Algorithms for the universal decomposition algebra
Romain Lebreton, Éric Schost

To cite this version:
Romain Lebreton, Éric Schost. Algorithms for the universal decomposition algebra. 2012. <hal-00669806>

HAL Id: hal-00669806
https://hal-polytechnique.archives-ouvertes.fr/hal-00669806
Submitted on 14 Feb 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Algorithms for the universal decomposition algebra

Romain Lebreton
Équipe MAX
LIX, École polytechnique
Palaiseau, France
lebreton@lix.polytechnique.fr

Éric Schost
ORCCA and CS Department
University of Western Ontario
London, ON, Canada
eschosl@uwo.ca

Abstract

Let $k$ be a field and let $f \in k[T]$ be a polynomial of degree $n$. The universal decomposition algebra $A$ is the quotient of $k[X_1, \ldots, X_n]$ by the ideal of symmetric relations (those polynomials that vanish on all permutations of the roots of $f$). We show how to obtain efficient algorithms to compute in $A$. We use a univariate representation of $A$, i.e., an isomorphism of the form $A \simeq k[T]/Q(T)$, since in this representation, arithmetic operations in $A$ are known to be quasi-optimal. We give details for two related algorithms, to find the isomorphism above, and to compute the characteristic polynomial of any element of $A$.

1. INTRODUCTION

Let $k$ be a field and let $f = X^n + \sum_{i=1}^{n} (-1)^i f_i X^{n-i}$ in $k[X]$ be a degree $n$ separable polynomial. We let $\mathcal{R} := \{\alpha_1, \ldots, \alpha_n\}$ be the set of roots of $f$ in an algebraic closure of $k$. The ideal of symmetric relations $\mathcal{I}_n$ is the ideal

$$\{ P \in k[X_1, \ldots, X_n]| \forall \sigma \in \mathcal{S}_n, P(\alpha_{\sigma(1)}, \ldots, \alpha_{\sigma(n)}) = 0\}. $$

It is generated by $(E_i - f_i)_{i=1, \ldots, n}$, where $E_i$ is the $i$th elementary symmetric function on $X_1, \ldots, X_n$. Finally, the universal decomposition algebra is the quotient algebra $A := k[X_1, \ldots, X_n]/\mathcal{I}_n$, of dimension $\delta := n!$. For all $P \in A$, we denote by $X_{P,A}$ its characteristic polynomial in $A$, that is, the characteristic polynomial of the multiplication-by-$P$ endomorphism of $A$. Stickelberger’s theorem shows that

$$X_{P,A}(T) = \prod_{\sigma \in \mathcal{S}_n} (T - P(\alpha_{\sigma(1)}, \ldots, \alpha_{\sigma(n)})) \in k[T].$$

This polynomial is related to the absolute Lagrange resolvent

$$L_P(T) := \prod_{\sigma \in \mathcal{S}_n/\text{Stab } P} (T - P(\alpha_{\sigma(1)}, \ldots, \alpha_{\sigma(n)})) \in k[T],$$

where $\mathcal{S}_n/\text{Stab } P$ are the right cosets of the stabilizer of $P$ in the symmetric group $\mathcal{S}_n$; indeed, these polynomials satisfy the relation $X_{P,A} = L_P^{\text{Stab } P}$.

Computing Lagrange resolvents is a fundamental question, motivated for instance by applications to Galois theory or effective invariant theory. There exists an abundant literature on this question [27, 39, 41, 2, 3, 28, 43, 33, 4]; known symbolic methods rely on techniques involving resultants, symmetric functions, standard bases or invariants (we will make use of some of these ingredients as well). However, little is known about the complexity of these methods. As it turns out, almost all algorithms have at least a quadratic cost $\delta^2$ in the general case.

In some particular cases, though, it is known that resolvents can be computed in quasi-linear time [13]. Our goal in this article is thus to shed some light on these questions, from the complexity viewpoint: is it possible to give fast algorithms (as close to quasi-linear time as possible) for general $P$? What are some particular cases for which better solutions exist? To answer these questions, we measure the cost of our algorithms by the number of arithmetic operations in $k$ they perform. Practically, this is well adapted to cases where $k$ is a finite field; over $k = \mathbb{Q}$, a combination of our results and modular techniques should be used.

