# Quotient Multiplication Table

Fabrice Rouillier

*Deliverable 3.2.2.3.1*

# Contents

# 1 Introduction

We suppose that $(f_1, \ldots, f_s) \in D[X_1, \ldots, X_n]$ is a zero-dimensional system (with a finite number of solutions) of multivariate polynomials with coefficients in a domain $D$. In this case, if we note $K$ the fraction field associated to $D$, $A = \frac{K[X_1,\ldots,X_n]}{\langle f_1,\ldots,f_s \rangle}$ is a finite vector space. After having computed a monomial basis $\mathcal{B} = \{\omega_1, \ldots, \omega_D\}$ of $A$ through a reduced Groebner basis $\mathcal{G} = \{g_1, \ldots g_u\}$ of $I = \langle f_1, \ldots, f_s \rangle$, the goal is to compute efficiently the multiplication table of $A$ (i.e. : all the values $\omega_i \omega_j$ expressed in $A$ with respect to the basis $\mathcal{B}$.), avoiding or optimizing, as most as possible the use of the *NormalForm* function.

Let's take a simple example :
Consider $\mathcal{I} = \langle X_1^3 - 2, X_2^3 - 3 \rangle \subset K[X_1, \ldots X_2]$. A monomial basis of $K[X_1, \ldots X_2]/\mathcal{I}$ is given by :

$$\mathcal{B} = \left\{ \begin{array}{l} \omega_1 = 1, \omega_2 = X_1, \omega_3 = X_1^2, \omega_4 = X_2, \omega_5 = X_2^2, \\ \omega_6 = X_1X_2, \omega_7 = X_1^2X_2, \omega_8 = X_1X_2^2, \omega_9 = X_1^2X_2^2 \end{array} \right\},$$

and the monomials that define the multiplication table are :

| $\times$ | $\omega_1$ | $\omega_2$ | $\omega_3$ | $\omega_4$ | $\omega_5$ | $\omega_6$ | $\omega_7$ | $\omega_8$ | $\omega_9$ |
|---|---|---|---|---|---|---|---|---|---|
| $\omega_1$ | $1$ | $X_1$ | $X_1^2$ | $X_2$ | $X_2^2$ | $X_1X_2$ | $X_1^2X_2$ | $X_1X_2^2$ | $X_1^2X_2^2$ |
| $\omega_2$ | $X_1$ | $X_1^2$ | $X_1^3$ | $X_1X_2$ | $X_1X_2^2$ | $X_1^2X_2$ | $X_1^3X_2$ | $X_1^2X_2^2$ | $X_1^3X_2^2$ |
| $\omega_3$ | $X_1^2$ | $X_1^3$ | $X_1^4$ | $X_1^2X_2$ | $X_1^2X_2^2$ | $X_1^3X_2$ | $X_1^4X_2$ | $X_1^3X_2^2$ | $X_1^4X_2^2$ |
| $\omega_4$ | $X_2$ | $X_1X_2$ | $X_1^2X_2$ | $X_2^2$ | $X_2^3$ | $X_1X_2^2$ | $X_1^2X_2^2$ | $X_1X_2^3$ | $X_1^2X_2^3$ |
| $\omega_5$ | $X_2^2$ | $X_1X_2^2$ | $X_1^2X_2^2$ | $X_2^3$ | $X_2^4$ | $X_1X_2^3$ | $X_1^2X_2^3$ | $X_1X_2^4$ | $X_1^2X_2^4$ |
| $\omega_6$ | $X_1X_2$ | $X_1^2X_2$ | $X_1^3X_2$ | $X_1X_2^2$ | $X_1X_2^3$ | $X_1^2X_2^2$ | $X_1^3X_2^2$ | $X_1^2X_2^3$ | $X_1^3X_2^3$ |
| $\omega_7$ | $X_1^2X_2$ | $X_1^3X_2$ | $X_1^4X_2$ | $X_1^2X_2^2$ | $X_1^2X_2^3$ | $X_1^3X_2^2$ | $X_1^4X_2^2$ | $X_1^3X_2^3$ | $X_1^4X_2^3$ |
| $\omega_8$ | $X_1X_2^2$ | $X_1^2X_2^2$ | $X_1^3X_2^2$ | $X_1X_2^3$ | $X_1X_2^4$ | $X_1^2X_2^3$ | $X_1^3X_2^3$ | $X_1^2X_2^4$ | $X_1^3X_2^4$ |
| $\omega_9$ | $X_1^2X_2^2$ | $X_1^3X_2^2$ | $X_1^4X_2^2$ | $X_1^2X_2^3$ | $X_1^2X_2^4$ | $X_1^3X_2^3$ | $X_1^4X_2^3$ | $X_1^3X_2^4$ | $X_1^4X_2^4$ |

This table contains 81 monomials but only 25 are distinct.

Our objective is to represent the quotient $A$ as a coefficient ring. We describe in the report the internal implementation of the multiplication table and we give in the conclusion a detailed workplan that will drive to a full integration of a new type of coefficients inside the library.

