular equations simply as a convenience. We will see how this helps us in Section 3.4. Since all equations are digested, we have finished using *NCProcess*1 (see Section 3 in Section 2.3). As we shall see, this output spreadsheet leads directly to the theorem about factoring systems.

3.4. *The End Game*

The first step of the end game is to run *NCProcess*2 on the last spreadsheet which was produced in Section 3.3. The aim of this run of *NCProcess*2 is to shrink the spreadsheet as aggressively as possible without destroying important information. The spreadsheet produced by *NCProcess*2 is the same as the last spreadsheet which was produced[23] in Section 3.3.

Note that it is necessary that all of the equations in the spreadsheet have solutions, since they are implied by the original equations. The equations involving only knowns play a key role. In particular, they say precisely that, there must exist a projection P_1 such that

$$P_1AP_1 = P_1A \qquad \text{and} \qquad P_1BCP_1 = P_1A - AP_1 + BCP_1 \tag{3.4}$$

are satisfied.

The converse is also true and can be verified with the assistance of the above spreadsheet. To do this, we assume that the matrices A, B, C and P_1 are given and that (3.4) holds, and wish to define m_1, m_2, n_1, n_2, a, b, c, e, f and g such that each of the equations in the above spreadsheet hold. If we can do this, then each of the equations from the starting polynomial equations (FAC) given in Section 3.2 will hold and we will have shown that a minimal factorization of the $[A, B, C, 1]$ system exists.

(1) Since $P_1^2 = P_1$, it is easy to show that there exists (not necessarily square) matrices m_1 and n_1 such that $n_1m_1 = 1$ and $m_1n_1 = P_1$. These are exactly the equations in the $\{n_1, m_1\}$-Category of the above spreadsheet.

(2) Since $(1 - P_1)^2 = 1 - P_1$, it is easy to show that there exists (not necessarily square) matrices m_2 and n_2 such that $n_2m_2 = 1$ and $m_2n_2 = 1 - P_1$. These are exactly the equations in the $\{n_2, m_2\}$-Category of the above spreadsheet together with the equations in the $\{n_2, m_2, n_1, m_1\}$-Category of the above spreadsheet.

(3) Since we have defined m_1, m_2, n_1 and n_2, we can define a, b, c, e, f and g by setting $a = n_1Am_1$, $b = n_1B$, $c = Cm_1$, $e = n_2Am_2$, $f = n_2B$ and $g = Cm_2$. These are exactly the equations in the singleton category.

[23] It is not hard to see that *NCProcess*2 would not have an effect, since the set of equations found on the previous spreadsheet can be easily seen to be minimal. We include the run here for pedagogical reasons.

Here we have used the fact that we are working with matrices and not elements of an abstract algebra.

With the assignments made above, every equation in the spreadsheet above holds. Thus, by backsolving through the spreadsheet, we have constructed the factors of the original system $[A, B, C, 1]$. This proves

THEOREM ([BGKvD]). *The system $[A, B, C, 1]$ can be factored if and only if there exists a projection P_1 such that $P_1AP_1 = P_1A$ and $P_1BCP_1 = P_1A - AP_1 + BCP_1$.*

Note that the known equations can be neatly expressed in terms of P_1 and $P_2 = 1 - P_1$. Indeed, it is easy to check with a little algebra that these are equivalent to (3.1). It is a question of taste, not algebra, as to which form one chooses.

For a more complicated example of an end game, see Section 6.3.

3.5. *Concluding Remarks*

We saw that this factorization problem could be solved with a 2-prestrategy. It was not a 1-prestrategy because there was at least at one point in the session where the user had to make a decision about an equation in two unknowns. On the other hand, the assignment (3.3) was a motivated unknown. We will see in Section 5.1 that this is a 1-strategy. For example, the equation

$$(1 - m_1 n_1)\, Am_1 - (1 - m_1 n_1)\, BCm_1 = 0 \tag{3.5}$$

in the two unknowns m_1 and n_1 can be transformed into an equation in the one unknown $m_1 n_1$ by multiplying by n_1 on the right:

$$(1 - m_1 n_1)\, Am_1 n_1 - (1 - m_1 n_1)\, Bcm_1 n_1 = 0. \tag{3.6}$$

If we do not restrict ourselves to the original variables but allow constructions of new variables (according to certain very rigid rules), then the factorization problem is solvable using a generalization of a 1-prestrategy, called a 1-strategy. Section 5 will describe 1-strategies.

The brevity of this presentation in a journal suppresses some of the advantages and some of the difficulties. For example, one might not instantly have all of the insight which leads to the second spreadsheet. In practice, a session in which someone "discovers" this theorem might use many spreadsheets. All that matters is that one makes a little bit of progress with each step. Also, since this paper discusses mathematics and does not explain how to use the code, it is not easy to see that there is very little which needs to be typed and that the computer computations are fast. For example, when performing the computations in Section 2.2.2, one does *not* type in the 5 equations of (2.3). One instead, types in the 2×2 block matrix and tells the computer that it is an isometry and that U is unitary.

4. EXAMPLES: MATRIX COMPLETION PROBLEMS

4.1. *The Partially Prescribed Inverse Problem*

Now we consider a type of problem known as a matrix completion problem. We pick one suggestion by Hugo Woerderman and we are grateful to him for discussions.

Given matrices a, b, c and d, we wish to determine under what conditions there exists matrices x, y, z and w such that the block two by two matrices

$$\begin{bmatrix} a & x \\ y & b \end{bmatrix} \qquad \begin{bmatrix} w & c \\ d & z \end{bmatrix} \tag{4.1}$$

are inverses of each other. Also, we wish to find formulas for x, y, z and w.

This problem was solved in a paper by W. W. Barrett, C. R. Johnson, M. E. Lundquist and H. Woerderman [BJLW] where they showed it splits into several cases depending upon which of a, b, c and d are invertible. In our next example, we assume that a, b, c and d are invertible and derive the result which they obtain. If one runs NCProcess1 on the polynomial equations which state that a, b, c and d are invertible together with the eight polynomial equations which come from the two matrices above being inverses of each other, one gets the spreadsheet:

THE ORDER IS NOW THE FOLLOWING:
$a < a^{-1} < b < b^{-1} < c < c^{-1} < d < d^{-1} \ll z \ll x \ll y \ll w$

YOUR SESSION HAS DIGESTED
THE FOLLOWING RELATIONS

THE FOLLOWING VARIABLES HAVE BEEN SOLVED FOR:
$\{w, x, y\}$
The corresponding rules are the following:
$w \to a^{-1}d^{-1}zbd \qquad x \to d^{-1} - d^{-1}zb \qquad y \to c^{-1} - bzc^{-1}$

The expressions with unknown variables $\{\}$
and knowns $\{a, b, c, d, a^{-1}, b^{-1}, c^{-1}, d^{-1}\}$
$aa^{-1} \to 1 \qquad bb^{-1} \to 1 \qquad cc^{-1} \to 1 \qquad dd^{-1} \to 1$
$a^{-1}a \to 1 \qquad b^{-1}b \to 1 \qquad c^{-1}c \to 1 \qquad d^{-1}d \to 1$

USER CREATIONS APPEAR BELOW

SOME RELATIONS WHICH APPEAR BELOW
MAY BE UNDIGESTED

THE FOLLOWING VARIABLES HAVE NOT BEEN SOLVED FOR:
$\{z\}$

The expressions with unknown variables $\{z\}$
and knowns $\{a, b, c, d\}$
$zbz \to z + dac$

‘

This spreadsheet shows that, if a, b, c and d are invertible, then one can find x, y, z and w such that the matrices in (4.1) are inverses of each other if and only if $zbz = z + dac$. The spreadsheet also gives formulas for x, y, and w in terms of z.

In [BJLW] they also solve the problem in the case that a is not invertible—the answer is more complicated and involves conditions on ranks of certain matrices. It is not clear whether or not these can be derived in a purely algebraic fashion.

4.2. *Unitary Case of Parrot's Lemma*

We begin by stating Parrot's Lemma.

Lemma 4.2 [Y]. *Let a be an $m \times m$ matrix, c be an $n \times m$ matrix and d be an $n \times n$ matrix. There exists an $m \times n$ matrix z such that*

$$\begin{bmatrix} a & z \\ c & d \end{bmatrix} \tag{4.3}$$

is a contraction if and only if

$$\begin{aligned} a^T a + c^T c &\leqslant 1; \\ c c^T + d d^T &\leqslant 1. \end{aligned} \tag{4.4}$$

Since we cannot handle inequalities, we analyze the case where the unknown matrix z must be chosen to make the matrix of (4.3) unitary.

Per our usual advice, when just starting a problem, we take most matrices to be invertible. Since only a and d are square, we assume that a and d are invertible.

If a and d are invertible, then the matrix of (4.3) is unitary if and only if the following polynomials are zero.

$$\begin{matrix} -1 + a^T a + c^T c & a^T z + c^T d & d^T c + z^T a & -1 + d^T d + z^T z \\ -1 + a a^T + z z^T & a c^T + z d^T & c a^T + d z^T & -1 + c c^T + d d^T \\ a^{-1} a - 1 & a a^{-1} - 1 & d^{-1} d - 1 & d d^{-1} - 1 \\ a^{-1T} a^T - 1 & a^T a^{-1T} - 1 & d^{-1T} d^T - 1 & d^T d^{-1T} - 1. \end{matrix} \tag{4.5}$$

When we run NCProcess1 on these equations, we obtain the following spreadsheet:

THE ORDER IS NOW THE FOLLOWING:
$a < a^T < a^{-1} < a^{-1T} < c < c^T < d < d^T < d^{-1} < d^{-1T} \ll z \ll z^T$

YOUR SESSION HAS DIGESTED
THE FOLLOWING RELATIONS

THE FOLLOWING VARIABLES HAVE BEEN SOLVED FOR:
$\{z, z^T\}$
The corresponding rules are the following:
$z \rightarrow -ac^T d^{-1T}$ $\quad$ $z^T \rightarrow -d^T ca^{-1}$

The expressions with unknown variables $\{\}$
and knowns $\{a, c, d, a^{-1}, d^{-1}, a^{-1T}, d^{-1T}, a^T, c^T, d^T\}$
$aa^{-1} \rightarrow 1$ $\quad$ $dd^{-1} \rightarrow 1$ $\quad$ $dd^T \rightarrow 1 - cc^T$ $\quad$ $a^{-1}a \rightarrow 1$
$d^{-1}d \rightarrow 1$ $\quad$ $a^{-1T}a^T \rightarrow 1$ $\quad$ $d^{-1T}d^T \rightarrow 1$ $\quad$ $a^T a^{-1T} \rightarrow 1$
$c^T c \rightarrow 1 - a^T a$ $\quad$ $d^T d^{-1T} \rightarrow 1$

USER CREATIONS APPEAR BELOW

SOME RELATIONS WHICH APPEAR BELOW
MAY BE UNDIGESTED

THE FOLLOWING VARIABLES HAVE NOT BEEN SOLVED FOR:
$\{\}$

Since the above spreadsheet contains the equations $z = -ac^T d^{-1T}$ and $z^T = -d^T ca^{-1}$, the equation $-ac^T d^{-1T} = (-d^T ca^{-1})^T$ follows from the above spreadsheet. The above spreadsheet and the observation just made shows that, if a, c and d are invertible, then there exists a matrix z such that the matrix of (4.3) is unitary if and only if $c^T c + a^T a = 1$, $dd^T + cc^T = 1$ and $-ac^T d^{-1T} = (-d^T ca^{-1})^T$. Moreover, if z exists, then $z = -ac^T d^{-1T}$.

To show that under the above invertibility assumptions, there exists an $m \times n$ matrix z such that the matrix of (4.3) is unitary if and only if $c^T c + a^T a = 1$, $dd^T + cc^T = 1$, it is necessary[24] to show that the equation $-ac^T d^{-1T} = (-d^T ca^{-1})^T$ follows from the equations in the spreadsheet which do not involve either z or z^T. One can either show this by hand or run NCProcess1 on the equations in the above spreadsheet which do

[24] If NCProcess used a symmetric version of Small Basis By Category (see Section 12.7), then either $z = -ac^T d^{-1T}$ or $z^T = -d^T ca^{-1}$ would have appeared on the spreadsheet, but not both. Therefore we would not have to perform this additional step. A symmetric version of the Small Basis By Category option is being written at the time of the writing of this paper. This option will be available in a future version of the software.

not involve either z or z^T together with[25] the equation $-ac^Td^{-1T} = (-d^Tca^{-1})^T$ and see that this equation is redundant. In summary, if a, c and d are invertible, there exists a matrix z such that the matrix of (4.3) is unitary if and only if $c^Tc + a^Ta = 1$ and $dd^T + cc^T = 1$.

Since we assumed that a and d were invertible, the above calculation has a "back of the envelope" flavor. Now that our "back of the envelope" calculation assuming invertibility was successful, it is easy to remove the invertibility assumptions. If we do not assume that a and d are invertible, NCProcess1 produces the spreadsheet:

THE ORDER IS NOW THE FOLLOWING:
$a < a^T < c < c^T < d < d^T \ll z_1 \ll z^T$

YOUR SESSION HAS DIGESTED
THE FOLLOWING RELATIONS

The expressions with unknown variables $\{\}$
and knowns $\{a, c, d, a^T, c^T, d^T\}$
$dd^T \rightarrow 1 - cc^T$
$c^Tc \rightarrow 1 - a^Ta$

USER CREATIONS APPEAR BELOW

SOME RELATIONS WHICH APPEAR BELOW
MAY BE UNDIGESTED

THE FOLLOWING VARIABLES HAVE NOT BEEN SOLVED FOR:
$\{z, z^T\}$

The expressions with unknown variables $\{z\}$
and knowns $\{a, d, a^T, c^T, d^T\}$
$zd^T \rightarrow -ac^T$
$a^Tz \rightarrow -c^Td$

The expressions with unknown variables $\{z^T\}$
and knowns $\{a, c, d, a^T, d^T\}$
$dz^T \rightarrow -ca^T$
$z^Ta \rightarrow -d^Tc$

The expressions with unknown variables $\{z^T, z\}$
and knowns $\{a, d, a^T, d^T\}$
$zz^T \rightarrow 1 - aa^T$
$z^Tz \rightarrow 1 - d^Td$

[25] It is important that the equation $-ac^Td^{-1T} = (-d^Tce^{-1})^T$ appears after the other equations for this run.

One could use these equations to push through to the unitary case of Parrot's Lemma. However, since the theorem is well known, there is not much point in publishing this derivation.

Notice from the spreadsheet above that if d is invertible, then one can solve for z. So far, we only have a result for the case that both a and d are invertible. Let us consider the case when only d is invertible and see what happens. If we do not assume that a or a^T is invertible, but still assume that d is invertible (and so, of course, assume that d^T is invertible), then we obtain the following spreadsheet:

THE ORDER IS NOW THE FOLLOWING:
$a < a^T < c < c^T < d < d^T < d^{-1} < d^{-1T} \ll z \ll z^T$

YOUR SESSION HAS DIGESTED
THE FOLLOWING RELATIONS

THE FOLLOWING VARIABLES HAVE BEEN SOLVED FOR:
$\{z, z^T\}$
The corresponding rules are the following:
$z \rightarrow -ac^T d^{-1T}$
$z^T \rightarrow -d^{-1} c a^T$

The expressions with unknown variables $\{\}$
and knowns $\{a, c, d, d^{-1}, d^{-1T}, a^T, c^T, d^T\}$
$dd^{-1} \rightarrow 1$
$dd^T \rightarrow 1 - cc^T$
$d^{-1}d \rightarrow 1$
$d^{-1T}d^T \rightarrow 1$
$c^T c \rightarrow 1 - a^T a$
$d^T d^{-1T} \rightarrow 1$
$ac^T d^{-1T} d^{-1} c a^T \rightarrow 1 - aa^T$

USER CREATIONS APPEAR BELOW

SOME RELATIONS WHICH APPEAR BELOW
MAY BE UNDIGESTED

THE FOLLOWING VARIABLES HAVE NOT BEEN SOLVED FOR:
$\{\}$

Notice that the last expression in the spreadsheet ($ac^T d^{-1T} d^{-1} c a^T = 1 - aa^T$) shows that $a(c^T d^{-1T} d^{-1} c + 1)\, a^T = 1$. Therefore, the matrix a is onto. Since a is a square matrix, a is invertible. Thus, by using the spreadsheet and the special properties of matrices (rather than elements of an arbitrary abstract algebra) we have discovered that if d is invertible, then a is invertible. At this point in the session we would add the fact that a is invertible, run NCProcess1 and one would obtain the first spreadsheet of Section 4.2.