The heart of the article, and the key to obtain better algorithms, is the question of which representation should be used for $A$. A commonly used representation is triangular. The divided differences, also known as Cauchy modules [14, 33], are defined by $C_i(X_1) := f(X_1)$ and

$$C_{i+1} := \frac{C_i(X_1, \ldots, X_i) - C_i(X_1, \ldots, X_{i-1}, X_{i+1})}{X_i - X_{i+1}}$$

for $1 \leq i < n$. They form a triangular basis of $\mathcal{I}_n$, in the sense that $C_i$ is in $k[X_1, \ldots, X_i]$, monic in $X_i$, and reduced with respect to $(C_1, \ldots, C_{i-1})$. In particular, they define a tower of intermediate algebras $A_k$ for $1 \leq i \leq n$:

$$A_1 := k[X_1]/\langle C_1 \rangle;$$

$$A_m := k[X_1, \ldots, X_m]/\langle C_1, \ldots, C_m \rangle.$$ 

In this approach, elements of $A$ are represented by means of multivariate polynomials reduced modulo $(C_1, \ldots, C_m)$. For all $m \leq n$, $A_m$ has dimension $\delta_m := n!/(n-m)!$; its elements are represented as polynomials in $X_1, \ldots, X_m$.

Introducing these intermediate algebras makes it possible for us to refine our problem: we will also consider the question of fast arithmetics, and in particular characteristic polynomial computation for $A_m$. The characteristic polynomial of $P \in A_m$ will be written $X_{P,A_m} \in k[T]$; it has degree $\delta_m$ and admits the factorization

$$X_{P,A_m} = \prod_{\alpha_1, \ldots, \alpha_m \in \mathcal{R}} (T - P(\alpha_1, \ldots, \alpha_m)).$$

Divided differences are inexpensive to compute via their recursive formula, but it is difficult to make computations in $A_m$ efficient with this representation. To review known results, it will be helpful to consider two extreme cases: when $m$ is small (typically, $m$ is a constant), and when $m$ is close to $n$. Note that the first case covers some useful cases for Galois theory (such as the computation of resolvents associated to simple polynomials of the form $X_1X_2 + X_3X_4, \ldots$).
When \( m \) is fixed (say \( m = 4 \) in the above example) and \( n \to \infty \), \( \delta_m = n/(m - n)! \) is equivalent to \( n^m \). In this case, there exist algorithms of cost \( O'(\delta_m) = O'(n^m) \) for multiplication and inversion (when possible) in \( \mathbb{A}_m \) [17, 30]. Here, and everywhere else in this paper, the \( O' \) notation indicates the omission of logarithmic factors. For characteristic polynomial computation, it is possible to deduce from [29] a cost estimate of \( O'(\delta_m n^2) = O'(n^{m+1}) \).

However, all these algorithms hide exponential factors in \( m \) in their big-O, which makes them unsuitable for the case \( m \gg n \). For the case \( m = n \), the paper [7] gives a multiplication algorithm of cost \( O(\delta_n) \), but this algorithm hides high degree logarithmic terms in the big-O. There is no known quasi-linear algorithm for inverting elements of \( \mathbb{A}_m \).

The second representation we discuss is univariate. For \( m \leq n \), an element \( P \) of \( \mathbb{A}_m \) will be called primitive if the \( k \)-algebra \( k[P] \) spanned by \( P \) is equal to \( \mathbb{A}_m \) itself. If \( \Lambda \) is a primitive linear form in \( \mathbb{A}_m \), a univariate representation of \( \mathbb{A}_m \) consists of polynomials \( \bar{P} = (Q, S_1, \ldots, S_m) \) in \( k[T] \) with \( Q = X_{P, \mathbb{A}_m} \) and \( \deg(S_i) < \delta_m \) for all \( i \leq m \) such that we have a \( k \)-algebra isomorphism

\[
\mathbb{A}_m = k[X_1, \ldots, X_m]/(C_1, \ldots, C_m) \to k[T]/\langle Q \rangle \\
X_1, \ldots, X_m \to S_1, \ldots, S_m \\
\Lambda \to T.
\]

When using univariate representations, the elements of \( \mathbb{A}_m \cong k[T]/\langle Q \rangle \) are then represented as univariate polynomials of degree less than \( \delta_m \). As usual, we will thus denote by \( M(n) \) the cost of polynomial multiplication for polynomials of degrees bounded by \( n \), under the super-linearity assumptions of [22]. One can make \( T(n) = O(n \log(n) \log(\log(n))) \) using Fast Fourier Transform [36, 12].

