Proving formally the implementation of an efficient gcd algorithm for polynomials

Assia Mahboubi

INRIA Sophia-Antipolis
2204, routes des Lucioles - B.P. 93
06902 Sophia Antipolis Cedex, France,
Assia.Mahboubi@sophia.inria.fr

Abstract. We describe here a formal proof in the Coq system of the structure theorem for subresultants, which allows to prove formally the correction of our implementation of the subresultants algorithm. Up to our knowledge it is the first mechanized proof of this result.

1 Introduction

Automation in formal proofs can greatly benefit from the marriage of proof assistants and computer algebra tools. Unfortunately the languages of these two kinds of tools, both intending to do mathematics on a computer, were not designed to talk to each other and importing the art of computer algebra inside a proof assistant can become a challenging problem. There is at least two different approaches to bridge the gap either using (but not trusting) skillful oracles like Maple and prove correct the result of each computation [11, 6] or integrating computer algebra algorithms inside the proof assistant and provide a machine-checked correctness proof of the procedures [17]. The work we present here goes in this direction. We are using the Coq system [1], which is a proof assistant based on type theory and therefore containing a strongly typed programming language we use for computations. The recent introduction of a compiler [9] to the system allows to expect a reasonable efficiency, taking into account that the average user of a proof assistant does not seek for the same level of performances as the one of a computer algebra system. The latter will expect from the system fast computations that are beyond human reach because they involve large entries, while the former will need a proof producing automated tool for small (but very tedious and repetitive) goals.

We present here an algorithm for computing efficiently greatest common divisors (gcd) for polynomials with coefficients in a unique factorization domain (UFD) (see [8]). A UFD is a ring where it makes sense to define gcd but may be less than a field, like it is the case for integers or polynomial rings on a UFD. Computing polynomial gcds is a fundamental concern of algebraic manipulations: simplification of polynomial expressions, computation of partial fraction expansions, mechanization of proofs in geometry...

The algorithms for polynomial gcd computations implemented in computer algebra systems merely fall into three main classes : pseudo-remainder sequences

based algorithms whose most efficient variant is called subresultant algorithm (introduced by [5]), sparse modular algorithms based on Hensel lemma (see [8]), including probabilistic versions [20], and heuristics, like taking benefit from trivial factorizations or tricks based again on finite field decompositions or on reduction to integer gcds.

Choosing which algorithm will be the most efficient, even on a given entry, is quite tricky and there is no decision procedure for that problem. Most systems implement several methods, and define a default behavior while allowing the user to customize it. According to the study made in [14], Maple will apply sparse modular methods when its heuristic fails, and Macsyma as well as Mathematica try a [20]-like procedure before calling a subresultant algorithm. This paper also points out that the subresultant algorithm turns out to be the fastest in half of the benchmarking problems. Moreover, a subresultants algorithm can be defined and used on arbitrary UFDs, whereas the two other kinds of procedures apply for polynomials with integer base constants. We have chosen to implement and prove formally a subresultant algorithm.

Up to our knowledge the only directly related work has been the preliminary study of Boulmé in [3], which did not lead to a formalization, and it seems that there is not mechanized proof of this result available. However the motivations of our work are strongly connected with the formalization of Buchberger algorithm by Théry [17]. Our feeling is that this well-known algorithm of computer algebra had not been formalized before because, despite an abundant computer algebra literature on this topic (see for example the survey[18]), proofs are technical and seemed to require a large amount of preliminary formalizations. One contribution of this paper is to provide a tractable proof, combining the presentations of [4] and [2] for the so-called structure theorem of subresultants. We also provide an implementation in Coq of this algorithm (following [2]), on top of a library for polynomial arithmetic we have certified in the system. We prove formally the fundamental theorem of subresultants, which leads to the correction of the algorithm implemented.

The paper is organized as follows : in section 2 we introduce our representation of polynomials and defined *pseudo-remainder sequences* (PRS) which generalize the euclidean remainder sequence. Then section 3, after pointing out the complexity trade-off in polynomial gcd computation, defines *polynomial determinants*, studies their links with PRS and describes the corresponding formalization. Finally, in section 4 we define *subresultant polynomials*, prove the fundamental theorem and explain how it yields to the subresultant algorithm. We also give an insight of the complexity and give an example of computation in Coq, before concluding with section 5.

For the sake of readability we try to avoid Coq syntax as much as possible and present here the *informal* proof which underlies our formal development. Coq files can however be retrieved from:

http://www-sop.inria.fr/marelle/Assia.Mahboubi/rech-eng.html

2 Preliminary definitions and formalizations

2.1 Polynomials

In the sequel, we will work with polynomials in $D[X]$, where D is a UFD with characteristic 0. In fact, our motivation is to implement these algorithms for polynomials in $\mathbb{Q}[X_1, \ldots, X_n]$, which are represented as univariate polynomials in X_n with coefficients in $\mathbb{Q}[X_1] \ldots [X_{n-1}]$. In this latter case, $\mathbb{Q}$ being a field, the polynomial ring $D = \mathbb{Q}[X_1] \ldots [X_{n-1}]$ will always be a unique factorization domain.

Polynomials of $D[X]$ are implemented in the *sparse Horner* representation. Given the set D of coefficients, elements of $D[X]$ are inductively defined as being either constants, built from an element of D, or of the form $P \times X^n + p$ where P is an element of $D[X]$, n is a positive integer and p is an element of D.

