MAPLE V Procedure for Computation of Gröbner Bases by the Gröbner Walk

Klaus Aigner

March 24, 1997

Abstract

GWalk is a MAPLE V Procedure which converts a given Gröbner basis of a polynomial ideal I of arbitrary dimension with respect to the tdeg ordering to a Gröbner basis with respect to the plex ordering. The implementation is based on the theory of the Gröbner Walk developed by S.Collart, M. Kalkbrenner and D. Mall.

1 Introduction

The size and the form of a Gröbner basis of a polynomial ideal [2] depends on the chosen term ordering. Also the complexity of its computation is strongly influenced by the underlying term ordering. The choice of the term ordering usually depends on the problem we have to solve, e.g. for elimination we need lexicographic orderings. Unfortunately these lexicographic orderings are rather cumbersome to compute, whereas we can compute total degree Gröbner bases much faster. Therefore, it is often advantageous to compute first a Gröbner basis with respect to a total degree ordering and then to convert it to a lexicographic Gröbner basis by a special conversion algorithm. This combined process can be considerably faster then the direct application of the Buchberger algorithm to compute a lexicographic Gröbner basis. Such a conversion algorithm was given by Faugère, Gianni, Lazard and Mora in [4] for 1-dimensional ideals. Another conversion method is the Gröbner Walk developed by Collart, Kalkbrenner and Mall [3]. This method works for ideals of arbitrary dimension. We implemented the MAPLE V Procedure GWalk, which is based on the method of the Gröbner Walk.

The GWalk Procedure takes as input a list of polynomials which are the basis of an ideal I and a list of ordered variables. It first computes a Gröbner basis G with respect to the tdeg ordering. Following a path in the Gröbner fan we construct a finite sequence of term orderings, $tdeg = \prec_0, \ldots, \prec_m = plex$ and lists of polynomials $G = G_1, \ldots, G_m$, such that G_k is a Gröbner basis of I with respect to $\prec_k$. As G_{k+1} lies in the neighborhood of G_k, that is the corresponding cones of the Gröbner Fan [5] of I are adjacent, G_{k+1} can be derived from G_k relatively fast. In general the Gröbner walk works for any arbitrary two different admissible term orderings.

Our implementation is written in the MAPLE V Programming Language and uses parts of the MAPLE V Package Grobner. For small examples the procedure may present some overhead. For larger examples (> 100s) we achieve considerable speedup factors of 2 - 50, and we can compute Gröbner bases where the MAPLE V procedure gbasis fails.

2 Mathematical Foundations

The theoretical background in this and the subsequent chapters is taken from [3]. Let I be an ideal in the polynomial ring $R := K[x_1, \ldots, x_n]$, where K is an arbitrary field. We denote the *set of terms* in the variables $x_1, \ldots, x_n$ by T^n. Let f be a polynomial in R, G a subset of R, and $\langle G \rangle$ the ideal generated by G. For a term ordering $<$ in T^n we denote by $in_<(f)$ the *initial monomial* of f w.r.t. $<$ and by $G_<$ the set $\{in_<(g) | g \in G\}$. G is a Gröbner basis of I if and only if $G_<$ is a Gröbner basis of the monomial ideal $\langle I_< \rangle$. The set

$$\Omega^n := \{(\omega_1, \ldots, \omega_n) | \omega_i \in \mathbb{Q}, \omega_i \geq 0, 1 \leq i \leq n\}$$

is called the set of *weight vectors*. Let $\omega \in \Omega^n$, f a nonzero polynomial in R, $t = a x_1^{i_1} \ldots x_n^{i_n}$ a monomial, then we define the *ω-degree* of t as

$$deg_\omega(t) := \sum_{j=1}^{n} i_j \omega_j$$

and the *ω-degree* of f as

$$deg_\omega(f) := max(\{deg_\omega(t) | t \text{ a monomial of } f\}).$$

We call f *ω-homogenous* iff all monomials of f have the same ω-degree. Let $f = h_1 + \ldots + h_r$ where h_i's are ω-homogenous and $deg_\omega(h_i) > deg_\omega(h_j)$ for $i < j$, then we call the h_i's the *ω-homogenous components* of f, h_1 the *initial form* of f, denoted by $in_\omega(f)$ and the ideal I *ω-homogenous* iff for every $f \in I$ all ω-homogenous components of f are in I. Furthermore we denote by G_ω the set $\{in_\omega(g) | g \in G\}$.

A weight vector $\omega \in \Omega^n$ induces a partial order $\prec(\omega)$ by

$$t_1 \quad \prec(\omega) \quad t_2 \text{ iff } deg_\omega(t_1) < deg_\omega(t_2), \text{ for } t_1, t_2 \in T^n.$$

For a term ordering $<$ we define the term ordering $\omega^<$ by

$$t_1\ \omega^<\ t_2 \text{ iff } t_1 \prec (\omega) t_2 \text{ or } deg_\omega(t_1) = deg_\omega(t_2) \text{ and } t_1 < t_2.$$

We say that $<$ *refines* ω iff $\prec \subseteq <$. If $<$ refines $\prec$ then $\langle I_< \rangle = \langle \langle I_\omega \rangle_< \rangle$. If there exists a term ordering such that $\langle I_< \rangle = \langle I_\omega \rangle$ then we say ω *represents* $<$ for I. The following lemma gives a criterion whether a weight vector represents a term ordering for I.

Lemma 1 *Let G be a reduced Gröbner basis of I w.r.t. $<$. Then a weight vector ω represents $<$ iff $in_\omega(g) = in_<(g)$ for all $g \in G$.* $\square$

For the Gröbner Walk we will have to represent arbitrary term orderings.

Lemma 2 *Every term ordering $<$ can be expressed by a sequence S of n rational vectors which span $\mathbb{Q}^n$ and the first element of S is the unique weight vector ω (up to a scalar) s.t. $<$ refines ω.* $\square$

We now mention some important facts on the Gröbner fan of I, which will be used for constructing the path for the Gröbner Walk. For more details on the Gröbner fan and the proofs of the given theorems we refer to [5]. We define the *Gröbner cone* of I w.r.t. $<$ by

$$cone(<) := cl(\{\omega \in \Omega^n | \langle I_< \rangle = \langle I_\omega \rangle \})$$

where cl denotes the usual topological closure of $\mathbb{Q}^n$.

Theorem 3 *The $cone(<)$ is a convex polyhedral cone in $\mathbb{Q}^n$.* $\square$

We call $F(I) := \{cone(<) |\ <$ a term ordering$\}$ the *Gröbner fan* of I. For the termination property of the Gröbner Walk we will need:

Theorem 4 *The Gröbner fan has finite cardinality.* $\square$

The following easy lemma can be used as termination condition in the algorithm. It follows immediately from Lemma 1.

