Computer Science Journal of Moldova, vol.4, no.2(11), 1996

Rewrite Rules and Simplification of Matrix Expressions

John J Wavrik

Abstract

This paper concerns the automated simplification of expressions which involve non-commuting variables. The technology has been applied to the simplification of matrix and operator theory expressions which arise in engineering applications. The non-commutative variant of the Gröbner Basis Algorithm is used to generate rewrite rules. We will also look at the phenomenon of infinite bases and implications for automated theorem proving.

1 Introduction

This paper had its genesis in an attempt to simplify some large matrix expressions which had been produced by machine computation. Here is an expression for a Hamiltonian which arose in a computation in H^∞ Control Theory:

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
```

fig 1.1

The notation: ** is used for matrix multiplication, tp[] for transpose, inv[] for inverse. All other symbols are names for particular (but unspecified) matrices.

This expression can be seen to be a polynomial expression in a set of simpler expressions. Introduce the symbols

a = A	h = inv[-1+XY]	o = tp[C2]
b = B1	i = inv[-1+YX]	p = tp[x]
c = B2	j = inv[Y]	q = tp[z]
d = C1	k = tp[A]	r = x
e = C2	l = tp[B1]	s = z
f = X	m = tp[B2]	
g = Y	n = tp[C1]	

We find that H becomes

H = qjigoeghjs - qjigoeghjr - qjigoeghfs + qjigoeghfr - qfigoeghjs + qfigoeghjr + qfigoeghfs - qfigoeghfr - pjigoeghjs + pjigoeghjr + pjigoeghfs - pjigoeghfr + pfigoeghjs - pfigoeghjr - pfigoeghfs + pfigoeghfr + qoeghjs - qoeghjr - qoeghfs + qoeghfr + qjigoes - qjigoer - qfigoes + qfigoer - poeghjs + poeghjr + poeghfs - poeghfr - pjigoes + pjigoer + pfigoes - pfigoer + qjbljs - qjbljr + qfcmfs - qfcmfr - qfblfs + qfblfr - pjbljs + pjbljr - pfcmfs + pfblfs + qkjs - qkjr - qkfs + qkfr + qjas - qjar - qfas + qfar + pndr - pkjs + pkjr + pkfs - pjas + pjar + pfas

fig 1.2

This is a polynomial with 57 terms in 19 non-commuting variables. The largest term has 10 factors. In this paper we will consider matrix expressions which, like H, can be written as polynomials in a set of subexpressions (which we regard as "atomic"). A simplification of this expression, using the technology introduced in this paper, will be given in Section 3.

The use of rewrite rules to simplify expressions formalizes the process used to simplify these expressions by hand.

Example 1 *It is easy to see, for example, that* $1 - x^{-1}x + x^2$ *simplifies to* x^2*. Here we replace the subexpression* $x^{-1}x$ *by the equivalent, but simpler, expression* 1*. We have applied the rewrite rule* $x^{-1}x \to 1$*. This example uses an obvious rewrite rule which arises when a matrix is adjacent to its inverse in a term. Not all simplification rules are this obvious.*

Example 2 *The expression* $1 - x^{-1}(1 - xy)^{-1}x$ *simplifies to* $1 - (1 - yx)^{-1}$*. Here we use the rule* $(1 - xy)^{-1}x \to x(1 - yx)^{-1}$ *which is true although not obvious.*

An expert in this area has a repertoire of matrix identities which can be used for simplification. Expressions are scanned for subexpressions to which these rules may be applied. This is the essence of simplification by the use of rewrite rules. We would like to automate the process. If we have sufficiently many rules available, we might hope not only to produce simplified expressions but also to determine whether two matrix expressions are equivalent by reducing them to the same simplified form. We will also discuss simplification and equivalence.

2 Rewrite Rules and Simplification

Let E be a set (expressions) equipped with a partial ordering, $\geq$ (notion of simplicity), and an equivalence relation, $\sim$. A **reduction process** is a set of pairs $(f,g) \in E \times E$ with $f \geq g$ and $f \sim g$ in this case we write $f \rightarrow g$ and think of it as a process for getting from the expression f to an equivalent and simpler expression g. We may iterate this reduction process. Let $\rightarrow^*$ denote the reflexive, transitive closure of $\rightarrow$.

We will say that $g \in E$ is **irreducible** (or **reduced**) with respect to $\rightarrow$ if there is no h with $g \rightarrow h$. We will say that g is a normal form for f ($g = normal\_form(f)$) if $f \rightarrow^* g$ and g is irreducible.

In what follows, we will assume that $\rightarrow$ is **noetherian**: there are no infinite chains of the form $f_1 \rightarrow f_2 \rightarrow \ldots$. Thus any f can be reduced to a normal form by repeatedly applying $\rightarrow$. It should be noted that we do not assume that a given f is the source of only one arrow (and this is not the case in our application). There will usually be several ways in which f can be reduced to a normal form. The normal form is not, in general, unique. We say that $\rightarrow$ is **complete** if reduction to a normal form gives the same result no matter what sequence of reductions is used.

A **simplifier** is an effective procedure $\mathcal{S} : E \rightarrow E$ with the properties $e \geq \mathcal{S}(e)$ and $\mathcal{S}(e) \sim e \quad \forall e \in E$. Thus a simplifier is a process for replacing expressions by equivalent but simpler expressions. In the setting of a reduction process we may take $\mathcal{S}(e) = normal\_form(f)$. A **canonical simplifier** is an effective procedure $\mathcal{S} : E \rightarrow E$ with the properties $\mathcal{S}(e) \sim e$ and $e \sim f \iff \mathcal{S}(e) = \mathcal{S}(f) \quad \forall e, f \in E$. Thus $\mathcal{S}$ selects a unique representative, $\mathcal{S}(e)$ (called the **canonical form** of e) from the equivalence class of e.

If we have a reduction process, $\leftrightarrow^*$, the reflexive, symmetric, transitive closure of $\rightarrow$ is an equivalence relation on E. If we take the equivalence relation $\sim$ to be $\leftrightarrow^*$ then there are general criteria for the normal form simplifier $\mathcal{S}$ described above to be a canonical simplifier (or, equivalently, for $\rightarrow$ to be complete). Notation: $f \downarrow g$ will mean $\exists\, h$ with $f \rightarrow^* h$ and $g \rightarrow^* h$.

Proposition 1 *The following conditions are equivalent [see BuchLoos].*

(1) $\rightarrow$ is complete
(2) (Church-Rosser Property) $f \leftrightarrow^* g \quad \Leftrightarrow \quad f \downarrow g$
(3) (Confluence) $h \rightarrow^* f$ and $h \rightarrow^* g \;\Rightarrow\; f \downarrow g$
(4) (Local Confluence) $h \rightarrow f$ and $h \rightarrow g \;\Rightarrow\; f \downarrow g$

We will now specialize to the case in which E is $F\langle x_1, .., x_n\rangle$, the ring of polynomials, in a finite set of non-commuting variables over a field F (usually the rational numbers). We will see that, in this case, the test for local confluence can be restricted to a smaller (sometimes finite) collection of (f, g, h). We will also introduce a completion algorithm that attempts to extend a reduction process to a complete reduction (with the same equivalence relation).

The reduction process will be given by rewrite rules. A rewrite rule is a pair (LHS, RHS) where LHS is a monomial, RHS is a polynomial. A rewrite rule $r : LHS \rightarrow RHS$ is applied to a polynomial f by scanning the terms of f for an occurrence of LHS as a factor. If such a term is $c\mathcal{L}(LHS)\mathcal{R}$ (where c is the coefficient and $\mathcal{L}$ and $\mathcal{R}$ are monomials) we replace this term by $c\mathcal{L}(RHS)\mathcal{R}$ to obtain a polynomial g. Thus $g = f - c\mathcal{L}(LHS)\mathcal{R} + c\mathcal{L}(RHS)\mathcal{R}$. The notation for this is $f \rightarrow_r g$ and we will say that f reduces to g (by applying the rule r). A set, S, of rewrite rules defines a reduction process where $f \rightarrow_S g$ if $f \rightarrow_r g$ for some $r \in S$.

The following lemma is an adaptation of [BeckWeis Lemma 5.25].

Lemma 2 *(Translation Lemma)*
Let $f, g, h, h_1 \in E$, and let $\rightarrow$ denote a reduction process given by a set, S, of rewrite rules.

(i) If $f - g = h$ and $h \rightarrow^* h_1$ then $\exists f_1, g_1 \in E$ with $f_1 - g_1 = h_1,\ f \rightarrow^* f_1,\ g \rightarrow^* g_1$

(ii) If $f - g \rightarrow^* 0$, then $f \downarrow g$ and so, in particular, $f \leftrightarrow^* g$

Proof: (ii) follows from (i) with $h = 0$. We prove (i) by induction on the number of steps, k, in the reduction $h \rightarrow^* h_1$. If $k = 0$ then $h_1 = h$ and we may take $f_1 = f$ and $g_1 = g$. Suppose, then, that $h \rightarrow^* h_1$ takes $n + 1$ steps and that h_2 is the result of applying the first n steps. By induction, we have f_2 and g_2 so that $f \rightarrow^* f_2$, $g \rightarrow^* g_2$ and $f_2 - g_2 = h_2$. Moreover $h_2 \rightarrow h_1$ in one step. Thus h_2 has a term $c\mathcal{L}(LHS)\mathcal{R}$ corresponding to some rule in S and $h_1 = h_2 - c\mathcal{L}(LHS)\mathcal{R} + c\mathcal{L}(RHS)\mathcal{R}$ Let $c_1\mathcal{L}(LHS)\mathcal{R}$ be the term with monomial part $\mathcal{L}(LHS)\mathcal{R}$ in f_2 (take $c_1 = 0$ if there is no such term) and similarly $c_2\mathcal{L}(LHS)\mathcal{R}$ for g_2. Let f_1 and g_1 be the reductions obtained by replacing LHS by RHS in these terms. We then have $f_1 - g_1 = h_1$ as required.