Then, multiplications and inversions (when possible) in \( \mathbb{A}_m \) can be performed in respective times \( O(M(\delta_m)) \) and \( O(M(\delta_m) \log(\delta_m)) \). For characteristic polynomial, the situation is not as good, as no quasi-linear algorithm is known: the best known result [38] is \( O(M(\delta_m)\delta_m^{1/2} + \delta_m^{(m+1)/2}) \). Here, \( \omega \) so that we can multiply \( n \times n \) matrices with \( O(n^\omega) \) ring operations on any ring \( R \). The best known bound on \( \omega \) is \( \omega \leq 2.3727 \) [16, 40, 42], resulting in a \( O(\delta_m^{1/3}) \) characteristic polynomial algorithm.

Computing a univariate representation for \( \mathbb{A}_m \) is expensive: for \( m = n \), starting from the defining equations of \( T_z \), it takes time \( O'(\delta_n^2) \) with the geometric resolution algorithm [23]. Starting from the divided differences, the RUR algorithm [34] or the FGLM algorithm [20] take time \( O(\delta_n^2) \); a recent improvement of the latter [21] could reduce the exponent using sparse linear algebra techniques. Some other algorithms are specifically designed to take as input a triangular set (such as the divided differences) and convert it to a univariate representation, such as [10] or [32]; the latter performs the conversion for any \( m \) in quadratic time \( O'(\delta_m^2 + \delta_m^{(m+1)/2}) \), which is \( O'(\delta_m^{69}) \).

Thus, the triangular representation for \( \mathbb{A}_m \) is easy to compute but leads to rather inefficient algorithms to compute in \( \mathbb{A}_m \). On the other hand, computing a univariate representation is not straightforward, but once it is known, some computations in \( \mathbb{A}_m \) become faster. Our main contribution in this paper is to show how to circumvent the downsides of univariate representations, by providing fast algorithms for their construction (for \( \mathbb{A}_m \), itself, or for each \( \mathbb{A}_m \)) in many cases. We also show how to use fast univariate arithmetics in \( \mathbb{A}_m \) to compute characteristic polynomials efficiently.

We give two kinds of estimates, depending on whether \( m \) is fixed or not. In the first case, we are interested in what happens when \( n \to \infty \); the big-O estimates may hide constants depending on \( m \). In the second case, when both \( m \) and \( n \) can vary, a statement of the form \( f(m, n) = O(g(m, n)) \) means that there exists \( K \) such that \( f(m, n) \leq K g(m, n) \) holds for all \( m, n \). For univariate representations, our algorithms are Las Vegas: we give expected running times.

**Theorem 1.** Let \( m \leq n \) and suppose that the characteristic of \( k \) is zero, or at least \( 2\delta_m \). Then we can compute characteristic polynomials and univariate representations in \( \mathbb{A}_m \) with costs as specified in the following table.

<table>
<thead>
<tr>
<th>( m ) fixed</th>
<th>( m \leq n/2 )</th>
<th>any ( m )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X_{P, \mathbb{A}_m} )</td>
<td>( O(M(\delta_m)) ) for ( P ) linear</td>
<td>( O(n^{m+1}/m(\delta_m^2)) )</td>
</tr>
<tr>
<td>( \mathbb{A}_m )</td>
<td>( O(nmM(\delta_m)) ) for ( P ) linear</td>
<td>( O(n(\delta_m^{2/3})^2 m(\delta_m^2)) )</td>
</tr>
<tr>
<td>( \mathbb{A}_m )</td>
<td>( O(n^{m+1}/m(\delta_m^{2/3})) )</td>
<td>( O(n^{m+1}/m^2(\delta_m^2)) )</td>
</tr>
</tbody>
</table>

In particular, when \( m \) is fixed, we have optimal algorithms (up to logarithmic factors) for characteristic polynomials of linear forms. For arbitrary \( P \), the results in the last item are not optimal: when \( m \) is fixed, the running time of our algorithm is \( O(n^{m+1}/m^{69}) \), for an output of size \( n^{m} \). For small values of \( m \), say \( m = 2, 3, 4 \), this is a significant overhead. However, these results do improve on the state-of-the-art.