The positive integer n in the non constant case allows more compact representations for sparse polynomials, but at the same time enables even more terms to represent the same mathematical object. For example, X^2 can be represented as $((1) \times X) \times X + 0$ or as $(1) \times X^2 + 0$. It is however possible to choose as a normal form the most compact of these representations : the one with no head zeros and the one factorized as much as possible. Here is the Coq syntax for this definition:

Inductive $Pol(D : Set) : Set :=$
$|Pc : D \rightarrow Pol\ D$
$|PX : Pol\ D \rightarrow \mathbb{N} \rightarrow D \rightarrow Pol\ D.$

Now assuming that D is equipped with a ring structure, and a decidable equivalence relation which is taken as an equality relation over D (we mean all the operations on D will be compatible with it), it is possible to endow $(Pol\ D)$ with a decidable equality, equaling all the representations of a polynomial, together with a compatible ring structure. We also implement usual operations on polynomials like degree, leading coefficient...

We suppose now that a partial operation of division is available on D. Given two elements x and y of D we suppose that if there exist $a \in D$ such that $y = ax$ then $div(y, x) = a$. Partiality is a sensitive issue in type theory : we choose here to make div total, and $div(y, x)$ will be 0 if the division fails. Therefore $div(y, x)$ fails iff $div(y, x) = 0$ and $y \neq 0$. From now on it is possible to program a (partial) *euclidean division* over $D[X]$, in the usual way, but again giving zero quotient and remainder as soon as one of the divisions performed on coefficients fails.

Remark 2.11 *A nice property of our representation is that it allows the definition of euclidean division by structural induction over the divisor, which leads to a smooth formalization in Coq, which would otherwise require a termination proof for such a recursive definition.*

2.2 Pseudo-remainder and sequences

In the sequel, for $P \in D[X]$, the degree is denoted by $deg(P)$ and the leading coefficient by $lcoef(P)$. In the process of the euclidean division of P by Q, the only "denominators" we may introduce are powers of the leading coefficients of Q and in fact the process of euclidean division of $lcoef(Q)^{deg(P)-deg(Q)+1}P$ by Q, performed in $D[X]$, will never fail. This was already observed and used by Jacobi in 1836 [12] and we call $lcoef(Q)^{deg(P)-deg(Q)+1}P$ the Jacobi factor.

Definition 21 (Pseudo-division) *Let $P = p_0 + \cdots + p_nX^n$ and $Q = q_0 + \ldots q_mX^m$, with $p_n, q_m \neq 0$ and $n \geq m$, be two elements of $D[X]$.*

The unique remainder (resp. the quotient) of the euclidean division of $q_m^{n-m+1}P$ by Q is called pseudo-remainder *(resp.* pseudo-quotient*) of P by Q and denoted by $prem(P,Q)$ (resp. $pquo(P,Q)$). This operation is called* pseudo-division *of P by Q, and $prem(P,Q), pquo(P,Q) \in D[X]$.*

Example 2.21 *$P = X^2$, $Q = 2X + 1$: $pquo(P,Q) = 2X - 1$, $prem(P,Q) = 1$. $P = 2X^2 + 2X$, $Q = 2X + 2$: $pquo(P,Q) = 4X$, $prem(P,Q) = 0$.*

Definition 22 (Similar elements) *Let and $P, Q \in D[X]$. P and Q are* similar *($P \sim Q$) if there exists $a, b \in D$ such that $aP = bQ$.*

The Euclidean algorithm computes the gcd of two polynomials with coefficients in a field by a sequence of euclidean divisions. We can now generalize this algorithm to the case of a ring. Here each step of euclidean division is replaced by a step of *pseudo*-euclidean division, but since the correcting factors we have introduced to be able to perform the division may not be the smallest possible, we introduce the possibility of scalar factorizations in the polynomials by requiring only similarity to the pseudo-remainders:

Definition 23 (Pseudo-remainder sequences (PRS)) *Let $F_1, F_2 \in D[X]$, with $deg(F_1) > deg(F_2)$. Let $F_1, \ldots, F_k \in D[X]$ be a sequence of non-zero polynomials such that:*

$$F_i \sim prem(F_{i-2}, F_{i-1}) \text{ for } i = 3 \ldots k, \quad prem(F_{k-1}, F_k) = 0$$

This sequence is called a pseudo-remainder sequence. From the definitions above,

$$\forall i = 3 \ldots k, \quad \exists \alpha_i, \beta_i \in D \text{ and } \exists Q_i \sim pquo(F_{i-2}, F_{i-1}) \text{ such that}$$

$$\beta_i F_i = \alpha_i F_{i-2} - Q_i F_{i-1} \quad deg(F_i) < deg(F_{i-1})$$

Informally, α ensures that we can perform a euclidean division inside $D[X]$, and β is a scalar factor we know we can remove from the remainder. Again to circumvent the problem of partiality, such sequences of polynomials are encoded in Coq as infinite sequences in $(D[X])_{n\in\mathbb{N}}$, whose elements are zero from a certain k on (we even know thanks to degree decreasing that $k \leq deg(Q) + 2$).

2.3 Reduction to primitive polynomials

Finally, for $P \in D[X]$, one defines its *content* ($cont(P)$) as a gcd of its coefficients. It is unique up the multiplication by units of D. If the coefficients of P are relatively prime, P is said to be *primitive*. If not, the primitive part $pp(P)$ of P is defined by $P = cont(P)pp(P)$.