A connection can be established between the ideal theory of the polynomial ring and a reduction process using rewrite rules. This connection depends on a choice of ordering for the terms of polynomials. A rewrite rule $LHS \rightarrow RHS$ corresponds to the polynomial $LHS - RHS$. In the other direction, a term ordering will give each polynomial, g, a leading term, $LT(g)$. We will set $LHS = LT(g)$ and $RHS = LT(g) - g$. This is made precise in the following way:

Let $\mathcal{W}$ be the set of words on the alphabet $x_1, .., x_n$. A **term ordering** is a total ordering on the words in $\mathcal{W}$ which satisfies

(1) $S \geq T \Rightarrow RS \geq RT$ and $SR \geq TR \quad \forall R, S, T \in \mathcal{W}$

(2) $\geq$ is a well-ordering (no strictly descending infinite chains)

(3) $S \geq 1 \quad \forall S \in \mathcal{W}$, where 1 is the empty word.

Graded lexicographic orderings satisfy these properties: these are orderings induced by choosing an ordering for the polynomial variables, and then defining an ordering on words by:

$T \geq S$ if and only if either $length(T) > length(S)$

or $length(T) = length(S)$ and T comes after S in a dictionary[1]

Example 3 *If $x_1 > x_2 > x_3$, the words of length 3 are ordered as follows:*

$x_1x_1x_1 > x_1x_1x_2 > x_1x_1x_3 > x_1x_2x_1 > x_1x_2x_2 > x_1x_2x_3 > x_1x_3x_1 >$
$x_1x_3x_2 > x_1x_3x_3 > x_2x_1x_1 > x_2x_1x_2 > x_2x_1x_3 > x_2x_2x_1 > x_2x_2x_2 >$
$x_2x_2x_3 > x_2x_3x_1 > x_2x_3x_2 > x_2x_3x_3 > x_3x_1x_1 > x_3x_1x_2 > x_3x_1x_3 >$
$x_3x_2x_1 > x_3x_2x_2 > x_3x_2x_3 > x_3x_3x_1 > x_3x_3x_2 > x_3x_3x_3$

Once an ordering is chosen, every non-zero polynomial, $f \in E$, can be uniquely written as a finite sum $f = \sum_1^k c_i W_i$ where $W_1 > W_2 > \ldots > W_k$. $c_1 W_1$ will be called the **leading term** of f and denoted $LT(f)$; c_1 is the **leading coefficient**, denoted $LC(f)$, and W_1 will be called the **leading monomial** and denoted $LM(f)$.

Once a term ordering is chosen, a set, $\mathcal{G}$, of polynomials gives rise to a set of rewrite rules, and hence a reduction process. The requirement that the term ordering be a well-ordering makes this a noetherian reduction process. We may now connect the reduction process with the ideal $\mathcal{I} = \langle \mathcal{G} \rangle$ generated by the set $\mathcal{G}$. It is quite clear that $f \rightarrow g \Rightarrow f - g \in \mathcal{I}$ (i.e. $f \equiv g \ mod\ \mathcal{I}$). It is therefore clear that if $f \rightarrow^* 0$ then $f \in \mathcal{I}$. In general, the converse will not hold. A set $\mathcal{G}$ for which the converse holds (i.e. for which $f \in \mathcal{I} \Rightarrow f \rightarrow^* 0$) is called a **Gröbner Basis** for $\mathcal{I}$.

Lemma 3 *([BeckWeis Lemma 5.26])*
Let $f, g \in E$, $\rightarrow$ the reduction process defined by $\mathcal{G} \subseteq E$, and $\mathcal{I} = \langle \mathcal{G} \rangle$ the ideal generated by $\mathcal{G}$. Then $f \equiv g \ mod\ I \Leftrightarrow f \leftrightarrow^* g$.

Example 4 *Consider the set $\mathcal{G} = \{ca - c, ab - a\}$. The polynomial cab can be reduced in two ways, depending upon which subexpression is*

[1] Pure dictionary (lexicographic) order, $\gg$, is not a well-ordering because xy $\gg$ xxy $\gg$ xxxy $\gg \ldots$ is an infinite descending chain.

replaced first. We have $cab = c(ab) \to ca \to c$ *and we also have* $cab = (ca)b \to cb$*. Both* cb *and* c *are irreducible, so are normal forms for* cab*. Thus the reduction process defined by* $\mathcal{G}$ *is not complete. Notice, that, indeed, this is a failure of local confluence: We have* $cb \leftarrow cab \to c$ *but we do not have* $cb \downarrow c$*. The source of the problem should be clear:* cb *and* c *are equivalent with respect to the reduction process, but we do not have enough rules to reduce them to the same (canonical) form. The solution is to extend the set* $\mathcal{G}$ *to include more rules, without changing the notion of equivalence. In fact, as our later discussion will show,* $\mathcal{G}^* = \mathcal{G} \cup \{cb - c\}$ *does give a complete set of reduction rules. It is certainly obvious that, in this example, adding* $cb - c$ *to the basis takes care of the fact that* cab *does not have a unique normal form. It is not obvious that adding just this polynomial will assure that every polynomial has a canonical form. An algorithm for extending a basis will be discussed in the next section.*

2.1 The Basis Algorithm

We are given a basis, $\mathcal{G}$, for an ideal. The idea behind a completeness algorithm is to find situations in which local confluency might fail and add new rules to take care of such situations. The local confluence criterion requires that we examine all cases in which a polynomial h reduces in two ways. We show that it is enough to examine h which arise in a specific way from pairs of basis elements. In the case of polynomials in commuting variables [Buch], it is enough to check, local confluence in the case in which $f, g \in \mathcal{G}$ and where $h = lcm(LM(f), LM(g))$. We have two obvious reductions: $h \to_f h_1$ and $h \to_g h_2$. If h_1 and h_2 do not reduce to a common normal form (so local confluence fails in this case), then $normal\_form(h_1 - h_2, \mathcal{G})$ is added to the basis. $\mathcal{G}$ is a Gröbner Basis if $normal\_form(h_1 - h_2, \mathcal{G}) = 0$ for all f and g. Something similar holds in the non-commutative case as well: the local confluence needs only to be checked for several possible minimal multiples, h, of $LT(f)$ and $LT(g)$.

Definition 1 *We say that two words, S and T,* **overlap** *if one of the following conditions holds. In each case we indicate a non-trivial common multiple, M, of S and T obtained from the overlap.*

(1)	one of the words is a proper subword of the other	
	If $S = lTr$	Set $M = S$
	If $T = lSr$	Set $M = T$
(2)	$S = lw, T = wr$ for some $w \neq 1$.	Set $M = Sr = lT$
(3)	$S = wr, T = lw$ for some $w \neq 1$	Set $M = lS = Tr$.

Here is a picture of an overlap:

$$\begin{array}{c} \quad\mathbf{xy}xx \\ y\mathbf{xy}\quad \end{array}$$

Example 5 *The words xy and xz have no overlap. The words xyy and yyx have two overlaps. The words $yxyy$ and $yyxy$ have 4 overlaps.*

Definition 2 *A* **match** *for $f, g \in \mathcal{G}$ is a sextuple $(f, g, l_1, r_1, l_2, r_2)$ where $S = LM(f)$ and $T = LM(g)$ overlap and where $M = l_1 S r_1 = l_2 T r_2$ is the common multiple determined by the overlap as in the definition. We denote the set of matches for f, g by* $\mathbf{Matches(f,g)}$. *This set may be empty.*

Definition 3 *If $m = (f, g, l_1, r_1, l_2, r_2) \in Matches(f, g)$ the* **S-polynomial of m** *is $Spol(m) = l_1 f r_1 - l_2 g r_2$*

Theorem 4 *(S-polynomial Criterion - see [FMora])*
A set, $\mathcal{G}$, of polynomials is a Gröbner Basis for the ideal it generates if $\forall f, g \in \mathcal{G}$, $\forall m \in Matches(f, g)$ we have $normal\_form(Spol(m), \mathcal{G}) = 0$ (we can take any normal form – but in this case it can be shown that all normal forms are the same).