5. STRATEGIES AND MOTIVATED UNKNOWNS

In a prestrategy unknowns refers to the original unknowns presented in the problem. However, it is often the case that there will be some combination of variables which is a "natural" unknown for the problem. A particular class of these new unknowns, called *motivated unknowns*, are gotten from the 1-Decompose operation. Unfortunately, the Decompose operation is an idealization and the authors do not know an approximation to it which is implementable.

Also, even if Decompose were implementable, one would not be able to use it effectively, since the operation associates a composition of polynomials (and therefore, motivated unknowns) to a *particular* polynomial and applying Decompose to each polynomial in the ideal would not be practical.

For example, suppose that a polynomial $p(a_1, ..., a_r, x_1, ..., x_s)$ appears on a spreadsheet and has the property that there are other polynomials $L(a_1, ..., a_r, x_1, ..., x_s)$ and $R(a_1, ..., a_r, x_1, ..., x_s)$ for which LpR has a 1-decomposition

$$LpR = k(a_1, ..., a_r, q(a_1, ..., a_r, x_1, ..., x_s))$$

where k is a polynomial in one unknown. By the definition of an ideal, LpR is in the ideal represented by the output of the spreadsheet. The polynomial LpR will *not* appear on the spreadsheet, since p appears on the spreadsheet.[26] Therefore, in practice, a person must recognize the L and R, which yield the 1-decomposition. We formalize this as follows.

Definition 5.1. A polynomial p motivates any unknown y via the equation $y = q(a_1, ..., a_r, x_1, ..., x_s)$ if there exist polynomials $L(a_1, ..., a_r, x_1, ..., x_s)$ and $R(a_1, ..., a_r, x_1, ..., x_s)$ and there exists a polynomial in one unknown $k(a_1, ..., a_r, y)$ such that $LpR = k(a_1, ..., a_r, q(a_1, ..., a_r, x_1, ..., x_s))$.

Of course, from the perspective of finding zeros on collections of polynomials, if p has a zero, then LpR has a zero and so k has a zero. Since k is a polynomial in only one unknown variable, finding the zeros of k is bound to be easier than finding the zeros of p.

The definition we give here is not the most general useful definition which we could have given. A more general definition would involve finding a 1-decomposition using any polynomial in the ideal generated by p rather than those of the form LpR. We have found that the definition we give above is sufficient for the problems which we are considering.

[26] The polynomial LpR will not appear on the spreadsheet, since its normal form with respect to p is zero.

While we do not know how to implement the Decompose operation (Section 1.2.1), there is a certain type of “collect” command which we have found very helpful. This “collect” command assists the user in performing decompositions of the polynomial at hand and helps in finding other polynomials in the ideal which would produce motivated unknowns. (See the discussion around equation (3.5).)

This section gives a definition of a strategy and then describes a command which “collects” knowns and products of knowns out of expressions. For example, suppose that A and B are knowns and X, Y and Z are unknowns. The collected form of

$$XABX + XABY + YABX + YABY + AX + AY \tag{5.2}$$

is

$$(X + Y)\,AB(X + Y) + A(X + Y). \tag{5.3}$$

Clearly this suggests a decomposition of (5.2) and indeed the collect command helps find decompositions of much more complicated polynomials.

Next we give a demonstration of how collect enters the NCProcess commands.

5.1. *Strategy*

The idea of a *strategy* is the similar to that of a prestrategy, except that it incorporates motivated unknowns. The idea of a *strategy* is:

The input to a strategy is a set of equations C.

(S0′) Set $C' = \{\}$.

(S1′) Run NCProcess1 which creates a display of the output (see O1–O5 in Section 2.2) which contains collected forms of the polynomial equations. Look at the list of polynomials involving only one unknown or one *motivated* unknown (say a particular equation E_{17} only depends upon the motivated unknown $x_3x_2 + x_1$).

(S2′) The user must now make a decision about equations in $x_3x_2 + x_1$ (e.g., E_{17} is a Riccati equation so I shall not try to simplify it, but leave it for Matlab). Now the user declares a new unknown y and adds the relation $y = x_3x_2 + x_1$ as a user creation. The user would also select the equation E_{17} as important. User selecting an equation corresponds to adding it to the set C'. See Section 5.2.2 for variants on this step.

(S3′) Either do the “End game” (see Section 2.3.1) or Go to S1′.

The above listing is, in fact, a statement of a 1-*strategy*. Sometimes one needs a 2-*strategy* in that the key is equations in 1 or 2 unknowns (motivated or not). Typically, if a problem is solved with a 1-strategy, then the computer has done a lot for you, while if the problem requires a 2-strategy, then it has done less, and with a 3-strategy much less.

As with a prestrategy, the point is to isolate and to minimize what the user must do. This is the crux of a strategy.

5.2. *Computing Decompositions*

In this section we give examples of how one implements the 1-Decompose operation with human intervention.

We use an example to remind the reader of the notion of a decomposition of a polynomial (see Section 1.2.1).

Example 5.4. Suppose A, A^T, B and B^T are knowns and b, d, b^T and d^T are unknown. Consider the polynomial

$$\begin{aligned} p(A, A^T, B, B^T, b, b^T, d, d^T) = 1 - B^TB - dd^T + B^Tbd^T - B^Tbb^TB \\ + db^TB + dA^TAd^T - B^TbA^TAd^T \\ - dA^TAb^TB + B^TbA^TAb^TB \end{aligned} \tag{5.5}$$

It can be written as

$$p = 1 - B^TB - (d - B^Tb)(d^T - b^TB) + (d - B^Tb)\, A^TA(d^T - b^TB) \tag{5.6}$$

which is neatly written as a composition

$$p = k(A, A^T, B, B^T, q, q^T) = 1 - B^TB - qq^T + qA^TAq^T$$

where $q = d - B^Tb$. Note that k is a polynomial whose only unknowns are q and q^T.

We do not know a way to compute decompositions and hopefully this paper's need for a decomposition operator will stimulate a serious quest for a way to compute decompositions. For now, the best we can hope for is a modest amount of computer assistance which is why we turn to the collect-type commands of the next section.

5.2.1. *NCCollectOnVariables*

This subsection concerns a command NCCollectOnVariables, abbreviated NCCV, which is implementable (indeed, it is implemented in

NCAlgebra). A technical description is given in the appendix. Often human intervention can be coupled with NCCV to discover decompositions.

In addition to being an option for the NCProcess commands, it can be called as a stand-alone command. It is used in a session where knowns and unknowns have been declared.

NCCollectOnVariables is a command which "collects" maximal products of knowns and products of knowns out of expressions. Since it collects on monomials which are products of knowns, it typically presents important combinations of knowns in its output.

For example, if in a computer session where A and B are set to be knowns and X, Y and Z are set to be unknowns, then, when NCCollectOnVariables is applied to $XABZ + YABZ + AX + AY$, it returns $(X+Y)\,ABZ + A(X+Y)$. Therefore, $XABZ + YABZ + AX + AY$ depends on a motivated unknown $X+Y$ and an unknown Z.

As another example, if we are in a computer session and A, A^T, B_1, B_1^T, B_2, B_2^T, C_1, C_1^T, C_2, and C_2^T are set to be knowns and all of the other variables are set to be unknowns, then when the command NCCollectOnVariables is applied to

$$\begin{aligned}
&-E_{22}E_{12}^{-1}E_{11}B_2B_2^TE_{11}E_{21}^{-1}E_{22}-E_{21}AE_{21}^{-1}E_{22}+E_{21}B_1B_1^TE_{12}\\
&\quad-E_{21}B_2B_2^TE_{12}-E_{22}E_{12}^{-1}A^TE_{12}+E_{21}B_2C_1E_{21}^{-1}E_{22}\\
&\quad+E_{22}E_{12}^{-1}C_1^TB_2^TE_{12}-E_{21}B_1B_1^TE_{11}E_{21}^{-1}E_{22}\\
&\quad+E_{21}B_2B_2^TE_{11}E_{21}^{-1}E_{22}+E_{22}E_{12}^{-1}E_{11}AE_{21}^{-1}E_{22}\\
&\quad-E_{22}E_{12}^{-1}E_{11}B_1B_1^TE_{12}+E_{22}E_{12}^{-1}E_{11}B_2B_2^TE_{12}\\
&\quad+E_{22}E_{12}^{-1}A^TE_{11}E_{21}^{-1}E_{22}-E_{22}E_{12}^{-1}E_{11}B_2C_1E_{21}^{-1}E_{22}\\
&\quad-E_{22}E_{12}^{-1}C_1^TB_2^TE_{11}E_{21}^{-1}E_{22}+E_{22}E_{12}^{-1}E_{11}B_1B_1^TE_{11}E_{21}^{-1}E_{22}
\end{aligned}$$

we obtain

$$\begin{aligned}
&E_{22}E_{12}^{-1}A^T(E_{12}-E_{11}E_{21}^{-1}E_{22})+(E_{21}-E_{22}E_{12}^{-1}E_{11})\,AE_{21}^{-1}E_{22}\\
&\quad-(E_{21}-E_{22}E_{12}^{-1}E_{11})\,B_1B_1^T(E_{12}-E_{11}E_{21}^{-1}E_{22})\\
&\quad+(E_{21}-E_{22}E_{12}^{-1}E_{11})\,B_2B_2^T(E_{12}-E_{11}E_{21}^{-1}E_{22})\\
&\quad-E_{22}E_{12}^{-1}C_1^TB_2^T(E_{12}-E_{11}E_{21}^{-1}E_{22})\\
&\quad-(E_{21}-E_{22}E_{12}^{-1}E_{11})\,B_2C_1E_{21}^{-1}E_{22}.
\end{aligned}$$

In the problem which this expression comes from (see Section 6), the transpose of $E_{21}-E_{22}E_{12}^{-1}E_{11}$ is $E_{12}-E_{11}E_{21}^{-1}$ (see Appendix A). Therefore, the above expression motivates the introduction of motivated unknowns $Q=E_{21}-E_{22}E_{12}^{-1}E_{11}$ and $Q^T=E_{12}-E_{11}E_{21}^{-1}E_{22}$.

As these examples show, NCCollectOnVariables is an extremely important option for NCProcess.

5.2.2. *Introducing a New Motivated Unknown*

As in the introduction (Section 1.2.1), suppose we are in a context where there are knowns $\{a_1, ..., a_r\}$ and unknowns $\{x_1, ..., x_s\}$. We now describe how we shall be using the NCCV command to help discover decompositions. (In the forthcoming example (see Section 6), Step 3 below is easy enough that it is done by inspection. No run of the GBA is required.) NCCV can be used as follows:

(1) Apply NCCV to a polynomial p and pay particular attention to the terms containing the most knowns. That is, for each term, compute the degree of that term with respect to the set of knowns and use the ones with the highest degree in this sense.

(2) The collected form of p may suggest a decomposition of p or suggest that there is another polynomial in the ideal which has a decomposition. Let us suppose that one of the parenthesized summands of the collected form of p has a decomposition as $k(a_1, ..., a_r,\ q_1(a_1, ..., a_r, x_1, ..., x_s),\ q_2(a_1, ..., a_r, x_1, ..., x_s))$. Therefore, we obtain

$$p=k(a_1, ..., a_r, q_1(a_1, ..., a_r, x_1, ..., x_s), q_2(a_1, ..., a_r, x_1, ..., x_s))+s_1 \tag{5.7}$$

from this operation where k is a polynomial and s_1 is a polynomial.

(3) Declare both of the new variables $Q_1=q_1(a_1, ..., a_r, x_1, ..., x_s)$ and $Q_2=q_2(a_1, ..., a_r, x_1, ..., x_s)$, set Q_1 and Q_2 lower in the ordering than all other unknowns and run the GBA. This will convert (5.7) to

$$p=k_2(a_1, ..., a_r, q_1(a_1, ..., a_r, x_1, ..., x_s), q_2(a_1, ..., a_r, x_1, ..., x_s))+s_2.$$

If s_2 contains no unknowns, then we have found the desired decomposition.

For example, if, in a computer session, A, B, A^T and B^T are set to be knowns and b, d, b^T and d^T are set to be unknowns, then

$$1-dd^T+B^T(bd^T-(1+bb^T)\,B)+db^TB-(d-B^Tb)\,A^TA(b^TB-d^T) \tag{5.8}$$

is the output of NCCollectOnVariables when applied to (5.5). This suggests we set $Q=d-B^Tb$ and $Q^T=d^T-b^TB$. Step 3 above converts (5.8) to

$$p=1-B^TB-QQ^T+QA^TAQ^T \tag{5.9}$$

which gives the decomposition $k(A, A^T, B, B^T, Q, Q^T)$ for p as described in Example 5.4.

6. EXAMPLE: SOLVING THE H^∞ CONTROL PROBLEM

In this section we give an example of solving a problem using a strategy. We will repeatedly call the *NCProcess*1 command and from time to time the Decompose operation will be approximated with a little human intervention.

A basic problem in systems engineering is to make a given system dissipative by designing a feedback law. We now give a demonstration of how one discovers the algebraic part of the solution to this problem. The engineering motivation is in Appendix D.

6.1. *Problem Statement*

Let H_{xx}, H_{xz}, H_{zx}, and H_{zz} be defined as follows.

$$\begin{aligned}
H_{xx} &= E_{11}A+A^TE_{11}+C_1^TC_1+E_{12}bC_2+C_2^Tb^TE_{12}^T+E_{11}B_1b^TE_{12}^T\\
&\quad +E_{11}B_1B_1^TE_{11}+E_{12}bb^TE_{12}^T+E_{12}bB_1^TE_{11}\\
H_{xz} &= E_{21}A+\frac{a^T(E_{21}+E_{12}^T)}{2}+c^TC_1+E_{22}bC_2+c^TB_2^TE_{11}^T\\
&\quad +\frac{E_{21}B_1b^T(E_{21}+E_{12}^T)}{2}+E_{21}B_1B_1^TE_{11}^T\\
&\quad +\frac{E_{22}bb^T(E_{21}+E_{12}^T)}{2}+E_{22}bB_1^TE_{11}^T\\
H_{zx} &= A^TE_{21}^T+C_1^Tc+\frac{(E_{12}+E_{21}^T)\,a}{2}+E_{11}B_2c+C_2^Tb^TE_{22}^T\\
&\quad +E_{11}B_1b^TE_{22}^T+E_{11}B_1B_1^TE_{21}^T\\
&\quad +\frac{(E_{12}+E_{21}^T)\,bb^TE_{22}^T}{2}+\frac{(E_{12}+E_{21}^T)\,bB1^TE_{21}^T}{2}\\
H_{zz} &= E_{22}a+a^TE_{22}^T+c^Tc+E_{21}B_2c+c^TB_2^TE_{21}^T+E_{21}B_1b^TE_{22}^T\\
&\quad +E_{21}B_1B_1^TE_{21}^T+E_{22}bb^TE_{22}^T+E_{22}bB_1^TE_{21}^T.
\end{aligned}$$

The math problem we address is:

(HGRAIL) *Let A, B_1, B_2, C_1, C_2 be matrices of compatible size be given. Solve $H_{xx}=0$, $H_{xz}=0$, $H_{zx}=0$, and $H_{zz}=0$ for a, b, c and for E_{11}, E_{12}, E_{21} and E_{22}. When can they be solved? If these equations can be solved, find formulas for the solution.*

We shall make the strong assumption that each E_{ij} is invertible. While this assumption turns out to be valid, making it at this point is cheating. Ironically, we recommend strongly that the user make heavy invertibility assumptions at the outset of a session. Later, after the main ideas have been discovered, the user can selectively relax them and thereby obtain more general results.