Lemma 5 *Let $<$ and $\ll$ be term orderings, $G = \{g_1, \ldots, g_r\}$ the reduced Gröbner basis of I w.r.t. $<$. Then $cone(<) = cone(\ll)$ iff $in_<(g_i) = in_\ll(g_i)$ for all $1 \leq i \leq r$.* $\square$

As a consequence we obtain that $cone(<) = cone(\ll)$ implies $G = H$, where G and H are reduced Gröbner bases w.r.t. $<$ and $\ll$, respectively. Hence we can speak about a reduced Gröbner basis of I over a $cone(<)$.

3 The Walk

We assume that we are given a Gröbner basis G of I with respect to a given term ordering $<$. Our goal is to compute a Gröbner basis of I with respect to another term ordering $\ll$.

By Lemma 2 we can express the term orderings $<$ and $\ll$ by sequences of rational vectors $S_<$ and $S_\ll$, respectively. Their first elements, denoted by σ and τ are weight vectors which refine $<$ and $\ll$, respectively. For σ, $\tau \in \Omega^n$ we denote the line segment in Ω^n between σ and τ by $\overline{\sigma\tau}$, i.e.

$$\overline{\sigma\tau} := \{(1-t)\sigma + t\tau | 0 \le t \le 1\}$$

It follows from Theorem 3 and Theorem 4 that there exist finitely many weight vectors $\sigma = \omega_0, \omega_1, \ldots, \omega_m = \tau$ in $\overline{\sigma\tau}$ and pairwise different cones $cone(<) = C_1, \ldots, C_m = cone(\ll)$ in the Gröbner fan of I such that for every $k \in \{1, \ldots, m\}$ the line segment $\overline{\omega_{k-1}\omega_k}$ is the maximal line segment in $\{\overline{\omega_{k-1}\psi} \subseteq C_k | \psi \in \overline{\omega_{k-1}\tau}\}$. We denote the reduced Gröbner basis of I over C_k by G_k.

Now we have to compute $\omega_1, \ldots, \omega_{m-1}$ and $G_2, \ldots, G_m$ successively. First of all by using Lemma 5 we can decide if $C_k = cone(\ll)$ for a given $G_k = \{g_1, \ldots, g_r\}$ and therefore $m = k$ and G_k is already the reduced Gröbner basis of I with respect to $\ll$.

If not, we have to determine the next weight vector, that is the point on our walk where we leave the cone C_k. By Lemma 5 we have to detect a change in the initial forms, i.e. we compute a weight vector $\omega = \omega(t) := \omega_k + t(\tau - \omega_k), 0 < t \le 1$, on the line $\overline{\omega_k\tau}$ with the property

$$\begin{aligned} t = min(\{s \in \mathbb{Q} \cap (0,1] | deg_{\omega(s)}(p_1) = deg_{\omega(s)}(p_i), \\ \text{for some } g = p_1 + \ldots + p_n \in G, 2 \le i \le n\}). \end{aligned}$$

This can be done by the calculation of two scalar products in $\mathbb{Q}^n$ and one rational coefficient per monomial. Thus, if t is defined, it is a positive rational number. The value of t provides some more information: if $t = 1$ then the next conversion is the last step of the walk and if t is undefined then we have already determined the final Gröbner basis.

It remains to convert the Gröbner basis G_k over the cone C_k to a Gröbner basis G_{k+1} over the cone C_{k+1}. Let $\prec_1$ and $\prec_2$ be term orderings which refine ω_k and such that $cone(\prec_1) = C_k$ and $cone(\prec_2) = C_{k+1}$. For computing G_{k+1} we need the following lemmata.

Lemma 6 *The set G_{ω_k} is a Gröbner basis of $\langle I_{\omega_k}\rangle$ w.r.t. $<$.* $\square$

Since the S-polynomial of two ω-homogenous polynomials is ω-homogenous we obtain the following lemma.

Lemma 7 *Every reduced Gröbner basis of an ω-homogenous ideal consists of ω-homogenous polynomials.* $\square$

Let $\{m_1, \ldots, m_s\}$ be a reduced Gröbner basis of $\langle I_{\omega_k} \rangle$ w.r.t. $\ll$. It follows from the lemmata that for every $1 \leq i \leq s$ there exist ω-homogenous polynomials $h_{i1}, \ldots, h_{ir}$ s.t.

$$m_i = \sum_{j=1}^{r} h_{ij} in_\omega(g_j) \text{ and } deg_\omega(m_i) = deg_\omega(h_{ij} in_\omega(g_j)) \text{ for } 1 \leq j \leq r \text{ with } h_{ij} \neq 0.$$

Using the h_{ij}'s we can lift the m_i's to a Gröbner basis of I with respect to $\prec_2$. Replacing $in_\omega(g_j)$ by g_j we obtain

$$f_i := \sum_{j=1}^{n} h_{ij} g_j \text{ and } F := \{f_1, \ldots, f_s\}$$

Theorem 8 *The set F is a GB of I w.r.t. $\prec_2$.* $\square$

Proof: The polynomials h_{ij} are ω_k-homogenous, therefore we have

$$h_{ij} in_{\omega_k}(g_j) = in_{\omega_k}(h_{ij} g_j)$$

for every $i \in \{1, \ldots, s\}$ and $j \in \{1, \ldots, r\}$. This implies $in_{\omega_k}(f_i) = m_i$. Since $\prec_2$ refines ω_k,

$$in_{\prec_2}(m_i) = in_{\prec_2}(f_i).$$

Hence $\langle F_{\prec_2} \rangle = \langle I_{\prec_2} \rangle$ and since $F \subseteq I$, F is a Gröbner basis of I with respect to $\prec_2$. $\square$

This lifted Gröbner basis is not reduced (see section 4.2.). Now we can formulate an algorithm.

Gröbner Walk

Input:
  a weight vector $<$ which refines ω
  a weight vector $\ll$ which refines τ
  G reduced Gröbner Basis of I with respect to $<$
Output:
  G reduced Gröbner Basis of I with respect to $\ll$

```
t := 0
while defined(t)
begin
    G_ω := initials(G, ω)
    G_ω := gröbner-basis(G_ω, ω^≪)
    G = lift(G_ω, G, ω^≪)
    G := reduce(G_ω, ω^≪)
    t := determine-border(G, ω, τ)
    if defined(t) then ω := ω + t(τ − ω)
end
RETURN(G)
```

Table 1: The Gröbner Walk Algorithm

4 Speeding up the Walk

There are several methods to speed up the computation of the Gröbner Walk, recommended by Amrhein, Gloor and Küchlin [1], which we implemented in our procedure.