Remark 1 *As pointed out above, this is the test for local confluence applied to a restricted set of triples.*

We now give a sketch of a simplifed version of the Gröbner Basis Algorithm. (A more detailed version is found in [FMora])

Procedure: Make-Basis

$NEW := \mathcal{G}$
While $NEW \neq \emptyset$
    $\mathcal{G} := \mathcal{G} \cup NEW$
    Fill_$WAITING$
    $NEW := \emptyset$
    While $WAITING \neq \emptyset$
        Choose $m \in WAITING$
        $WAITING := WAITING - m$
        $h := normal\_form(Spol(m), \mathcal{G} \cup NEW)$
        If $h \neq 0$ then $NEW := NEW \cup \{h\}$

Procedure: Fill_WAITING

$WAITING := \emptyset$
$\forall g \in \mathcal{G}, n \in NEW$
    $WAITING := WAITING \cup Matches(g, n)$

The algorithm makes use of two auxiliary sets. The set WAITING holds the matches yet to be processed while the set NEW holds new polynomials to be added to the basis. It is essential for the correctness of the algorithm, particularly since it may not terminate, to ensure that every match between basis elements would eventually be processed if the program ran indefinitely. Notice that the mechanism of WAITING and NEW delays the addition of matches involving newly produced basis elements until the previous batch of matches has been processed, and then only matches are added which involve a new basis element. This simplified version does not specify the order in which matches are selected for processing; the sequence in which existing rules are applied; or mechanisms for detecting matches which can, a priori, be eliminated from consideration. It also does not reduce existing elements of $\mathcal{G}$ by newly produced elements. These are factors which can affect the execution time of the algorithm. In practice we add a final step, reducing every $g \in \mathcal{G}$ by $\mathcal{G} - \{g\}$ to produce a *reduced* Gröbner Basis.[2]

The algorithm adds elements to the set $\mathcal{G}$. If this algorithm terminates, it must be because the S-polynomials for all matches of pairs of elements in $\mathcal{G}$ reduce to 0. The S-polynomial criterion shows that $\mathcal{G}$ is a Grobner Basis in this case. In the case of polynomials in *commuting* variables, the algorithm always does terminate and so always produces a Gröbner Basis.

In the *non-commutative* case the algorithm may fail to terminate (some examples are found in the next section). In this case, a Gröbner Basis can often still be determined by analysing the elements being added to $\mathcal{G}$. If patterns are found, it may be possible to conjecture the eventual output if the algorithm ran indefinitely. The S-Polynomial criterion can then be applied to provide a proof that the conjecture is correct. The algorithm should be used in an environment in which the output being generated can be observed, and the algorithm halted (yielding the current $\mathcal{G}$) when desired.

There are many formulations of the criterion and algorithm (see [Ufn] for a discussion and further references)

3 Examples From Operator Theory

The matrix or operator expressions which occur in this section are polynomials in a given set of "**atomic expressions**". A collection of atomic expressions with an imposed order will be called a **model**. In our first model, we look at matrix expressions which are polynomials in the atomic expressions x, x^{-1}, $(1-x)^{-1}$. The atomic expressions are treated as variables in a polynomial ring. The starting basis for the Basis Algorithm consists of some obvious **relations** among these variables: i.e. polynomials which become 0 if we substitute for x any matrix or operator for which x and $(1-x)$ are invertible. All of the polynomials obtained by the Gröbner Basis Algorithm are in the ideal

[2]We have found it preferable (to facilitate analysis of the algorithm) to leave this step for last rather than make changes in existing basis elements as the program executes.

generated by the starting relations, and so also become 0 under any meaningful substitution.

Model A

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

Comment This is the simplest example for which the Gröbner Algorithm produces new relations. This is called RESOL in [HWS] because operator theorists refer to $(1-x)^{-1}$ as the resolvent of x. The relations A_0 and A_1 are called the **defining relations** for x^{-1} since they define what is meant by inverse (similarly A_2 and A_3 are the defining relations for $(1-x)^{-1}$).

Starting Relations

$$\begin{aligned}
A_0 &= +x^{-1}x - 1 \\
A_1 &= +xx^{-1} - 1 \\
A_2 &= +(1-x)^{-1}x - (1-x)^{-1} + 1 \\
A_3 &= +x(1-x)^{-1} - (1-x)^{-1} + 1
\end{aligned}$$

Ending Relations

$$\begin{aligned}
A_0 &= +x^{-1}x - 1 \\
A_1 &= +xx^{-1} - 1 \\
A_2 &= +(1-x)^{-1}x - (1-x)^{-1} + 1 \\
A_3 &= +x(1-x)^{-1} - (1-x)^{-1} + 1 \\
A_4 &= +(1-x)^{-1}x^{-1} - (1-x)^{-1} - x^{-1} \\
A_5 &= +x^{-1}(1-x)^{-1} - (1-x)^{-1} - x^{-1}
\end{aligned}$$

Observations The two new relations, A_4 and A_5 are known to operator theorists as the Resolvent Identities. The fact that these identities were produced by the Gröbner Algorithm means that they are in the ideal generated by the starting relations. One can easily trace the Algorithm to derive these new relations directly from the starting relations:

$$A_4 = (1-x)^{-1}A_1 - A_2\,x^{-1}$$
$$A_5 = A_0(1-x)^{-1} - x^{-1}A_3$$

Model B

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

Comment This is Model A applied to two different letters.

Starting Relations

$$\begin{aligned}
B_0 &= +x^{-1}x - 1\\
B_1 &= +xx^{-1} - 1\\
B_2 &= +y^{-1}y - 1\\
B_3 &= +yy^{-1} - 1\\
B_4 &= +(1-x)^{-1}x - (1-x)^{-1} + 1\\
B_5 &= +x(1-x)^{-1} - (1-x)^{-1} + 1\\
B_6 &= +(1-y)^{-1}y - (1-y)^{-1} + 1\\
B_7 &= +y(1-y)^{-1} - (1-y)^{-1} + 1
\end{aligned}$$

Ending Relations

$$\begin{aligned}
B_0 &= +x^{-1}x - 1\\
B_1 &= +xx^{-1} - 1\\
B_2 &= +y^{-1}y - 1\\
B_3 &= +yy^{-1} - 1\\
B_4 &= +(1-x)^{-1}x - (1-x)^{-1} + 1\\
B_5 &= +x(1-x)^{-1} - (1-x)^{-1} + 1\\
B_6 &= +(1-y)^{-1}y - (1-y)^{-1} + 1\\
B_7 &= +y(1-y)^{-1} - (1-y)^{-1} + 1\\
B_8 &= +(1-x)^{-1}x^{-1} - (1-x)^{-1} - x^{-1}\\
B_9 &= +(1-y)^{-1}y^{-1} - (1-y)^{-1} - y^{-1}\\
B_{10} &= +x^{-1}(1-x)^{-1} - (1-x)^{-1} - x^{-1}\\
B_{11} &= +y^{-1}(1-y)^{-1} - (1-y)^{-1} - y^{-1}
\end{aligned}$$

Observations Notice that the starting relations are the disjoint union of the relations for x and for y. The ending relations are also the disjoint union of the separate sets of ending relations. We can see that this is true in general:

Theorem 5 *Let $F\langle x, y\rangle$ be a polynomial ring in two sets of non-commuting variables. Let $S \subset F\langle x\rangle$ and $T \subset F\langle y\rangle$ be two finite sets. Let S^* and T^* be the output sets of Basis Algorithm applied separately to S and T. Then $(S \cup T)^* = S^* \cup T^*$. If S^* and T^* satisfy the Gröbner property, then so does $S^* \cup T^*$.*

Proof. New basis elements are added by reducing S-Polynomials of matches. The formation of a match between two polynomials requires an overlap of the leading monomials. If the polynomials are in disjoint variables, no match can occur. When a match does occur (between two polynomials in the same set of variables) reductions only occur using polynomials in the same set of variables. The Gröbner Property can be verified by applying the S-Polynomial Criterion. Here again, matches and reductions will occur among polynomials in the same set of variables.

Model C

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

Comment This is called EB in [HWS] because expressions involving these atomic expressions underlie energy balance equations in H^∞-Control. The starting relations are the defining relations for the 4 inverses.