6.2. *Solving (HGRAIL) Using NCProcess*

The first step is to assemble all of the key polynomial equations in executable form:

The polynomials we shall input to NCProcess1 are naturally thought of in several groups. First, to enforce that the 2×2 matrix $(E_{i,\,j})_{i=1,\,2,\,j=1,\,2}$ is symmetric, we require each of the following polynomials to be zero:

$$E_{11}^T - E_{11}, \qquad E_{12}^T - E_{21}, \qquad E_{21}^T - E_{12}, \qquad E_{22}^T - E_{22}, \qquad E_{11}^{-1T} - E_{11}^{-1},$$
$$E_{12}^{-1T} - E_{21}^{-1}, \qquad E_{21}^{-1T} - E_{12}^{-1}, \qquad E_{22}^{-1T} - E_{22}^{-1}.$$

We also assume that $E_{ij}$is invertible for $i=1, 2$ and $j=1, 2$. We assume the following polynomials are zero:

$$E_{11}^{-1}E_{11}-1 \qquad E_{12}^{-1}E_{12}-1 \qquad E_{21}^{-1}E_{21}-1 \qquad E_{22}^{-1}E_{22}-1$$
$$E_{11}^TE_{11}^{-1T}-1 \quad E_{12}^TE_{12}^{-1T}-1 \quad E_{21}^TE_{21}^{-1T}-1 \quad E_{22}^TE_{22}^{-1T}-1.$$

Naturally we also assume the following polynomials are zero:

$$H_{xx}, \qquad H_{xz}, \qquad H_{zx}, \qquad H_{zz}.$$

The multigraded lexicographic order (see Section 7) which we use is: $A < A^T < B_1 < B_1^T < B_2 < B_2^T < C_1 < C_1^T < C_2 < C_2^T \ll E_{12} \ll E_{12}^T \ll E_{21} \ll E_{21}^T \ll E_{22} \ll E_{22}^T \ll E_{11} \ll E_{11}^T \ll E_{12}^{-1} \ll E_{12}^{-1T} \ll E_{21}^{-1} \ll E_{21}^{-1T} \ll E_{22}^{-1} \ll E_{22}^{-1T} \ll E_{11}^{-1} \ll E_{11}^{-1T} \ll b \ll b^T \ll c \ll c^T \ll a \ll a^T$.

We ran NCProcess1 for 2 iterations with the option *NCCollectOnVariables* turned on. *NCCollectOnVariables* was applied to each equation. The algorithm did not run the full two iterations but finished after one. Our program produced a message saying that, in fact, the output is a Gröbner Basis (rather than a partial GB).

6.2.1. *Step* 1: *Process and Collect*

There is no point in listing the full spreadsheet here. Indeed, the only nontrivial undigested polynomial equations are:

The expressions with unknown variables $\{b^T, b, E_{21}^{-1}, E_{12}^{-1}, E_{11}\}$
and knowns $\{A, B_1, C_1, C_2, A^T, B_1^T, C_1^T, C_2^T\}$

$bb^T + bC_2E_{12}^{-1} + E_{12}^{-1}C_2^Tb^T + E_{12}^{-1}E_{11}AE_{21}^{-1} + E_{12}^{-1}E_{11}B_1b^T + E_{12}^{-1}A^TE_{11}E_{21}^{-1} + E_{12}^{-1}C_1^TC_1E_{21}^{-1} + (b + E_{12}^{-1}E_{11}B_1)\, B_1^TE_{11}E_{21}^{-1} = 0$

The expressions with unknown variables $\{c^T, c, E_{21}^{-1}, E_{12}^{-1}, E_{11}, E_{22}, E_{21}, E_{12}\}$
and knowns $\{A, B_1, B_2, C_1, A^T, B_1^T, B_2^T, C_1^T\}$

$c^Tc + (E_{21} - E_{22}E_{12}^{-1}E_{11})\, B_2c + c^TB_2^T(E_{12} - E_{11}E_{21}^{-1}E_{22}) - E_{22}E_{12}^{-1}A^T(E_{12} - E_{11}E_{21}^{-1}E_{22}) - E_{22}E_{12}^{-1}C_1^T(c - C_1E_{21}^{-1}E_{22}) - (E_{21} - E_{22}E_{12}^{-1}E_{11})\, AE_{21}^{-1}E_{22} + (E_{21} - E_{22}E_{12}^{-1}E_{11})\, B_1B_1^T(E_{12} - E_{11}E_{21}^{-1}E_{22}) - c^TC_1E_{22}^{-1}E_{22} = 0$

6.2.2. *Step* 2: *The User Attacks*

Now the reader must apply his expertise to the nontrivial polynomial equations left undigested by the *NCProcess*1 command. A key observation is that the first key polynomial equation contains b but not c and the second key polynomial equation contains c but not b. In other words, b and c appear in decoupled equations.

Observe that the first key polynomial from the above spreadsheet is quadratic in b. We could complete the square and put the polynomial in the form

$$(b + \mu)(b^T + \mu^T) + v \tag{6.10}$$

where μ and v are expressions involving C_2, C_2^T, B_1, B_1^T, A, A^T, E_{21}^{-1}, E_{12}^{-1} and E_{11}. Since there are many unknowns in the problem, there is probably excess freedom. Let us investigate what happens when we take $b + \mu = 0$. This yields the polynomial equation

$$b = -E_{12}^{-1}C_2^T - E_{12}^{-1}E_{11}B_1 \tag{6.11}$$

which we could add to the starting polynomial equations and proceed. We can also complete the square for the expression in c and put that expression in the form

$$(c + \lambda)(c^T + \lambda^T) + \gamma. \tag{6.12}$$

We also assume that $c+\lambda=0$. This defines c by the following equation

$$c=-B_2^T E_{12}+C_1 E_{21}^{-1} E_{22}+B_2^T E_{11} E_{21}^{-1} E_{22}. \tag{6.13}$$

Since we have now solved for b and c, we can use these equations to solve for b^T and c^T.

6.2.3. *Step* 3

The starting polynomial equations for this step will be the output from the first call to *NCProcess*1 (above) as well as the four new equations that we have just derived. We ran *NCProcess*1 for two iterations.[27]

Once again we go directly to the spreadsheet which *NCProcess*1 created. There is no need to record all of it here, since at this stage we shall be concerned only with the undigested polynomial equations. There are only two undigested polynomial equations which are not banal.

The expressions with unknown variables $\{E_{11}\}$
and knowns $\{A, B_1, C_1, C_2, A^T, B_1^T, C_1^T, C_2^T\}$

$E_{11}B_1C_2 \rightarrow E_{11}A+A^TE_{11}+C_1^TC_1-C_2^TC_2-C_2^TB_1^TE_{11}$

The expressions with unknown variables $\{E_{21}^{-1}, E_{12}^{-1}, E_{11}, E_{22}, E_{21}, E_{12}\}$
and knowns $\{A, B_1, B_2, C_1, A^T, B_1^T, B_2^T, C_1^T\}$

$E_{22}E_{12}^{-1}A^T(E_{12}-E_{11}E_{21}^{-1}E_{22})+(E_{21}-E_{22}E_{12}^{-1}E_{11})\,AE_{21}^{-1}E_{22}-(E_{21}-E_{22}E_{12}^{-1}E_{11})\,B_1B_1^T(E_{12}- E_{11}E_{21}^{-1}E_{22})+(E_{21}-E_{22}E_{12}^{-1}E_{11})\,B_2B_2^T(E_{12}-E_{11}E_{21}^{-1}E_{22})-E_{22}E_{12}^{-1}C_1^TB_2^T(E_{12}-E_{11}E_{21}^{-1}E_{22})- (E_{21}-E_{22}E_{12}^{-1}E_{11})\,B_2C_1E_{21}^{-1}E_{22}=0$

6.2.4. *Step* 4.

Now we analyze these two polynomial equations.

The first polynomial equation is an equation in E_{11}, which fortunately is a Riccati–Lyapunov equation. Numerical methods for solving Riccati equations are common. For this reason assuming that a Riccati equation has a solution is a socially acceptable necessary condition throughout control engineering. Thus we can consider E_{11} to be known.

A first glance at the second equation reveals that the same products of unknowns appear over and over. Also we can see that this equation is symmetric. It would not take an experienced person long to realize that by multiplying this equation on the left by $E_{12}E_{22}^{-1}$ and on the right by $E_{22}^{-1}E_{21}$, we will have an equation in one unknown.

[27] In addition, we bound iterations using the options DegreeCap → 6, DegreeSumCap → 9 which limit the degrees of polynomials occurring as the GBA runs. See Section 13.

Now we can replace $E_{11}-E_{12}E_{22}^{-1}E_{21}$ with a new variable X. This yields

$$-XA-A^TX+XB_2C_1+C_1^TB_2^TX-XB_1B_1^TX+XB_2B_2^TX.$$

Observe that this is an equation in the one unknown X.

6.3. *End Game*

Now let us compare what we have found to the well known solution of (*HGRAIL*). In that theory there are two Riccati equations due to Doyle, Glover, Khargonekar and Francis. These are the DGKF X and Y equations. One can read off that the E_{11} equation which we found is the DGKF equation for Y^{-1}, while the Riccati equation which we just analyzed is the DGKF X equation.

Indeed what we have proved is that if (HGRAIL) *has a solution with E_{ij} invertible and if b and c are given by formulas* (6.11) *and* (6.13), *then*

(1) *the* DGKF *X and Y^{-1} equations must have a solution*;

(2) *X and Y are self-adjoint*;

(3) *$Y^{-1}-X$ is invertible.*

Now we turn to the converse. The straightforward converse of the above italicized statement would be: If items (1), (2), and (3) above hold, then (*HGRAIL*) has a solution with E_{ij} invertible and b and c are given by formulas (6.11) and (6.13). There is no reason to believe (and it is not the case) that b and c *must* be given by the formulas (6.11) and (6.13). These two formulas came about in Section 6.2.2 and were motivated by "excess freedom" in the problem. The converse which we will attempt to prove is:

Proposed Converse 6.14. *If items* (1), (2), *and* (3) *above hold, then* (*HGRAIL*) *has a solution with E_{ij} invertible.*

To obtain this proposed converse, we need a complete spreadsheet corresponding to the last stages of our analysis. The complete spreadsheet is:

THE ORDER IS NOW THE FOLLOWING:

$A < A^T < B_1 < B_1^T < B_2 < B_2^T < C_1 < C_1^T < C_2 < C_2^T < X < X^{-1} < Y < Y^{-1} \ll E_{12} \ll E_{21} \ll E_{22} \ll E_{12}^T \ll E_{21}^T \ll E_{22}^T \ll E_{11} \ll E_{11}^T \ll E_{11}^{-1} \ll E_{11}^{-1T} \ll E_{12}^{-1} \ll E_{21}^{-1} \ll E_{22}^{-1} \ll E_{12}^{-1T} \ll E_{21}^{-1T} \ll E_{22}^{-1T} \ll b \ll b^T \ll c \ll c^T \ll a \ll a^T$

YOUR SESSION HAS DIGESTED
THE FOLLOWING RELATIONS

THE FOLLOWING VARIABLES HAVE BEEN SOLVED FOR:

$\{a, b, c, E_{11}, E_{11}^{-1}, a^T, b^T, c^T, E_{11}^T, E_{12}^T, E_{21}^T, E_{22}^T, E_{11}^{-1T}, E_{12}^{-1T}, E_{21}^{-1T}, E_{22}^{-1T}\}$

The corresponding rules are the following:

$a \to -E_{12}^{-1}A^TE_{12} + E_{12}^{-1}C_1^TB_2^TE_{12} + E_{12}^{-1}C_2^TB_1^TE_{12} + E_{12}^{-1}E_{11}B_2B_2^TE_{12} - E_{12}^{-1}C_1^TC_1E_{21}^{-1}E_{22} - E_{12}^{-1}E_{11}B_2C_1E_{21}^{-1}E_{22} - E_{12}^{-1}C_1^TB_2^TE_{11}E_{21}^{-1}E_{22}^{-1} - E_{12}^{-1}E_{11}B_2B_2^TE_{11}E_{21}^{-1}E_{22}$

$b \to -E_{12}^{-1}C_2^T - E_{12}^{-1}E_{11}B_1$

$c \to -B_2^TE_{12} + C_1E_{21}^{-1}E_{22} + B_2^TE_{11}E_{21}^{-1}E_{22}$

$E_{11} \to Y^{-1}$

$E_{11}^{-1} \to Y$

$a^T \to -E_{21}AE_{21}^{-1} + E_{21}B_1C_2E_{21}^{-1} + E_{21}B_2C_1E_{21}^{-1} + E_{21}B_2B_2^TE_{11}E_{21}^{-1} - E_{22}E_{12}^{-1}C_1^TC_1E_{21}^{-1} - E_{22}E_{12}^{-1}E_{11}B_2C_1E_{21}^{-1} - E_{22}E_{12}^{-1}C_1^TB_2^TE_{11}E_{21}^{-1} - E_{22}E_{12}^{-1}E_{11}B_2B_2^TE_{11}E_{21}^{-1}$

$b^T \to -C_2E_{21}^{-1} - B_1^TE_{11}E_{21}^{-1}$

$c^T \to -E_{21}B_2 + E_{22}E_{12}^{-1}C_1^T + E_{22}E_{12}^{-1}E_{11}B_2$

$E_{11}^T \to E_{11}$	$E_{12}^T \to E_{21}$	$E_{21}^T \to E_{12}$	$E_{22}^T \to E_{22}$
$E_{11}^{-1T} \to E_{11}^{-1}$	$E_{12}^{-1T} \to E_{21}^{-1}$	$E_{21}^{-1T} \to E_{12}^{-1}$	$E_{22}^{-1T} \to E_{22}^{-1}$

The expressions with unknown variables $\{\}$
and knowns $\{A, B_1, B_2, C_1, C_2, X, Y, X^{-1}, Y^{-1}, A^T, B_1^T, B_2^T, C_1^T, C_2^T\}$

$XX^{-1} \to 1$
$YY^{-1} \to 1$
$X^{-1}X \to 1$
$Y^{-1}Y \to 1$
$Y^{-1}B_1C_2 \to Y^{-1}A + A^TY^{-1} + C_1^TC_1 - C_2^TC_2 - C_2^TB_1^TY^{-1}$
$XB_2B_2^TX \to XA + A^TX - XB_2C_1 - C_1^TB_2^TX + XB_1B_1^TX$

USER CREATIONS APPEAR BELOW

$E_{11}^{-1} \to Y$
$E_{12}E_{22}^{-1}E_{21} \to E_{11} - X$

SOME RELATIONS WHICH APPEAR BELOW
MAY BE UNDIGESTED

THE FOLLOWING VARIABLES HAVE NOT BEEN SOLVED FOR:

$\{E_{12}, E_{21}, E_{22}, E_{12}^{-1}, E_{21}^{-1}, E_{22}^{-1}\}$

The expressions with unknown variables $\{E_{12}^{-1}, E_{12}\}$
and knowns $\{\}$

$E_{12}E_{12}^{-1} \to 1$
$E_{12}^{-1}E_{12} \to 1$

The expressions with unknown variables $\{E_{21}^{-1}, E_{21}\}$
and knowns $\{\}$

$E_{21}E_{21}^{-1} \to 1$
$E_{21}^{-1}E_{21} \to 1$

The expressions with unknown variables $\{E_{22}^{-1}, E_{22}\}$
and knowns $\{\}$

$E_{22}E_{22}^{-1} \to 1$
$E_{22}^{-1}E_{22} \to 1$

The expressions with unknown variables $\{E_{22}^{-1}, E_{11}, E_{21}, E_{12}\}$
and knowns $\{X\}$

$\Uparrow E_{12}E_{22}^{-1}E_{21} \to E_{11} - X$

In the spreadsheet, we use conventional X, Y^{-1} notation rather than "discovered" notation so that our arguments will be familiar to experts in the field of control theory.