We propose two approaches; both of them rely on classical ideas. The first one (in Section 3) computes characteristic polynomials by means of their Newton sums, following previous work of [41, 2, 13], but is limited to simple polynomials, such as linear forms; this will establish the first two items in the theorem. The second one (in Section 4) relies on iterated resultants [27, 39, 28, 33] and provides the last statements in the theorem. The last section gives experimental results.

In addition to the general results given in the theorem above, the following sections also mention other examples for which our techniques, or slight extensions thereof, yield quasi-linear results – as of now, we do not have a complete classification of all examples for which this is the case.

In all the paper, our focus is on computing characteristic polynomials rather than resultants. From this, one can deduce resultants by root extraction, but it is of course preferable to compute the resultant directly, by cleaning multiplicities as early as possible. The basic ideas we use are known to make this possible: we mention it in the next section for the Newton sums approach and [28, 33, 5] discuss the resultant-based approach. However, quantifying the complexity gains of this improvement is beyond the scope of this paper. Note also that for cases where \( P \) is fixed, such as \( P := X_1 X_2 + X_3 X_4 \), and \( n \to \infty \), we can save only a constant factor in the running time with such considerations.

## 2. PRELIMINARIES

### 2.1 The Newton representation

Let \( g \) be monic of degree \( n \) in \( k[X] \), and let \( \beta_1, \ldots, \beta_n \) its roots in an algebraic closure of \( k \). For \( i \in \mathbb{N} \), we let \( S_i(g) \in k \) be the \( i \)-th Newton sum of \( g \), defined by \( S_i(g) := \sum_{\ell=0}^{n-1} \beta_\ell \), and for \( m \in \mathbb{N} \) we write \( S(g, m) := (S_i(g))_{0 \leq i < m} \).
The conversion from coefficients to the Newton representation $S(g, m)$ and back can be done by the Newton-Girard formulas, but this takes quadratic time in $m$. To achieve a quasi-linear complexity, we recall a result first due to Schönhage \cite{Sch69}; see \cite{Ber84} for references and a more detailed exposition, including the proofs of the results we state below.

**Lemma 2.** Let $g$ be a monic polynomial of degree $n$ in $k[X]$. Then, for $m \in \mathbb{N}$, one can compute $S(g, m)$ in time $O(M(m))$. If the characteristic of $k$ is either zero or greater than $n$, one can recover $g$ from $S(g, n)$ in time $O(M(n))$.

In particular, knowing $S(g, n)$, we can compute $S(g, n')$ for any $n' \geq n$ in time $O(M(n'))$.

The Newton representation is useful to speed up certain polynomial operations, such as multiplication and exact division, since $S_i(gh) = S_i(g) + S_i(h)$ for all $i \in \mathbb{N}$. Other improved operations include the composed sum and composed product of $g$ and another polynomial $h$, with roots $\gamma_1, \ldots, \gamma_m$; they are defined by
\[
\begin{align*}
g \oplus h &:= \prod_{i=1}^{m} (X - (\beta_i + \gamma_i)), \\
g \otimes h &:= \prod_{i=1}^{m} (X - (\beta_i \gamma_i)).
\end{align*}
\]

**Lemma 3.** Let $g, h$ be monic polynomials in $k[X]$, and suppose that $S(g, r)$ and $S(h, r)$ are known. Then one can compute $S(g \oplus h, r)$ in time $O(r)$; if the characteristic of $k$ is either zero or greater than $r$, one can compute $S(g \otimes h, r)$ in time $O(M(r))$.

We write $\otimes_{NS}(S(g, r), S(h, r))$ and $\oplus_{NS}(S(g, r), S(h, r))$ for these algorithms; the subscript $NS$ shows that the inputs are in the Newton representation.

### 2.2 Univariate representations

We recall a few facts on univariate representations. Let us fix $m \leq n$. Then, a linear form $\Lambda$ is primitive for $\mathbb{A}_m$ if and only if $\Lambda$ takes distinct values on the points of the variety defined by $\mathcal{I}_m \cap k[X_1, \ldots, X_m]$. This is the case if and only if the minimal polynomial of $\Lambda$ coincides with its characteristic polynomial $\chi_{\Lambda,k_m}$, if and only if $\chi_{\Lambda,k_m}$ is squarefree. For instance when $m = n$, $\Lambda$ is primitive in $\mathbb{A}_n$ if and only if the values $\Lambda(\alpha_{(1)}, \ldots, \alpha_{(n)})$ are all distinct for $\sigma \in S_n$.