Starting Relations

$$\begin{aligned}
C_0 &= +x^{-1}x - 1 \\
C_1 &= +xx^{-1} - 1 \\
C_2 &= +y^{-1}y - 1 \\
C_3 &= +yy^{-1} - 1 \\
C_4 &= +(1-xy)^{-1}xy - (1-xy)^{-1} + 1 \\
C_5 &= +xy(1-xy)^{-1} - (1-xy)^{-1} + 1 \\
C_6 &= +(1-yx)^{-1}yx - (1-yx)^{-1} + 1 \\
C_7 &= +yx(1-yx)^{-1} - (1-yx)^{-1} + 1
\end{aligned}$$

The ending relations are found on the next page. Notice that the algorithm terminates and so the ending relations are a Gröbner Basis. C_4 and C_6, which appear in the starting relations above, do not appear in the ending relations. They are reduced to zero by the others. Here, as in other reports of output, we have preserved the numbering of the starting relations in the list of ending relations. Relations which are reduced to zero are left out of the list, but the numbers are not changed. The "quasi-commutativity" relations C_{12} and C_{13} are very important in later examples. Notice that they are a consequence of the starting relations in this model:

$$\begin{aligned}
C_{12} &= (1-yx)^{-1}y\,C_5 - C_6\,y(1-xy)^{-1} \\
C_{13} &= (1-xy)^{-1}x\,C_7 - C_4\,x(1-yx)^{-1}
\end{aligned}$$

Ending Relations

$$
\begin{aligned}
C_0 &= +x^{-1}x - 1\\
C_1 &= +xx^{-1} - 1\\
C_2 &= +y^{-1}y - 1\\
C_3 &= +yy^{-1} - 1\\
C_5 &= +xy(1-xy)^{-1} - (1-xy)^{-1} + 1\\
C_7 &= +yx(1-yx)^{-1} - (1-yx)^{-1} + 1\\
C_8 &= +(1-yx)^{-1}x^{-1} - y(1-xy)^{-1} - x^{-1}\\
C_9 &= +(1-xy)^{-1}y^{-1} - x(1-yx)^{-1} - y^{-1}\\
C_{10} &= +x^{-1}(1-xy)^{-1} - y(1-xy)^{-1} - x^{-1}\\
C_{11} &= +y^{-1}(1-yx)^{-1} - x(1-yx)^{-1} - y^{-1}\\
C_{12} &= +(1-yx)^{-1}y - y(1-xy)^{-1}\\
C_{13} &= +(1-xy)^{-1}x - x(1-yx)^{-1}
\end{aligned}
$$

The large expression which started this paper (Fig 1.1) contains only a few variables for which there are simplifying relations. The other variables are disjoint. We may simplify this expression using just the relations found in this model. The starting relations have no effect, so the expression cannot be simplified by locating expressions adjacent to inverses. When we apply the ending relations we obtain

H = +qjbljs-qjbljr+qfcmfs-qfcmfr-qfblfs+qfblfr-pjbljs
+pjbljr-pfcmfs+pfblfs-qoes+qoer+qkjs-qkjr-qkfs
+qkfr+qjas-qjar-qfas+qfar+poes-poer+pndr-pkjs
+pkjr+pkfs-pjas+pjar+pfas

Which, while still complicated, now has only 29 terms (rather than 57) the largest of which has only 6 factors (rather than 10). The definition of simplicity we are using is based primarily on the number of factors in a term. We expect reduction to lower the number of factors. On the other hand, we are using a Gröbner Basis for the reduction – so that equivalent terms reduce to the same canonical form. In this case, the reduction process discloses terms which can be combined or cancelled – which explains why, in practice, there is a decrease in the number of terms.

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

Comment This is the first example in which the Gröbner Basis is infinite. It is called preNF in [HW] because it is a preliminary to the Nagy-Foias model. Notice that D_{12} and D_{13} are actually generated from the other starting relations (and they appeared in Model C). It was found, however, that they occur fairly late in the output and that they are very effective in reducing other relations. So they are included in the starting relations for the sake of efficiency. The starting relations are the union of those for Models B and C.

Starting Relations

$$
\begin{aligned}
D_0 &= +x^{-1}x - 1\\
D_1 &= +xx^{-1} - 1\\
D_2 &= +y^{-1}y - 1\\
D_3 &= +yy^{-1} - 1\\
D_4 &= +(1-x)^{-1}x - (1-x)^{-1} + 1\\
D_5 &= +x(1-x)^{-1} - (1-x)^{-1} + 1\\
D_6 &= +(1-y)^{-1}y - (1-y)^{-1} + 1\\
D_7 &= +y(1-y)^{-1} - (1-y)^{-1} + 1\\
D_8 &= +(1-xy)^{-1}xy - (1-xy)^{-1} + 1\\
D_9 &= +xy(1-xy)^{-1} - (1-xy)^{-1} + 1\\
D_{10} &= +(1-yx)^{-1}yx - (1-yx)^{-1} + 1\\
D_{11} &= +yx(1-yx)^{-1} - (1-yx)^{-1} + 1\\
D_{12} &= +(1-xy)^{-1}x - x(1-yx)^{-1}\\
D_{13} &= +(1-yx)^{-1}y - y(1-xy)^{-1}
\end{aligned}
$$

The ending relations for this model are found on the next page. In this case, the output of the Basis Algorithm ultimately produces instances of 8 parametrized families of relations together with a collection of "special" relations which are part of the basis but are not instances of the parametrized families. The S-Polynomial criterion can be used to show that this is indeed a Gröbner Basis.

This model results from combining Model B and Model C. These two models have finite Gröbner bases. All of of the infinite families, and the special relations D_SPC_1 and D_SPC_2 result from interaction between the two sets of relations. D_SPC_1, for example, results from a match between B_6 and C_7. The family $D\_I[n]$ starts from a match of B_5 and C_{13} (the later members of this family are obtained recursively by matches of their predecessor with C_{13}). In section 4 we will take a closer look at some of the ways that infinite bases arise.

Ending Relations – Special Relations

$$
\begin{aligned}
D\_SPC_1 &= +(1-y)^{-1}x(1-yx)^{-1}-(1-y)^{-1}(1-yx)^{-1} \\
&\quad -x(1-yx)^{-1}+(1-y)^{-1} \\
D\_SPC_2 &= +(1-x)^{-1}y(1-xy)^{-1}-(1-x)^{-1}(1-xy)^{-1} \\
&\quad -y(1-xy)^{-1}+(1-x)^{-1} \\
D\_SPC_3 &= +yx(1-yx)^{-1}-(1-yx)^{-1}+1 \\
D\_SPC_4 &= +xy(1-xy)^{-1}-(1-xy)^{-1}+1 \\
D\_SPC_5 &= +(1-yx)^{-1}x^{-1}-y(1-xy)^{-1}-x^{-1} \\
D\_SPC_6 &= +(1-yx)^{-1}y-y(1-xy)^{-1} \\
D\_SPC_7 &= +(1-xy)^{-1}y^{-1}-x(1-yx)^{-1}-y^{-1} \\
D\_SPC_8 &= +(1-xy)^{-1}x-x(1-yx)^{-1} \\
D\_SPC_9 &= +(1-y)^{-1}y^{-1}-(1-y)^{-1}-y^{-1} \\
D\_SPC_{10} &= +(1-y)^{-1}y-(1-y)^{-1}+1 \\
D\_SPC_{11} &= +(1-x)^{-1}x^{-1}-(1-x)^{-1}-x^{-1} \\
D\_SPC_{12} &= +(1-x)^{-1}x-(1-x)^{-1}+1 \\
D\_SPC_{13} &= +y^{-1}(1-yx)^{-1}-x(1-yx)^{-1}-y^{-1} \\
D\_SPC_{14} &= +y^{-1}(1-y)^{-1}-(1-y)^{-1}-y^{-1} \\
D\_SPC_{15} &= +y^{-1}y-1 \\
D\_SPC_{16} &= +x^{-1}(1-xy)^{-1}-y(1-xy)^{-1}-x^{-1} \\
D\_SPC_{17} &= +x^{-1}(1-x)^{-1}-(1-x)^{-1}-x^{-1} \\
D\_SPC_{18} &= +x^{-1}x-1 \\
D\_SPC_{19} &= +y(1-y)^{-1}-(1-y)^{-1}+1 \\
D\_SPC_{20} &= +yy^{-1}-1 \\
D\_SPC_{21} &= +x(1-x)^{-1}-(1-x)^{-1}+1 \\
D\_SPC_{22} &= +xx^{-1}-1
\end{aligned}
$$

The special relations are listed in decreasing order of their leading term. It should be noted that, with the exception of D_SPC_1 and D_SPC_2, each of the special relations occurs in the Gröbner bases for Models B and C.

Ending Relations – Infinite Families

$$
\begin{aligned}
D\_I[n] &= x(1-yx)^{-n}(1-x)^{-1} - (1-xy)^{-n}(1-x)^{-1} \\
&\quad +(1-xy)^{-n} \\
D\_II[n] &= x(1-yx)^{-n}(1-y)^{-1} - (1-xy)^{-n}(1-y)^{-1} \\
&\quad -x(1-yx)^{-n} + (1-xy)^{-(n-1)}(1-y)^{-1} \\
D\_III[n] &= y(1-xy)^{-n}(1-x)^{-1} - (1-yx)^{-n}(1-x)^{-1} \\
&\quad -y(1-xy)^{-n} + (1-yx)^{-(n-1)}(1-x)^{-1} \\
D\_IV[n] &= y(1-xy)^{-n}(1-y)^{-1} - (1-yx)^{-n}(1-y)^{-1} \\
&\quad +(1-yx)^{-n} \\
D\_V[n] &= (1-x)^{-1}(1-yx)^{-n}(1-x)^{-1} - (1-x)^{-1}(1-xy)^{-n} \\
&\quad (1-x)^{-1} - (1-yx)^{-n}(1-x)^{-1} + (1-x)^{-1} \\
&\quad (1-xy)^{-n} \\
D\_VI[n] &= (1-x)^{-1}(1-yx)^{-n}(1-y)^{-1} - (1-x)^{-1} \\
&\quad (1-xy)^{-n}(1-y)^{-1} - (1-yx)^{-n}(1-y)^{-1} \\
&\quad -(1-x)^{-1}(1-yx)^{-n} + (1-x)^{-1}(1-xy)^{-(n-1)} \\
&\quad (1-y)^{-1} + (1-yx)^{-n} \\
D\_VII[n] &= (1-y)^{-1}(1-yx)^{-n}(1-x)^{-1} - (1-y)^{-1}\Sigma(1-x)^{-1} \\
&\quad +\Sigma(1-x)^{-1} + (1-y)^{-1}\Sigma - \Sigma - (1-y)^{-1}(1-x)^{-1} \\
D\_VIII[n] &= (1-y)^{-1}(1-yx)^{-n}(1-y)^{-1} - (1-y)^{-1} \\
&\quad (1-xy)^{-n}(1-y)^{-1} + (1-xy)^{-n}(1-y)^{-1} \\
&\quad -(1-y)^{-1}(1-yx)^{-n}
\end{aligned}
$$

Where $\Sigma = \sum_{k=0}^{n}(1-xy)^{-k}$

Model E

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

Comment We have in mind the situation in which x and y are operators on a Hilbert space with x, y, $1-x$ and $1-y$ invertible, and h

a function analytic on the spectrum of xy and yx. The relations E_8 and E_9 can be shown to hold under these conditions. This example is the genesis of the GENR rules in [HWS]. We again obtain an infinite basis.