The gcd of two elements of $D[X]$ is the product of their contents by the gcd of their primitive parts. Moreover, it two polynomials F_1 and F_2 in $D[X]$ are primitive, so is their gcd, hence with the notations of the definition 23, $gcd(F_1, F_2) = pp(F_k)$. We will suppose that we are able to compute gcd on D (again we are interested in the case where D is a polynomial ring $\mathbb{Q}[X_1, \ldots X_n]$) hence from now on we will confine our study to the problem of computing the gcd and subresultants of two primitive polynomials, with distinct degrees.

3 Polynomial determinants and PRS

In this section, we consider $F_1 = p_0 + \cdots + p_n X^n$ and $F_2 = q_0 + \ldots q_m X^m$ where $p_n, q_m > 0$ and $n > m$ two polynomials in $D[X]$ and $(F_i)_{i=1\ldots k}$ a PRS, such that for $i = 3...k$:

$$\beta_i F_i = \alpha_i F_{i-2} - Q_i F_{i-1} \quad deg(F_i) < deg(F_{i-1}) \quad (1)$$

We denote $n_i = deg(F_i)$ and $c_i = lcoef(F_i)$ for $i = 1...k$ and call (1) a *pseudo-euclidean relation*.

3.1 Control over the growth of coefficients

Computing efficiently the gcd of two polynomials is computing efficiently the last non zero element of their PRS. The naivest way of computing a PRS is choosing $F_i = prem(F_{i-2}, F_{i-1})$. This PRS is called the *Euclidean PRS* (after [5]). Unfortunately this may lead to a dramatical increase in the size of the coefficients of the polynomials in the PRS. In fact, the bit-size of the coefficient grows exponentially: an exponential lower bound is given by Yap in [19] and Knuth describes this phenomenon in [13]:"Thus the upper bound [...] would be approximately $N^{0.5^{(2.414)^n}}$ and experiments show that the simple algorithm does in fact have this behavior, the number of digits in the coefficients grows exponentially at each step!". Moreover according to [18] : "In a single division, say with random inputs, one cannot do much better than Jacobi's pseudo-division in trying to keep the remainder integral. But the results in Euclid's algorithm are so highly dependent that there are always large factors that can be extracted".

On the other hand, choosing $F_i = pp(prem(F_{i-2}, F_{i-1}))$ minimizes the growth of theses coefficients. This PRS is called the *primitive PRS* (again after [5]). But recursive computations of gcds for each division step is in the general case too expensive.

The *Subresultant PRS algorithm* were are going to present is a compromise between the two preceding solutions, removing at each pseudo-division step a significant factor which is easier to compute than the content. This means it

predicts suitable values for the α_i's and β_i's (using notations of 23), which ensure a reasonable (bit-size quadratic) growth for the coefficients of the polynomials.

3.2 An example of computations

We reprint here the same example as in [18] to compare these three approaches. The α_i's are always the Jacobi factor of the pseudo-division, and the β_i's are factors we can extract from the pseudo-remainder.

i	α_i	β_i	F_i
1			$9X^6 - 27X^4 - 27X^3 + 72X + 18X - 45$
2			$3X^4 - 4X^2 - 9X + 21$
3	$3^3 = 27$	1	$-297x^2 - 729X + 1620$
4	-26198073	1	$3245333040X - 4899708873$
5	10532186540515641600	1	-1659945865306233453993

Euclidean PRS : no factorization ($\beta_i = 1$), exponential growth.

i	α_i	β_i	F_i
1			$9X^6 - 27X^4 - 27X^3 + 72X + 18X - 45$
2			$3X^4 - 4X^2 - 9X + 21$
3	$3^3 = 27$	3	$-11X - 27X + 60$
4	-1331	9	$18320X - 27659$
5	335622400	1959126851	-1

Primitive PRS : optimal factorization, expensive recursive computations.

i	α_i	β_i	F_i
1			$9X - 27X^4 - 27X^3 + 72X + 18X - 45$
2			$3X - 4X^2 - 9X + 21$
3	27	3	$297X^2 + 729X - 1620$
4	26198073	-243	$13355280X - 20163411$
5	178363503878400	2910897	9657273681

Subresultant PRS : a compromise, we remove smaller factors than in the Primitive case but computations are much cheaper.

3.3 Polynomial determinants

We now roughly follow the presentation of [2] to introduce *polynomial determinants*. Let $\mathcal{F}_n$ be the set of polynomials in $D[X]$ whose degrees are less than n. It is a finitely generated free module, equipped with the usual monomial basis $\mathcal{B} = X^{n-1} \dots X, 1$.

Proposition 31 (Polynomial determinant definition) *Let $m \leq n$, two integers. There exists a unique multi-linear antisymmetric mapping, denoted* pdet*, from $(\mathcal{F}_n)^m$ to $\mathcal{F}_{n-m+1}$ such that for every $n > i_1 > \dots i_{m-1} > i$:*

$$\begin{cases} pdet_{n,m}(X^{i_1}, \dots, X^{i_{m-1}}, X^i) = X^i & \text{if for every } j < m \ i_j = n - j \\ pdet_{n,m}(X^{i_1}, \dots, X^{i_{m-1}}, X^i) = 0 & \text{otherwise} \end{cases}$$

Proof. Uniqueness comes from antisymmetry and multilinearity, after decomposing the arguments on the basis $\mathcal{B}$.

