Computer Simplification of Engineering Systems Formulas *

J. William Helton † Mark Stankus† John Wavrik†

ABSTRACT

Currently, the three most popular commercial computer algebra systems are Mathematica, Maple, and MACSYMA (the 3 M's). These systems provide a wide variety of symbolic computation facilities for commutative algebra and contain implementations of powerful algorithms in that domain. The Gröbner Basis Algorithm, for example, is an important tool used in computation with commutative algebras and in solving systems of polynomial equations.

On the other hand, most of the computation involved in linear control theory is performed on matrices and these do not commute. A typical issue of IEEE TAC is full of A B C D type linear systems and computations with the A B C D's or partitions of them into block matrices. The 3 M's are weak in the area of non-commutative operations. They allow a user to declare an operation to be non-commutative, but provide very few commands for manipulating such operations and no powerful algorithmic tools.

It is the purpose of this article to report on applications of a powerful tool: a non-commutative version of the Gröbner Basis Algorithm. The commutative version of this algorithm is implemented on each of the three M's. It has many applications ranging from solving systems of equations to computations involving polynomial ideals. The noncommutative version is relatively new [Mora]. Our application to the simplification of expressions which occur in systems theory is unique. We will describe the Gröbner Basis for several elementary situations which arise in systems theory. These give (in a sense to be made precise) a "complete" set of simplifying rules for formulas which arise in these situations. We have found that this process elucidates the nature of simplifying rules and provides a practical means of simplifying some types of complex expressions.

The research required the use of software suited for computing with non-commuting symbolic expressions. Most of the research was performed using a special-purpose system developed for the project by J. Wavrik. This system uses a new approach to the development of mathematical software. It provides the flexibility needed for experimentation with algorithms, data representation, and data analysis.

In another direction, Helton, Miller and Stankus have written packages for Mathematica called NCAlgebra [1] which extend many of Mathematica's commands to symbolic expressions in non-commutative algebras. We have incorporated in these packages some of the results on simplification described in this paper.

1 THE GENERAL IDEA

Start with simple matrix variables and expressions like x, x^{-1} and $(1-x)^{-1}$. Repeatedly perform arithmetic operations (addition, subtraction and multiplication) on these. Very complex expressions can be obtained in this way. It is quite possible for two expressions which look quite different to be equivalent in the sense that they represent the same matrix. We would like to find the simplest possible expression among those representing it.

Indeed, we would like a mechanical procedure which selects one which is simplest in some sense. There is, however, no absolute concept of simplicity. By "simpler" we could mean "takes less computer time to evaluate using a particular software system and computer" or "has fewer terms" or "has fewer factors in each term". Whatever the concept of "simple", it must have the property that if A is simpler than B, and B is simpler than C, then A is simpler than C. Also, we will use simplification techniques which require that if A is simpler than B and B is simpler than A, then A equals

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

†Department of Mathematics, University of California, San Diego, California 92093 helton@osiris.ucsd.edu, mstankus@osiris.ucsd.edu, jjwavrik@ucsd.edu

[1] contact ncalg@osiris.ucsd.edu

B. In other words, it must be a partial order on the set of expressions. A simplifier is a process which, when applied to a expression, f, produces a expression, r, which is mathematically equivalent to f and satisfies $r \leq f$.

The simplifier we will study here is a generalization to polynomials of several variables of the familiar division algorithm and might be called a Gröbner type of simplifier. Progress can be made with this type of simplification and it shows evidence of being in practice a very valuable tool.

1.1 Replacement Rules

Crucial to our simplification procedures is finding a list of rules for simplifying expressions.

A replacement rule consists of a left hand side (LHS), which will always be a monomial, and a right hand side (RHS) which will always be a polynomial. Our notation is $LHS \to RHS$. Naturally we are unwilling to substitute RHS for LHS unless the equation $LHS - RHS = 0$ is true. Thus replacement rules are associated with equations.