The most interesting aspect of this model is that it is a collection of models: for various choices of h. Examples of h which satisfy the conditions are $h(s) = (1-s)^{-1}$, $h(s) = \sqrt{1-s}$, $h(s) = (1-s)^{-n}$, $h(s) = e^s$. Thus the simplification rules we obtain hold very generally. It should not be expected, for a particular h, that we find all relations for this h by specializing this general set of rules. In particular, for $h(s) = (1-s)^{-1}$ we have the same atomic expressions as in Model D, but the starting relations here do not include the defining relations for $(1-xy)^{-1}$ and $(1-yx)^{-1}$ (D_{10} to D_{13}).

Starting Relations

$$
\begin{aligned}
E_0 &= +x^{-1}x - 1 \\
E_1 &= +xx^{-1} - 1 \\
E_2 &= +y^{-1}y - 1 \\
E_3 &= +yy^{-1} - 1 \\
E_4 &= +(1-x)^{-1}x - (1-x)^{-1} + 1 \\
E_5 &= +x(1-x)^{-1} - (1-x)^{-1} + 1 \\
E_6 &= +(1-y)^{-1}y - (1-y)^{-1} + 1 \\
E_7 &= +y(1-y)^{-1} - (1-y)^{-1} + 1 \\
E_8 &= +h(xy)x - xh(yx) \\
E_9 &= +h(yx)y - yh(xy)
\end{aligned}
$$

Ending Relations - Infinite Families

$$
\begin{aligned}
E\_I[n] &= +xh(yx)^n(1-x)^{-1} - h(xy)^n(1-x)^{-1} + h(xy)^n \\
E\_II[n] &= +yh(xy)^n(1-y)^{-1} - h(yx)^n(1-y)^{-1} + h(yx)^n \\
E\_III[n] &= +x^{-1}h(xy)^n(1-x)^{-1} - h(yx)^n(1-x)^{-1} \\
& \quad -x^{-1}h(xy)^n \\
E\_IV[n] &= +y^{-1}h(yx)^n(1-y)^{-1} - h(xy)^n(1-y)^{-1} \\
& \quad -y^{-1}h(yx)^n \\
E\_V[n] &= +(1-x)^{-1}h(yx)^n(1-x)^{-1} - (1-x)^{-1}h(xy)^n \\
& \quad (1-x)^{-1} - h(yx)^n(1-x)^{-1} + (1-x)^{-1}h(xy)^n \\
E\_VI[n] &= +(1-y)^{-1}h(yx)^n(1-y)^{-1} - (1-y)^{-1}h(xy)^n \\
& \quad (1-y)^{-1} + h(xy)^n(1-y)^{-1} - (1-y)^{-1}h(yx)^n
\end{aligned}
$$

These hold for $n = 0, \ldots$

Ending Relations - Special Relations

$$
\begin{aligned}
E\_SPC_1 &= +h(yx)x^{-1} - x^{-1}h(xy) \\
E\_SPC_2 &= +h(yx)y - yh(xy) \\
E\_SPC_3 &= +h(xy)y^{-1} - y^{-1}h(yx) \\
E\_SPC_4 &= +h(xy)x - xh(yx) \\
E\_SPC_5 &= +(1-y)^{-1}y^{-1} - (1-y)^{-1} - y^{-1} \\
E\_SPC_6 &= +(1-y)^{-1}y - (1-y)^{-1} + 1 \\
E\_SPC_7 &= +(1-x)^{-1}x^{-1} - (1-x)^{-1} - x^{-1} \\
E\_SPC_8 &= +(1-x)^{-1}x - (1-x)^{-1} + 1 \\
E\_SPC_9 &= +y^{-1}y - 1 \\
E\_SPC_{10} &= +x^{-1}x - 1 \\
E\_SPC_{11} &= +yy^{-1} - 1 \\
E\_SPC_{12} &= +xx^{-1} - 1
\end{aligned}
$$

Notice that the starting relations hold if we substitute $h(xy)^n$ for $h(xy)$ and $h(yx)^n$ for $h(yx)$. As a result, the same must be true of the ending relations. In particular, all of the infinite families are obtained by making this substitution in the relations for $n = 1$. Notice also that the rules are symmetrical upon interchanging x with y (and $h(xy)$ with $h(yx)$) The basis can be completely described by a finite set of relations, called $GENR$ in [HWS] (which also contains an interpretation in terms of the functional calculus in operator theory).

GENR RELATIONS

1 The relations B_0 .. B_{11}

2 The relations

$$\begin{aligned}
GENR_0 &= +h(xy)x - xh(yx)\\
GENR_1 &= +h(yx)x^{-1} - x^{-1}h(xy)\\
GENR_2 &= +xh(yx)(1-x)^{-1} - h(xy)(1-x)^{-1} + h(xy)\\
GENR_3 &= +x^{-1}h(xy)(1-x)^{-1} - h(yx)(1-x)^{-1} - x^{-1}h(xy)\\
GENR_4 &= +(1-x)^{-1}h(yx)(1-x)^{-1} - (1-x)^{-1}h(xy)\\
&\quad (1-x)^{-1} - h(yx)(1-x)^{-1} + (1-x)^{-1}h(xy)
\end{aligned}$$

3 The relations

$GENR_0$ – $GENR_4$ with x and y interchanged

Model F

$$\begin{aligned}
&x < y < x^{-1} < y^{-1} < (1-x)^{-1} < (1-y)^{-1} < (1-xy)^{-1}\\
&\quad < (1-yx)^{-1} < (1-xy)^{1/2} < (1-yx)^{1/2}
\end{aligned}$$

Comment This is called NF in [HW] because expressions of this type arise in the Nagy-Foias Model in operator theory. It might be useful to remind the reader that, although the notation (intentionally) suggests an analytic interpretation, $(1-xy)^{1/2}$ is just the name for a variable in a polynomial ring. The fact that we intend to substitute a matrix expression involving a square root for this variable is indicated by the fact that that we take F_{12} as one of the starting relations.

As we observed in Model D, the relations F_{14} and F_{15} are actually algebraic consequences of the other starting relations. They are included for reasons of efficiency. The relations F_{16} and F_{17}, however, are not in the ideal generated by the other relations.

They can be easily shown to hold by an analytic argument (see Model E).