# 2 Description

The internal process that have been implemented is fully described in [1]. IT consists in a Hashing table that uses internally algebraic properties to order its elements.

```
Package:  RealSolving/Multi
type:     -*- C++ -*-
file:     MultTable.H
date:     23 February 1996
author:   Fabrice Rouillier
```

- Configurations of the PoSSo-Library supported : All
- Limitations in the use : none.

The multiplication table is modelized inside the C++ library by a class with only one constructor :

```
PL_MultTable (PL_GroebnerReductor * g);
```

Creates (and compute) the table associated to a Groebner basis g when not using a Gb server. Warning : the basis must be computed before calling the constructor .
Then many features are accessible through methods of the `PL_MultTable` class :

```
PL_HVector & redForm(const unsigned int i,const unsigned int j)
```

returns the expression of $\omega_i\omega_j$ in $A$ with respect to the monomial basis $\{\omega_1, \ldots, \omega_D\}$.

```
PL_HVector & redFormX(const unsigned int i,const unsigned int j)
```

returns the expression of $X_i\omega_j$ in $A$ with respect to the monomial basis $\{\omega_1, \ldots, \omega_D\}$.

```
unsigned int position(const unsigned int i,const unsigned int j)
```

returns the position where the expression of $\omega_i\omega_j$ in $A$ with respect to the monomial basis $\{\omega_1, \ldots, \omega_D\}$ is stored in the table.

```
PL_HVector & value(const unsigned int i)
```

returns the expression stored at the $ith$ position in the table.

```
PL_HMatrix & matMultVar(const unsigned int i)
```

returns the matrix of multiplication by the $(i + 1)th$ variable in $A$.

```
PL_HMatrix & matMultFL(PL_HVector & v)
```

returns the matrix of multiplication by the the vector $v$ (element of $A$) in $A$.

```
int fieldDimension()
```

returns the dimension of $A$ as $K$-vector space.

```
int tableDimension()
```

returns the number of distinct elements in the table.

```
PL_HVector & ReducedExpression(PL_Poly &);
```

converts a polynomial into an element of $A$ (canonical projection).

```
PL_RSList<PL_RSMon>  GetBasis()
```

returns the monomial basis of $A$.

```
PL_PolyList &  GetGroebnerBasis()
```

returns the Groebner basis associated with $A$.

```
PL_GroebnerReductor &  GetGroebnerReductor()
```

returns the GroebnerReductor associated with $A$.

# 3 Example of Use

## 3.1 using the C++ library

Let suppose that a Groebner basis has been computed and stored inside the variable *gr*.

```
/* computation of the multiplication table */
  PL_MultTable mt(&g);

/* Get the matrix of multiplication by the first variable */
  PL_HMatrix Mx(mt.matMultVar(0);
```

## 3.2 Using the PoSSo-Server

Inside the PoSSo-server, the multiplication may be computed only giving it as parameter to other methods (like Hermite or RUR). Also, the methods are not accessible.

```
%
%put the ordering directives into (0) (DRL-ordering)

@
PR_Open
{ {0},
  {4,
        {C, {1, 1, 1, 1}},
        {V, {1, 1, 1, 1}},
        {N, {x, y, z, t}}
  }
}

% declaration of register (1) : List of polynomials
% with respect to (0)

@
PL_Open (0)


% put the system into (1)
%

@
PL_Put (1)
{{1t2,-1t,2z2,2y2,2x2}
,{2tz,2zy,2yx,-1z}
,{2ty,1z2,2zx,-1y}
,{1t,2z,2y,2x,-1}
}

% declaration of register (2) : Groebner Reductor
% with respect to (0)
```

```
@
GR_Open (0)

% Compute the Groebner basis of the system
% (the result is stored into register (2)

@
GR_ComputeBasis (2) (1)  % compute the Groebner Basis
@


% declaration of register (3) : Multiplication table
% associated to (2)

RST_Open (2) % open RST (3)
```

# 4 Conclusion

Our initial goal was to make efficient the computation of the reduction of all the possible pairs of two elements of $A$. As we can see in table [1], this objective has been reached.

Compared with the version that was existing before the FRISCO project, many improvement has been made and the multiplication table can be run now with any type of generic coefficients. The packages HMatrix and HVector used in the computation have now to be optimized in order to avoid unusefull simplifications when using coefficients that are in a field.

A new problem appears now (this is now the only factor that prevents from computing the multiplication table) : the memory occupation of the table. As we can see for some examples in table [2], a large part of the coefficients in the vectors of the table are null. Also, it will be convenient to implement a class of compressed vectors, storing only the non null coordinates.

For using the multiplication table in order to define $A$ as a coefficient ring, only a little interface is needed. An element of $A$ is internally coded by a `HVector`. Many basic operations (addition, substraction, test to zero, ...) are trivial using the linear algebra package. The product function of two elements of $A$ uses in background the Multiplication table.

Before this one must integrate some conversions and extras definitions inside the generic coefficient package of the PoSSo-Lib.