We first give a simple illustration of the reduction of polynomials by a list of rules. The definition of $(1-xy)^{-1}$ implies that $(1-xy)(1-xy)^{-1} = 1$ which we write in the form

$$xy(1-xy)^{-1} - (1-xy)^{-1} + 1 = 0\,, \tag{1.1}$$

which states that a polynomial [2] in x, y and $(1-xy)^{-1}$ equals 0. The 3 replacement rules we could associate to this are

$$1 \to (1-xy)^{-1} - xy(1-xy)^{-1} \tag{1.2}$$

$$(1-xy)^{-1} \to xy(1-xy)^{-1} + 1 \tag{1.3}$$

and

$$xy(1-xy)^{-1} \to (1-xy)^{-1} - 1\,. \tag{1.4}$$

We wish to use the rule to make expressions less complicated so we choose the last rule which replaces the "most complicated" monomial in (1.1) by a sum of simpler ones. Now we apply this rule on an expression. It is an algebraic fact that the polynomial

$$(1-xy)^{-2} - 2xy(1-xy)^{-2} + xyxy(1-xy)^{-2} - 1$$

simplifies to 0. We now claim that two applications of the replacement rule (1.4) can be used to establish this fact. The first application converts this to

$$(1-xy)^{-2} - 2((1-xy)^{-1} - 1)(1-xy)^{-1} +$$

$$xy((1-xy)^{-1} - 1)(1-xy)^{-1} - 1$$

which after distributing the products over the sums is

$$-(1-xy)^{-2} + 2(1-xy)^{-1} + xy(1-xy)^{-2} - xy(1-xy)^{-1} - 1$$

and the second application converts this to

$$-(1-xy)^{-2} + 2(1-xy)^{-1} + ((1-xy)^{-1} - 1)(1-xy)^{-1} -$$

$$((1-xy)^{-1} - 1) - 1$$

which is seen to be zero after cancelling like terms.

[2] the polynomial is $xyz - z + 1$ where we take $z = (1-xy)^{-1}$.

1.2 Complete lists of rules

At last we come to one of the more basic points which is the concept of 'completeness' of a list of rules. We illustrate this with an example.

Example 1.5 Consider x, x^{-1} and $(1-x)^{-1}$ and rules based on the definition of inverse

$$\begin{array}{ll} xx^{-1} \to 1 & x(1-x)^{-1} \to (1-x)^{-1} - 1 \\ x^{-1}x \to 1 & (1-x)^{-1}x \to (1-x)^{-1} - 1\,. \end{array} \tag{1.6}$$

There are polynomials in x, x^{-1} and $(1-x)^{-1}$ such as

$$x^{-1}(1-x)^{-1} - (1-x)^{-1}x^{-1} \tag{1.7}$$

which equals 0, but clearly the computer can not decide that it is 0 through repeated application of the rules (1.6). In this case (and most others) one needs an enlarged set of rules. Add to (1.6) the rules

$$\begin{array}{l} x^{-1}(1-x)^{-1} \to x^{-1} + (1-x)^{-1} \\ (1-x)^{-1}x^{-1} \to x^{-1} + (1-x)^{-1} \end{array} \tag{1.8}$$

(which an operator theorist calls the resolvent identity). It is clear that (1.7) reduces to 0 by applying the full list of rules (1.6) and (1.8) repeatedly (actually one pass will do and you do not really need (1.6)).

More generally one can prove after firming up the definitions (in the next section) that when this full list of rules is applied repeatedly to any polynomial p in x, x^{-1} and $(1-x)^{-1}$ one obtains one particular reduced polynomial q *regardless of the order in which the rules are applied.* That is q is a "canonical form" for p, which is in a certain sense as simple as possible. Also, one can prove that *if it is possible to derive using only algebra and the definitions of x^{-1} and $(1-x)^{-1}$ that $p = 0$, then it's canonical form is zero.*

Thus the set of rules (1.6) and (1.8) are "complete" in a profound sense. Such a set of rules is called a *Gröbner basis* or *GB* for x,x^{-1},$(1-x)^{-1}$ with a particular ordering on monomials which will be described in the next subsection. We shall not give a formal definition of a GB but refer the reader to **[Mora]**.