4.1 Path Perturbation

Whenever we do one step in the Gröbner Walk, i.e. we pass from one cone to another, some of the initial monomials $in_w(g)$ of the Gröbner basis become initial forms (true polynomials). In points where only two cones meet, we usually obtain very small polynomials. At points of intersection of some cones the initial polynomials become larger and it is much more effort to perform the conversion step. Hence we want to avoid to walk through such meeting-points. Particularly in the starting point (the starting weight vector σ) and the end point (the target weight vector τ), which usually consist of very simple weight vectors, it often happens that we are in meeting points. By perturbing these points slightly we can improve this situation. We can guarantee to stay within the right cone by controlling the initial polynomials. We may have to walk through more cones on the perturbed path. However, the individual conversions can be done much faster.

4.2 Reduction of Intermediate Gröbner Bases

As mentioned in [3] the Gröbner Walk works also by using minimal Gröbner bases instead of reduced Gröbner bases. So we could omit the reduction of the lifted Gröbner bases, which causes a lot of extra cost. But if we do not have a reduced Gröbner basis, when computing the next border, we may erronously conclude that we are leaving a cone (because an initial form of a non-reduced polynomial changes). This can lead to lots of additional steps (however, the termination property still holds).Experiments show that we gain more by the elimination of the extra Gröbner bases than we loose by the reduction steps. Hence we use intermediate reductions in our procedure.

4.3 Integer Weight Vectors

Every comparison of two monomials involves costly rational arithmetic with the rational weight vectors. Since the weight vectors ω and $\lambda\omega$, $\lambda \in \mathbb{Z}$, lead to the same partial ordering, we can scale the weight vectors until they are integer vectors. So we can use integer arithmetic, which is less time consuming.

4.4 A Specialized Buchberger Algorithm for Initial Forms

In a perturbed walk most of the initials are monomials. Since a critical pair of two monomials is unnecessary (its S-polynomial is always zero) we do not have to consider them. Furthermore experiments show that Gröbner bases of two adjacent cones are very similar, i.e. the computation of the initial Gröbner basis usually adds just one or two new polynomials to the basis. As most of the S-polynomials reduce to 0 in one step, there is no use spending time on sophisticated selection strategies.

5 Remarks on the Implementation

We restrict the Gröbner Walk to start with a Gröbner basis with respect to tdeg ordering and to convert it to a Gröbner basis with respect to plex ordering in our implementation, although the algorithm would work for any admissible ordering, but MAPLE V only supports the computation of leading terms of a polynomial for these two orderings by internal functions. All other orderings which must be constructed by matrices lead to a very inefficient computation of leading terms. So we try to avoid introducing new orderings as much as possible. If we start with the tdeg ordering we can use the MAPLE V Procedure *gbasis* to compute the first Gröbner basis. Furthermore if we use the plex ordering as the target of the Gröbner walk we can apply the following technique for the computation of leading terms with respect to the intermediate orderings: First we compare monomials with respect to the current weight vector and if the ω-degrees are equal we break ties with respect to the lexicographic ordering.

6 Description of the GWalk Procedure

FUNCTION: GWalk - compute a reduced Gröbner basis with respect to the lexicographical ordering

CALLING SEQUENCE:

GWalk(F, X1)
GWalk(F, X1, X2)

PARAMETERS:

F	-list of polynomials
X1	-list of indeterminates
X2	-list of indeterminates

SYNOPSIS:

- The command GWalk($F, X1$) computes the reduced, minimal Gröbner basis of the polynomials F in the indeterminates $X1$ with respect to the lexicographical ordering. The algorithm first computes a Gröbner basis with respect to tdeg ordering by using the function grobner[gbasis] and converts it to a lexicographical Gröbner basis by the use of the Gröbner Walk algorithm.
- The list of indeterminates $X1 := [x1, x2, \ldots, xn]$ induces the ordering $x1 > x2 > \ldots > xn$.
- If the procedure is called with 3 arguments then the list of indeterminates $X2 := [x1, x2, \ldots, xn]$ induces the ordering $x1 > x2 > \ldots > xn$ and is used for the computation of the tdeg Gröbner basis. Otherwise, $X1$ is used for this computation.
- The function is in the grobnerwalk package, and so can be used in the form GWalk(...) only after performing the command with(grobnerwalk). The function can always be accessed in the long form grobnerwalk[GWalk].

EXAMPLES:

```
> GWalk([t^2+y^2+z^2-1 ,t^2+z^2-y,  t-z], [t, y, z]);
```

$$[\, t - z, 2\, z^2 - y, 4\, z^4 + 2\, z^2 - 1\,]$$

```
> GWalk([ 1+x^2+z,x*z+y^2*z],[z,x,y],[x,y,z]);
```

$$[\, 1 + x^2 + z, y^2\, x^2 + x^3 + y^2 + x\,]$$

SEE ALSO: grobner package

7 Examples

First we give a small example with all important intermediate results. The Gröbner walk is done in two steps.

```
> GWalk([ 1+x^2+z,x*z+y^2*z],[z,x,y]);

starting base:   [x*z+y^2*z, 1+x^2+z]

starting weight vector:   [1, 1, 1]

Base Change at:   [18, 9, 8]

Initial Forms:   [y^2*z, x^2+z]

Changed Initial Base:   [x^2+z, -y^2*x^2]

New Grobner Basis:   [1+x^2+z, -y^2*x^2-x^3-y^2-x]

Base Change at:   [11, 2, 1]

Initial Forms:   [z, -y^2*x^2-x^3]

Changed Initial Base:   [z, -y^2*x^2-x^3]

New Grobner Basis:   [1+x^2+z, y^2*x^2+x^3+y^2+x]

Final Grobner Basis:
```

$$[\,1+x^2+z, y^2\,x^2+x^3+y^2+x\,]$$

For the next examples we just give the results and the intermediate timings, where the abbriviations have the following meaning:

firstbasis:	computation of the first Gröbner basis w.r.t. tdeg ordering
firstweight:	perturbation of the starting weight vector
B:	determination of the next border
I:	computation of the ω-initials of a Gröbner basis
G:	conversion of the ω-initial Gröbner basis
T:	computation of the transformation matrix
L:	lifting and reduction of the new Gröbner basis

```
> F:=[3*x^2+2*y*z-2*x*l,2*x*z-2*y*l,2*x*y-2*z-2*z*l,x^2+y^2+z^2-1]:
> GWalk(F,[l,x,y,z]);

firstbasis   4.233
firstweight   .584
B:   .183   I:   .083   G:   .667   T:   .250   L:   .350
B:   .183   I:   .084   G:   .516   T:   .234   L:   .716
B:   .167   I:   .067   G:   .500   T:   .150   L:   .233
B:   .117   I:   .050   G:   .316   T:   .134   L:   .283
B:   .117   I:   .050   G:   .766   T:   .134   L:   .233
B:   .117   I:   .050   G:   .300   T:   .116   L:   .200
B:   .117   I:   .050   G:   .333   T:   .100   L:   .217
B:   .117   I:   .050   G:   .400   T:   .100   L:   .200
B:   .117   I:   .317   G:   .350   T:   .116   L:   .184
```

```
complete time:   14.183
```

$$[3835\,x\,z - 1404\,z^3 + 2556\,z + 3835\,y\,z^2 - 1152\,z^5,\, x^2 + y^2 + z^2 - 1,$$
$$3835\,x\,y + 25987\,z^3 - 6403\,z - 19584\,z^5,$$
$$3835\,y^3 + 11778\,z^3 - 2562\,z - 3835\,y + 3835\,y\,z^2 - 9216\,z^5,$$
$$7670\,l - 11505\,x + 477321\,z^4 - 134419\,z^2 - 11505\,y\,z - 335232\,z^6,$$
$$-6912\,z^5 + 10751\,z^3 - 3839\,z + 3835\,z\,y^2,$$
$$1152\,z^6 - 1605\,z^4 + 453\,z^2 - 118\,y\,z^3 + 118\,y\,z,$$
$$655\,z^3 - 44\,z - 1763\,z^5 + 1152\,z^7]$$

```
> f:=x*y^4+x*y^5+x^2*y^3+x^5*y^2-19*x^2*y^5-53*x^3*y^4+x^5*y+
x^5*y^5+x^5+43*x^3*y^3+x^4*y^3+12*x^4*y^4+57*x^3*y^5-19*x^5*y^3-
36*x^4*y^5+y^5+21*x^5*y^4-15*x^3*y^2:
> G1 := GWalk([f,diff(f,x),diff(f,y)],[x,y]);
```

```
firstbasis   743.234
firstweight   .250
B:   .383   I:   .083   G:   .350   T:   .084   L:   .333
B:   .133   I:   .083   G:   .250   T:   .067   L:   .317
B:   .150   I:   .066   G:   .517   T:   .083   L:   .234
B:   .116   I:   .334   G:   .183   T:   .067   L:   .266
B:   .134   I:   .050   G:  1.133   T:   .083   L:   .234
complete time:   749.367
```

$$G1 := [3438383845\,x^4 - 12575090766471\,y^7 + 1375353538\,x\,y^3$$
$$+ 13014758581693\,y^8 - 1843098709713\,x\,y^7 + 687676769\,y^4$$
$$- 30945454605\,x^2\,y^2 + 2674873695\,y^5 + 5785077584\,x\,y^4$$
$$- 1755643410145\,x\,y^5 + 10315151535\,x^2\,y^3 - 6010028812787\,y^9$$
$$+ 1152663930821\,y^{10} + 4404019364745\,y^6$$
$$+ 3619088759496\,x\,y^6,\, -77304505780298\,y^7 + 41260606140\,x^3\,y$$
$$- 5501414152\,x\,y^3 + 74935053741478\,y^8 - 13570725327662\,x\,y^7$$
$$- 6876767690\,y^4 - 4126060614\,x^2\,y^2 - 29156770318\,y^5$$
$$- 25279174024\,x\,y^4 - 13371628797510\,x\,y^5 - 119655757806\,x^2\,y^3$$
$$- 31802642685402\,y^9 + 5690570375581\,y^{10}$$
$$+ 28476297280509\,y^6 + 27096916531768\,x\,y^6,\, 39818\,x\,y^8$$
$$- 225434\,y^7 + 278424\,y^8 - 119454\,x\,y^7 - 39818\,x\,y^5 - 145798\,y^9$$
$$+ 26495\,y^{10} + 66313\,y^6 + 119454\,x\,y^6,\, 188338878913\,y^7$$
$$- 176904063992\,y^8 + 36190241747\,x\,y^7 + 16823105\,y^5$$
$$+ 45034158\,x\,y^4 + 37475526969\,x\,y^5 - 687676769\,x^2\,y^3$$
$$+ 687676769\,x^2\,y^4 + 71947321111\,y^9 - 12417053597\,y^{10}$$
$$- 70981905540\,y^6 - 73710802874\,x\,y^6,$$
$$-y^{11} - 5\,y^7 + 10\,y^8 - 10\,y^9 + 5\,y^{10} + y^6]$$

8 Empirical Results

For testing our implementation we used MAPLE V Release 3 running on an SGI (100MHz IP20 Processor, 96MB RAM). The following examples should give a rough idea of how long it takes to compute a reduced lexicographical Gröbner basis of an polynomial ideal by using the GWalk procedure compared with the direct application of the MAPLE V procedure gbasis.

	Variables	Polynomials
CKM	$z > x > y$	$1 + x^2 + z, x * z + y^2 * z$
Ex1	$x > y > z$	$4 * z - 4 * x * y^2 - 16 * x^2 - 1,$ $2 * y^2 * z + 4 * x + 1, 2 * x^2 * z + 2 * y^2 + x$
COX	$l > x > y > z$	$3 * x^2 + 2 * y * z - 2 * x * l, 2 * x * z - 2 * y * l,$ $2 * x * y - 2 * z - 2 * z * l, x^2 + y^2 + z^2 - 1$
Ex2	$x > y > z$	$3 * x + z + x * y * z + x^2 * y * z + x * z^3,$ $3 * x * y + x^2 * y + x^3 * y + x^3 * z + 2 * x * z^3$
Ex3	$x > y > z$	$x^3 + y + x * y^2 + y^3 + z^2,$ $2 * x^3 * y + y^2 + x * y^2 + x^3 * z + x * z^3$
Ex4	$x > y > z$	$2 * x^2 * y + x^3 * y + 2 * x * y^2 * z,$ $x * y^3 + y^4 + y * z^2 - z^3 - 2 * x * z^3,$ $2 - 3 * x^2 * y + 2 * x^3 * y + y * z^3$
CAN	$x > y > z > t > u$	$-x^3 - x * y + z, -y^3 + t - 2,$ $-x^2 * y - x * y^2 + u$
CAN2	$x > y > z > t > u$	$x^3 - x * y^2 - z, x^2 * y - t - 2,$ $y^3 - x^2 - u$
TRI1	$v > u > t > z > y > x$	$45 * y + 35 * u - 165 * v - 36,$ $35 * y + 25 * z + 40 * t - 27 * u,$ $25 * y * u - 165 * v^2 + 15 * x - 18 * z + 30 * t,$ $15 * y * z + 20 * t * u - 9 * x,$ $-11 * v^3 + x * y + 2 * z * t,$ $-11 * u * v + 3 * v^2 + 99 * x$
WIN	$x > y$	$f := x * y^4 + x * y^5 + x^2 * y^3 + x^5 * y^2$ $-19 * x^2 * y^5 - 53 * x^3 * y^4 + x^5 * y$ $+x^5 * y^5 + x^5 + 43 * x^3 * y^3 + x^4 * y^3$ $+12 * x^4 * y^4 + 57 * x^3 * y^5 - 19 * x^5 * y^3$ $-36 * x^4 * y^5 + y^5 + 21 * x^5 * y^4$ polysystem $:= \{f, f_x, f_y\}$