Starting Relations

$$
\begin{aligned}
F_0 &= +x^{-1}x - 1 \\
F_1 &= +xx^{-1} - 1 \\
F_2 &= +y^{-1}y - 1 \\
F_3 &= +yy^{-1} - 1 \\
F_4 &= +(1-x)^{-1}x - (1-x)^{-1} + 1 \\
F_5 &= +x(1-x)^{-1} - (1-x)^{-1} + 1 \\
F_6 &= +(1-y)^{-1}y - (1-y)^{-1} + 1 \\
F_7 &= +y(1-y)^{-1} - (1-y)^{-1} + 1 \\
F_8 &= +(1-xy)^{-1}xy - (1-xy)^{-1} + 1 \\
F_9 &= +xy(1-xy)^{-1} - (1-xy)^{-1} + 1 \\
F_{10} &= +(1-yx)^{-1}yx - (1-yx)^{-1} + 1 \\
F_{11} &= +yx(1-yx)^{-1} - (1-yx)^{-1} + 1 \\
F_{12} &= +(1-xy)^{1/2}(1-xy)^{1/2} + xy - 1 \\
F_{13} &= +(1-yx)^{1/2}(1-yx)^{1/2} + yx - 1 \\
F_{14} &= +(1-xy)^{-1}x - x(1-yx)^{-1} \\
F_{15} &= +(1-yx)^{-1}y - y(1-xy)^{-1} \\
F_{16} &= +(1-xy)^{1/2}x - x(1-yx)^{1/2} \\
F_{17} &= +(1-yx)^{1/2}y - y(1-xy)^{1/2}
\end{aligned}
$$

The special relations from Model D together with:

Ending Relations – Special Relations

$$
\begin{aligned}
F\_SPEC_0 &= +y^{-1}(1-yx)^{1/2}(1-y)^{-1}-(1-xy)^{1/2} \\
&\quad (1-y)^{-1}-y^{-1}(1-yx)^{1/2} \\
F\_SPEC_1 &= +x^{-1}(1-xy)^{1/2}(1-x)^{-1}-(1-yx)^{1/2} \\
&\quad (1-x)^{-1}-x^{-1}(1-xy)^{1/2} \\
F\_SPEC_2 &= +(1-yx)^{1/2}(1-yx)^{1/2}+yx-1 \\
F\_SPEC_3 &= +(1-yx)^{1/2}(1-yx)^{-1}-(1-yx)^{-1}(1-yx)^{1/2} \\
F\_SPEC_4 &= +(1-yx)^{1/2}x^{-1}-x^{-1}(1-xy)^{1/2} \\
F\_SPEC_5 &= +(1-yx)^{1/2}y-y(1-xy)^{1/2} \\
F\_SPEC_6 &= +(1-xy)^{1/2}(1-xy)^{1/2}+xy-1 \\
F\_SPEC_7 &= +(1-xy)^{1/2}(1-xy)^{-1}-(1-xy)^{-1}(1-xy)^{1/2} \\
F\_SPEC_8 &= +(1-xy)^{1/2}y^{-1}-y^{-1}(1-yx)^{1/2} \\
F\_SPEC_9 &= +(1-xy)^{1/2}x-x(1-yx)^{1/2}
\end{aligned}
$$

There are a total of 16 infinite families. The 8 families from Model D together with the following

Ending Relations – Infinite Families

$$\begin{aligned}
F_I[n] &= x(1-yx)^{-n}(1-yx)^{1/2}(1-x)^{-1}-(1-xy)^{-n}\\
&\quad (1-xy)^{1/2}(1-x)^{-1}+(1-xy)^{-n}(1-xy)^{1/2}\\
F_II[n] &= x(1-yx)^{-n}(1-yx)^{1/2}(1-y)^{-1}-(1-xy)^{-n}\\
&\quad (1-xy)^{1/2}(1-y)^{-1}-x(1-yx)^{-n}(1-yx)^{1/2}\\
&\quad +(1-xy)^{-(n-1)}(1-xy)^{1/2}(1-y)^{-1}\\
F_III[n] &= y(1-xy)^{-n}(1-xy)^{1/2}(1-x)^{-1}-(1-yx)^{-n}\\
&\quad (1-yx)^{1/2}(1-x)^{-1}-y(1-xy)^{-n}(1-xy)^{1/2}\\
&\quad +(1-yx)^{-(n-1)}(1-yx)^{1/2}(1-x)^{-1}\\
F_IV[n] &= y(1-xy)^{-n}(1-xy)^{1/2}(1-y)^{-1}-(1-yx)^{-n}\\
&\quad (1-yx)^{1/2}(1-y)^{-1}+(1-yx)^{-n}(1-yx)^{1/2}\\
F_V[n] &= (1-x)^{-1}(1-yx)^{-n}(1-yx)^{1/2}(1-x)^{-1}-(1-x)^{-1}\\
&\quad (1-xy)^{-n}(1-xy)^{1/2}(1-x)^{-1}-(1-yx)^{-n}\\
&\quad (1-yx)^{1/2}(1-x)^{-1}+(1-x)^{-1}(1-xy)^{-n}(1-xy)^{1/2}\\
F_VI[n] &= (1-x)^{-1}(1-yx)^{-n}(1-yx)^{1/2}(1-y)^{-1}-(1-x)^{-1}\\
&\quad (1-xy)^{-n}(1-xy)^{1/2}(1-y)^{-1}-(1-yx)^{-n}\\
&\quad (1-yx)^{1/2}(1-y)^{-1}-(1-x)^{-1}(1-yx)^{-n}\\
&\quad (1-yx)^{1/2}+(1-x)^{-1}(1-xy)^{-(n-1)}(1-xy)^{1/2}\\
&\quad (1-y)^{-1}+(1-yx)^{-n}(1-yx)^{1/2}\\
F_VII[n] &= (1-y)^{-1}(1-yx)^{-n}(1-yx)^{1/2}(1-x)^{-1}-(1-y)^{-1}\\
&\quad \Sigma'(1-x)^{-1}+\Sigma'(1-x)^{-1}+(1-y)^{-1}\Sigma'\\
&\quad -\Sigma'-(1-y)^{-1}(1-yx)^{1/2}(1-x)^{-1}\\
F_VIII[n] &= (1-y)^{-1}(1-yx)^{-n}(1-yx)^{1/2}(1-y)^{-1}-(1-y)^{-1}\\
&\quad (1-xy)^{-n}(1-xy)^{1/2}(1-y)^{-1}+(1-xy)^{-n}\\
&\quad (1-xy)^{1/2}(1-y)^{-1}-(1-y)^{-1}\\
&\quad (1-yx)^{-n}(1-yx)^{1/2}
\end{aligned}$$

where $\Sigma' = \sum_{k=1}^{n}(1-xy)^{-k}(1-xy)^{1/2}$

4 Infinite Bases

We have seen several examples in which the Basis Algorithm does not terminate and the result is an infinite basis. As previously noted, these results are obtained by a three step process: (1) Observations are made of the new elements being added to $\mathcal{G}$ and the process is halted when enough information is obtained to conjecture the ultimate form of the output; (2) An analysis of the current $\mathcal{G}$ is made to find formulas for the infinite families; (3) The S-Polynomial Criterion is used to prove that the conjectured basis has the Gröbner property (in the process, it is shown that the basis elements would eventually arise if the procedure were allowed to run indefinitely). At the moment, steps (2) and (3) are not completely automated, so the whole procedure involves human assistance. In particular, the verification of the S-Polynomial criterion can involve the detailed examination of a large number of "pairings" between parametrized families to identify both the general form for matches and for their reduction.

It should be pointed out, however, that both the truncated $\mathcal{G}$ and the description of the infinite families of rules can be quite useful in practical computation. Since our chosen ordering depends on the number of factors in a term, simplification of any given polynomial expression, f, will only require rules whose left hand sides are lower in order than the leading term of f. It is therefore only necessary to store a finite set of rules to simplify any f up to a certain level of complexity. One can also actually implement the infinite collection of rules. In practice we have stored a large finite subset and we can generate any other rules as they are needed. A simple hashing algorithm is used to identify cases in which the leading term of an infinite family occurs as a factor and when a rule needs to be computed.

4.1 Infinite Bases and Ordering

It should be noticed that the set of rules depends on the chosen ordering. There are cases in which a change in the term ordering will produce both finite and infinite bases. Here is an example:

Example 6 *Let x and y be invertible matrices which commute with each other. The starting relations are*

$$\begin{array}{llllll} A_1 = & x^{-1}x - 1 & A_3 = & y^{-1}y - 1 & A_5 = & yx - xy \\ A_2 = & xx^{-1} - 1 & A_4 = & yy^{-1} - 1 & & \end{array}$$

Case A: $x < x^{-1} < y < y^{-1}$

In this case the algorithm terminates with the following additional rules

$$\begin{aligned} A_6 &= yx^{-1} - x^{-1}y \\ A_7 &= y^{-1}x - xy^{-1} \\ A_8 &= y^{-1}x^{-1} - x^{-1}y^{-1} \end{aligned}$$

Case B: $x < y < x^{-1} < y^{-1}$

In this case we have $B_i = A_i$ for $i = 1 \ldots 5$. There are the same three additional rules (but note the change in order in the first) together with two infinite rules

$$\begin{aligned} B_6 &= x^{-1}y - yx^{-1} \\ B_7 &= y^{-1}x - xy^{-1} \\ B_8 &= y^{-1}x^{-1} - x^{-1}y^{-1} \\ I(n) &= xy^n x^{-1} - y^n \\ II(n) &= yx^{-n}y^{-1} - x^{-n} \end{aligned}$$

To see that this is a Gröbner Basis we need only check matches which involve the two infinite classes (the other matches are checked in the process of running the Basis algorithm and can also be easily verified by machine). We recall that a match is a sextuple $\{f, g, l_1, r_1, l_2, r_2\}$ with $l_1 LM(f) r_1 = l_2 LM(g) r_2$ which comes from an overlap of $LM(f)$ with $LM(g)$.

f	g	l_1	r_2		$Spol(m)$
$II(m)$	$I(n)$	xy^{n-1}	$x^{-(m-1)}y^{-1}$		$y^{n-1}II(m-1) \triangleright -I(n-1)x^{-(m-1)}$
$I(n)$	B_1	x^{-1}	$y^n x^{-1}$	*	$-x^{-1}y^n + y^n x^{-1}$
B_1	$I(n)$	xy^n	x	*	$y^n x - xy^n$
$I(n)$	B_5	y	$y^n x^{-1}$		$I(n+1)$
B_6	$I(n)$	xy^n	y		$-I(n+1)$
$I(n)$	B_7	y^{-1}	$y^n x^{-1}$	1	$xy^{-1}y^n x^{-1} - y^{-1}y^n$
$II(n)$	B_3	y^{-1}	$x^{-n}y^{-1}$	*	$-y^{-1}x^{-n} + x^{-n}y^{-1}$
B_3	$II(n)$	yx^{-n}	y	*	$x^{-n}y - yx^{-n}$
$II(n)$	B_6	x^{-1}	$x^{-n}y^{-1}$		$II(n+1)$
B_7	$II(n)$	yx^{-n}	x	2	$yx^{-n}xy^{-1} - x^{-n}x$
B_8	$II(n)$	yx^{-n}	y^{-1}		$-II(n+1)$

Notes:

* If $ba \to ab$ is a reduction rule then $b^n a - ab^n$ reduces to 0.

1 $xB_3y^{n-1}x^{-1} \triangleright -B_3y^{n-1} \triangleright I(n-1)$

2 $-yx^{-(n-1)}B_1y^{-1} \triangleright +x^{-(n-1)}B_1 \triangleright -II(n-1)$

The notation $g_1 \triangleright g_2 \triangleright \ldots$ indicates the sequence of multiples of basis elements used in reduction. In practice we reduce the first term in an expression in which the LHS of some rule occurs, and we use the first occurrence (when scanning from left to right) within a term.