Finding such a set of rules in any situation where you are working is a very useful accomplishment. The process of computing a noncommutative Gröbner basis often is a time consuming process [3], but once it has been found it is very easy to use and provides the user with a powerful computational tool.

1.3 Ordering

The point of rules is to replace complicated monomials by sums of simpler monomials. Now we describe how one formalizes the notion of complicated verses simple. This is done by placing an order on monomials. Once we have imposed an order $>$ on monomials we will insist that any replacement rule respects this order, in other words, complicated monomials are replaced by sums of simpler ones.

We say that the monomial $M \leq N$ if and only if

$$\text{either length}(M) < \text{ length}(N) \text{ or}$$

$$\text{both length}(M) = \text{ length}(N)$$

and M would come before N in the dictionary.

This is called **graded lexicographic ordering** of the monomials.

Every polynomial f is a sum of monomials. Exactly one of these monomials is largest in the graded lexicogrpaphic ordering. We call this the leading term of f and denote it by $LT(f)$.

Example 1.9 Let us write a graded lexicographic order down for the case when we are working with the three symbols x, y and $(1-xy)^{-1}$. Let us take the ordering on degree one monomials to be

$$x < y < (1-xy)^{-1}$$

This plays the role of alphabetical order in the lexicographic order. It naturally induces the order

$$xx < xy < x(1-xy)^{-1} < yx < yy < y(1-xy)^{-1} <$$
$$(1-xy)^{-1}x < (1-xy)^{-1}y < (1-xy)^{-1}(1-xy)^{-1}$$

on degree 2 monomials and by analogy on degree 3 monomials, etc.

In this paper, the **only orderings** considered are graded lexicographic. This is the ordering we use to convert equations to rules. Namely, in any equation put the highest order monomial on the LHS of the rule so that it gets replaced by the negation of the sum of the lower order monomials. The reason we insist on having a total order is to rule out the possibility of infinite cycles where one makes one replacement and a few steps later reverse it and then repeat.

[3] If the Mora Algorithm terminates (with a finite basis) this is automatically a Grobner Basis. If the Mora Algorithm produces an infinite set, additional work is required to show that this set is a Gröbner Basis.

2 SOME WIDELY APPLICABLE RESULTS

Now that the we have the basic ideas, we give a simple but practical list of simplifying rules which apply to many situations. The first list, called RESOL rules, is a generalization of the example presented above which involved x, x^{-1} and $(1-x)^{-1}$.

RESOL Rules

$$(RESOL0) \qquad (\lambda - x)^{-1}x \to \lambda(\lambda - x)^{-1} - 1$$

$$(RESOL1) \qquad x(\lambda - x)^{-1} \to \lambda(\lambda - x)^{-1} - 1$$

$$(RESOL2) \qquad (\mu - x)^{-1}x \to \mu(\mu - x)^{-1} - 1$$

$$(RESOL3) \qquad x(\mu - x)^{-1} \to \mu(\mu - x)^{-1} - 1$$

$(RESOL4)$

$$(\mu-x)^{-1}(\lambda-x)^{-1} \to \frac{1}{\mu-\lambda}(\lambda-x)^{-1} + \frac{1}{\lambda-\mu}(\mu-x)^{-1}$$

$(RESOL5)$

$$(\lambda-x)^{-1}(\mu-x)^{-1} \to \frac{1}{\mu-\lambda}(\lambda-x)^{-1} + \frac{1}{\lambda-\mu}(\mu-x)^{-1}$$

for all operators x and y on a Hilbert space $\mathcal{H}$ and distinct complex numbers λ and μ. The following theorem is an easy generalization of a corresponding result from **[HW]**.

Theorem 2.1. *The list of RESOL rules is a Gröbner basis in the symbols x, $(\lambda-x)^{-1}$ and $(\mu-x)^{-1}$ where λ and μ are distinct complex numbers and the monomials are ordered using a graded lexicographical order with $x < (\lambda-x)^{-1} < (\mu-x)^{-1}$.*

A Gröbner basis can be found for polynomials in $x < y < x^{-1} < y^{-1} < (1-x)^{-1} < (1-y)^{-1} < S < T$ where S and T are variables and we take as starting relations the defining relations for the inverses and the relations $Sx = xT$ and $yS = Ty$ (see **[HW]**). If λ is a nonzero complex number, then this basis produces a very interesting and powerful set of simplifying rules which hold for all operators x, y on a Hilbert space with x, y, $\lambda - x$ and $\lambda - y$ invertible and for all functions h analytic on the spectrum of xy and yx.