Let $Mat'(\mathcal{P})$ be the square matrix whose $m-1$ first lines are the $m-1$ first lines of $Mat(\mathcal{P})$, and the last line is built with the polynomials $P_1, \ldots, P_m$:

$$Mat'(\mathcal{P}) = \begin{bmatrix} p^1_{n-1} & \cdots & p^m_{n-1} \\ \vdots & & \vdots \\ p^1_{n-m+1} & \cdots & p^m_{n-m+1} \\ P_1 & \ldots & P_m \end{bmatrix}$$

Now $pdet_{n,m} = det(Mat'(\mathcal{P}))$, where *det* is the usual determinant of matrices, here with polynomial coefficients. □

3.4 Formalization of multilinear applications in Coq

To date there exists no distributed contribution for multilinear algebra in the Coq system. Building such a theory of multilinear algebra on top of the existing Coq contributions on linear algebra finally appeared as a very intricate solution for our purpose: first because of the complexity of the inheritance mechanisms in formalized algebraic structures but also because we did not need the whole theory leading to the construction of determinants over rings. We have hence chosen to define determinants as (computable) functions and to prove on demand their required properties.

To fit the usual definition of multilinear applications, like determinants, let us consider a field K and E_K a vectorial space over K. Let l be the list of elements of E_K, we will compute the multilinear application by recursion over the length of l. We need to specify how to extract coordinates on the basis we have chosen, and therefore assume an extra parameter *coord* : nat $\to E_K \to K$. The recursion will transform the problem into the same but with determinants of size one less. Since our coordinate parameter is global, it also depends on an integer n, which is the dimension of the vector space. Now we claim that if n is the length of l, under the assumption that we know how to compute *det* for arguments of size $n-1$, we can compute (*det* l) recursively and this job will be performed by the *rec_det* function (whose description we postpone).

Definition det l := det_aux (length l) l.
let rec det_aux(n:nat)(l:list E_K):K :=
match n with
|$O \Rightarrow 1_K$
|$n_1 + 1 \Rightarrow$ rec_det (coord n) (det_aux n_1) l nil
end.

To compute the determinant, we will develop along, say one line, and recursively compute the cofactors, and finally build the appropriate linear combination of the cofactors, with alternate signs. Cofactors are determinants of the sublists

of length $n-1$ of l. The line along which the development is performed is chosen by the values of *coord*. Now *rec_det* is defined by induction on the structure of l.

let rec rec_det(f:$E_K \to K$)(rec : list $E_K \to K$)($l_1 l_2$:list E_K):K:=
match l_1 with
|$nil \Rightarrow 0_K$
|$a :: l_3 \Rightarrow$ f(a)*[rec (app l_2 l_3)] - [rec_det f rec l3 (app l2 (a::nil))]
end.

The assumptions we have made on K and E_K were only for sake of clarity : in fact the only requirements are that:

- K is a commutative integral ring
- E_K is a set equipped with an internal additive law and an external linear product $E_K \to K \to E_K$, and a linear *coord* operator.

Proving that we can develop a determinant along a line is granted from the definition of *det*. We can now also prove formally that *det* is multilinear, antisymmetric, and alternate, by proving it successively for *det_rec* and *det_aux*, of course under the assumption that *coord* is linear. We can also formalize the notion of triangular system, and obtain the value of such a determinant as a product of diagonal values of *coord*.

Taking $K = \mathbb{Z}$ and $E_K = list\ \mathbb{Z}$, we can encode the usual determinant of a square matrix of integers, just by taking:

Definition coord n l := nth (n - 1) l 0.

where *(nth n l a)* computes the n-th element of the list l and that the result is a if n is out of bounds.

We would like to define the polynomial determinant as an application of type $(list\ D[X]) \to D[X]$, and this definition of *det* allows us to do so. Here are the signatures of the auxiliary functions:

$$coord\ :\ nat \to D[X] \to D[X]$$
$$det\_aux\ :\ nat \to (list\ D[X]) \to D[X]$$
$$rec\_det\ :\ (D[X] \to D[X]) \to ((list\ D[X]) \to D[X]) \to (list\ D[X]) \to D[X]$$

We also need to precise the definition of *coord*. It corresponds to the development of the matrix $Mat'(\mathcal{P})$ along the penultimate line, because this is the way to get a definition of *pdet* by induction on the number of polynomials. We also need to give as a parameter *max_degree* (it was $n-1$ in definition 31) the maximal degree of polynomials involved, which determines the number of possible zero lines on top of $Mat'(\mathcal{P})$. This leads to the following definition:

Definition coord max_degree j : $\to D[X] \to D[X]$:=
if max_degree + 2 $\leq$ j
then (fun $P : Pol \Rightarrow P0$)
else
match j with
|$O \Rightarrow$ (fun $P : Pol \Rightarrow P0$)
|$1 \Rightarrow$ (fun $P : Pol \Rightarrow P$)
|_ $\Rightarrow$ (fun $P : Pol \Rightarrow (-1)^{j+1} p_{\text{max\_degree}-j+2)}$)
end.

where p_k is the coefficient of P on X^k, here viewed as a constant polynomial. Now we are ready to define *subresultant polynomials*.