We see in this example one of the mechanisms by which an infinite family of relations is produced. Notice that the main consequence of the change in ordering is the change in "sidedness" of rule A_6. The family $I(n)$ would not appear if this rule were applied in its original order. Rule B_6, however, does not permit y to be moved to the right past x^{-1}. Another point of view is that $I(n)$ is really just the rule $xx^{-1} - 1$ which is multiplied on the left by y^n then reduced by B_6. This, in fact, is the way the sequence $I(n)$ is produced: The S-polynomial of

$I(n)$ and B_5 produces $I(n+1)$. In any event, the infinite families seen here result from the choice of an ordering in which a rule which could be used to simplify them has the wrong "handedness". In this case, a change in the ordering of polynomial variables produces a rewrite rule with the proper handedness to eliminate the infinite families. Some of the infinite families that appear in the models of the previous section are of this sort – although, in those cases, changes which eliminate some families produce others.

While it is unlikely that rearrangement of the variables in a graded lexicographic ordering will make the families in our models finite, we have found an ordering for which they have finite Gröbner bases. In the case of Model D, for example, we let $<_1$ be the graded lexicographic ordering with $x < y < x^{-1} < y^{-1} < (1-x)^{-1} < (1-y)^{-1}$ and we let $<_2$ be the graded lexicographic ordering with $(1-xy)^{-1} < (1-yx)^{-1}$ The **wreath product** $<_1 \wr <_2$is a term ordering on the full set of variables. (see [Sims]). If we use the wreath product, we obtain a finite basis for Model D. Notice, however, that this ordering does not reduce the number of factors in a term so it is not as well suited to the task of simplifying expressions.

If X and Y are two sets of letters whose words are ordered by $<_X$by $<_Y$ the wreath product on the words in $X \cup Y$ is defined by comparing two such words written in the form $W = W_0 a_1 W_1 \ldots a_s W_s$ and $Z = Z_0 b_1 Z_1 \ldots b_t Z_t$. where the W_i and Z_i are (possibly empty) words in the letters in X while the a_i and b_i are letters in Y. We then say $W < Z$ if

either	$a_1 \ldots a_s <_Y b_1 \ldots b_t$ (as words in Y)
or	$a_1 \ldots a_s = b_1 \ldots b_t$
	and $(W_0, , W_s) <^* (Z_0, , Z_s)$

where $<^*$ is the lexicographic product ordering defined by $<_X$ (i.e. for some j, $W_i = Z_i$, $i = 0 \ldots j-1$ but $W_j <_X Z_j$)

Model D
Wreath Ordering

$$
\begin{array}{rcl}
Dwr_1 & = & +x^{-1}x-1 \\
Dwr_2 & = & +xx^{-1}-1 \\
Dwr_3 & = & +y^{-1}y-1 \\
Dwr_4 & = & +yy^{-1}-1 \\
Dwr_5 & = & +(1-x)^{-1}x-(1-x)^{-1}+1 \\
Dwr_6 & = & +x(1-x)^{-1}-(1-x)^{-1}+1 \\
Dwr_7 & = & +(1-y)^{-1}y-(1-y)^{-1}+1 \\
Dwr_8 & = & +y(1-y)^{-1}-(1-y)^{-1}+1 \\
Dwr_9 & = & +(1-xy)^{-1}xy-(1-xy)^{-1}+1 \\
Dwr_{10} & = & +xy(1-xy)^{-1}-(1-xy)^{-1}+1 \\
Dwr_{11} & = & +(1-yx)^{-1}-y(1-xy)^{-1}x-1 \\
Dwr_{12} & = & +(1-x)^{-1}x^{-1}-(1-x)^{-1}-x^{-1} \\
Dwr_{13} & = & +(1-y)^{-1}y^{-1}-(1-y)^{-1}-y^{-1} \\
Dwr_{14} & = & +(1-xy)^{-1}y^{-1}-(1-xy)^{-1}x-y^{-1} \\
Dwr_{15} & = & +(1-xy)^{-1}x(1-y)^{-1}-(1-xy)^{-1}(1-y)^{-1} \\
 & & -(1-xy)^{-1}x+(1-y)^{-1} \\
Dwr_{16} & = & +x^{-1}(1-x)^{-1}-(1-x)^{-1}-x^{-1} \\
Dwr_{17} & = & +x^{-1}(1-xy)^{-1}-y(1-xy)^{-1}-x^{-1} \\
Dwr_{18} & = & +y^{-1}(1-y)^{-1}-(1-y)^{-1}-y^{-1} \\
Dwr_{19} & = & +(1-x)^{-1}y(1-xy)^{-1}-(1-x)^{-1}(1-xy)^{-1} \\
 & & -y(1-xy)^{-1}+(1-x)^{-1}
\end{array}
$$

It should be noted that the wreath product does not produce simplification in the same sense as the graded lexicographic orders used in the earlier models. In particular, it does not reduce the number of factors in terms. It should also be noted that there are finitely generated ideals which have an infinite Gröbner Basis for every ordering [TMora].

4.2 Infinite Bases and Transformations

Model E in section 3 provides another way in which infinite families can occur. Notice that without the two relations involving $h(xy)$ and $h(yx)$

this is the same as Model B, which has a finite basis. The introduction of two new variables and the "pseudo-commutative" relations in which they occur produces infinite families. The power transformation which replaces $h(xy)$ by $h(xy)^n$ and $h(yx)$ by $h(yx)^n$ converts the starting relations to a set of relations which can be reduced by the starting relations. This will also be true of any relations which are obtained, using the Basis Algorithm, from the starting relations. We expect, therefore, that new relations can be obtained by applying this transformation to simpler relations. The infinite families for Model E are all obtained from the instance for $n = 1$ by using this transformation. These have been called the GENR rules. This provides an example of infinite families which arise by applying a sequence of transformation rules to a finite set of relations.

Notice that the first 4 families (E_I to E_IV) are also like the families in Case B above: they would reduce to 0 if the "handedness" of a simpler rule is changed. The remaining rules (E_V and E_VI) are not of this type. They can be derived from simpler rules by the power transformation, but cannot be reduced by a change in "handedness" of simpler rules.

4.3 Infinite Bases and Recursion

The the infinite families are generated by a process which produces higher order rules from lower order. We may trace this process to obtain recursion relations for these families. The recursion relations can be an important tool in proofs which involve the infinite bases. In the example above we have the following formulas:

Recursive Formulas for Case B	
$I[n] =$	$yI[n-1] - B_5 y^{n-1} x^{-1}$
$I[0] =$	B_2
$II[n] =$	$x^{-1} II[n-1] - B_6 x^{-(n-1)} y^{-1}$
$II[0] =$	B_4

A much more complicated set of recursion formulas comes from the infinite families in Model D. In this case, most of the recursion relations express the instance for n of a rule in terms of the instance for $n - 1$

of the same rule. This is true of the rules I-IV. In the other cases, the instance for n of the given rule uses the instance for n of another rule. In each case we have also explicitly written the rule for $n = 0$ or $n = 1$ from which the recursion starts. A list of the special rules needed in the recursion has been included for reference.

	Recursive Formulas for Model D
$I[n] =$	$(1-xy)^{-1}I[n-1] - SPC_8(1-yx)^{-(n-1)}(1-x)^{-1}$
$I[0] =$	SPC_{21}
$II[n] =$	$(1-xy)^{-1}II[n-1] - SPC_8(1-yx)^{-(n-1)}$
	$\{(1-y)^{-1} - 1\}$
$II[1] =$	$x(1-yx)^{-1}(1-y)^{-1} - (1-xy)^{-1}(1-y)^{-1}$
	$- x(1-yx)^{-1} + (1-y)^{-1}$
$III[n] =$	$(1-yx)^{-1}III[n-1] - SPC_6(1-xy)^{-(n-1)}$
	$\{(1-x)^{-1} - 1\}$
$III[1] =$	$y(1-xy)^{-1}(1-x)^{-1} - (1-yx)^{-1}(1-x)^{-1}$
	$- y(1-xy)^{-1} + (1-x)^{-1}$
$IV[n] =$	$(1-yx)^{-1}IV[n-1] - SPC_6(1-xy)^{-(n-1)}$
$IV[0] =$	SPC_{19}
$V[n] =$	$(1-x)^{-1}I[n] - SPC_{12}(1-yx)^{-n}(1-x)^{-1}$
$V[0] =$	0
$VI[n] =$	$(1-x)^{-1}II[n] - SPC_{12}(1-yx)^{-n}$
	$\{(1-y)^{-1} - 1\}$
$VI[1] =$	$(1-x)^{-1}(1-yx)^{-1}(1-y)^{-1} - (1-x)^{-1}$
	$(1-xy)^{-1}(1-y)^{-1} - (1-yx)^{-1}(1-y)^{-1}$
	$- (1-x)^{-1}(1-yx)^{-1} + (1-x)^{-1}(1-y)^{-1}$
	$+ (1-yx)^{-1}$
$VII[n] =$	$VII[n-1] - (1-y)^{-1}III[n] + SPC_{10}$
	$\{(1-x)^{-1} - 1\}$
$VII[1] =$	$(1-y)^{-1}(1-yx)^{-1}(1-x)^{-1} - (1-y)^{-1}$
	$(1-xy)^{-1}(1-x)^{-1} + (1-xy)^{-1}(1-x)^{-1}$
	$+ (1-y)^{-1}(1-xy)^{-1} - (1-y)^{-1}(1-x)^{-1}$
	$- (1-xy)^{-1}$
$VIII[n] =$	$SPC_{10}(1-xy)^{-n}(1-y)^{-1} - (1-y)^{-1}IV[n]$
$VIII[0] =$	0

$$
\begin{aligned}
SPC_6 &= (1-yx)^{-1}y - y(1-xy)^{-1} \\
SPC_8 &= (1-xy)^{-1}x - x(1-yx)^{-1} \\
SPC_{10} &= (1-y)^{-1}y - (1-y)^{-1} + 1 \\
SPC_{12} &= (1-x)^{-1}x - (1-x)^{-1} + 1 \\
SPC_{19} &= y(1-y)^{-1} - (1-y)^{-1} + 1 \\
SPC_{21} &= x(1-x)^{-1} - (1-x)^{-1} + 1
\end{aligned}
$$

5 Software Considerations

The results obtained in this paper required computer assistance. The simplification methods and the algorithms used are beyond the scope of hand computation. Moreover, a great deal of the work involved generating and analysing the output of computer runs and the operation of algorithms. This type of work benefits from a congenial computing environment. It is therefore appropriate to make some comments about the software used for this project.