GENR Rules

$$(Gr0) \qquad h(xy)x \to xh(yx)$$

$(Gr1)\qquad h(yx)x^{-1} \to x^{-1}h(xy)$

$(Gr2)\quad xh(yx)(\lambda - x)^{-1} \to \lambda h(xy)(\lambda - x)^{-1} - h(xy)$

$(Gr3)$
$x^{-1}h(xy)(\lambda - x)^{-1} \to \lambda^{-1}x^{-1}h(xy) + \lambda^{-1}h(yx)(\lambda - x)^{-1}$

$(Gr4)$
$$(\lambda - x)^{-1}h(yx)(\lambda - x)^{-1} - (\lambda - x)^{-1}h(xy)(\lambda - x)^{-1} \to$$
$$\lambda^{-1}(h(yx)(\lambda - x)^{-1} - (\lambda - x)^{-1}h(xy))$$

Operators of the form $h(xy)$ arise, of course, in what is called the *functional calculus* of the operator xy. Examples are $h(xy) = (1 - xy)^{-1}$, $h(xy) = (1 - xy)^{-1/2}$ and $h(xy) = e^{xy}$ provided the eigenvalues of xy are in the right location.

For the case of $\lambda = 1$, the (GENR) rules are obtained by substituting $h(xy)$ for S and $h(yx)$ for T in the Gröbner basis from **[HW]**. It is easy to check by hand that the (GENR) rules hold for $\lambda \neq 1$. Observe that each of the rules in either (RESOL) or (GENR) are easy to prove directly. For example, the following calculation verifies (Gr3).

$$\begin{aligned} x^{-1}h(xy)(\lambda - x)^{-1} &= h(yx)x^{-1}(\lambda - x)^{-1} \\ &= h(yx)(\lambda^{-1}x^{-1} \\ &\qquad + \lambda^{-1}(\lambda - x)^{-1}) \\ &= \lambda^{-1}h(yx)x^{-1} + \\ &\qquad \lambda^{-1}h(yx)(\lambda - x)^{-1} \end{aligned}$$

Possibly, this set of rules is of very general interest and might be worth teaching to all people beginning in the area of matrix or operator algebra. Already (RESOL4), called the resolvent identity, is universally taught while only the special case (Gr0) is common in introductory courses.

From the rules (RESOL) and (GENR) we obtain, for each particular analytic function h, an expanded set of rules: we may replace $h(xy)$ with $h(xy)^n$ and $h(yx)$ with $h(yx)^n$ for any positive integer n; and we may interchange x with y. The rules obtained in this way are all obviously valid rules. The expanded set is discussed more thoroughly in **[HW]**. It is not, in general, a Gröbner basis, because it does not contain rules which come from the special properties of a particular h. For example, if $h(xy) = (1 - xy)^{-1}$, the replacement rule $xyh(xy) \to h(xy) - 1$ is not a consequence of the rules in the expanded set even though it holds for this particular h. On the other hand, this rule is not universally applicable. We think of the expanded set as a complete set of universally applicable rules. We have found that extremely effective simplification can be performed just using the expanded set supplemented by some of the obvious rules for a particular h.

3 FURTHER COMMON SITUATIONS

We have also found GB for reducing polynomials in

(EB) x, y, x^{-1}, y^{-1}, $(1 - xy)^{-1}$ and $(1 - yx)^{-1}$

(preNF) x, y, x^{-1}, y^{-1}, $(1 - x)^{-1}$, $(1 - y)^{-1}$, $(1 - xy)^{-1}$ and $(1 - yx)^{-1}$.

(NF) x, y, x^{-1}, y^{-1}, $(1 - x)^{-1}$, $(1 - y)^{-1}$, $(1 - xy)^{-1}$, $(1 - yx)^{-1}$, $(1 - xy)^{-1/2}$ and $(1 - yx)^{-1/2}$.