For these examples we get the following timings:

	tdeg	Walk	$\sum$	plex
CKM	0.100s	0.317s	0.417s	0.100s
Ex1	0.683s	2.317s	3.000s	0.543s
COX	4.366s	10.234s	14.600s	17.950s
Ex2	2.300s	20.366s	22.666s	18.217s
Ex3	2s	104s	106s	692s
Ex4	25s	400s	425s	>1800s
CAN	131s	706s	837s	>30000s
CAN2	117s	434s	551s	>9000s
TRI1	17s	33s	50s	>13000s
WIN	743s	6s	749s	19205s

9 Conclusions

Although the computation of leading terms for the intermediate term orderings of the Gröbner Walk is rather slow, since MAPLE V does not support their computation by internal functions, for larger examples it is much faster to compute a lexicographical Gröbner basis by the GWalk procedure than applying the MAPLE V procedure gbasis directly. With our implementation, we achieved speedup factors from 2-50 for reasonaby large examples. If MAPLE V would support the computation of leading terms of polynomials with respect to any admissible term orderings, we could easily generalize our procedure to arbitrary starting and target weight vectors. Furthermore, this would speed up the whole computation, since the determination of the intermediate reduced Gröbner bases could be performed faster.

References

[1] B. Amrhein, O. Gloor, and W. Küchlin, *Walking Faster*, in: Design and Implementation of Symbolic Computation Systems (Proc. DISCO'96), J. Calmet and C. Limongelli (eds.), 150-161, Springer Verlag LNCS 1128 (1996).

[2] B. Buchberger, *An algorithmic method in polynomial ideal theory*, in: N.K. Bose, editor, Recent Trends in Multidimensional Systems Theory, chapter 6. Reidel, (1985). (Also Report CAMP-83.29, U.Linz, 1983).

[3] S. Collart, M. Kalkbrenner, D. Mall, *The Gröbner Walk*, manuscript(1995). To appear in JSC.

[4] J. Faugère, P. Gianni, D. Lazard, and T. Mora, *Efficient computation of zero-dimensional Gröbner Bases by change of ordering*, JSC 16, 329-344, (1993).

[5] T. Mora and L. Robbiano , *The Gröbner Fan of an ideal*, JSC 6, 183-208, (1988).

[6] V. Weispfenning, *Admissible orders and linear forms*, SIGSAM Bulletin 21, 16-18, (1987).