By Zippel-Schwartz lemma \cite{Sie09, Zie77}, for $K \subseteq \mathbb{N}$, a random linear form $\Lambda$ will be primitive for $\mathbb{A}_m$ with probability greater than $1 - 1/(2K)$ if its coefficients are taken in a set of cardinality $K^2$; this still holds if we set $\lambda_1 := 1$. One can find primitive linear forms for $\mathbb{A}_m$ in a (non-uniform) deterministic manner, but with a cost polynomial in $\delta_m$ \cite{Ber84}.

When $\Lambda$ is primitive, in the univariate representation $\mathcal{F} = (Q, S_1, \ldots, S_n)$ corresponding to $\Lambda$, we obtain $Q$ as $Q = \chi_{\Lambda,k_m}$. The polynomials $S_i$ are called parametrizations because they are the images of the variables $X_i$ by the isomorphism $\lambda_m \simeq k[T]$. We will now argue that any "reasonable" algorithm that computes $Q$ will also give us the parametrizations for a moderate overhead.

Let us extend the base field $k$ to $k':=k(L_1, \ldots, L_m)$, where $L_i$ are new indeterminates. Let $\mathcal{A}_m := \mathbb{A}_m \otimes_k k'$ be obtained by adding $L_1, \ldots, L_m$ to the ground field in $\mathbb{A}_m$, and let finally $\chi_{\mathcal{A}_m,k'} \in k'[T]$ be the characteristic polynomial of $L := L_1 X_1 + \cdots + L_m X_m$. Then, the following holds:
\[
S_i = \frac{\partial \chi_{\mathcal{A}_m,k'}}{\partial L_i} \bigg|_{L_1 = \cdots = L_m = \alpha_1, \ldots, \alpha_m} \mod \chi_{\mathcal{A}_m,k'}
\]

see for instance \cite{Ber84, Ber85, Ber86, Ber87, Ber88}.

We can avoid working with $m$-variate rational function coefficients, as the formula above implies that we can obtain $S_i$ as follows. Let $k_a := k[\epsilon]/(\epsilon^2)$. For a given $\lambda$, and for $i \in m$, let $\chi_i$ be the characteristic polynomial of $\chi_i := \Lambda_i + \epsilon X_i$, computed over $k_a$. Then, $\chi_i$ takes the form $\chi_i = \chi_{\Lambda_i,k_m} + \epsilon R_i$, and we obtain $S_i$, together with $R_i$, by $\chi_{\mathcal{A}_m,k'}(X_i) \mod \chi_{\mathcal{A}_m,k'}$.

We will require that the algorithm computing $\chi_{\Lambda,k_m}$ performs no zero-test or division (other than by constants in $k$, since those can be seen as multiplications by constants). Since any ring operation $(+, \times)$ in $k_a$ costs at most $3$ operations in $k$, given such an algorithm that computes the characteristic polynomial of any linear form in $\mathbb{A}_m$ in time $O(n)$, we can deduce an algorithm that computes each $S_i$ in time $O(n)$, and $S_1, \ldots, S_m$ in time $O(mn)$.

### 3. NEWTON SUMS TECHNIQUES

In this section, we give our first algorithm for computing characteristic polynomials in $\mathbb{A}_m$. This approach is based on the following proposition and as such applies only to polynomials satisfying certain assumptions; the main result in this section is in Proposition 5 below. Our approach relies on Newton sums computations, following \cite{Nie02, Nie03, Nie04, Nie05}; an analogue of the following result can be found in \cite{Nie02} for the special cases $P = X_1 + \cdots + X_m$ and $P = X_1 \cdots X_m$. See also \cite{Ber84} for similar considerations in the bivariate case.