These objects were chosen for extensive study because they form the cornerstone of computations in various subjects. The expressions (EB) occur in energy balance formulas in H^∞ control. See section 4. These combined with basic inverses often called resolvents constitute (preNF). The relations (NF) are extremely important to those working with 2×2 block unitary matrices or with the Nagy-Foias **[NF]** model which is the paradigm for discrete time energy conserving systems. Due to space constraints, we only list the answer for (EB) (see **[HW]**,**[HWS]** for the other lists).

3.1 A Gröbner basis for EB

The indeterminants which are used in EB and their ordering which we use is as follows.

$$x < y < x^{-1} < y^{-1} < (1 - xy)^{-1} < (1 - yx)^{-1} .$$

The set of relations of EB is the set of defining relations of x^{-1}, y^{-1}, $(1 - xy)^{-1}$ and $(1 - yx)^{-1}$ (EB_0 through EB_7 below). These relations are not a Gröbner basis. The following theorem shows that one can extend this list of relations to obtain a Gröbner basis.

Theorem 3.1([HW]). *The following relations constitute a finite Gröbner basis for* (EB).

$$\begin{aligned}
\mathrm{EB}_0 &= x^{-1}\,x - 1 \\
\mathrm{EB}_1 &= xx^{-1} - 1 \\
\mathrm{EB}_2 &= y^{-1}\,y - 1 \\
\mathrm{EB}_3 &= yy^{-1} - 1 \\
\mathrm{EB}_4 &= xy\,(1 - xy)^{-1} - (1 - xy)^{-1} + 1 \\
\mathrm{EB}_5 &= yx\,(1 - yx)^{-1} - (1 - yx)^{-1} + 1 \\
\mathrm{EB}_6 &= (1 - xy)^{-1}xy - (1 - xy)^{-1} + 1 \\
\mathrm{EB}_7 &= (1 - yx)^{-1}yx - (1 - yx)^{-1} + 1 \\
\mathrm{EB}_8 &= (1 - yx)^{-1}\,x^{-1} - y(1 - xy)^{-1} - x^{-1} \\
\mathrm{EB}_9 &= (1 - xy)^{-1}\,y^{-1} - x\,(1 - yx)^{-1} - y^{-1} \\
\mathrm{EB}_{10} &= x^{-1}\,(1 - xy)^{-1} - y\,(1 - xy)^{-1} - x^{-1} \\
\mathrm{EB}_{11} &= y^{-1}\,(1 - yx)^{-1} - x\,(1 - yx)^{-1} - y^{-1} \\
\mathrm{EB}_{12} &= (1 - yx)^{-1}\,y - y\,(1 - xy)^{-1} \\
\mathrm{EB}_{13} &= (1 - xy)^{-1}\,x - x\,(1 - yx)^{-1} .
\end{aligned}$$

Here we follow the algebraist convention and suppress the $= 0$ or the rule ('$\rightarrow$') notation. Also, clearly one only needs to remember 7 rules and the fact that one can derive the other 7 by swapping x and y. Note: EB_6 and EB_7 can be eliminated using the other rules and we still have a GB.

3.2 How to find a Gröbner Bases

In the commutative setting, there is a standard algorithm (Buchberger) which always terminates and provides a Gröbner Basis. We have used an adaptation of the algorithm to polynomials in non-commuting variables due to F. Mora. The details of this algorithm, which we refer to as Mora's Algorithm, are found in **[Mora]** and **[HW]**. In contrast to the commutative case, the Mora Algorithm does not always terminate. In this case, Mora's algorithm does not guarantee a Gröbner basis, so additional techniques are needed.