Now we use the above spreadsheet to verify the proposed converse. To do this, we assume that matrices A, B_1, B_2, C_1, C_2, X and Y exist, that X and Y are invertible, that X and Y are selfadjoint, that $Y^{-1}-X$ is invertible and that the DGKF X and Y^{-1} equations hold. That is, the two following polynomial equations hold.

$$Y^{-1}B_1C_2 = Y^{-1}A + A^TY^{-1} + C_1^TC_1 - C_2^TC_2 - C_2^TB_1^TY^{-1}$$

$$XB_2B_2^TX = XA + A^TX - XB_2C_1 - C_1^TB_2^TX + XB_1B_1^TX$$

We wish to assign values for the matrices E_{12}, E_{21}, E_{22}, E_{11}, a, b and c such that each of the equations on the above spreadsheet hold. If we can do this, then each of the equations from the starting polynomial equations from Section 6.1 will hold and the proposed converse will follow.

(1) Note that all of the equations in the $\{\}$-Category of the above spreadsheet hold since X and Y solve the DGKF equations and are both invertible.

(2) Set E_{11} equal to the inverse of Y. This assignment is dictated by the user selects. Note that $E_{11}=E_{11}^T$ follows since Y is self-adjoint.

(3) Let E_{12} and E_{21} be any invertible matrices such that $E_{12}^T=E_{21}$. For example, one could choose E_{12} and E_{21} to both be the identity matrix.

(4) Note that there is there is a user select $E_{12}E_{22}^{-1}E_{21}=E_{11}-X$ and that E_{12}, E_{21} are invertible. Since $Y^{-1}-X$ is invertible and $E_{11}=Y^{-1}$, $E_{11}-X$ is invertible. Therefore, we set $E_{22}=E_{21}^{-1}(E_{11}-X)^{-1}E_{21}^{-1}$. Since $E_{12}^T=E_{21}$, $E_{11}^T=E_{11}$ and $X^T=X$, it follows that E_{22} is invertible and self-adjoint.

(5) Since E_{ij} has been set for $i, j=1, 2$, we can set a, b and c according to their formulas at the top of the spreadsheet.

With the assignments of E_{12}, E_{21}, E_{22}, E_{11}, a, b, and c as above, it is easy to verify by inspection that every polynomial equation on the spreadsheet above holds.

We have proven the proposed converse and, therefore, have proven the following approximation to the classical [DGKF] theorem.

Theorem 6.15. *If* (*HGRAIL*) *has a solution with invertible* E_{ij} *and* b *and* c *are given by the formulas* (6.11) *and* (6.13), *then the DGKF* X *and* Y^{-1}

equations have solutions X and Y which are symmetric matrices with X, Y^{-1} and $Y^{-1}-X$ invertible. The DGKF X and Y^{-1} equations have solutions X and Y which are symmetric matrices with X, Y^{-1} and $Y^{-1}-X$ invertible, then (*HGRAIL*) *has a solution with invertible E_{ij}.*

Note that we obtained this result with an equation in the one unknown X and an equation with the one unknown $E_{11}=Y^{-1}$. From the strategy point of view, the first spreadsheet featured an equation in the single unknown b (and its transpose) and an equation in the single unknown c (and its transpose) and so is the most complicated. For example, (6.10) decomposes as

$$p=q_1^T q_1+q_2 \tag{6.16}$$

where $q_1=b+E_{12}^{-1}C_2^T+E_{12}^{-1}E_{11}B_1$ and q_2 is a symmetric polynomial which does not involve b. This forces us to say that the proof of the necessary side of Theorem 6.15 was done with a 2-strategy.

A more aggressive way of selecting knowns and unknowns allows us to obtain this same result with a symmetrized 1-strategy. In particular, one would set a, b and c to be the only unknowns to obtain a first spreadsheet. The first spreadsheet contains key equations like (6.16) which is a symmetric 1-decomposition because q_2 does not contain a, b or c. Once we have solved for a, b and c, we turn to the next spreadsheet by declaring the variables involving E_{ij} (e.g., E_{11}, E_{11}^{-1}, ...) to be unknown. At this point, the computer run is the same as Steps 2, 3 and 4 above.

7. MONOMIAL ORDERS

At the heart of the NCProcess1 and NCProcess2 commands is a GBA and central to this is the notion of monomial order. The effect of the SetKnowns and SetUnknowns commands is to prescribe a monomial order. It is essential to set a monomial order before running NCProcess. Indeed this is all that is required; one need not think about knowns and unknowns.

In this section we give more thorough descriptions of the types of monomial orders which are natural for these applications. They provided more flexibility in attacking problems.

Many possible choices of ordering are possible. We will briefly discuss three types of orderings: graded lex, pure lex and multigraded lex. The discussion here gives a straightforward generalization of the discussion of graded lex and pure lex monomial orders from commutative polynomial rings to noncommutative polynomial rings. See [CLS]. See also Appendix C.

Graded Lex (*Notation*: $a<b<c<d$). A graded lexicographic order on monic monomials in variables, say $a, b, c, \ldots$ compares two monic monomials by first comparing their total degrees. If their total degrees are not equal, then the monomial with the smaller total degree is considered smaller in the graded lex order. If their total degrees are equal, then the monomials are compared using a "dictionary" type order. To specify a graded lex order, one needs to specify an order on the variables, denoted $a<b<c<\cdots$, from which this "dictionary" order is derived. For example, if we use a graded lex order on monomials in the variables a and b such that $a<b$, then the monomials of total degree $\leqslant 2$ are ordered by

$$a<b<aa<ab<ba<bb.$$

More generally, for $a<b<c<\cdots$ and monic monomials M and N, we say that

> M is less than N (with respect to graded lex) if either the total degree of M is less than the total degree of N or M comes before N in the dictionary.

This type of order often works well when the main objective is not to eliminate particular variables but primarily to reduce the degree of the polynomial $p-q$ in the polynomial equations $p=q$.

Pure Lex (*Notation*: $a \ll b \ll c \ll d$). Pure lex order is a type of order used to transform a collection of equations into a collection of polynomial equations in triangular form, as described in Section 2.1. The pure lex order induced by $a, b, c, \ldots$ is denoted by $a \ll b \ll c \ll \cdots$ (contrast this to the notation for graded lex and multigraded lex). This notation is consistent with [CLS].

For those unfamiliar with pure lex orders, it is instructive to note that, in a pure lex order on monic monomials in a and b such that $a \ll b$, any monomial M_1 containing b is higher in the order than any monomial M_2 which does not contain b, even if M_1 has a much higher degree than M_2. Also two monic monomials containing b are ranked by first comparing the number of occurrences of the variable b in the monomial. If that is not enough to determine which is larger, a "dictionary" type order is used.

For example, if we use a pure lex graded lex order on monic monomials in a and b such that $a \ll b$, then the monic monomials of total degree $\leqslant 2$ are ordered by

$$a<aa<b<ab<ba<bb.$$

Multigraded Lex (*Notation: $a<b \ll c<d$*). Multigraded lex is a combination of pure lex and graded lex. We defer the explanation of this

order until Appendix C. The notation for multigraded lex orders consists of a list of variables separated by either $<$ or $\ll$ (e.g., $a<b\ll c<d$, $a\ll b\ll c$ or $a<b<c$).[28] If all of the separators are $<$, then the multigraded lex order is a graded lex order. If all of the separators are $\ll$, then the multigraded lex order is a pure lex order. We give one example.

EXAMPLE 7.1. We consider a multigraded lex order on monic monomials in the variables a, b, c and d which is denoted $a<b\ll c<d$.

This order on the monic monomials of total degree $\leqslant 2$ is

$$a<b<aa<ab<ba<bb<c<d<ac<bc<ca<cb$$
$$<ad<bd<da<db<cc<cd<dc<dd.$$

More generally the following statements hold:

(1) If M_1 and M_2 are monic monomials in a and b, then they are ordered by the graded lex order $a<b$.

(2) If M_1 and M_2 are monic monomials in c and d, then they are ordered by the graded lex order $c<d$.

(3) M_1 and M_2 are monic monomials in a and c (or in a and d or in b and c or in b and d), then they are ordered by the pure lex order $a\ll c$ (or $a\ll d$ or $b\ll c$ or $b\ll d$).

Knowns and Unknowns. Much of this paper is phrased in terms of knowns and unknowns. We use orders to put this distinction into a Gröbner Basis Algorithm. Obviously, one wants to solve for (or eliminate) unknowns, not knowns. Thus we will always be using orders satisfying

$$\text{knowns} \ll \text{unknowns}$$

and such that monic monomials in the knowns are ordered using a graded lex order. Indeed, this property could be taken to be the formal definition of known. In the notation of multigraded lex, the knowns are listed with $<$ between them and the unknowns are listed with a combination of $<$ and $\ll$ between them.

In the examples in this paper, the monic monomials in the unknowns are ordered using a pure lex order. In the notation of multigraded lex, this is expressed by listing the unknowns with $\ll$ between them.

We have found that one could use either a pure lex or certain multigraded lex orders on the unknowns and obtain the same results. The examples in this paper were initially done with a multigraded lex order on the unknowns which was not a pure lex order.

[28] This notation is a modification of the notation found in [CLS].

8. SUMMARY

Let us review the basic ideas. We start with knowns and unknowns and a set C of polynomial equations. As a strategy proceeds, more and more equations are digested by the user and placed in a set C'. Also, more and more unknowns become knowns. Thus we ultimately have two classes of knowns: original knowns $\mathscr{K}_0$ and user designated knowns $\mathscr{K}_U$.

Often prominent in the statement of a theorem is the subset $C_{\mathscr{K}}$ of C' consisting of equations involving only knowns $\mathscr{K} := \mathscr{K}_U \cup \mathscr{K}_0$.

Indeed many classical theorems (at least the ones we present here) have the form:

> The equations in C have a solution if and only if the equations in $C_{\mathscr{K}}$ have a solution.

Now the "only if" part of this is clear from the derivation using a strategy, but proving the "if" part is a less precise process. Proving the "if" part is the business of what we call the "end game," which is the act of using the last spreadsheet generated during a prestrategy to write down a theorem and its proof. Clearly one of the first steps is to translate the equations on the spreadsheet into palatable language. For example, if one obtains the equation $P^2 = P$ and $P = P^*$ and $P \in \mathscr{K}$, then one would say that P is a projection. A slightly more complicated example would be if $E_{12}E_{22}^{-1}E_{21} = Y^{-1} - X$ where E_{12}, E_{21}, E_{22} and Y are invertible and $E_{12} = E_{21}^T$ and E_{22} was self-adjoint. If Y^{-1} and X were in $\mathscr{K}$, then one could say that $Y^{-1} - X$ would have to be *both* symmetric and invertible.

When one performs the approach of the above paragraph to obtain the "if" part of the theorem, one tries to record enough necessary conditions so that the conditions are also sufficient. Once such conditions (in the form of polynomial equations) are selected from the spreadsheet,[29] then one assumes that each of these equations hold for certain unspecified values of the knowns (those variables in $\mathscr{K}$) and tries to find values for the remaining variables such that each of the equations on the final spreadsheet holds. If one can find values for these remaining variables, then, since each of the equations of the final spreadsheet holds, each of the equations on the initial spreadsheet holds and one has proven the desired "if" part of the theorem. Fortunately, the spreadsheet often has a block triangular

[29] If one does not find all of the necessary conditions that one needs at this point, it will, of course, become clear when trying to prove the converse, because one will not be able to continue with the proof after a certain point. Attempting to prove the converse when one has found most of the necessary conditions will help in pointing out the additional necessary conditions which are needed.

form which is amenable to backsolving. One of the benefits of putting the equations in categories is that this reveals this triangular structure (see Section 2.1).

These "end game" techniques are demonstrated in Section 3.4 and Section 6.3.

9. APPENDIX TO PART I: MORE DETAILS ON NCCOLLECTONVARIABLES

As described in Section 5, when performing computations, it is very helpful to distinguish between different *representations* of a polynomial equation. For example, the polynomials in (5.2) and (5.3) are equal, but (5.3) is much nicer, since it makes it clear that the polynomial depends on A, B, $X+Y$ and Z rather than just on A, B, X, Y, and Z. For this reason, it is helpful to have a way to compute such "parenthesized" representation of a polynomial. One might then be interested in computing all such "parenthesized" representation of a polynomial.

The key to computing "parenthesized" representations of a polynomial is the use of NCCollectOnVariables which has the NCAlgebra command NCCollect at its core. Before discussing the NCCollect command, we define a notion of homogeneous (noncommuting) polynomial.

Definition 9.1. Let p be a polynomial and V be a set of variables. p is homogeneous in V if for every $v \in V$, the number of occurrences of v is each term of p is independent of the term.

Of course, $xyzz^T + xx^T$ is homogeneous in $\{x, x^T\}$, $xyzz^T + xzy$ is homogeneous in $\{y, z\}$ and but $xyzx^T + xz$ is not homogeneous in $\{x, x^T\}$ (since the number of x^T is the first term is 1 and the number of x^T is the second term is 0).

When given a polynomial p and a set of variables V, NCCollect first writes the polynomial p as a sum of polynomials which are homogeneous. For each summand, NCCollect writes (to the extent possible) the polynomial into a "parenthesized" form using the rules

$$c_1 p_1 v + c_2 p_1 v p_2 = p_1 v (c_1 + c_2 p_2)$$

$$c_1 p_1 v p_3 + c_2 p_1 v = p_1 v (c_1 p_3 + c_2)$$

$$c_1 p_3 v p_2 + c_2 v p_2 = (c_1 p_3 + c_2)\, v p_2$$

$$c_1 v p_2 + c_2 p_1 v p_2 = (c_1 + c_2 p_1)\, v p_2$$

$$c_1 p_1 v p_3 + c_2 p_1 v p_2 = p_1 v(c_1 p_3 + c_2 p_2)$$

$$c_1 p_3 v p_2 + c_2 p_1 v p_2 = (c_1 p_3 + c_2 p_1)\, v p_2$$

$$c_1 v + c_2 v p_2 = v(c_1 + c_2 p_2)$$

$$c_1 v + c_2 p_1 v = (c_1 + c_2 p_1)\, v$$

where v is a variable in V, c_1, and c_2 are scalars, and p_1, p_2, and p_3 are polynomials.

If none of the above rules apply to a V-homogeneous polynomial, then we say that its only collected form is trivial. If a polynomial is a sum of homogeneous polynomials whose only collected form is trivial, then we say that this sums only collected form is trivial.

9.1. *Collecting against a Set of Expressions*

When given a polynomial p and a set of products of variables $\{q_1, ..., q_n\}$ (where each q_j is a product of variables), NCCollect begins by creating new variables $\{v_1, ..., v_n\}$, transforms p by replacing instances of the polynomial q_j in p with v_j, performs NCCollect as described in Section 9 and then replaces v_j with q_j.