4 Structure of subresultant polynomials

4.1 Definition and first properties

Definition 41 (Subresultant polynomials) *Let $P, Q \in D[X]$ be two polynomials with $deg(P) = n$, $deg(Q) = m$ and $n > m$. Then for $i = 0 \dots n$, the i-th* subresultant polynomial $S_i(P, Q)$ *is defined by:*

- $S_n(P, Q) = P$
- $S_i(P, Q) = 0 \quad$ *for* $m < i < n$
- $S_i(P, Q) = pdet_{n+m-i,n+m-2i}(X^{m-i-1}, \dots, XP, P, Q, XQ, \dots, X^{n-i-1}Q)$ *otherwise*

Going back to the interpretation of *pdet* as a determinant of a matrix of polynomials (section 3.3), we can observe that:

$$S_i(P,Q) = det\, \mathbf{M}_{n_i} \text{ where } \mathbf{M}_{n_i} = \begin{bmatrix} p_n & & 0 & q_m & & 0 \\ \vdots & \ddots & & \vdots & \ddots & \\ p_{n-m+i+1} & \cdots & p_n & & & \\ & & & q_{m-n+i+1} & \cdots & q_m \\ \vdots & & \vdots & \vdots & & \vdots \\ p_{2i+2-m} & \cdots & p_{i+1} & q_{2i+2-n} & \cdots & q_{i+1} \\ X^{m-i-1}P & \dots & P & X^{n-i-1}Q & \dots & Q \end{bmatrix}$$

Using the notations of section 3 and considering a PRS $F_1, \dots, F_k$, from the relation (1) we can obtain Bezout-like relations by induction on i. Indeed for all $i = 3 \dots k$, there exists $\gamma_i \in D$ and U_i, V_i in $D[X]$ such that :

$$U_i \times F_1 + V_i \times F_2 = \gamma_i F_i \quad deg(U_i) < m - n_{i-1}, \quad deg(V_i) < n - n_{i-1}$$

Reversing the problem, this relation can be seen as a system of linear equations in the coefficients of U_i and V_i, considering the relations equaling coefficients of like powers on both sides. Gathering the last $n_i + 1$, inhomogeneous, equations in a single linear polynomial equation, this system can be described by

$$\mathbf{M}_{n_i} \times V = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \gamma_i F_i \end{bmatrix}$$

where V is the column vector of the coefficients of U_i and V_i in decreasing order of subscript. There are in fact relations between the subresultant polynomials of two polynomials P and Q and the polynomials of a PRS starting

with P and Q. From now on, we will drop both subscripts of *pdet*; unless otherwise specified, $n+1$ will always be the maximal degree of polynomials given in arguments.

The following lemma (see [4]) establishes that $S_j(P,Q)$ is a multiple of $S_j(Q, prem(P,Q))$, and this shift will be the elementary step of our main proof.

Lemma 4.11 *Let F, G, H, B be non zero polynomials in $D[X]$, of degree $\phi, \gamma, \eta, \beta$, respectively, such that :*

$$F + BG = H \quad with \quad \phi \geq \gamma > \eta \quad and \quad \beta = \phi - \gamma$$

Then,

$$\begin{array}{lll} (i) & S_j(F,G) = (-1)^{(\phi-j)(\gamma-j)} g_\gamma^{\phi-\eta} S_j(G,H) & 0 \leq j < \eta \\ (ii) & S_\eta(F,G) = (-1)^{(\phi-\eta)(\gamma-\eta)} g_\gamma^{\phi-\eta} h_\eta^{\gamma-\eta-1} H & \\ (iii) & S_j(F,G) = 0 & \eta < j < \gamma - 1 \\ (iv) & S_{\gamma-1}(F,G) = (-1)^{\phi-\gamma+1} g_{\phi-\gamma+1} H & \end{array}$$

Proof. Recall that for $j < \gamma$:

$$S_j(F,G) = pdet(X^{\gamma-j-1}F, \ldots, XF, F, G, XG, \ldots, X^{\phi-j-1}G)$$

In the right hand side, replacing each F by H is adding to each one of the $\gamma - i$ first arguments a linear combination of the $\phi - i$ last ones. Indeed $X^k F + X^k BG = X^k H$ with $\beta + k \leq \phi - j - 1$. The polynomial determinant being multilinear and alternate, this replacement does not alter the value of the *pdet*. Notice that these operations mimic the euclidean division of F by G. We have then:

$$S_j(F,G) = pdet(X^{\gamma-j-1}H, \ldots, XH, H, G, XG, \ldots, X^{\phi-j-1}G)$$

We now come back to the matrix representation of this *pdet*. We denote g_k (resp. h_k) the coefficient of G (resp. H) on the monomial X^k and swap the two blocks of columns:

$$S_j(F,G) = (-1)^{(\phi-j)(\gamma-j)} det \begin{bmatrix} g_\gamma & & 0 & h_\phi & & 0 \\ \vdots & & & \vdots & \ddots & \\ \vdots & \ddots & \vdots & \vdots & & h_\phi \\ \vdots & & \vdots & \vdots & & \vdots \\ g_{\gamma-\phi+j+1} & \cdots & g_\gamma & h_{j+1} & \cdots & h_\gamma \\ \vdots & & \vdots & \vdots & & \vdots \\ g_{2i+2-\phi} & \cdots & g_{j+1} & h_{2j+2-\gamma} & \cdots & h_{j+1} \\ X^{\phi-i-1}G & \ldots & G & X^{\gamma-j-1}H & \ldots & H \end{bmatrix}$$

If $j \geq \eta$, then the matrix is triangular, and

$$S_\eta(F,G) = (-1)^{(\phi-\eta)(\gamma-\eta)} g_\gamma^{\phi-\eta} h_\eta^{\gamma-\eta-1} H \text{ for } \eta \leq j \leq \gamma - 1$$

This proves $(ii) - (iv)$. Now if $j < \eta$, the determinant has the block form:

$$S_j(F,G) = (-1)^{(\phi-j)(\gamma-j)} det \begin{bmatrix} A & 0 \\ B & S_j(G,H) \end{bmatrix}$$

where A is a triangular square block of size $\phi - \eta$, with all elements on its main diagonal equal to g_ϕ, which proves (i). □

Remark 4.11 *The formal proof of this lemma relies on the fact that every polynomial in $D[X]$ of degree less than d is equal to a linear combination of monomials of degree less than d. Due to the choice of our representation (see section 2), we had then to provide a theorem of equivalence of representation. Hereafter we can switch at any moment to the most convenient representation for the current goal to be proved.*

Remark 4.12 *This kind of polynomial identities, like properties of determinants, are proved formally mainly by two kinds of small steps:*

(i) Rewriting of previously established lemmas on polynomials and/or coefficients;

(ii) Reasoning modulo the ring axioms of both polynomial and coefficient structures.