4 AN APPLICATION OF A GRÖBNER BASIS

In H^∞ control one deals with a Hamiltonian H on the state space (x, z) of the closed loop system. Here x is a state of the plant and z is a state of the compensator. The unknowns in H are a quadratic form ε which is to be a storage function of the closed loop system and the a, b, c, d of the unknown compensator. As usual take $d = 0$. If a solution ε exists one can derive that some controller, called the central controller, a, b, c must be given by certain formulas. These formulas do not imply that a solution to the H^∞ control problem exist and to see if it does one must plug the central controller formulas back into H and see if $H \leq 0$ for all states (x, z) of the closed loop system. Let's see how this computation goes with the methods of section 3.

```
H= tp[x] ** X ** A ** z + tp[x] ** inv[Y] ** A ** x -
  tp[x] ** inv[Y] ** A ** z + tp[x] ** tp[A] ** X ** z +
  tp[x] ** tp[A] ** inv[Y] ** x - tp[x] ** tp[A] ** inv[Y] ** z +
  tp[x] ** tp[C1] ** C1 ** x + tp[z] ** X ** A ** x -
  tp[z] ** X ** A ** z - tp[z] ** inv[Y] ** A ** x +
  tp[z] ** inv[Y] ** A ** z + tp[z] ** tp[A] ** X ** x -
  tp[z] ** tp[A] ** X ** z - tp[z] ** tp[A] ** inv[Y] ** x +
  tp[z] ** tp[A] ** inv[Y] ** z + tp[x] ** X ** B1 ** tp[B1] ** X ** z -
  tp[x] ** X ** B2 ** tp[B2] ** X ** z +
  tp[x] ** inv[Y] ** B1 ** tp[B1] ** inv[Y] ** x -
  tp[x] ** inv[Y] ** B1 ** tp[B1] ** inv[Y] ** z +
  tp[z] ** X ** B1 ** tp[B1] ** X ** x -
  tp[z] ** X ** B1 ** tp[B1] ** X ** z -
  tp[z] ** X ** B2 ** tp[B2] ** X ** x +
  tp[z] ** X ** B2 ** tp[B2] ** X ** z -
  tp[z] ** inv[Y] ** B1 ** tp[B1] ** inv[Y] ** x +
  tp[z] ** inv[Y] ** B1 ** tp[B1] ** inv[Y] ** z -
  tp[x] ** X ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** x +
  tp[x] ** X ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** z +
  tp[x] ** inv[Y] ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** x -
  tp[x] ** inv[Y] ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** z -
  tp[x] ** tp[C2] ** C2 ** Y ** inv[-1 + X ** Y] ** X ** x +
  tp[x] ** tp[C2] ** C2 ** Y ** inv[-1 + X ** Y] ** X ** z +
  tp[x] ** tp[C2] ** C2 ** Y ** inv[-1 + X ** Y] ** inv[Y] ** x -
  tp[x] ** tp[C2] ** C2 ** Y ** inv[-1 + X ** Y] ** inv[Y] ** z +
  tp[z] ** X ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** x -
  tp[z] ** X ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** z -
  tp[z] ** inv[Y] ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** x +
  tp[z] ** inv[Y] ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** z +
  tp[z] ** tp[C2] ** C2 ** Y ** inv[-1 + X ** Y] ** X ** x -
  tp[z] ** tp[C2] ** C2 ** Y ** inv[-1 + X ** Y] ** X ** z -
tp[z] ** tp[C2] ** C2 ** Y ** inv[-1 + X ** Y] ** inv[Y] ** x +
tp[z] ** tp[C2] ** C2 ** Y ** inv[-1 + X ** Y] ** inv[Y] ** z +
tp[x] ** X ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** Y **
 inv[-1 + X ** Y] ** X ** x -
tp[x] ** X ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** Y **
 inv[-1 + X ** Y] ** X ** z -
tp[x] ** X ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** Y **
 inv[-1 + X ** Y] ** inv[Y] ** x +
tp[x] ** X ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** Y **
 inv[-1 + X ** Y] ** inv[Y] ** z -
tp[x] ** inv[Y] ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** Y **
 inv[-1 + X ** Y] ** X ** x +
tp[x] ** inv[Y] ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** Y **
 inv[-1 + X ** Y] ** X ** z +
tp[x] ** inv[Y] ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** Y **
 inv[-1 + X ** Y] ** inv[Y] ** x -
tp[x] ** inv[Y] ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** Y **
 inv[-1 + X ** Y] ** inv[Y] ** z -
tp[z] ** X ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** Y **
 inv[-1 + X ** Y] ** X ** x +
tp[z] ** X ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** Y **
 inv[-1 + X ** Y] ** X ** z +
tp[z] ** X ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** Y **
 inv[-1 + X ** Y] ** inv[Y] ** x -
tp[z] ** X ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** Y **
 inv[-1 + X ** Y] ** inv[Y] ** z +
tp[z] ** inv[Y] ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** Y **
 inv[-1 + X ** Y] ** X ** x -
tp[z] ** inv[Y] ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** Y **
 inv[-1 + X ** Y] ** X ** z -
tp[z] ** inv[Y] ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** Y **
 inv[-1 + X ** Y] ** inv[Y] ** x +
tp[z] ** inv[Y] ** inv[-1 + Y ** X] ** Y ** tp[C2] ** C2 ** Y **
 inv[-1 + X ** Y] ** inv[Y] ** z
```