9.2. *NCCollectOnVariables*

A key concept in understanding the NCCollectOnVariables option for NCProcess, which is abbreviated as NCCV, is the notion of a maximal product of knowns within a monic monomial.

Definition 9.2. $v_a v_{a+1} \cdots v_b$ is a maximal product of knowns in $v_1 \cdots v_n$ if

(1) $1 \leqslant a \leqslant b \leqslant n$;

(2) v_j is a known for $a \leqslant j \leqslant b$;

(3) if $a > 1$, then v_{a-1} is unknown;

(4) if $b < n$, then v_{b+1} is unknown.

NCCollectOnVariables takes as input a polynomial p. On each term of p, NCCollectOnVariables computes a list of maximal products of knowns. NCCollectOnVariables then repeatedly applies NCCollect to transform p into a parenthesized expressions with respect to the maximal products of knowns. NCCollectOnVariables uses the maximal products of knowns with the largest degree before using the maximal products of knowns with lower degrees. For example, if A, B, C, D, E, F, and G are knowns and a, b, d, e, and h are unknowns, then when NCCollectOnVariables is applied to $abABCd + De + FGhAC$, it tries to collect with respect to $\{ABC\}$ and then collect with respect to $\{FG, AC\}$ and then collect with respect to $\{D\}$.

II. THEORY AND MORE DETAILS

10. BACKGROUND ON IDEALS AND GRÖBNER BASES

The Gröbner Basis Algorithm requires that we have imposed an ordering (a "term order") on the monic monomials in the ring of polynomials in several variables $x_1, ..., x_n$ over a field K as in Section 1. Once a term order is defined, one can define the notion of a leading term, leading monomial and leading coefficient of a nonzero polynomial p. If p is a polynomial, $c_0 \in K \backslash \{0\}$ and M_0 is a monic monomial, then $c_0 M_0$ is the leading term of p if $c_0 M_0$ appears as a term of p and for every term cM of p (with $c \in K \backslash \{0\}$ and M a monic monomial), M_0 is greater than or equal to M in the term order. In this case, c_0 is called the leading coefficient of p and M_0 is called the leading monomial of p. For a nonzero polynomial f, let $lt(f)$ denote the leading term of f (with respect to the given ordering), $lc(f)$ denote the leading coefficient of f and $lm(f)$ denote the leading monomial of f. Note that $lt(f) = lc(f)\, lm(f)$.

10.1. *The Reduction Process*

A key operation in the theory of Gröbner Basis is the idea of reducing a polynomial p by a set of polynomials F. Once a term order is defined, reducing a polynomial p by a set of polynomials F involves constructing a polynomial p_0 such that

(1) $p - p_0$ lies in the ideal generated by F (so that p and p_0 determine the same cosets of $K[x_1, ..., x_n]/\mathcal{I}_F$)

(2) the leading monomial of p_0 is less than the leading monomial of p.

We now give details involving reducing a polynomial by a set of polynomials.

Given a monomial order, let $F = \{f_1, ..., f_k\}$ be a set of polynomials. Now let f be any polynomial. We say that f is reducible to g with respect to F if there exists $f_i \in F$ such that $g = f - cuf_iv$ where c is a constant and

u, v are monomials chosen so that the leading term of cuf_iv coincides with one of the terms of f. The effect is to replace a term of f with terms of lower order. The polynomial p is *irreducible* with respect to F if the leading term of f_i does not divide the leading term of f for any $f_i \in F$.

A reduction step can be conceived of as a replacement LHS → RHS of a term in f, which contains LHS as a factor, by a term or sum of terms of lower order. If, for example, $xx^{-1} \to 1$ is a replacement rule[30] and a term of f contains $x\,x^{-1}$, then xx^{-1} can be replaced by 1. The description of the reduction procedure in terms of replacement rules corresponds to the way it is commonly implemented. The "handedness" of replacement rules is determined by the term ordering. The LHS is always taken to be the leading term of the polynomial, while RHS is the negative of the sum of the remaining terms (which are lower in the monomial order).

When one applies a list of rules F repeatedly until no further reduction can occur to any polynomial p one obtains a *normal form* of p with respect to F. A normal form of p with respect to F is irreducible with respect to F.

10.2. *The Basis Algorithm*

G is a Gröbner Basis for an ideal I if G is a set of polynomials in I having the property that a polynomial f is in I if and only if 0 *is a normal form of f with respect to G*. This agrees with the definition in the commutative case, but in the commutative case, a finite Gröbner Basis exists for any ideal and can be obtained by the Buchberger Algorithm applied to any set of generators for the ideal. For ideals in a noncommutative polynomial ring, it need not be true that an ideal has a finite Gröbner Basis. An adaptation of Buchberger's Algorithm to the case of noncommutative polynomial rings is due to F. Mora [FMora].

We now discuss some of the particulars of the GBA given by [FMora]. Critical to a GBA is the construction of, from pairs of polynomials (f, g), common multiples for the leading terms of f and g. Now we recall the appropriate notion of common multiple.

Let S be the free semigroup generated by a finite alphabet A (i.e., the collection of words in the letters of A). Let (m_1, m_2) be an ordered pair of elements of S. By a match of (m_1, m_2) we mean a 4-tuple (l_1, r_1, l_2, r_2) of elements of S which satisfy one of the following conditions:

(1) $l_1 = r_1 = 1, m_1 = l_2 m_2 r_2$.

(2) $l_2 = r_2 = 1, m_2 = l_1 m_1 r_1$.

[30] Recall that x^{-1} and x are two different indeterminates and that one needs to explicitly add the equations $xx^{-1} = 1$ and $x^{-1}x = 1$.

(3) $l_1 = r_2 = 1$, $l_2 \neq 1$, $r_1 \neq 1$, there is a $w \neq 1$ with $m_1 = l_2 w$, $m_2 = w r_1$.

(4) $l_2 = r_1 = 1$, $l_1 \neq 1$, $r_2 \neq 1$, there is a $w \neq 1$ with $m_1 = w r_2$, $m_2 = l_1 w$.

These conditions make $l_1 m_1 r_1 = l_2 m_2 r_2$. This is a common multiple of m_1 and m_2 which is minimal in some sense.

In the commutative case, the Basis Algorithm makes use of a kind of resolvent of two polynomials called the *S-Polynomial.* S-Pol$(f, g) = c_2 u_1 f_1 - c_1 u_2 f_2$ where $c_i = lc(f_i)$ and where u_i is chosen so that $u_i lm(f_i)$ is the least common multiple of $lm(f_1)$ and $lm(f_2)$ for $i = 1, 2$. In the noncommutative case, there are several such resolvents—one for each match. If M is a 6-tuple $(f_1, f_2, l_1, r_1, l_2, r_2)$ where (l_1, r_1, l_2, r_2) is a match for $(lm(f_1), lm(f_2))$ and $c_i = lc(f_i)$, then we set

$$\text{S-Pol}(M) = c_2 l_1 f_1 r_1 - c_1 l_2 f_2 r_2.$$

Example. Consider the polynomials

$$f_1 = aaba + ab;$$

$$f_2 = abaa + ba.$$

There are four matches for (f_1, f_2)

1. $(aba, 1, 1, aba)$. In this case the S-Polynomial is

$$(aba)(f_1) - (f_2)(aba) = -baaba + abaab.$$

2. $(ab, 1, 1, ba)$. In this case the S-polynomial is

$$(ab)(f_1) - (f_2)(ba) = -baba + abab.$$

3. $(1, baa, aab, 1)$. In this case the S-Polynomial is

$$(f_1)(baa) - (aab)(f_2) = abbaa - aabba.$$

4. $(1, a, a, 1)$. In this case the S-Polynomial is

$$(f_1)(a) - (a)(f_2) = 0.$$

The algorithm is iterative and starts with a finite set of polynomials G_1. The kth step has available to it a set G_k such that the ideal generated by G_1 equals the ideal generated by G_k. The kth step of the algorithm creates a set G_{k+1} by setting it equal to the union of G_k and the set of all nonzero reductions of S-Polynomials for all pairs in G_k. The process repeats as long as there are S-Polynomials with nonzero reductions. In other words, the process repeats until $G_k = G_{k+1}$.

11. A GRÖBNER BASIS THEOREM ON ELIMINATION IDEALS

A classic theorem for commutative polynomial rings (Theorem 2 of Chapter 3 Section 1 of [CLS]) says that if G is a Gröbner Basis for an ideal I with respect to certain types of orders (an order of elimination type) and J is a certain type of ideal contained in I (an elimination ideal), then $G \cap J$ is a Gröbner Basis for J. In this section we prove that this theorem generalizes to noncommutative polynomial rings. We shall see that this new theorem can help us solve systems of polynomial equations which are in triangular form and implies that NCProcess (up to the vagaries of stopping in a finite time and choosing the correct type of order) eliminates variables as well as is theoretically possible.

We now define *elimination ideals*, a notion critical to this section.

Definition 11.1. Let I be an ideal of $K[x_1, ..., x_n]$ and $1 \leqslant j \leqslant n$. The jth elimination ideal I_j is the ideal of $K[x_{j+1}, ..., x_n]$ defined by

$$I_j = I \cap K[x_{j+1}, ..., x_n].$$

The NCProcess command generates output which is displayed as a list of V-categories (see Section 2.2) and the V-categories are defined in such a way that if one of the polynomials in a category is in the elimination ideal, then the entire category is a subset of the elimination ideal.

It is helpful to be able to find a generating set (or even a Gröbner Basis) not only for I but also for the jth elimination ideals. If one has a Gröbner Basis with respect to certain types of monomial orders, then, if one considers the subset of the Gröbner Basis which lies in the jth elimination ideal, then this set is itself a Gröbner Basis and generates the jth elimination ideal. This is the content of Theorem 11.3.

To layout the correspondence of the rest of this section to Chapter 3 of [CLS], note that the definition of an elimination ideal (Section 1 Definition 1) an elimination order (Section 1 Exercise 5), an Elimination Theorem (Section 1 Theorem 2) and an Extension Theorem (Section 1 Theorem 3) are given in [CLS]. We now give the corresponding definitions, a corresponding Elimination theorem and discuss issues involving extensions in the noncommuative case.

Definition 11.2 [CLS]. Let j and n be natural numbers such that $1 \leqslant j \leqslant n$. A monomial order is of jth elimination type provided that any monic monomial involving $x_1, ..., x_{j-1}$ or x_j is greater than any monic monomial of $K[x_{j+1}, ..., x_n]$.

In this section, we follow the ordering convention used in [CLS]. With the definition of elimination order given above, $x_a > x_b$ if $1 \leqslant a \leqslant j < b \leqslant n$.

For instance, $x_1 > x_n$. In the discussions in the rest of the paper, we have always taken $x_1 < x_2 < \cdots < x_n$.

If one considers a multigraded lex order (Section 7 and Appendix) under which

$$x_1 > x_2 > x_3 > \cdots > x_n \qquad \text{and} \qquad x_j \gg x_{j+1},$$

then this multigraded lex order is of jth elimination type. Note that a pure lex order $x_1 \gg x_2 \gg \cdots \gg x_n$ is of jth elimination type for any j such that $1 \leqslant j \leqslant n$.

The following theorem is the main result of this section and shows that a Gröbner Basis for an ideal I with respect to an jth elimination order yields a Gröbner Basis with respect to the jth elimination ideal.

Theorem 11.3. *Let $R = K[x_1, ..., x_n]$, let $\succ$ be a term order on the monic monomials of R, let I be an ideal of R and let G be a Gröbner Basis of I with respect to $\succ$. If $1 \leqslant j \leqslant n$ and $\succ$ is of jth elimination type, then $G \cap K[x_{j+1}, ..., x_n]$ is a Gröbner Basis for $I \cap K[x_{j+1}, ..., x_n]$.*

If a pure lex order is used, if one runs NCProcess *until the GBA being used by NCProcess generates a Gröbner Basis* and all of the shrinking (see Section 12) parts of the NCProcess commands are turned off, then the categories which this outputs can be used to determine generating sets for the elimination ideals. More precisely, the union of the categories which are subsets of the jth elimination ideal is a generating set for the jth elimination ideal.

The comments of the last paragraph hold when a multigraded lex order is used rather than a pure lex order *and* one only considers the jth elimination ideal only if the order is described by placing a "$\ll$" between x_j and x_{j+1}.

We begin by proving the following lemma.

Lemma 11.4. *If f is a nonzero polynomial in $K[x_1, ..., x_n]$, if $\succ$ is a monomial order, if $\succ$ is of jth elimination type and if the leading monomial of f with respect to $\succ$ is in $K[x_{j+1}, ..., x_n]$, then $f \in K[x_{j+1}, ..., x_n]$.*

Proof. Let $lm(f)$ denote the leading monomial of f with respect to the order $\succ$.

Suppose $c_1, ..., c_s \in K \backslash \{0\}$ and $m_1, ..., m_s$ are monic monomials in $K[x_1, ..., x_n]$ such that $f = \sum_{r=1}^{s} c_r m_r$ and $m_1 \succ m_2 \succ \cdots \succ m_s$. Suppose there exists an r_0 such that $m_{r_0} \notin K[x_{j+1}, ..., x_n]$. Definition 11.2 and the assumption that $lm(f) \in K[x_{j+1}, ..., x_n]$ imply that $m_{r_0} \succ lm(f)$. But $lm(f) \succ m_{r_0}$ since $lm(f)$ is the leading monomial of f. But then $m_{r_0} \succ m_{r_0}$

which is a contradiction. Therefore, $m_r \in K[x_{j+1}, ..., x_n]$ for $r = 1, ..., s$. Therefore, $f \in K[x_{j+1}, ..., x_n]$. This completes the proof of Lemma 11.4.

We now move to the proof of Theorem 11.3.

Proof of Theorem 11.3. Throughout this proof, for any nonzero polynomial p, $lm(p)$ will denote the leading monomial of p with respect to the order $\succ$.

Let $f \in I \cap K[x_{j+1}, ..., x_n]$ be nonzero. Since $f \in I$ and G is a Gröbner Basis, f has a finite d-representation with respect to G [FMora]. That is, there exist scalars $c_1, ..., c_s \in K\backslash\{0\}$, there exists $f_1, ..., f_s \in G$, and there exist monic monomials $\ell_1, ..., \ell_s, r_1, ..., r_s$ such that

$$f = c_1 \ell_1 f_1 r_1 + \cdots + c_s \ell_s f_s r_s \tag{11.5}$$

$$lm(f) = \ell_1\, lm(f_1)\, r_1 \tag{11.6}$$

and

$$\ell_m lm(f_m)\, r_m \succ \ell_{m+1} lm(f_{m+1})\, r_{m+1} \quad \text{for} \quad m = 1, ..., s-1. \tag{11.7}$$

We wish to show that $\ell_i, f_i, r_i \in K[x_{j+1}, ..., x_n]$ for $1 \leqslant i \leqslant s$. Since $f \in K[x_{j+1}, ..., x_n]$, its leading term $lm(f)$ is in $K[x_{j+1}, ..., x_n]$. Therefore, (11.6) implies that $\ell_1, r_1, lm(f_1) \in K[x_{j+1}, ..., x_n]$. An application of Lemma 11.4 yields $f_1 \in K[x_{j+1}, ..., x_n]$. For $1 \leqslant i \leqslant s$,

$$K[x_{j+1}, ..., x_n] \ni lm(f) = \ell_1\, lm(f_1)\, r_1 \succ \ell_i lm(f_i)\, r_i \tag{11.8}$$

Now, (11.8) implies that $\ell_i, r_i, lm(f_i) \in K[x_{j+1}, ..., x_n]$. Since $\succ$ is of jth elimination type, an application of Lemma 11.4 yields $f_i \in K[x_{j+1}, ..., x_n]$ for $1 \leqslant i \leqslant s$. Thus, f has a d-representation in $K[x_{j+1}, ..., x_n]$ with respect to $G \cap K[x_{j+1}, ..., x_n]$. Since every element of $I \cap K[x_{j+1}, ..., x_n]$ has a finite d-representation with respect to $G \cap K[x_{j+1}, ..., x_n]$ an application of Proposition 2.2 of [FMora] proves that $G \cap K[x_{j+1}, ..., x_n]$ is a Gröbner Basis. The completes the proof of Theorem 11.3. ▮

Note that we have not analyzed what is called extendibility of solutions in commutative theory [CLS]. The main theorem involving extendibility of solutions can be viewed as giving sufficient conditions, in the commutative case, as to when backsolving is possible. We now give a brief description of these ideas.