In Coq, a carrier equipped with an equivalence relation is called a setoid *(see [7]). Such a relation can be rewritten, provided that the occurrence concerned is under only* morphisms*, which are functions proved compatibles with the equivalence relation.*

These rewritings (i) may become tedious because of the interweaving of setoids and many-arguments morphisms and this is now very smooth thanks to the recent major improvements of equational reasoning made available in the Coq system (see [16]).

Automatizing (ii) has been made possible by the introduction of a re-shaped tactic for automating proofs and normalization of ring identities ([10]), which enhances the previously available tactic, specially by providing an efficient and convenient tool on abstract (axiomatically defined) structures.

4.2 Subresultants and PRS

In all the sequel $F_1, F_2, \ldots, F_k$ is a PRS in $D[X]$ with notations of section 3. The lemma 4.11 was detailing the behavior of subresultants when combined with euclidean division. We carry on this result and study the link between subresultants and PRS. As we use here multilinearity of *pdet*, we have no longer multiple but similar polynomials. The following lemma is a straightforward corollary of the preceding one:

Lemma 4.21 *For $i = 3 \ldots k$,*

(i) For $0 \leq j < n_i$,

$$S_j(F_{i-2}, F_{i-1})\alpha_i^{n_{i-1}-j} = S_j(F_{i-1}, F_i)\beta_i^{n_{i-1}-j} c_{i-1}^{n_{i-1}-n_i}(-1)^{(n_{i-2}-j)(n_{i-1}-j)}$$

(ii) $S_{n_i}(F_{i-2}, F_{i-1})\alpha_i^{n_{i-1}-n_i} = F_i\beta_i^{n_{i-1}-n_i} c_i^{n_{i-1}-n_i-1} c_{i-1}^{n_{i-2}-n_i}(-1)^{(n_{i-2}-n_i)(n_{i-1}-n_i)}$

(iii) $S_j(F_{i-2}, F_{i-1}) = 0$ *for* $n_i < j < n_{i-1} - 1$

(iv) $S_{n_{i-1}-1}(F_{i-2}, F_{i-1})\alpha_i = F_i\beta_i c_{i-1}^{n_{i-1}-n_{i-1}+1}(-1)^{n_{i-1}-n_{i-1}+1}$.

Proof. This is exactly lemma 4.11 for $F = \alpha_i F_{i-1}$, $G = F_{i-1}$, $H = \beta_i F_i$, $B = -Q_i$, using the multilinear property : $S_j(aF, bG) = a^{\gamma-j}b^{\phi-j}S_j(F, G)$. □

We are now ready to state and prove the fundamental theorem describing the structure of the sequence of subresultant polynomials, telling that once the possible zero polynomials occurring in the subresultant polynomials sequence have been removed, polynomials in the sequence obtained are pairwise similar to ones of the PRS. For sake of readability, we give the exact values of the similarity coefficients only in the proof.

Theorem 4.22 (Fundamental theorem) *With the same notations as above,*

(i) $S_j(F_1, F_1) = 0, \quad$ *for* $0 \leq j < n_k$
(ii) $S_{n_i}(F_1, F_2)$ *and* F_i *are similar.*
(iii) $S_j(F_1, F_1) = 0$ *for* $n_i < j < n_{i-1} - 1$
(iv) $S_{n_{i-1}-1}(F_1, F_2)$ *and* F_i *are similar, for* $i = 3, \ldots k$.

Proof. The exact formulas we are going to prove for (ii) and (iv) are respectively:

$$(ii): \quad S_{n_i}(F_1, F_2) \prod_{l=3}^{i} \alpha_l^{n_{l-1}-n_i} = F_i c_i^{n_{i-1}-n_i-1} \prod_{l=3}^{i} [\beta_l^{n_{l-1}-n_i} c_{l-1}^{n_{l-2}-n_l} (-1)^{(n_{l-2}-n_i)(n_{l-1}-n_i)}]$$

and:

$$(iv): \quad S_{n_{i-1}-1}(F_1, F_2) \prod_{l=3}^{i} \alpha_l^{n_{l-1}-n_{i-1}+1} = F_i c_{i-1}^{1-n_{i-1}+n_i} \prod_{l=3}^{i} [\beta_l^{n_{l-1}-n_{i-1}+1} c_{l-1}^{n_{l-2}-n_l} (-1)^{(n_{l-2}-n_{i-1}+1)(n_{l-1}-n_{i-1}+1)}]$$

Iteration of lemma 4.21(i) for $0 \leq j < n_{i-1}$ and $3 \leq i \leq k+1$ lead to $S_j(F_1, F_2) \sim S_j(F_{i-2}, F_{i-1})$ with the following coefficients:

$$(2) \quad S_j(F_1, F_2) \prod_{l=3}^{i-1} \alpha_l^{n_{l-1}-j} = S_j(F_{i-1}, F_{1-2}) \prod_{l=3}^{i-1} [\beta_l^{n_{l-1}-j} c_{l-1}^{n_{l-2}-n_l} (-1)^{(n_{l-2}-j)(n_{l-1}-j)}]$$

When $i = k + 1$, it means $S_j(F_1, F_2) \sim S_j(F_{k-1}, F_k)$. For $0 \leq j \leq n_k$, lemma 4.11 proves that $S_j(F_{k-1}, F_k) = 0$ hence (i).