Here, with the usual normalization,

(4.1)

$$\varepsilon(x, z) = \left(x^T \, z^T \right) \begin{pmatrix} Y^{-1} & -(Y^{-1} - X) \\ -(Y^{-1} - X) & Y^{-1} - X \end{pmatrix} \begin{pmatrix} x \\ z \end{pmatrix}$$

and we have taken relations called the Doyle Glover Kargonekar Francis simplifying assumptions which greatly reduce the complexity of the formulas.

Here we have used the same notation that one finds in our NCAlgebra program to give a feel for how this type of computer algebra goes. *tp* stands for transpose while *inv* stands for inverse and '**' for multiply.

The rules (RESOL) together with (EB) are stored in a function NCSimplifyRational in NCAlgebra which applies them repeatedly to an expression until no change occurs. We get the considerably simpler expression.

```
HYYI = NCSimplifyRational[H]=
 tp[x] ** X ** A ** z + tp[x] ** inv[Y] ** A ** x -
  tp[x] ** inv[Y] ** A ** z + tp[x] ** tp[A] ** X ** z +
  tp[x] ** tp[A] ** inv[Y] ** x - tp[x] ** tp[A] ** inv[Y] ** z +
  tp[x] ** tp[C1] ** C1 ** x - tp[x] ** tp[C2] ** C2 ** x +
  tp[x] ** tp[C2] ** C2 ** z + tp[z] ** X ** A ** x -
  tp[z] ** X ** A ** z - tp[z] ** inv[Y] ** A ** x +
  tp[z] ** inv[Y] ** A ** z + tp[z] ** tp[A] ** X ** x -
  tp[z] ** tp[A] ** X ** z - tp[z] ** tp[A] ** inv[Y] ** x +
  tp[z] ** tp[A] ** inv[Y] ** z + tp[z] ** tp[C2] ** C2 ** x -
  tp[z] ** tp[C2] ** C2 ** z + tp[x] ** X ** B1 ** tp[B1] ** X ** z -
  tp[x] ** X ** B2 ** tp[B2] ** X ** z +
  tp[x] ** inv[Y] ** B1 ** tp[B1] ** inv[Y] ** x -
  tp[x] ** inv[Y] ** B1 ** tp[B1] ** inv[Y] ** z +
  tp[z] ** X ** B1 ** tp[B1] ** X ** x -
  tp[z] ** X ** B1 ** tp[B1] ** X ** z -
  tp[z] ** X ** B2 ** tp[B2] ** X ** x +
  tp[z] ** X ** B2 ** tp[B2] ** X ** z -
  tp[z] ** inv[Y] ** B1 ** tp[B1] ** inv[Y] ** x +
  tp[z] ** inv[Y] ** B1 ** tp[B1] ** inv[Y] ** z
```

Notice that everything of the form inv[1-Y**X] and inv[1-X**Y] have been eliminated from H. This took 27 seconds on a SPARC II. In general we have found NCSR very effective in simplifying expressions with inverses of (very simple) polynomials.