Elimination ideals (and, therefore, Gröbner Basis with respect to term orders of elimination type) can be used to simplify the problem of finding common zeros of sets of polynomial equations. In other words, these elimination ideals can facilitate the process of backsolving. One process of finding a common zero of the set of polynomials in an ideal I involves first finding a common zero of the polynomials in a jth elimination ideal for j_1 close to n. This common zero will only assign values to the variables $x_{j_1+1}, ..., x_n$. One then tries to extend this common zero to a common zero of the j_2-elimination ideal for $j_2 < j_1$. Extending the common zero consists of trying to assign values to the variables $x_{j_2+1}, ..., x_j$. The procedure repeats until one cannot extend a common zero (in which case one has to backtrack and pick a different common zero in the previous steps if there are any other choices) or until one finds a common zero of the entire ideal.

For the case of common zeros of commutative polynomials whose coefficients are complex numbers, Chapter 3 Section 1 of [CLS] gives a theorem which guarantees the extendibility of common zeros from one elimination ideal to the next under certain conditions. The authors do not know of any theorem of this type in the noncommutative case. The main reason we have not looked into it ourselves is that in the examples we have run (many besides those presented here), it was obvious that backsolving was possible for any choice of solution at each stage. Possibly more complicated theorems in systems or operator theory will require such a theory. This remains to be seen.

12. FINDING A SMALL GENERATING SET FOR AN IDEAL

A principal part of the NCProcess commands is the use of the Gröbner Basis algorithm. A Gröbner Basis can be infinite and even when a Gröbner Basis is finite, it can be very large. One often finds that there are many polynomial equations which are generated which do not enhance our understanding of the mathematics. We begin with an example.

Example 12.1. The GB generated by the set $\{PTP - TP, P^2 - P\}$ is the set $\{PT^nP - T^nP: n \geqslant 1\} \cup \{P^2 - P\}$ regardless of the term order used. No smaller GB exists.

Here just two polynomials equations generate infinitely many. One way to view this example is that the computer discovers that if the range of P is invariant for a linear transformation T, then it is invariant for T^n for every $n \geqslant 1$. The GBA tries to generate this infinite set of polynomial equations and, at any time, has generated a finite subset of them. Since the NCProcess commands are used in the discovery of theorems (as described in Part I), it would not be helpful to display $PT^nP = T^nP$, for n in a large set of natural numbers, on the spreadsheet, since these equations are mathematically and conceptually redundant. Indeed, one would not choose to state, either as a hypothesis or as a conclusion to a theorem, that the range of P is invariant for T^n for all $n \geqslant 1$, since the equivalent statement that the range of P is invariant for T would suffice.

This introduces the next topic which is shrinking a set of polynomial equations to eliminate redundancy. Our desire is to take the generated basis and to remove mathematical redundancy from the generating set without destroying the information which was gained while running the GBA. Also we point out that our examples in Part I used shrinking of the type described here heavily and that the undigested polynomial equations in the spreadsheets survived the deleting of many polynomial equations from a partial GB.

To be more precise, in many cases, we would like to compute a minimal length generating set for an ideal (that is, given an ideal $\mathscr{I}$, a minimal length generating set for the ideal would be a set Y such that every generating set for $\mathscr{I}$ has cardinality equal to or greater than the cardinality of Y). Determining an algorithm for finding a minimal length generating set for an ideal is an open problem for commutative polynomial rings and is probably unsolvable in the noncommutative case.

We say that a set of polynomials X is a minimal generating set if no proper subset of X generates $\mathscr{I}_X$. A more practical goal is to try to find, given a finite set X of polynomials, a minimal generating subset of X (that is, a subset Y of X which generates the same ideal which X generates and which is a minimal generating set).

In the case of ideals of a commutative polynomial ring, it is well known that every minimal generating subset of a given set of polynomials can be found. In fact, one approach to finding such minimal generating subsets would be to use an algorithm based upon using reduction and using a finite number of runs of the Gröbner Basis Algorithm. Since the Gröbner Basis Algorithm is guaranteed to finish in a finite amount of time, one can compute minimal generating subsets.

In contrast to the case of commutative polynomial rings, for an ideal of a noncommutative polynomial ring, there does not exist an algorithm which can,[31] given a finite set X of polynomials, construct every minimal generating subset of X. We prove this assertion in the following lemma.

LEMMA 12.2. *There does not exist an algorithm which can, in every case, given a finite set X of polynomials, construct every minimal generating subset of X.*

Proof. Suppose that one could write such an algorithm. Given a finite set X and a polynomial p, one could use this hypothetical algorithm to find a minimal generating subset Y of X. By using this hypothetical algorithm

[31] In every case.

again, we can find a $k \geqslant 1$ and find the minimal generating subsets $\{Z_n : n=1, ..., k\}$ of $Y \cup \{p\}$. Note that no Z_n could be a proper subset of Y. With this set up, p is a member of the ideal generated by X if and only if one of the Z_n equals Y. Therefore, if such a hypothetical algorithm existed, then one could solve the membership problem for finitely generated ideals of noncommutative polynomial rings. The membership problem for noncommutative polynomial rings is known to be unsolvable ([TMora]). ▌

12.1. *The Need to Consider Small Generating Sets*

Lemma 12.2 shows that, for noncommutative polynomial rings, it is in general algorithmically impossible to compute minimal generating subsets. Therefore, we content ourselves with finding *small* generating subsets rather than minimal generating subsets. We now show that even if we could find minimal generating subsets, we would not necessarily want to compute them.

In a strategy, it is often the case that equations with no unknowns are the basic compatibility conditions for the problem being considered. Thus some redundancy, redundancy which promotes the retaining of equations with no unknowns, can be helpful. The next two example illustrate this. The first example shows that it is not always desirable to find a minimal generating set. The second example shows that, when seeking a generating set, it is helpful to direct ones search so that one keeps all equations which do not involve any unknowns. This is described further in Section 12.7.

The following example shows that, when given a set X, one does not always want the smallest subset of X which generates the same ideal.

EXAMPLE 12.3. Consider the case of the spreadsheet following Eq. (3.3). That spreadsheet contained the following polynomials.

$$\begin{array}{lll} P_1^2 - P_1 & P_1 A P_1 - P_1 A & m_1 n_1 - P_1 \\ n_1 m_1 - 1 & n_2 m_2 - 1 & m_2 n_2 - 1 + m_1 n_1 \end{array}$$

There is only one proper subset of the polynomials listed above which generates the same ideal. That subset consists of the above list of polynomials with $P_1^2 - P_1$ removed. It would, however, be a tactical mistake to remove this polynomial since it reveals important information about the known P_1, namely that it is idempotent. This information can be derived, of course, from the remaining equations, but it is preferable for it to be shown explicitly.

In addition, there are cases when it is *not* advantageous to find an arbitrary minimal generating set, but a specific minimal generating with certain properties is desired. Consider the following example.

EXAMPLE 12.4. Suppose that A and P are known and that m and n are unknown. Let X be the set $\{PAP - AP, P - mn, nAmn - Amn, nm - 1\}$. There are two minimal subsets of X which generate the same ideal as X. These subsets are $\{PAP - AP, P - mn, nm - 1\}$ and $\{P - mn, nAmn - Amn, nm - 1\}$. Clearly, the first minimal generating set is preferable to the second since the first contains the polynomial $PAP - AP$ which involves only knowns.

12.2. *Preview of Operations*

Example 12.1 shows that, in some cases, we seek a minimal generating set for an ideal; Example 12.3 shows that in some cases we prefer certain minimal generating sets for an ideal to other minimal generating sets; Example 12.4 shows that in some cases we prefer finding *small generating sets*. The choice of how to shrink a generating set is further complicated by the desire to have shrinking occur quickly on the computer.

The remainder of this section describes six different operations to convert from a particular basis for an ideal to a smaller one. They differ in approach, speed and functionality. We now give a brief description of each of these six operations.

(1) SmallBasis can be used to find smaller generating sets and is relatively fast. SmallBasis is described in Section 12.3. SmallBasis uses the GBA heavily.

(2) ShrinkBasis is used to find *all minimal generating subsets* and can be *very* slow. ShrinkBasis is often too slow to use in practice. ShrinkBasis is described in Section 12.4. ShrinkBasis uses the GBA heavily.

(3) RemoveRedundant is used to find smaller generating sets and is *very* fast. The RemoveRedundant operation is described in Section 12.6. The RemoveRedundant operation requires the recording of information during the previous run of the GBA and uses that information once the run is completed. For this reason, the explanation of RemoveRedundant requires some initial theoretical discussions which are in Section 12.5. The run time of RemoveRedundant is very fast since it implements a graph search and does not invoke a GBA.

(4, 5) The fourth and fifth operations are SmallBasisByCategory and RemoveRedundantByCategory. These two operations find smaller subsets

‘

while respecting the fact that retaining polynomials which do not involve unknowns is desirable and they act on each category individually. These last two operations are described in Section 12.7. These two operations use SmallBasis and RemoveRedundant, respectively, heavily.

(6) The sixth operation is RemoveRedundantProtect. This command combines the results of three RemoveRedundant runs and is very fast. This command is used to prevent equations not involving unknowns from being removed by polynomials which involve unknowns. Also, the command prevents digested polynomials from being removed by polynomials which are undigested. This command is described in Section 12.8.

Recall, for the remainder of the section, that the ideal in $K[x_1, ..., x_n]$ generated by $X \subset K[x_1, ..., x_n]$ is denoted $\mathscr{I}_X$.

12.3. *The SmallBasis Operation*

One natural operation is called SmallBasis. We begin by describing an idealized version of it. SmallBasis associates to a finite sequence $X = \{p_1, ..., p_m\} \subset K[x_1, ..., x_n]$, a subset Y of X such that $p_j \notin Y$ if and only if $p_j \in \mathscr{I}_{\{p_1, ..., p_{j-1}\}}$, for each $j = 2, ..., m$.

Note that *SmallBasis*(X) depends on the order of the p_j's in the sequence X.

The reason that SmallBasis cannot be fully implemented on a computer is that one cannot in general decide whether or not $p_j \in \mathscr{I}_{\{p_1, ..., p_{j-1}\}}$.

12.3.1. *Approximation of the SmallBasis Operation*

Our approach to approximating the SmallBasis operation is to replace the test $p_j \in \mathscr{I}_{\{p_1, ..., p_{j-1}\}}$ with the process of running the GBA with $\{p_1, ..., p_{j-1}\}$ and some monomial order as input for a small number of iterations and testing if the normal form of p_j is zero with respect to the output of the GBA.

As a final note, this approximation of SmallBasis has the property that: If $p_j \notin$ *SmallBasis*$(\{p_1, ..., p_n\})$, then $p_j \in \mathscr{I}_{\{p_1, ..., p_{j-1}\}}$.

12.4. *The ShrinkBasis Operation*

The second idealized operation is called ShrinkBasis. For a set $X = \{p_1, ..., p_m\} \subset K[x_1, ..., x_n]$, ShrinkBasis associates to X every minimal generating subset of X, that is, the collection of all subsets X_0 such that X_0 generates the same ideal that X does and no proper subset of X_0 generates the same ideal that X does. We will denote this collection by *ShrinkBasis*(X).

One (very inefficient) way to implement *ShrinkBasis*(X) is by using SmallBasis. Specifically, if we view X as a sequence, the idea is to form

$$\{\mathit{SmallBasis}(\sigma(X)) \colon \sigma \text{ is a permutation of } X\}. \tag{12.5}$$

The result of *ShrinkBasis*(X) is the set of members of the set of (12.5) such that no subset of it lies in (12.5).

A more efficient way to perform this computation is to determine whether or not x lies in the ideal generated by $X\backslash\{x\}$ for each $x \in X$. *ShrinkBasis*(X) will be the collection containing the single set X if x is not a member of $\mathscr{I}_{X\backslash\{x\}}$ for every $x \in X$ and otherwise it is the union of all sets *ShrinkBasis*$(X\backslash\{x\})$ such that $x \in X$ lies in the ideal generated by $X\backslash\{x\}$. Of course, one can not determine whether or not an element x is in a specified ideal and so one uses the GBA for a small number of iterations.

12.5. *Gröbner Graph*: *Background for the RemoveRedundant Operation*

12.5.1. *Gröbner Basis Background*

As discussed in Section 9, the principal computation which the Gröbner Basis Algorithm performs is the computation of an S-polynomial s from two polynomials (say p_1 and p_2) and then the computation of this S-polynomial's normal form (say r) using other polynomials (say, $q_1, ..., q_\ell$). One of the side effects of this type of computation is the fact that s lies in the ideal generated by $\{p_1, p_2, q_1, ..., q_\ell\}$. If this ideal membership relationship is stored during the process of running the GBA and one wants to find a smaller generating set for the output of the GBA, then it is helpful to use these ideal membership relationships.

12.5.2. *The Graph*

We begin with a definition involving graphs. Let $G=(V, E)$ be a graph and v be a vertex of G. We say that the *immediate ancestors* of v are members of the set $T(v, G)$ of vertices from which there is an edge leading to v (i.e., $T(v, G)=\{w \in V \colon (w, v) \in E\}$).

One way to graphically record the fact that a polynomial s lies in the ideal generated by $\{p_1, p_2, q_1, ..., q_\ell\}$ is to create a graph whose vertices are $\{s, p_1, p_2, q_1, ..., q_\ell\}$ and edges are $\{(p_1, s), (p_2, s), (q_1, s), ..., (q_\ell, s)\}$ (each of the edges is directed toward s). By taking a finite union of graphs of the type just described, one has a graph whose vertices are polynomials and the following property holds:

> Every vertex v of the graph has no immediate ancestors or lies in the ideal generated by its immediate ancestors.

Since the GBA is based on generating S-polynomials, its flow can be described as the construction of a graph with the above property and this graph will be directed and acyclic. Our implementation of the GBA actually constructs this graph and this graph is used in Section 12.6. Also, note that if v is a vertex, then v is a starting polynomial equation for the run if and only if it has no immediate ancestors.

We now lay these observations into an abstract framework in the next section.

12.5.3. *Definition of a Gröbner Graph*

We now shift our discussion from referring to S-polynomials to using graph theory. We begin by stating the following definition in which we define a Gröbner graph to encapsulate the observations of Section 12.5.2.

Definition 12.6. Let $G = (V, E)$ be a graph such that $V \subset K[x_1, ..., x_n]$. We say that G is *Gröbner graph* if G is a directed acyclic graph and for every $v \in V$, either $T(v, G)$ is the empty set or v lies in the ideal generated by $T(v, G)$.

Definition 12.7. Let $G = (V, E)$ be a graph and v be a vertex of G. We will say that v is a *starting vertex* if $T(v, G)$ is the empty set.

The following proposition follows easily from the previous two definitions and mathematical induction. We omit the proof.

Proposition 12.8. *Let $G = (V, E)$ be a Gröbner graph, V be a finite set and v be a vertex of G. Let V_0 be the set of starting vertices of G. Every element of V lies in the ideal generated by V_0.*

Remark 12.9. The hypothesis of Proposition 12.8 can be strengthened. One can replace the hypothesis that V be a finite set with the hypothesis that every element of V has only finitely many ancestors.