Then let $0 \leq i \leq k$. For $j = n_i$, lemma 4.21 (ii) combines with (2) to prove (ii). When $n_i < j < n_{i-1} - 1$, lemma 4.21 (iii) together with (2) proves (iii). And finally 4.21 (iv) and (2) yield (iv). □

4.3 Subresultant polynomials as a pseudo-remainders chain

By definition, every subresultant polynomial is in $D[X]$. The fundamental theorem establishes that, up to deletion of some zeros, subresultant polynomials are pairwise similar to the polynomials of a PRS. Now the idea of a subresultants algorithm is to choose α_i's and β_i's such that the non zeros subresultant polynomials are exactly the F_i of a PRS, the subresultant PRS. Fine customizations are possible in the choice of these α_i's and β_i's and this leads to several algorithms, which are all called subresultant algorithms. The structure theorem for subresultants describes this situation and the choice of α_i's and β_i's. Here we follow the presentation of [2] and implement the corresponding algorithm, computing successive subresultant polynomials by euclidean divisions, and using the following, recursively defined, α_i and β_i:

Theorem 4.31 (Structure theorem for subresultants polynomials) *Let* P *and* Q *be two polynomials in* $D[X]$ *and* $deg(P) = p > q = deg(Q)$. *We denote:* $t_j = lcoef\ S_j(P, Q)$ *and* s_j *the coefficient of* $S_j(P, Q)$ *on* X^j *(which may be zero). Let* $0 \leq j < i \leq p + 1$. *Suppose that* $S_{i-1}(P, Q) \neq 0$ *and is of degree* j.

- *If* $S_{j-1}(P,Q) = 0$ *then* $S_{i-1}(P,Q) = gcd(P,Q)$ *and for* $l \leq j-1, S_l(P,Q) = 0$
- *If* $S_{j-1}(P,Q) \neq 0$ *and* $k = degS_{j-1}(P,Q)$*, then there exists* $Q \in D[X]$ *such that:*
$$s_j t_{i-1} S_{k-1}(P,Q) = QS_{j-1}(P,Q) + s_k t_{j-1} S_{i-1}(P,Q)$$
- *If* $j \leq q, k < j-1$*, then:*
$S_l(P,Q) = 0$ *for* $k < l < j-1$ *and* $t_{j-1}S_k(P,Q) = s_k S_{j-1}(P,Q)$

Notice that unlike in the example given in 3.2, we do not choose the Jacobi factor for α_i. To prove formally the correctness of our procedure we still need to show that this procedure satisfies the properties of 4.31 (but it was implemented on purpose) and also to instantiate 4.22 with the appropriate values of α_i's and β_i's, to prove theorem 4.31.

4.4 Implementation and complexity issues

In particular, the unique factorization domain can be instantiated by integers, and we can go back to our example in the section 3.2. Defining $P := 9X^6 - 27X^4 - 27X^3 + 72X + 18X - 45$ and $Q := 3X^4 - 4X^2 - 9X + 21$, here is the output (instantly) computed by our Coq implementation:

```
Eval compute in (Pol_subres_list P Q).
     = PX (PX (PX (PX (PX (Pc 9) 2 -27) 1 -27) 1 72) 1 18) 1 -45
       :: PX (PX (PX (Pc 3) 2 -4) 1 -9) 1 21
          :: PX (PX (Pc -33) 1 -81) 1 180
             :: PX (Pc 18320) 1 -27659
                :: Pc -1471921 :: nil
     : list Pol
```

Note that this result is slightly better than the one given in the former example, we are indeed here not do far form the primitive PRS, thanks to the efficient choice of α_i's.

One can find in [2] a detailed complexity analysis of the algorithm yield by theorem 4.22. To summarize it, we cite the Prop. 8.43 ([2] p.298)

Theorem 4.41 (Size of the remainders) *If* P *and* Q *have degree* p *and* q *and have coefficients in* $\mathbb{Z}[Y_1, \ldots, Y_k]$*, which have degree* d *in* $Y_1, \ldots, Y_k$ *and are of bit-size* τ *then the degree of* $S_j(P,Q)$ *in* $Y_1, \ldots, Y_k$ *is at most* $d(p+q-2j)$ *and the bit-sizes of the coefficients of* $S_j(P,Q)$ *are at most* $(\tau+\nu)(p+q-2j)+k\mu$ *where* ν *is the bit-size of* $p+q$ *and* μ *is the bit-size of* $(p+q)d+1$*.*

The theoretical (word operations) runtime complexity is $O(n^6)$, for input polynomials of degree $\leq n$ and bit-size of coefficients $\leq n$. Benchmarking Coq's output to get runtime results is not easy, because Coq's time measurement tool is not precise enough. Anyway, our implementation running in Coq is never slower that 20 times the implementation of [14] on the 4 case problems where the subresultant algorithm wins the competition. This means for example that gcds of relatively prime polynomials of degree 10 with 5 variables are computed in less than one second.