This work involves symbolic computation. The currently most popular commerical symbolic algebra systems (Mathematica, Maple, and Macsyma) are rather weak in their provisions for non-commutative operations. They allow operators to be declared non-commutative, but they have very few algorithms and functions dealing with non-commutative operations. While they do have matrix packages, these treat matrices of specific finite dimension, but they do not effectively deal with general symbolic matrices of arbitrary dimension. Thus this project could not have been done with packaged software. Some of the results of this research, however, have been incorporated in a Mathematica package [NCA] designed primarily for engineers.

The research for this project was conducted using an approach in which a core language, designed for the specific research project, is written first. This core system is then modified and expanded as the research is conducted. The end result is a special-purpose software system designed to meet the needs of a specific research project. There are several benefits to this approach to research software:

1 The software provides an interactive computing environment.
2 The system is extensible
3 Modifications can be made with relative ease
4 The language and commands are natural to the researcher
5 It reduces the gap between the concepts of the mathematics and the means of expression provided by programming language
6 The data representation can be tailored to the project to provide for efficiency
7 The inner workings of algorithms can be examined with relative ease
8 It provides a good medium for experimenting with data representation and algorithms
9 The researcher retains a high degree of control over, and knowledge of, the software

A suitable base language is needed to carry out this approach. The computer language Forth has been found to be very effective. Forth can be seen as a language for writing languages. It is built around an expandable dictionary which contains a compiled form of the commands (words) it currently knows. These words can either be executed interactively or used in the definition of new words. Programming in Forth is equivalent to extending the dictionary. The end product of programming is an "application": a collection of words related to a task. These words can be executed interactively or used in the definition of still further words. As in most interactive environments, data is persistent in the sense that values are retained after a command performs its action.

This is not the place for a tutorial on the Forth language. It is important, however, to at least say something about the structure of dictionary entries and the way that compilation is carried out. A new word is defined in terms of previous words together with words that implement the traditional control structures[3]. The body of the new word's dictionary entry consists of a sequence of addresses of its component words punctuated by branching instructions which implement the control flow. Rather than passing source code through a separate program, Forth uses an unusual approach to compilation: words like IF and THEN are written in Forth, are part of the dictionary, and carry out actions during compilation. Thus when IF is encountered it puts in place a conditional jump instruction and leaves space for the address or displacement to the target of the jump (which is not known when IF is encountered). The action of THEN is to resolve the address of the conditional branch. The functions of a conventional compiler, therefore, are distributed to the action of words in the dictionary – and these actions can be modified and extended.

Since the act of programming in Forth consists in adding new words to the dictionary. Forth is extensible. This extensibility includes what, in conventional languages, would require extending and modifying the compiler. Most of a Forth system is written in Forth itself and the "virtual machine code" is easily decompiled. Thus Forth systems tend to be open and provide the user with a very high degree of control over language features. The Forth system is conceptually simple. The language can be learned relatively easily by understanding how it works. An important asset is that it is a language which can be used effectively by someone who is not a professional programmer. The effect is to bring to the working mathematician some of the control over language that is usually only available to a computer specialist.

It should be mentioned that the "virtual machine code" into which high level Forth is compiled executes relatively rapidly. If one adds the ability to code time-critical words directly into assembly language, a Forth application will run at a rate comparable to conventionally compiled programs. Some timings comparisons found in [HSW] show that some of the main algorithms coded in the Forth based research system run substantially faster (about 40x) than the same algorithms coded in Mathematica.

[3]There is also a provision for defining words in the assembly language of the host computer and for defining words for data structures.

Some references on Forth are included. I would like to thank Michael Gassanenko and Mikhail Kolodin for supplying references to the Russian literature on Forth and Anton Ertl for some helpful discussions about the Forth language.

Books and Articles

1 Baranov S.N., Nozdrunov N.R. Yazyk Fort i ego realizatsii. - L. Mashinostroenie, 1988. - 158 p. (Baranoff S.N., Nozdrunoff N.R. "The Forth Language and Its Implementations", in Russian.)[4]

2 Burago A.Yu., Kirillin V.A., Romanovskij I.V. Fort - yazyk dlya mikroprotsessorov. - L.:Znanie, 1989. - 36 p. (Forth: a Language for Microprocessors, in Russian.)

3 Broudi L. Nachalnyj kurs programmirovaniya na yazyke Fort. - M.:Finansy i statistika, 1990. - 352 p. (translation of Brodie L. Starting FORTH. Prentice-hall,Inc.:Englewood Cliffs,New Jersey.)

4 Kelli M., Spais N. Yazyk programmirovaniya Fort. - M.:Radio i svyaz', 1993. - 318 p. (translation of Kelly M., Spies N. "Forth: a Text and Reference". Prentice-Hall, N.J., 1986.)

5 Tuzov V.A. Funktsionalnye metody programmirovaniya. Instrumentalnye sredstva podderzhki programmirovaniya. - LIIAN:Leningrad, 1988. - p. 129-143[5]

6 Tuzov V.A. Yazyki predstavleniya znanij. LGU:Leningrad, 1990. (The Languages of Knowledge Representation, in Russian).[6]

[4]This is the best book on Forth published in Russian.

[5]An excellent paper about the technique known in the West as "data-driven programming".

[6]This book describes what an ideal language for Knowledge Representation should be, the system is an extension of Forth.

Forth Compilers

There are a large variety of Forth compilers, both commercial and public domain, available for many platforms. The research for this project used F-PC by Tom Zimmer and others (for MS-DOS, 16-bit). F-PC has many features, is well written and well documented. It is in the public domain and can be obtained free from many places (in particular, it can be obtained at the Web site `www.forth.org` or by FTP `ftp.taygeta.com`). It adheres primarily to the Forth-83 standard which is the language standard used in the books mentioned above.

A version of Forth which adheres to the recent ANS-Forth standard is Gforth written by Anton Ertl and Bernd Paysan. The kernel of Gforth is written in 'C' (rather than assembly language) so that it can be compiled on many platforms but the majority of the system is written in Forth. It is a 32-bit implementation and is distributed free under the conditions of the GNU General Public License. Copies of the most recent release can be obtained from `www.forth.org` and `www.complang.tuwien.ac.at/forth/gforth/`.

References

[BeckWeis] . Becker & V. Weispfenning, *Gröbner Bases, A Computational Approach to Commutative Algebra*, Springer-Verlag, Undergraduate Texts in Mathematics, 1992.

[Buch] . Buchberger "Grobner bases: an algorithmic method in polynomial ideal theory" *Recent Trends in multidimensional system theory*, Reidel (1985) pp.184–232.

[BuchLoos] uchberger, B. & Loos, R. "Algebraic Simplification" *Computer Algebra - Symbolic and Algebraic Computation*, Springer-Verlag (1982), pp.11–43.

[HSW] . W. Helton, M. Stankus and J. J Wavrik "Computer Simplification of Formulas in Linear Systems Theory.", submitted to *IEEE Transactions on Automatic Control*

[HW] . W. Helton and J. J. Wavrik "Rules for Computer Simplification of the formulas in operator model theory and linear systems", *Operator Theory: Advances and Applications* 73 (1994), pp.325–354.

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

[TMora] . Mora, "An introduction to commutative and noncommutative Gröbner Bases", *Theoretical Computer Science*, vol.134 (1994) pp.131–173.

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

[Sims] harles C. Sims, *Computation with Finitely Presented Groups* Cambridge Univ Press, 1994.

[Ufn] . Ufnarovski, "Combinatorial and asymptotic methods in algebra". (Russian) *Current problems in mathematics. Fundamental directions*, Vol.57 (Russian), 5–177, Itogi Nauki i Tekhniki, Akad. Nauk SSSR, Vsesoyuz. Inst. Nauchn. i Tekhn. Inform., Moscow, 1990.

John J Wavrik,
Department of Mathematics,
University of California,
San Diego, California, USA
e–mail: *jjwavrik@ucsd.edu*

Received 5 November, 1996