Appendix: Source Code

```
# Groebner Walk
#
# GWalk computes the reduced, minimal Groebner basis
# of the polynomials F in the indeterminates X with respect to the
# lexicographical ordering.
#
# param.:  F ... list of polynomials
#              X1 ... list of variables for the tdeg Groebner basis
#              X2 ... list of variables for the plex Groebner basis
#
# output:  G the Groebner basis of F w.r.t. plex ordering
# ---------------------------------------------------------------

`grobnerwalk/GWalk` := proc(F, X1, X2)

    global grobnerwalk;
    local G, X, Y, f,p;

#--------------------------------------------------------------
# check input parameters
#--------------------------------------------------------------

    if nargs < 2 then ERROR(`too few arguments`) fi;
    if not(type(F,list)) then ERROR(`first argument must be a list`) fi;
    if not(type(X1,list)) then ERROR(`second argument must be a list`) fi;
    if nargs = 3 then
        if not(type(X2,list)) then ERROR(`third argument must be a list`) fi;
        if convert(X1,set) <> convert(X2,set) then
            ERROR(`the variable list X1 and X2 contain different variables`)
        fi
    fi;
    if nops(X1)<1 then ERROR(`no indeterminates specified`) fi;
    if nops(F)<1 then ERROR(`no polynomials specified`) fi;
    if member(false, map(type,X1,name),'p') then
        ERROR(`bad variable list`)
    fi;

    Y := {};
    for f in F do
        if not(type(f,'polynom(anything,X1)')) then
            ERROR(`input must be polynomials over`,X)
        fi;
        Y := Y union indets(f);
    od;
    if Y minus {op(X1)} <> {} then ERROR(`coefficients must be rational`) fi;

    if nargs=2 then X := X1 else X:= X2 fi;

#--------------------------------------------------------------
# load procedures
#--------------------------------------------------------------

    grobnerwalk[grobnerwalk] := 'readlib(``grobnerwalk/grobnerwalk``)';
    grobnerwalk[winitials] := 'readlib(``grobnerwalk/winitials``)';
    grobnerwalk[lift] := 'readlib(``grobnerwalk/lift``)';
    grobnerwalk[detborder] := 'readlib(``grobnerwalk/detborder``)';
    grobnerwalk[initial] := 'readlib(``grobnerwalk/initial``)';
```

```
    grobnerwalk[intcone] := 'readlib(''grobnerwalk/intcone'')';
    grobnerwalk[memcone] := 'readlib(''grobnerwalk/memcone'')';
    grobnerwalk[weight] := 'readlib(''grobnerwalk/weight'')';
    grobnerwalk[normalform] := 'readlib(''grobnerwalk/normalform'')';
    grobnerwalk[mongb] := 'readlib(''grobnerwalk/mongb'')';
    grobnerwalk[transmat] :=   'readlib(''grobnerwalk/transmat'')';
    grobnerwalk[transvect] :=   'readlib(''grobnerwalk/transvect'')';

#-------------------------------------------------------------
# main
#-------------------------------------------------------------

     G := grobnerwalk[grobnerwalk](F, X1, X);

end:
```

```
# Groebner Walk
#
# grobnerwalk computes the reduced, minimal Groebner basis
# of the polynomials F in the indeterminates X with respect to the
# lexicographical ordering.
#
# param.:  F ... list of polynomials
#                X1 ... list of variables for the tdeg Groebner basis
#                X2 ... list of variables for the plex Groebner basis
#
# output:  G the Groebner basis of F w.r.t. plex ordering
# --------------------------------------------------------------

`grobnerwalk/grobnerwalk` := proc(F,X1,X)

    local G, w1, w2, n, i, Gw, tau, T, t, j, oldw, Gnew, d, neww, newGw,
          cflag, unique, Glpp, k,  Glp1, Glp2, ord1;

# --------------------------------------------------------------
# get number of variables and compute weight vectors
# --------------------------------------------------------------

    n := nops(X);
    w1 := [seq(1, i=1..n)];
    w2 := [seq(10^(n-i-1), i=1..n-1),0];
    oldw:= w1;

# -------------------------------------------------------------
# compute a Groebner basis w.r.t. tdeg ordering
# -------------------------------------------------------------

    G := grobner[gbasis](F, X1);

# -------------------------------------------------------------
# perturb starting weight vector
# -------------------------------------------------------------

    Gw := grobnerwalk[winitials](G, w1, X1);
    Glpp := [seq(grobner[leadmon](Gw[i], X1, tdeg),i=1..nops(Gw))];
    Glpp := [seq(Glpp[i][1]*Glpp[i][2], i=1..nops(Glpp))];
    unique := grobnerwalk[intcone](Gw);
    ord1 := linalg[matrix](n,n,1);
    for i to n do
        for j to n do
            if i=n+1-j then ord1[i,j] := 0 fi;
        od
    od;
    for i to n while not(unique) do
        cflag := false;
        j := 1;
        while not(cflag) do
            neww := [seq(10^j*w1[k]+ord1[i,k],k=1..n)];
            newGw := grobnerwalk[winitials](G,neww,X1);
            cflag := grobnerwalk[memcone](Glpp, newGw);
            j := j+1
        od;
        w1 := neww; Gw := newGw;
        unique :=  grobnerwalk[intcone](Gw);
    od;

# -------------------------------------------------------------
```

```
# main loop for the Groebner Walk
# -------------------------------------------------------------

    do

# -------------------------------------------------------------
# determine the new border
# -------------------------------------------------------------

        tau := [seq(w2[i]-w1[i], i=1..n)];
        t := grobnerwalk[detborder](w1, tau, G, Glpp,X1);

# -------------------------------------------------------------
# check if we have already a Groebner basis w.r.t. plex ordering and
# check if the perturbed target vector is member of the target cone
# -------------------------------------------------------------

        if t=1 then
            Glp1 := [seq(grobnerwalk[initial](G[i], X, w1),i=1..nops(G))];
            Glp2 := [seq(grobnerwalk[initial](G[i], X, w2),i=1..nops(G))];
            if Glp1 = Glp2 then
                RETURN(G)
            else
                w2 := [1, seq(0, i=1..n-1)]
            fi
        fi;

# -------------------------------------------------------------
# compute the new weight vector
# -------------------------------------------------------------

         oldw := w1;
         w1 := [seq(w1[i]*denom(t)+numer(t)*tau[i],i=1..n)];
         d := igcd(op(w1));
         w1 := [seq(w1[i]/d, i=1..n)];

# -------------------------------------------------------------
# compute the w-initials
# -------------------------------------------------------------

         Gw := grobnerwalk[winitials](G, w1, X);

# -------------------------------------------------------------
# convert the w-initial Groebner basis
# -------------------------------------------------------------

         Gnew :=  grobnerwalk[mongb](Gw, X, w1  );

# -------------------------------------------------------------
# compute the initial of the new Groebner basis
# -------------------------------------------------------------

        Glpp := [seq(grobnerwalk[initial](Gnew[i], X, w1 ),i=1..nops (Gnew))];

# -------------------------------------------------------------
# compute the transformation matrix
# -------------------------------------------------------------

        T := grobnerwalk[transmat](Gnew,Gw,X,oldw  );

# -------------------------------------------------------------
```

```
# lift the new Groebner basis
# ------------------------------------------------------------

        G := grobnerwalk[lift](G , T,Glpp,  X, w1 );

    od

end:
```

```
# Groebner Walk
#
#  detborder computes the next border of cones
#
# param.:  w ... weight vector
#                tau ...  weight vector
#                G ... list of polynomials
#                Glpp ... list of initials of G
#                X  ... list of variables
#
# output:  t the parameter for the next border on the line segment w-tau
# -------------------------------------------------------------

`grobnerwalk/detborder` := proc(w, tau, G,Glpp, X )