Here is a part the MTCoeff class implemetation that will be integrated in a future release of the the library (see `PL_Coeff` documentation): we define for example the two principal operations (addition and multiplication).

```
class PL_MTCoeff:PL_Coeff
{
public:
  PL_MTCoeff() {
     rep=new PL_HVector();
  }
#ifdef PL_CMM
  void traverse();
#endif

private :
```

```
  PL_HVector *rep;
}


class PL_MTCoeffRing:PL_CoeffRing
{
.
.
  PL_Coeff& add( const PL_Coeff& c1, const PL_Coeff& c2 ) const;
  PL_Coeff& mult( const PL_Coeff& c1, const PL_Coeff& c2 ) const;
.
.
.
private :
  PL_MultTable *mt;
}
```

Where the methods add and mult can be implemented as follow :

```
  PL_Coeff& PL_MTCoeffRing::add( const PL_Coeff& c1, const PL_Coeff& c2 )
  {
PL_MTCoeff &res=*new PL_MTCoef();
        *res.rep=(*((PL_MTCoeff &)c1).rep)+(*((PL_MTCoeff &)c2).rep);
        return((PL_Coeff &) res);
  }

  PL_Coeff& PL_MTCoeffRing::mult( const PL_Coeff& c1, const PL_Coeff& c2 )
  {
     PL_MTCoeff &res=*new PL_MTCoef();
// the affectation operator of HVector do not realize a copy
// of the vector, it only moves pointers
    res.rep=new PL_HVector(mt->fieldDimension(),createCoeff(0));
    PL_HVector *c1rep=((PL_MTCoeff& )c1).rep;
    PL_HVector *c2rep=((PL_MTCoeff& )c2).rep;
    int i,j;
    for (i=0;i<mt->fieldDimension();i++){
      (*res.rep)+=(((*c1)rep[i])*((*c2)rep[i]))*(mt->redForm(i,i));
      for (j=i+1;j<mt->fieldDimension();j++)
(*res.rep)+=(((*c1rep)[i])*((*c2rep)[j])+
     ((*c1rep)[j])*((*c2rep)[i]))*(mt->redForm(i,j));
    }
    return((PL_Coeff &) res);
  }
```

The tables [1] and [2] has been made using examples from the PoSSo test-suite. The correspondence is the following :
e1=caprasse, e2=cassou, e3=sendra, e4=katsura5, e5=katsura7, e6=cyclic5 , e7=reimer5, e8=discret3.
**Legend :**

- D. : dimension of $A$.
- V. : number of variables.

- R. : (reductions computed)/($D^2$)
- L. : binary size of the greatest integer in the result. (i.e : the leastest $i$ such that $2^{32*i}$ is greater).
- T : computation time (in sec.) on a DEC Alpha-600 - 175 Mhz station.
- G.B. : Groebner basis w.r.t. the DRL ordering.
- M.T. : multiplication table :
  - Prep. : initialization of the Hashing table.
  - Bord : computation of the products $X_i \omega_j$
  - Table : computation of the rest of the table.
- Nb : number of effective computations.

| | GB | M.T. | | | | | | | | | D. | V |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Prep. | Bord | | | Table | | | Total | | | |
| Ex. | L. | T. | T. | L. | Nb. | T. | L. | Nb. | T. | R. | | |
| e1 | 1 | 0,1 | 0,31 | 1 | 52 | 0,9 | 1 | 367 | 1,31 | 0,13 | 56 | 4 |
| e2 | 5 | 0,04 | 0,16 | 5 | 20 | 0,04 | 7 | 37 | 0,24 | 0,22 | 16 | 4 |
| e3 | 3 | 0,06 | 0,09 | 3 | 8 | 0,58 | 15 | 110 | 0,73 | 0,06 | 46 | 2 |
| e4 | 6 | 0,02 | 1,42 | 6 | 76 | 1,24 | 14 | 124 | 2,68 | 0,20 | 32 | 6 |
| e5 | 10 | 0,69 | 83,05 | 11 | 402 | 317,78 | 15 | 1489 | 401,52 | 0,12 | 128 | 8 |
| e6 | 1 | 0,11 | 1,05 | 1 | 137 | 1,15 | 1 | 357 | 2,31 | 0,10 | 70 | 5 |
| e7 | 2 | 0,44 | 50,19 | 3 | 217 | 93,16 | 123 | 1250 | 143,79 | 0,07 | 144 | 5 |
| e8 | 9 | 0,81 | 48,213 | 9 | 448 | 174,79 | 13 | 1611 | 223,813 | 0,13 | 128 | 8 |

**Table[1]**

**Legend :**

- NbCoefs : number of coefficients that appear inside the table,
- NbNullCoefs : number of null coefficients that appear inside the table,
- Rapp : NbCoefs/NbNullCoefs.

| example | NbCoeffs | NbNullCoeffs | Rapp |
|---|---|---|---|
| cassou | 1312 | 544 | 2.41 |
| caprasse | 21560 | 4024 | 5.35 |
| cyclic5 | 40880 | 6074 | 6.73 |
| cyclic6 | 290940 | 42755 | 6.80 |
| katsura5 | 8160 | 6190 | 1.31 |
| katsura6 | 48064 | 40660 | 1.18 |

**Table[2]**

# References


[1] F. Rouillier. Algorithmes efficaces pour l'étude des zéros réels des systèmes polynomiaux *Doctoral Thesis*, 1996.