If one takes a subgraph $G_0 = (V_0, E_0)$ of a Gröbner graph $G = (V, E)$ such that

> if v is a vertex of the graph G_0 and v is not a starting vertex of G_0, then $T(v, G_0) = T(v, G)$

then G_0 is a Gröbner graph. Other subgraphs of a Gröbner graph may be Gröbner, but the above mentioned subgraphs can be seen to be Gröbner from purely graph-theoretic arguments. We call these subgraphs subGröbner and formalize this with the following definition.

DEFINITION 12.10. A graph $G=(V,E)$ will be called a *subGröbner graph* with respect to a Gröbner graph $\tilde{G}=(\tilde{V},\tilde{E})$ if G is a subgraph of $\tilde{G}$ and for every $v\in V$, either v is a starting vertex of G or $T(v,G)=T(v,\tilde{G})$.

Note that if one is given a Gröbner graph $\tilde{G}=(\tilde{V},\tilde{E})$ and a set of vertices $V_0\subset\tilde{V}$, then there is a smallest graph $G=(V,E)$ such that $V_0\subset V$ and G is subGröbner with respect to $\tilde{G}$.

If $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$ are graphs, then we let $G_1\cap G_2$ denote the graph $(V_1\cap V_2, E_1\cap E_2)$ and let $G_1\cup G_2$ denote the graph $(V_1\cup V_2, E_1\cup E_2)$.

LEMMA 12.11. *Let $\mathcal{G}$ be a Gröbner graph. If $\mathcal{G}_1=(V_1,E_1)$ and $\mathcal{G}_2=(V_2,E_2)$ are subGröbner with respect to $\mathcal{G}$, then $\mathcal{G}_1\cup\mathcal{G}_2$ is subGröbner with respect to $\mathcal{G}$ and $\mathcal{S}(\mathcal{G}_1\cup\mathcal{G}_2)=(\mathcal{S}(\mathcal{G}_1)\cup(V_2\backslash V_1))\cap(\mathcal{S}(\mathcal{G}_2)\cup(V_1\backslash V_2))$ where $\mathcal{S}(\mathcal{H})$ denotes the set of starting vertices of $\mathcal{H}$ for any Gröbner graph $\mathcal{H}$.*

Proof. For ease of reference, let $X=\mathcal{S}(\mathcal{G}_1\cup\mathcal{G}_2)$, $Y_1=\mathcal{S}(\mathcal{G}_1)\cup(V_2\backslash V_1)$ and $Y_2=\mathcal{S}(\mathcal{G}_2)\cup(V_1\backslash V_2)$. We wish to show that $X=Y_1\cap Y_2$. Since $X\subset V_1\cup V_2$, $Y_1\subset V_1\cup V_2$ and $Y_2\subset V_1\cup V_2$, it suffices to show that $v\in X$ if and only if $v\in Y_1\cap Y_2$ under the conditions that $v\in V_1\cap V_2$, $v\in V_1\backslash V_2$ or $v\in V_2\backslash V_1$.

If $v\in V_1\cap V_2$, then $v\in Y_1$ if and only if $v\in\mathcal{S}(\mathcal{G}_1)$, $v\in Y_2$ if and only if $v\in\mathcal{S}(\mathcal{G}_2)$ and $T(v,\mathcal{G}_1\cup\mathcal{G}_2)=T(v,\mathcal{G}_1)\cup T(v,\mathcal{G}_2)$. Therefore, $v\in X$ if and only if $v\in\mathcal{S}(\mathcal{G}_1)\cap\mathcal{S}(\mathcal{G}_2)$ which occurs if and only if $v\in Y_1\cap Y_2$.

If $v\in V_1\backslash V_2$, then $v\in Y_1$ if and only if $v\in\mathcal{S}(\mathcal{G}_2)$, $v\in Y_2$ and $T(v,\mathcal{G}_1\cup\mathcal{G}_2)$ $=T(v,\mathcal{G}_1)$. Therefore, $v\in X$ if and only if $v\in\mathcal{S}(\mathcal{G}_1)$ which occurs if and only if $v\in Y_1\cap Y_2$.

If $v\in V_2\backslash V_1$, then $v\in Y_1$, $v\in Y_2$ if and only if $v\in\mathcal{S}(\mathcal{G}_2)$ and $T(v,\mathcal{G}_1\cup\mathcal{G}_2)$ $=T(v,\mathcal{G}_2)$. Therefore, $v\in X$ if and only if $v\in\mathcal{S}(\mathcal{G}_2)$ which occurs if and only if $v\in Y_1\cap Y_2$.

In summary, $\mathcal{S}(\mathcal{G}_1\cup\mathcal{G}_2)=(\mathcal{S}(\mathcal{G}_1)\cup(V_2\backslash V_1))\cap(\mathcal{S}(\mathcal{G}_2)\cup(V_1\backslash V_2))$.

Let v be a vertex of $\mathcal{G}_1\cup\mathcal{G}_2$ which is not a starting vertex. Therefore, $v\notin(\mathcal{S}(\mathcal{G}_1)\cup(V_2\backslash V_1))\cap(\mathcal{S}(\mathcal{G}_2)\cup(V_1\backslash V_2))$. Without loss of generality, assume that $v\notin\mathcal{S}(\mathcal{G}_1)\cup(V_2\backslash V_1)$. But then $v\in(V_1\cup V_2)\backslash(V_2\backslash V_1)\subset(V_1\cup V_2)\backslash V_2\subset V_1$ and $v\notin\mathcal{S}(\mathcal{G}_1)$. Therefore, $T(v,\mathcal{G})=T(v,\mathcal{G}_1)$ and so

$$T(v,\mathcal{G}_1\cup\mathcal{G}_2)\subseteq T(v,\mathcal{G})=T(v,\mathcal{G}_1)\subseteq T(v,\mathcal{G}_1\cup\mathcal{G}_2).$$

Therefore, $T(v,\mathcal{G}_1\cup\mathcal{G}_2)=T(v,\mathcal{G})$. This completes the proof of Lemma 12.11. ▌

The following lemma follows trivially from the definition of immediate ancestor, the definition starting vertex (Definition 12.7) and the definition of subGröbner graph (Definition 12.10).

LEMMA 12.12. *Let $\mathscr{G}$ be a Gröbner graph. If $\mathscr{G}_1=(V_1, E_1)$ and $\mathscr{G}_2=(V_2, E_2)$ are subGröbner with respect to $\mathscr{G}$, then $\mathscr{G}_1 \cap \mathscr{G}_2$ is subGröbner with respect to $\mathscr{G}$.*

The following technical lemma will be used in the proof of Proposition 12.14.

LEMMA 12.13. *Let $\mathscr{G}=(V, E)$ be a Gröbner graph, V_0 be a finite set, $V_0 \subset V$ and for each $v \in V_0$, let*

$$\mathscr{F}_v=\{\mathscr{H}: \mathscr{H} \text{ is subGröbner with respect to } \mathscr{G}, v \text{ is a vertex of } \mathscr{H}$$
$$\text{and every starting vertex of } \mathscr{H} \text{ is in } V_0 \backslash \{v\}\}.$$

If $\mathscr{H}_v=(V_v, E_v)$ is an element of $\mathscr{F}_v$ such that card$(\mathscr{H}) \leqslant$ card$(\mathscr{H}_v)$ for each $\mathscr{H} \in \mathscr{F}_v$, then $\mathscr{F}_w$ is empty for every starting vertex w of $\mathscr{H}_v$.

Proof. Suppose, for the purpose of proof by contradiction, that there exists a starting vertex w of $\mathscr{H}_v$ such that $\mathscr{F}_w$ is not empty. Since $\mathscr{F}_w$ is not empty, there exists $\mathscr{K}=(G_{\mathscr{K}}, E_{\mathscr{K}})$ which is subGröbner with respect to $\mathscr{G}$ such that v is a vertex of $\mathscr{K}$ and every starting vertex of $\mathscr{K}$ is a member of $V_0 \backslash \{v\}$. By Lemma 12.11, $\mathscr{H}_v \cup \mathscr{K}$ is subGröbner with respect to $\mathscr{G}$ and $\mathscr{H}_v \cup \mathscr{K} \in \mathscr{F}_w$. By the choice of $\mathscr{H}_v$, the fact that $\mathscr{H}_v$ is a subgraph of $\mathscr{H}_v \cup \mathscr{K}$ and the fact that $\mathscr{H}_v \cup \mathscr{K} \in \mathscr{F}_v$, the graph $\mathscr{H}_v \cup \mathscr{K}$ equals the graph $\mathscr{H}_v$ and so $\mathscr{K}$ must be a subgraph of $\mathscr{H}_v$. But then, since $\mathscr{K} \in \mathscr{F}_w$, w is a starting vertex of $\mathscr{H}_v$ and w is a vertex of $\mathscr{K}$, w is a member of $V_0 \backslash \{w\}$ which is a contradiction. Thus $\mathscr{F}_w$ is empty for every starting vertex of w of $\mathscr{H}_v$. This completes the proof of Lemma 12.13. ▮

12.6. *The RemoveRedundant Operation*

The third idealized operation which we will consider is called RemoveRedundant and is implementable. RemoveRedundant takes a Gröbner graph $\mathscr{G}=(V, E)$ (with a finite set V) and a set $V_0 \subset V$ and returns the set $X \subset V_0$ which is determined by the condition that $v \in X$ if and only if, using the notation of Lemma 12.13, $\mathscr{F}_w$ is the empty set.

The justification of the use of Remove Redundant requires the following proposition.

PROPOSITION 12.14. *Let $\mathscr{G}=(V, E)$ be a Gröbner graph, V be a finite set, $V_0 \subset V$ and X be the set which is generated by RemoveRedundant when it is applied to G and V_0. The ideal generated by V_0 equals the ideal generated by X.*

Proof. For each $v \in V_0$, let $\mathscr{F}_v$ be defined as in Lemma 12.13. Since $X \subseteq V_0$, $\mathscr{I}_X \subseteq \mathscr{I}_{V_0}$. To show that $\mathscr{I}_{V_0} \subseteq \mathscr{I}_X$, it suffices to show that $v \in \mathscr{I}_X$ for $v \in V_0 \backslash X$. Let $v \in V_0 \backslash X$. Since $v \notin X$, $\mathscr{F}_v$ is not the empty set. Let $\mathscr{H}_v$ be as in Lemma 12.13. $\mathscr{F}_w$ is empty for each starting vertex w of $\mathscr{H}_v$. By Proposition 12.8, v lies in the ideal generated by the starting vertices of $\mathscr{H}_v$ and so v lies in the ideal generated by X. This completes the proof of Proposition 12.14. ▌

RemoveRedundant can be a very fast operation since it requires only the searching of graphs (which is fast in comparison to GBA which is what the previous two idealized operations used).

12.7. *Respect for Categories*

Each of the above three operations, SmallBasis, ShrinkBasis and RemoveRedundant, is used to create a subset of the output of the GBA. While the output of these operations does generate the starting polynomial equations of the GBA run, they might not contain some particular polynomial equation which the "user feels is valuable" because a "shrinking" operation removed it. For example, it is possible that any of the above three operations introduced in this section will eliminate an entire category. Thus we need variations on these operations which reduce the chance of eliminating interesting polynomial equations.

12.7.1. *Idealized Shrinking by Category*

A protection against removing valuable polynomial equations is to not use the fact that the polynomial equations from a few categories may imply equations from a different category. This includes the case when two equations involving unknowns yield an equation involving only knowns. Certainly, if NCProcess finds an equation involving knowns, then this equation should be retained.

Recall that a spreadsheet consists[32] of

(1) Polynomial equations which solve for a variable.

(2) Polynomial equations which do not involve any unknowns.

(3) User selected and user created polynomial equations.

(4) Undigested polynomials.

The equations in (1), (2), and (3) above were referred to as digested equations. These items were displayed in terms of a list of smaller sets of

[32] See Section 2.2.1.

equations called categories.[33] This division of a set of polynomial equations into categories suggests the following possible ways to shrink a set of polynomial equations while preserving important equations.

I. Let $C_1, ..., C_\ell$ be the categories. The simplest way is to replace $C_j = \{p_j: 1 \leqslant j \leqslant k_1\}$ with a minimal set $\{q_j: 1 \leqslant j \leqslant k_2\}$ such that the p_j's and the q_j's generate the same ideal.

II. The most drastic way to shrink is to pick an ordering $\mathscr{C}\mathscr{O}$ on categories $C_1 < C_2 < \cdots < C_\ell$ of undigested polynomial equations (for example, the one induced by the ordering $\mathscr{O}$ underlying the run) and output the subset $X_0 \subset D$ and $X_j \subset C_j$ defined to satisfy

(1) A minimal generating set for the digested polynomials D. Call it X_0.

(2) A minimal set of the form $X_1 \cup X_0$ which is a generating set for the ideal generated by the union of the digested polynomials D and the category C_1.

(3) A minimal set of the form $X_2 \cup X_1 \cup X_0$ which is a generating set for the ideal generated by the union of the digested polynomial D and the categories C_1 and C_2.

(4) etc.

Here D is the set of digested polynomials.

III. An intermediate course which is less sensitive to ordering is to output the subset $X_0 \subset D$ and $X_j \subset C_j$ defined to satisfy

(1) A minimal generating set for the digested polynomials D. Call it X_0.

(2) A minimal set of the form $X_1 \cup X_0$ which is a generating set for the ideal generated by the union of the digested polynomials D and the category C_1.

(3) A minimal set of she form $X_2 \cup X_0$ which is a generating set for the ideal generated by the union of the digested polynomials D and the category C_2.

(4) etc.

Here D is the set of digested polynomials.

12.7.2. *Practical Shrinking by Category*

If one uses the approximation to the idealized operation SmallBasis (see Section 12.3.1) to find small generating sets (rather than minimal generating sets) for II in Section 12.7.1, then one obtains a SmallBasis operation

[33] See Section 2.2.3.

which respects the order of the categories. This SmallBasis operation is used in the NCProcess commands. Since the SmallBasis command runs a GBA, SmallBasis requires an iteration parameter to indicate at most how many iterations the GBA will run for.

If one uses the approximation to the idealized operation SmallBasis (see Section 12.3.1) to find small generating sets (rather than minimal generating sets) for III in Section 12.7.1, then one obtains the "Small Basis By Category" operation. The "Small Basis By Category" uses two iteration parameters. The first iteration parameter specifies how many iterations should be used in SmallBasis for III item 1 (in Section 12.7.1). The second iteration parameter specifies how many iterations should be used in SmallBasis for III items 2, 3, and 4 (in Section 12.7.1).

If one uses the RemoveRedundant operation (see Section 12.6) to find small generating sets (rather than minimal generating sets) for III in Section 12.7.1, then one obtains the "Remove Redundant By Category" operation.

12.8. *RemoveRedundantProtected*

The command RemoveRedundantProtected combines the results of three Remove Redundant runs. This command is used to prevent equations not involving unknowns from being removed by polynomials which involve unknowns and to prevent digested polynomials from being removed by polynomials which are undigested. The first step is to compute RemoveRedundant on only the polynomials which do not involve knowns. The second step is to compute RemoveRedundant on the polynomials which are digested. The third step is to compute RemoveRedundant on all of the polynomials. The result of using RemoveRedundantProtected is the union of the results of the three RemoveRedundant runs.

13. SPEEDING UP RUNS

While this section is called "Speeding up runs," which would be of interest to a practitioner, this section suggests open questions which might be of interest to the theorist.

13.1. *DegreeCap and DegreeSumCap*

One way to reduce the run time for the *NCProcess* commands is to use the options capping the degree of the polynomials that are produced in the course of running the *NCProcess* commands.