    local t, n, g, h, m, i, j, d, q, r, t1, p, k, q1, r1;

    t := 2;
    n := nops(X);

# -------------------------------------------------------------
# run trough all polynomials g of G which are no monomials
# -------------------------------------------------------------

    for k to nops(G) do
    if type(G[k], `+`) then
        g := G[k];
        h := Glpp[k]  ;
        m := nops(g);

# -------------------------------------------------------------
#  compute the dotproduct of tau and the exponent vector of the initial of g
#  compute the dotproduct of w and the exponent vector of the initial of g
# -------------------------------------------------------------

        r1 := 0; q1 := 0;
        for j to n do
            d := degree(h, X[j]);
            r1 := r1-tau[j]*d;
            q1 := q1+w[j]*d
        od;

# -------------------------------------------------------------
# run trough all monomials  of g and determine the possible parameter t
# -------------------------------------------------------------

        for i  to m do
            r := 0; q := 0;
            for j to n do
                d := degree(op(i, g), X[j]);
                r := r-tau[j]*d;
                q := q+w[j]*d
            od;
            if (r1-r) <> 0 then
                t1 := (q1-q)/(r1-r);
                if (t1 > 0) and (t1 < t) then t := t1 fi
            fi
        od

    fi
    od;
```

```
# -------------------------------------------------------------
# if no parameter t<1 is found then t:=1
# -------------------------------------------------------------

    if t > 1 then t:=1 fi;
    t

end:
```

```
# Groebner Walk
#
# winitials computes the list of w-initials of a list of polynomials G
# w.r.t. the weight vector w
#
# param.:  G ... list of polynomials
#                w ... weight vector
#                X ... list of variables
#
# output:  Gw list of w-initials
# --------------------------------------------------------------

`grobnerwalk/winitials` := proc(G, w, X)

    local Gw, g, n, initial, maxdeg, mondeg, i, j,m, mon;

    Gw := [];
    m := nops(w);
    for g in G do
        if type(g, `+`) then
            n := nops(g);
            initial := op(1,g);
            maxdeg := 0;
            for j from 1 to m do
                    maxdeg := maxdeg+w[j]*degree(initial,X[j])
            od;
            for i from 2 to n do
                mon := op(i,g);
                mondeg := 0;
                for j from 1 to m do
                    mondeg := mondeg+w[j]*degree(mon,X[j])
                od;
                if mondeg > maxdeg then
                    maxdeg := mondeg;
                    initial := mon
                elif mondeg = maxdeg then
                    initial := initial + mon
                fi
            od
        else
            initial := g
        fi;
        Gw := [op(Gw), initial]
    od;

    Gw

end:
```

```
# Groebner Walk
#
# mongb computes the reduced, minimal Groebner basis
# of the polynomials F in the indeterminates X with respect to the
# comparing monomials w.r.t. the weight vector w and breaking ties
# w.r.t. the lexicographical ordering.
#
# param.:  F ... list of polynomials
#                X  ... list of variables
#                w ... weight vector
#
# output:  G the Groebner basis of F w.r.t. ordering described above
# -------------------------------------------------------------

`grobnerwalk/mongb` := proc(F, X, w)

local G, G1, Glpp, polyflag, i, j, Cp, cp, spol , h, redundant,k,c,g, glpp,
         polyflag1,t;

# -------------------------------------------------------------
# compute the initials of the given polynomials and check which
# of the polynomials are monomials
# -------------------------------------------------------------

G := F;
Glpp := [seq(grobnerwalk[initial](G[i], X, w),i=1..nops(G))];
polyflag := [seq(type(G[i],`+`), i=1..nops(G))];

# -------------------------------------------------------------
# compute all critical pairs (never consider two monomials
# -------------------------------------------------------------

Cp := [];
for i from 1 to nops(G) do
    for j from i+1 to nops(G) do
        if polyflag[i] or polyflag[j] then
            Cp := [op(Cp), [ i ,  j ]];
        fi;
    od;
od;

# -------------------------------------------------------------
# apply the Buchberger algorithm to compute a Groebner basis
# -------------------------------------------------------------

while Cp <> [] do
    cp := Cp[1];
    Cp := subsop(1=NULL, Cp);
    c := tcoeff(lcm(Glpp[cp[1]], Glpp[cp[2]]),X,'h');
    spol :=  expand(c*(h/Glpp[cp[1]]*G[cp[1]]-h/Glpp[cp[2]]*G[cp[2]])) ;
    spol := grobnerwalk[normalform](spol, G, Glpp, X, w);
    if spol <> 0 then
        polyflag1 := type(spol, `+`);
        for  i to nops(G) do
            if polyflag1 or polyflag[i] then
                Cp := [op(Cp), [i, nops(G)+1]];
            fi;
        od;
        G := [op(G), spol];
        Glpp := [op(Glpp), grobnerwalk[initial](spol, X, w)];
        polyflag := [op(polyflag), polyflag1];
```

```
    fi;
od;

# --------------------------------------------------------------
# delete all redundant polynomials from the Groebner basis
# (minimal Groebner basis)
# --------------------------------------------------------------

redundant := true;
while redundant do
    redundant := false;
    for i to nops(G) while not(redundant) do
    for j to nops(G)  while not (redundant) do
        if divide(Glpp[i], Glpp[j], 't') and i<>j then redundant := true; k:=i  fi;
    od;
    od;
    if  redundant  then
         G  := subsop(k=NULL, G);
         Glpp  := subsop(k=NULL, Glpp);
    fi;
od;

# --------------------------------------------------------------
# reduce the Groebner basis
# --------------------------------------------------------------

for i to nops(G) do
        g := grobnerwalk[normalform](G[i]-Glpp[i], subsop(i=NULL, G ),
                subsop(i=NULL, Glpp), X,w, Glpp[i]);
        glpp := grobnerwalk[initial](g, X, w);
        G  := subsop(i= g, G );
        Glpp := subsop(i=glpp, Glpp)
    od;

G

end:
```

```
# Groebner Walk
#
#  initial computes the initial of a polynomial f w.r.t the term ordering
#  constructed by the weight vector w and breaking ties with the
#  plex ordering
#
# param.:  f ... a polynomial
#                X ... list of variables
#                w ... weight vector
#
# output: lp the initial of f
# -------------------------------------------------------------

`grobnerwalk/initial` := proc(f, X, w)

     local i,w1,wnew,lp,t,c;

     options system, remember;

# -------------------------------------------------------------
# if f is a real polynomial (no monomial) we compute a value for the
# exponent of each monomial of f and determine the maximum
# -------------------------------------------------------------

     if type(f,`+`) then
        c := coeffs(f, X,'t');
        t:=[t]; c:=[c];
        w1 := grobnerwalk[weight](t[1], X, w);
        lp := c[1]*t[1];
        for i from 2 to nops(t) do
            wnew := grobnerwalk[weight](t[i], X, w);
            if w1< wnew then
                w1 := wnew;
                lp := c[i]*t[i]
            fi
        od

# -------------------------------------------------------------
# if f is a monomial it is its initial
# -------------------------------------------------------------

   else
        lp := f
    fi;

   lp

end:
```

```
# Groebner Walk
#
# lifts the new initial Groebner basis to a Groebner basis of the ideal
# and reduces it
#
# param.:  G ... list of initial forms
#                T ... transformation matrix
#                Glpp ... list of initials of G
#                X ... list of variables
#                w ... weight vector
#
# output: G1 lifted and reduced Groebner basis
# --------------------------------------------------------------

`grobnerwalk/lift` := proc(G, T,Glpp,  X, w)