5 Conclusion

The benchmarks of the previous section give very satisfying results for a proof assistant system purpose. Computations are made by the reduction engine of Coq, with binary encoded integers but *not* machine integers. This tends to show that the experiment done in [6], where the algorithm rests on gcd computations performed by Maple, could be transposed in a decision procedure, using this kind of implementations (and proofs) to compute inside the system. The main asset of such a self-contained approach is that time of computing is time of proving (the correction proof of the algorithm is done once and for ever). Computations of subresultants are in fact heavily used in a decision procedure for real numbers we have implemented in Coq as well (see [15]). It is a cylindrical algebraic decomposition algorithm (see [5],[2]), for which numerous gcd computations are performed, even in the univariate case, and which relies on algebraic properties of subresultants for projections of multivariate problems. This formal proof is a piece of the (much larger) correction proof we would like to provide for this decision procedure.

One of the main difficulties of this work was to first write a pen-and-paper proof, which would be adapted to formal proof, and though the presentation we adopt here is not new (it mixes [4] and [2]), both papers we have used presented bottlenecks to formal treatments. The historical paper [4] does not not use the convenient definition of polynomial determinants and [2] use the fraction field of a UFD, which requires further development in Coq. We hope that this description is general enough to help the user of another interactive proof assistant wanting to prove this correction theorem and also that our work on formal definition of determinants will be reusable.

This work is still in progress : the correctness proof of the implemented algorithm is not finished and we rely on axiomatic specifications of our euclidean division. The code also needs to be cleaned up in order to be reusable as a stand-alone library for polynomial arithmetic.

We think that despite the large size of the formal proof, this work contributes to the construction of a certified platform in proof assistants enabling the user to forget about the basic manipulation of algebraic expressions. Recent improvements of the system (equational reasoning, automation, tactic metalanguage) have been decisive in the feasibility of such a proof. In [10], an efficient simplification tool was provided for ring expressions, which has now been integrated to the development version of Coq, this work should contribute to be able to enhance rational fraction handling and hence automation in the simplification of field expressions.

Acknowledgments: We would like to thank Laurent Théry for very fruitful discussions and in particular for his suggestions in the formalization of determinants, Laurence Rideau for her significant help in the pedestrian proofs of the ring axioms for polynomials and Marie-Françoise Roy for her detailed explanations on subresultants and elimination theory.

References

1. The coq system. Technical report. http://coq.inria.fr.
2. S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in real algebraic geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer Verlag, 2003. draft for snd edition available at http://name.math.univ-rennes1.fr/marie-francoise.roy/bpr-posted1.html.
3. S. Boulmé. Vers la spécification formelle d'un algorithme non trivial de calcul formel : le calcul de pgcd de deux polynômes par la chaîne de pseudo-restes de sous-résultants. Master's thesis, SPI team, Paris VI University, September 1997.
4. W. S. Brown and J. F. Traub. One euclid's algorithm and the the theory of subresultants. *Journal of the ACM*, 18(4):505–514, 1971.
5. G. E. Collins. Subresultant and reduced polynomial remainder sequences. *Journal of the ACM*, 14:128–142, 1967.
6. D. Delahaye and M. Mayero. Quantifier Elimination over Algebraically Closed Fields in a Proof Assistant using a Computer Algebra System. In *Proceedings of Calculemus 2005*, 2005.
7. V. C. G. Barthe and O. Pons. Setoids in type theory. *Journal of Functional Programming*, 13(2):261–293, March 2003.
8. K. Geddes, S. R. Czapor, and G. Labahn. *Algorithms for computer algebra*. Kluwer Academic Publishers, 1992.
9. B. Grégoire and X. Leroy. A compiled implementation of strong reduction. In *International Conference on Functional Programming 2002*, pages 235–246. ACM Press, 2002.
10. B. Grégoire and A. Mahboubi. Proving ring equalities done right in coq. In *TPHOLs'2005*, LNCS. Springer Verlag.
11. J. Harrison and L. Théry. A skeptic's approach to combining HOL and Maple. *Journal of Automated Reasoning*, 21:279–294, 1998.
12. C. Jacobi. De eliminatione variablilis e duabus aequationibus. *J. Reine Angew. Math*, 1836.
13. D. Knuth. *The Art of Computer Programming, Semi-numerical algorithmms*, volume 2. Addison-Wesley, 1998.
14. H.-C. Liao and R. J. Fateman. Evaluation of the heuristic polynomial gcd. In *ISSAC '95: Proceedings of the 1995 international symposium on Symbolic and algebraic computation*, pages 240–247. ACM Press, 1995.
15. A. Mahboubi. Programming and certifying a cad algorithm in the coq system. In *Mathematics, Algorithms, Proofs*, number 05021 in Dagstuhl Seminar Proceedings. IBFI, Schloss Dagstuhl, Germany, 2006.
16. C. Sacerdoti. A semi-reflexive tactic for (sub-)equational reasoning. In *TYPES 2004*, volume 3839 of *Lecture Notes in Computer Sciences*, pages 98–114. Springer Verlag, 2006.
17. L. Théry. A machine-checked implementation of buchberger's algorithm. *Journal of Automated Reasoning*, 26:107–137, 2001.
18. J. von zur Gathen and T. Lücking. Subresultants revisited. *Theoretical Computer Science*, (297):199–239, 2003.
19. C. K. Yap. *Fundamental Problems of Algorithmic Algebra*. Oxford University Press, 2000.
20. R. Zippel. *Effective Polynomial Computation*. Kluwer Academic Publishers, 1993.