This is valuable since the user will ordinarily know that a polynomial of a very high degree will not be useful to him and so there is no reason to produce it. It is not the time that it takes to produce a large polynomial that is the primary factor. Rather it is the reduction algorithms that will get bogged down trying to remove it. Degree caps prevent the algorithm from ever producing polynomials over a certain degree, or combining polynomials over a certain degree, and the user will still be left with a generating set for the ideal generated by the input equations. There are two different options associated with degree caps. For instance,

$$\text{DegreeCap} \to 8$$

would prevent a polynomial of degree 8 or higher from combining with a polynomial of higher degree.

$$\text{DegreeSumCap} \to 10$$

would prevent two polynomials whose degrees add up to 10 or more from combining. Degree caps could prevent an important relation from being created, so when there is a lack of progress, raising the degree caps as well as the iteration number would be the next step.

WE URGE USE OF DEGREE CAPS. THEY SAVE A LOT OF TIME.

13.2. *Monomial Orders vs Run Time*

The command $SetUnknowns[x_1, ..., x_s]$ imposes a pure lex order

$$x_1 \ll x_2 \ll \cdots \ll x_s$$

on monomials in the unknowns $x_1, ..., x_s$. In the examples of this paper, we did each of our experiments with a multigraded lex order. We found that this did not effect the answer. In the commutative case, a typical GBA runs faster with a graded lex order than a pure lex order. Thus, in practice, one might be advised to set a graded lex order on the unknowns using the command $SetUnknowns[\{x_1, ..., x_s\}]$.

14. DETAILS OF NCPROCESS

The following section gives pseudocode for the NCProcess1 and NCProcess2 commands.

This pseudocode uses the function NCMakeGB which implements a Gröbner Basis Algorithm which returns partial GB's which are reduced. In particular, running the function NCMakeGB for 0 iterations on a set F does not compute any S-polynomials, but does produce a set G which is reduced (see discussion following Lemma 12.2). G will be called a reduced form of F.

14.1. *NCProcess*1 *Command*

The input to NCProcess1 is a set of starting equations *start*, a number of iterations n for the GBA and a collection of user selects.

The steps taken by NCProcess1 are:

I. Preparation for the main call to NCMakeGB

(1) Run the GBA on the equations in *start* which do not involve any unknown variables together with the user selects for at most $n+1$ iterations. Let A denote this partial GB.

(2) Shrink A using the RemoveRedundantProtected operation. Call this shrunken set B.

II. The main call to NCMakeGB

(3) Run NCMakeGB with the input B together with *start* for at most n iterations. In this NCMakeGB run, S-polynomials between two elements of the partial GB of A are not computed. Let C denote this partial GB.

III. Shrinking the partial GB

(4) Shrink C using the RemoveRedundantProtected operation. Call this shrunken set D.

(5) Let D_k be the set of polynomials in D which do not involve any unknowns. Let $D_u = D \backslash D_k$. Let E_u be a set of the normal forms of the elements of D_u with respect to D_k. Let $E = D_k \cup E_u$.

(6) Let F be the union of E and the user selects. Let G be a reduced form of F (see the beginning of Section 14).

(7) Shrink G by SmallBasisByCategory using iteration parameters $n+1$ and $n+2$ (see Section 12.7.2). Call this shrunken set H.

IV. Attempt Decompose

(8) Construct the collected forms (as described in Section 9) of the polynomials in H.

V. Displaying the results

(9) For elements of H, if the polynomial's only collected form is trivial (see discussion following Definition 9.1), then display the rule corresponding to the polynomial, otherwise display the collected form of the polynomial. This is the step in which the "spreadsheets" of the results of this paper are constructed.

VI. Return a three tuple to the user for future use

(10) Return the triple (A, H_0, H_1) to the user where A is from item 1 above, H_0 is the set of polynomials in H which are digested and H_1 is the set of polynomials in H which are undigested.

14.2. *NCProcess*2 *Command*

The input to NCProcess2 is a set of starting equations *start*, a number of iterations n for SmallBasis and a collection of user selects.

The steps taken by NCProcess2 are:

I. Shrinking the input equations

(1) Shrink *start* using the RemoveRedundantProtected operation. Call this shrunken set D.

(2) Let D_k be the set of polynomials in D which do not involve any unknowns. Let $D_u = D \backslash D_k$. Let E_u be a set of the normal forms of the elements of D_u with respect to D_k. Let $E = D_k \cup E_u$.

(3) Let F be the union of E and the user selects. Let G be a reduced form of F. (see the beginning of Section 14).

(4) Shrink G by SmallBasis. Set H equal to the result of the shrinking.

II. The "Attempt Decompose" "Displaying the results" and "Return a three tuple to the user for future use" as in Section 14.1.

APPENDIX A. ADJOINTS

In Section 1.2.3, we used the notation $q(a_1, ..., a_r, x_1, ..., x_s)^*$ in conjunction with problems involving an algebra which possessed an involution, $w \rightarrow w^*$. We now make this notion precise.

We start by assuming that there is an involution on the coefficients, say $t \in K \rightarrow \bar{t} \in K$.

By relabelling the knowns and unknowns if necessary, let us suppose that

(1) The polynomial q depends upon the knowns $a_1, ..., a_{r_0}$ and upon the unknowns $x_1, ..., x_{s_0}$ where $r_0 \leqslant r$ and $s_0 \leqslant s$.

(2) The polynomial q does not depend on the knowns $a_{r_0+1}, ..., a_r$ and does not depend on the unknowns $x_{s_0+1}, ..., x_s$.

Let us also suppose that there is an injective map σ_1 from $\{1, ..., r_0\}$ to $\{1, ..., r\}$ and an injective map σ_2 from $\{1, ..., s_0\}$ to $\{1, ..., s\}$ where $r_0 \leqslant r$ and $s_0 \leqslant s$.

We now make the assumption which corresponds to the problem involving the algebra with involution:

> If the mathematical problem we are trying to investigate involves elements of an algebra $\mathscr{A}$ with involution, say $w \to w^*$, and we substitute $A_i \in \mathscr{I}$ for a_i and $X_j \in \mathscr{I}$ for x_j for $i = 1, ..., r$ and for $j = 1, ..., s$, then

$$A_i^* = A_{\sigma_1(i)} \quad \text{for} \quad i = 1, ..., r_0 \qquad \text{and} \qquad X_j^* = X_{\sigma_2(j)} \quad \text{for} \quad j = 1, ..., s_0.$$

There is a unique operation * from $K[a_1, ..., a_{r_0}, x_1, ..., x_{s_0}]$ to $K[a_1, ..., a_r, x_1, ..., x_s]$ such that the following four conditions hold:

(1) * is additive (i.e., if $p, q \in K[a_1, ..., a_{r_0}, x_1, ..., x_{s_0}]$, then $(p+q)^* = p^* + q^*$).

(2) if $p, q \in K[a_1, ..., a_{r_0}, x_1, ..., x_{s_0}]$, then $(pq)^* = q^* p^*$.

(3) $a_1^* = a_{\sigma_1(i)}$ for $i = 1, ..., r_0$.

(4) $x_j^* = x_{\sigma_2(j)}$ for $j = 1, ..., s_0$.

APPENDIX B. EXAMPLE OF DISCOVERING

EXAMPLE B.1. Suppose you are working with matrices a, b, c etc. If an NCProcess command discovers that $ab = bc$ and, at that point, you realize that it is reasonable to assume that no eigenvalue of a is an eigenvalue of c, then b must be zero from an analytic argument.[34] Therefore, one can

[34] In fact even if one realized from the beginning of the computation that a and c have no common eigenvalues, there is no way to express this hypothesis using polynomial equations. In fact, a complete analytic argument which showed that $b = 0$ would consist of an algebraic derivation of the identity $ab = bc$ followed by the use of the hypothesis that a and c have no common eigenvalues. One would not in general know in advance that $ab = bc$ and so it would be premature to add "$b = 0$" to the collection of polynomial equations at the beginning of the computation.

add the polynomial equation $b=0$ to the collection of polynomial equations and continue. As research progressed, one would add more and more analytic observations at the points where it became clear what those observations were.

The syllogism corresponding to the above type of argument would go as follow.

Fact B.2. If ⟨whatever hypothesis⟩, then $ab=bc$.

Fact B.3. If a, b and c are matrices, no eigenvalue of a is an eigenvalue of c and $ab=bc$, then $b=0$.

Fact B.4. If ⟨*whatever hypothesis*⟩ and $b=0$, then ⟨desired conclusion⟩.

Conclusion B.5. If ⟨whatever hypothesis⟩, then ⟨desired conclusion⟩.

Fact B.2 would be seen by using the algorithms in this paper. Fact B.3 is an observation from analysis. Fact B.4 would be seen by using the algorithms in this paper. Conclusion B.5 is a tautological conclusion of the Facts B.2, B.3, and B.4.

APPENDIX C. FORMAL DESCRIPTIONS OF PURE LEX AND MULTIGRADED LEX

C.1. *An ordering on* $\mathbf{N}^n$

We begin by giving the definition of pure lex order on $\mathbf{N}^n$ given in Section 2.2 of [CLS]. This order will play a central role in defining pure lex and multilex orders on the monic monomials of $K[x_1, ..., x_n]$.

Definition C.6 [CLS]. *Pure lex on* N^n. Let $\alpha=(a_1, ..., a_n)\in\mathbf{N}^n$ and $\beta=(b_1, ..., b_n)\in\mathbf{N}^n$. We say that α is greater than β if, in the vector difference $\alpha-\beta\in\mathbf{Z}^n$, the left-most nonzero entry is positive.

C.2. *Formal Description of Pure Lex*

For a monic monomial M and a variable v, let $occ(M, v)$ be the number of occurrences of v in M. (For example, $occ(abcdc, a)=1$, $occ(abcdc, b)=1$, $occ(abcdc, c)=2$, $occ(abcdc, d)=1$ and $occ(abcdc, e)=0$.)

Let M and N be monic monomials in the variables $x_1, ..., x_n$. M is less than N with respect to the pure lex order $x_1 \ll x_2 \ll \cdots \ll x_n$ if one of the following two conditions holds.

(1) $(occ(M, x_1), ..., occ(M, x_n))<(occ(N, x_1), ..., occ(N, x_n))$

(2) $(occ(M, x_1), ..., occ(M, x_n))=(occ(N, x_1), ..., occ(N, x_n))$ and M is less than N with respect to the graded lex order $x_1<x_2<\cdots<x_n$.

C.3. *Formal Description of Multigraded Lex*

To specify a multigraded lex order on monic monomials in the variables $x_1, ..., x_n$ such that $x_i < x_j$ if $1 \leqslant i \leqslant j \leqslant n$, then one needs to specify a subset of $\{1, ..., n-1\}$. If M and N are monic monomials in the variables $x_1, ..., x_n$, then M is less than N with respect to the multigraded lex order if one of the following two conditions hold:

(1) $(occ(M, V_1), ..., occ(M, V_\ell)) < (occ(N, V_1), ..., occ(N, V_\ell))$

(2) $(occ(M, V_1), ..., occ(M, V_\ell)) = (occ(N, V_1), ..., occ(N, V_\ell))$ and M is less than N with respect to the graded lex order $x_1 < \cdots < x_n$

where

(a) $V_k = \{x_j \colon a_{j-1} < k \leqslant a_j\}$ (we set $a_{-1} = 0$ and $a_{\ell+1} = n$) for $1 \leqslant j \leqslant \ell$

(b) $occ(M, V)$ is the sum of $occ(M, v)$ for all $v \in V$.

To denote the above order, we write down the sequence of characters

$$x_1 s_1 x_2 s_2 \cdots s_{n-1} x_n$$

where s_j is "$\ll$" if j is one of the a's and s_j is "$<$" otherwise.

Note that if the collection of subsets is the empty set, then the multigraded lex order is graded lex and if the collection of subsets is $\{1, 2, ..., n-1\}$, then the multigraded lex order is a pure lex order.

As another example, a multigraded lex order for monic monomials in a, b, c, d and e such that $a < b < c < d < e$ can be specified by specifying the set $\{2, 4\}$. This multigraded lex order would be denoted $a < b < c < d \ll e$.

ACKNOWLEDGMENTS

The authors thank Kurt Schneider and Stan Yoshinobu who wrote some of the Mathematica software and contributed heavily to the working of examples in this paper. In particular, they did substantial parts of the examples in Sections 6 and 3. Kurt Schneider did much of the work for the example in Section 6 and Stan Yoshinobu did much of the work in the example in Section 4.1. Victor Shih and Mike Moore contributed to the C^{++} coding behind the NCProcess command. We thank Johan Kaashoek, Nicholas Young, and Hugo Woerderman for providing problems and comments on our computer generated solutions. Johan Kaashoek deserves special thanks for detailed and valuable help. We also thank Roger Germundsson for helpful discussions involving commutative Gröbner Basis.

REFERENCES

[BGK] H. Bart, I. Gohberg, and M. A. Kaashoek, "Minimal Factorization of Matrix and Operator Functions," Birkhäuser, Basel, 1979.

[BJLW] W. W. Barrett, C. R. Johnson, M. E. Lundquist, and H. Woerderman, Completing a block diagonal matrix with a partially prescribed inverse, *Linear Algebra Appl.* **223/224** (1995), 73–87.

[Ch] S. C. Chou, “Mechanical Geometry Theorem Proving,” D. Reidel Publishing Company, Dordrecht, The Netherlands, 1988.

[CLS] D. Cox, J. Little, and D. O’Shea, “Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra,” Undergrad. Texts Math., Springer-Verlag, Berlin/New York, 1992.

[DGKF] J. C. Doyle, K. Glover, P. P. Khargonekar, and B. A. Francis, State–space solutions to standard H_2 and H_∞ control problems, *IEEE Trans. Automata Control* **34** (1989), 831–847.

[FMora] F. Mora, “Groebner Bases for Non-commutative Polynomial Rings,” Lecture Notes in Comput. Sci., Vol. 229, pp. 353–362, Springer-Verlag, Berlin/New York, 1986.

[FaFeGr] D. R. Farkas, C. D. Feustel, and E. L. Green, Synergy in the theories of Gröbner bases and path algebras, *Canad. J. Math.* **45**(4) (1993), 727–739.

[HSW] J. W. Helton, M. Stankus, and J. J. Wavrik, Computer simplification of engineering formulas, *IEEE Trans. Automat. Control* **43** (1998), 302–314.

[HW] J. W. Helton and J. J. Wavrik, Rules for computing simplifications of the formulas in operator model theory and linear systems, *Oper. Theory Adv. Appl.* **73** (1994), 325–354.

[NCA] J. W. Helton, R. L. Miller, and M. Stankus, “NCAlgebra: A Mathematica Package for Doing Non-Commuting Algebra,” available from the web site http://math.ucsd.edu/ ∼ ncalg.

[NCGBDoc] J. W. Helton and M. Stankus, “Non-Commutative Gröbner Basis Package,” available from ncalg@ucsd.edu.

[TMora] T. Mora, An introduction to commutative and noncommutative Gröbner Bases, *Theor. Comput. Sci.* **134** (1994), 131–173.

[Wu1] Wu Wen-tsun, On the decision problem and the mechanization of theorems in elementary geometries, *Scientia Sinica* **21** (1978), 159–172; Re-published in “Automated Theorem Proving: After 25 Years,” Contemporary Mathematics, Vol. 29, pp. 213–234, American Mathematics Society, Providence, Rhode Island, 1984.

[Wu2] Wu Wen-tsun, Toward mechanization of geometry—Some comments on Hilbert’s “Gründlagen der Geometrie,” *Acta Math. Scientia* **2** (1982), 125–138.

[Y] N. Young, “An Introduction to Hilbert Space,” Cambridge Math. Textbooks, Cambridge Univ. Press, Cambridge, UK, 1988.