    local G1,Gn, g, g1, m, n, k, i, f, G2, j, hpoly,Glpp1 ,c,pt,glpp;

# --------------------------------------------------------------
# setting initial values
# --------------------------------------------------------------

    G1 := [];
    Glpp1 := [];
    n := nops(T);
    m := nops(G);

# --------------------------------------------------------------
# aplying the transformation matrix T to G
# --------------------------------------------------------------

    for i from 1 to n do
        f := 0;
        for j to m do
             f := f+G[j]*T[i][j];
        od;
        f :=  primpart(expand(f));
        G1 := [op(G1), f ];
        tcoeff(Glpp[i],X,'pt' );
        c := coeff(subs(pt=`grobnerwalk/newvariable`, f),
                 `grobnerwalk/newvariable`);
        Glpp1 := [op(Glpp1), c*pt];
    od;

# --------------------------------------------------------------
# reduction of the Groebner basis G1
# --------------------------------------------------------------

    for i to nops(G1) do
        g := grobnerwalk[normalform](G1[i]-Glpp1[i], subsop(i=NULL, G1),
               subsop(i=NULL, Glpp1), X,w, Glpp1[i]);
        glpp := grobnerwalk[initial](g, X, w);
        G1 := subsop(i= g, G1);
        Glpp1 := subsop(i=glpp, Glpp1);
    od;

    G1;

end:
```

```
# Groebner Walk
#
#  transmat computes the matrix which transform the old Groebner basis G
#  to the new Groebner basis F
#
# param.:  F ...  list of polynomials
#                G ...  list of polynomials
#                X ... list of variables
#                w ... weight vector
#
# output:  Mat the transformation matrix
# ------------------------------------------------------------

`grobnerwalk/transmat` := proc(F,G,X,w)

    local poly,Mat,m;

# ------------------------------------------------------------
# compute the transformation vector for every polynomial in F
# ------------------------------------------------------------

    Mat := [];
    for poly in F do
        m:= grobnerwalk[transvect](poly,G,X,w);
        Mat := [op(Mat), m]
    od;
    Mat

end:
```

```
# Groebner Walk
#
# transvect reduces the polynomial poly to 0 w.r.t. G
#
# param.:  poly ... a polynomial
#                G ... list of polynomials
#                X ... list of variables
#                w ... weight vector
#
# output:  k the reduction vector
# --------------------------------------------------------------

`grobnerwalk/transvect` := proc(poly,G,X,w )

    local  HT,HC,m,f,k,ft,fc,i,j,Glpp,flpp,ht,t,u;

# --------------------------------------------------------------
# compute initial values
# --------------------------------------------------------------

    Glpp := [seq(grobnerwalk[initial] (G[j],X,w),j = 1 .. nops(G))];
    HC := []; HT:= [];
    for i to nops(Glpp) do
    HC := [op(HC), tcoeff(Glpp[i], X, 'ht')];
    HT := [op(HT), ht];
    od;
    m := nops(G);
    f := expand(poly);
    k := [seq(0 ,i=1..nops(G))];

# --------------------------------------------------------------
# reduce the polynomial to 0
# --------------------------------------------------------------

    while f <> 0 do
            for i from 1 to m do
            flpp := grobnerwalk[initial](f,X,w);
            fc := tcoeff(flpp,X, 'ft');
            if divide(ft, HT[i], 'u') then
                f := expand(-fc*u*G[i] + HC[i]*f);
                k := [seq(k[j]*HC[i] ,j=1..m)];
                k := subsop(i=k[i]+fc*u,k)
            fi
        od
     od;

     k

end:
```

```
# Groebner Walk
#
# checks if a weight vector is in the interior of a
# cone with respect to an ideal
#
# param.:  Gw ... list of w-initial forms
#
# output:  true if a weight vector is in the interior of a cone else false
# -------------------------------------------------------------

`grobnerwalk/intcone` := proc(Gw)

    local g;

# -------------------------------------------------------------
# if there exists an w-initial in Gw which is not a monomial
# then w is not in the interior of a cone
# -------------------------------------------------------------

    for g in Gw do
        if type(g, `+`) then RETURN(false) fi
    od;

    true

end:
```

```
# Groebner Walk
#
#  memcone checks if we do not leave the cone when
#  perturbing the starting vector
#
# param.:  Glpp ... list of initials
#                Gw ... list of initial forms
#
# output:  true if we stay in the cone, else false
# --------------------------------------------------------------

`grobnerwalk/memcone` := proc(Glpp, Gw)

    local g,i,j,inflag,p;

    inflag := true;

# --------------------------------------------------------------
# checks for every initial form in Gw if it contains a monomial of Glpp
# --------------------------------------------------------------

    for i from 1 to nops(Gw) do
        g := Gw[i];
        if type(g, `+`) then
            if not(member(primpart(Glpp[i]),[seq(primpart(op(j,g)),j=1..nops(g))],
                       'p')) then inflag := false fi
        else
            if primpart(g)<>primpart(Glpp[i]) then inflag := false fi
        fi
    od;

    inflag

end:
```

```
# Groebner Walk
#
# normalform computes the normal form of a polynomial f w.r.t to a
# list of polynomials F
#
# param.:  f ... a polynomial
#               F ... list of polynomials
#               Flpp ... list of initials of F
#               X ... list of variables
#               w ... a weight vector
#               ht2 ... initial of f (optional)
#
# output:  h the reduced polynomial
# ---------------------------------------------------------------

`grobnerwalk/normalform` := proc(f,F, Flpp,X,w, ht2)

    local g, redflag, m,n, gl, i,j,c,h,ht,t, Glpp, G ,htw, G1, G1lpp,ht1,u ;

# ---------------------------------------------------------------
# if the function is called with 5 arguments then ht2 is the initial of the
# polynomial ht2+f and ht2 is irreducible
# ---------------------------------------------------------------

    g := f;
    redflag := true;
    m := nops(G);
    if nargs = 5 then
        h := 0
    else
        h := ht2;
    fi;
    G := F;
    Glpp := Flpp;

# ---------------------------------------------------------------
# reduce the polynomial g w.r.t. F and eliminate elements from F if
# possible in every step
# ---------------------------------------------------------------

    while g<>0 do
        redflag := false;
        ht := grobnerwalk[initial](g, X, w);
        tcoeff(ht, X, 'ht1');
        htw := grobnerwalk[weight](ht1 , X, w);
        G1 := []; G1lpp := [];
        for j to nops(G) do
            if grobnerwalk[weight](Glpp[j], X, w) <= htw then
                G1 := [op(G1), G[j]];
                G1lpp := [op(G1lpp), Glpp[j]]
            fi
         od;
         if G1 = [] then
             RETURN(primpart(h+g))
         else
             G := G1;
             Glpp := G1lpp
         fi;
         for j to nops(G) while not(redflag) do
             if divide(ht,Glpp[j],'u') then
                 c:=tcoeff(u,X,'t');
```

```
                g :=  expand(denom(c)*g-numer(c)* t*G[j] ) ;
                h := h*denom(c);
                redflag := true
            fi
        od;
        if not(redflag) then
            g :=  g-ht ;
            h := h+ht
        fi;

    od;

    primpart(h)

end:
```

```
# Groebner Walk
#
#  calculates a value for the exponent of the monomial mon
#
# param.:  mon ... a monomial
#                X ... list of variables
#                w ... weight vector
#
# output:  s value of exponent
# ---------------------------------------------------------------

`grobnerwalk/weight` := proc(mon, X, w)

    local s,i, n ;

    options system, remember;

    n := nops(X);
    s := 0;
    for i to n  do

        s := s+degree(mon,X[i])*w[i]
    od;


    for i from 1 to n  do
         s := 1000*s + degree(mon,X[i])
    od;

end:
```