This is still a bad expression, but we have not yet used the fact that X and Y solve the two famous **[DGKF]** Ricatti equations $DGKFX = 0$ and $DGKFYI = 0$ where

$DGKFX = XA{+}A^TX{+}C_1^TC_1{+}XB_1B_1^TX{-}XB_2B_2^TX$

$DGKFYI = Y^{-1}A{+}A^TY^{-1}{+}C_1^TC_1{-}C_2^TC_2{+}Y^{-1}B_1B_1^TY^{-1}$.

Actually this is the Ricatti for Y^{-1}. Y^{-1} is denoted YI in NCAlgebra.

Now we show the effect of our simplification methods. First set an ordering on the indeterminants in these expressions. Say $A < B_1 < B_2 < C_1 < C_2 < x < X < Y^{-1} < z < A^T < B_1^T < B_2^T < C_1^T < C_2^T < x^T < z^T$. This is done in NCAlgebra using the command

$$SetMonomialOrder[\{A, B1, B2, C1, C2, x, X, YI,$$

$$z, tp[A], tp[B1], tp[B2], tp[C1], tp[C2], tp[x], tp[z]\}] .$$

Now we invoke the NCAlgebra command

$$GroebnerSimplify[HYYI, \{DGKFX, DGKFYI\}] .$$

This command finds a Gröbner basis for the ideal generated by $DGKFX$ and $DGKFYI$ and reduces $HYYI$ by that Gröbner basis. The command took 116 seconds and reduced HYI to 0.

This completes the core of the sufficient side of the famous **[DGKF]** theorem. Namely, that the central controller has desired H^∞ performance level. It was done automatically with generic methods.

Note in this example we computed a finite GB for a collection of symbols (matrices) satisfying two Ricatti equations and for a particular order. The order was chosen by default in the middle of a computer session. Since it worked, we did not explore other orders.

Incidently, the Gröbner basis was the following four relations:

$$-XA - A^TX - {C_1}^TC_1 - XB_1{B_1}^TX + XB_2{B_2}^TX$$

$$Y^{-1}A + A^TY^{-1} + {C_1}^TC_1 - {C_2}^TC_2 + Y^{-1}B_1{B_1}^TY^{-1}$$

$$\begin{gathered} -XAA + A^TA^TX + A^T{C_1}^TC_1 - {C_1}^TC_1A \\ -XAB_1{B_1}^TX + XAB_2{B_2}^TX + \\ XB_1{B_1}^TA^TX + XB_1{B_1}^T{C_1}^TC_1 \\ -XB_2{B_2}^TA^TX - XB_2{B_2}^T{C_1}^TC_1 - \\ {C_1}^TC_1B_1{B_1}^TX + {C_1}^TC_1B_2{B_2}^TX \end{gathered}$$

$$\begin{gathered} -Y^{-1}AA + A^TA^TY^{-1} + A^T{C_1}^TC_1 - \\ A^T{C_2}^TC_2 - {C_1}^TC_1A + {C_2}^TC_2A \\ -Y^{-1}AB_1{B_1}^TY^{-1} + \\ Y^{-1}B_1{B_1}^TA^TY^{-1} + Y^{-1}B_1{B_1}^T{C_1}^TC_1 \\ -Y^{-1}B_1{B_1}^T{C_2}^TC_2 - {C_1}^TC_1B_1{B_1}^TY^{-1} + \\ {C_2}^TC_2B_1{B_1}^TY^{-1} \end{gathered}$$

REFERENCES

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

[HW] J. W. Helton and J. J. Wavrik "Rules for Computer Simplification of the formulas in operator model theory and linear systems", Operator Theory and Its Applications, to appear.

[HSW] J. W. Helton, M. Stankus and J. J. Wavrik "Proving the Gröbner Property for certain infinite sets.", in preperation.

[Mora] F. Mora, "Groebner Bases for Non-commutative Polynomial Rings" Lecture Notes in Computer Science, number 229 (1986) pp 353-362.

[NCA] J.W. Helton and R.L. Miller "NCAlgebra: A Mathematica Package for Doing Non Commuting Algebra" available from ncalg@ucsd.edu

[NF] B. Sz-Nagy and C. Foias, Harmonic Analysis of Operators on Hilbert Space North Holland 1970