**Proposition 4.** Let $P \in \mathbb{A}_m$ be of the form $P(X_1, \ldots, X_m) := Q(X_1, \ldots, X_{m-1}) + R(X_m)$, with $Q$ in $\mathbb{A}_{m-1}$. For $1 \leq i \leq m-1$, define $\mathcal{P}_i := \{Q(X_1, \ldots, X_{m-1}) + R(X_i) \in \mathbb{A}_{m-1}\}$, and let $R_i := r(X_i) \in \mathbb{A}_1$. Then the following equality holds:
\[
\begin{align*}
\mathcal{X}_{P,\mathbb{A}_m} &= \mathcal{X}_{Q,\mathbb{A}_{m-1}} \otimes_{\mathbb{A}_m} \mathcal{X}_{R_1,\mathbb{A}_1} \\
\quad \bigg[ \prod_{i=1}^{m-1} (T - R_i) \bigg].
\end{align*}
\]

**Proof.** Let $\mathcal{R} = \{\alpha_1, \ldots, \alpha_m\}$ be the roots of $f$ and note that $\mathcal{X}_{R_1,\mathbb{A}_1} = \prod_{i=1}^{m} (T - R_i)$. We rewrite (3) as
\[
\begin{align*}
\mathcal{X}_{Q,\mathbb{A}_{m-1}} &= \mathcal{X}_{P,\mathbb{A}_{m-1}} - \prod_{i=1}^{m-1} (T - P(\alpha_1, \ldots, \alpha_{m-1})).
\end{align*}
\]

Thus, $\mathcal{X}_{Q,\mathbb{A}_{m-1}} \oplus \mathcal{X}_{R_1,\mathbb{A}_1}$ equals
\[
\prod_{\alpha_1, \ldots, \alpha_{m-1} \in \mathbb{R} \text{pairwise distinct, } \alpha_m \in \mathbb{R}} (T - P(\alpha_1, \ldots, \alpha_{m-1}, \alpha_m)).
\]

This product contains parasite factors compared to $\mathcal{X}_{P,\mathbb{A}_m}$, corresponding to cases where $\alpha_m = \alpha_i$ for some $i$ between $1$ and $m - 1$. For a given $i$, the factor due to $\alpha_m = \alpha_i$ is
\[
\prod_{\alpha_1, \ldots, \alpha_{m-1} \in \mathbb{R} \text{pairwise distinct}} (T - P(\alpha_1, \ldots, \alpha_{m-1}, \alpha_i)).
\]

This result can lead to a recursive algorithm, provided all recursive calls are well-defined (not all polynomials $P$ satisfy the assumptions of this proposition). We will consider a convenient particular case, when the input polynomial is linear. In this case, we can continue the recursion all the way down, remarking that for $m = 1$, the characteristic polynomial of $\chi_1$ is $f(T)$. We deduce our recursive algorithm CharNSRec, together with the top-level function CharNS; they compute $\chi_{\Lambda,k_m}$, for $\Lambda = \lambda_1 X_1 + \cdots + \lambda_m X_m$. \hfill $\square$
The algorithm `CharNSRec` uses the Newton sums representation for all polynomials involved; the only conversions are done in the top-level function `CharNS`. The algorithm thus takes as an extra argument the precision $\ell$, that is, the number of Newton sums we need. As in the previous proposition, we write $\Lambda_0 := \lambda_1 X_1 + \cdots + \lambda_{n-1} X_{m-1}$ and, for $i \leq m$, $\Lambda_i := \lambda_1 X_1 + \cdots + \lambda_{m-1} X_{m-1} + \lambda_m X_i$.

Algorithm `CharNSRec`

**Input:** $S(f, n)$, $m$, $\Lambda$, the precision $\ell$.

**Output:** $S(X_{\Lambda, \delta_m}, \ell)$.

1. $\ell' := \min(\ell, \delta_m)$
2. if $(m = 1)$ then $\text{out} := (S_i(f)\lambda_1^{\ell})_{0 \leq i < n}$ else
   a. $\text{out} := \text{CharNSRec}(S(f, n), m - 1, \Lambda, \ell')$
   b. $\text{out} := \oplus_{\text{NS}}(\text{out}, \text{CharNSRec}(S(f, n), 1, \lambda_m X_1, \ell'), \ell')$
   c. for $i$ from $1$ to $m - 1$
   \hspace{1cm} $\text{out} := \text{out} - \text{CharNSRec}(S(f, n), m - 1, \Lambda_i, \ell')$
3. if $(\ell' < \ell)$ then Extend the series “out” up to precision $\ell$
4. return $\text{out}$

The main algorithm follows; it uses a trick in the case $m = n$ to reduce the depth of the recursion by one unit.

Algorithm `CharNS`

**Input:** $f$, $m$, $\Lambda$.

**Output:** $X_{\Lambda, \delta_m}$.

1. if $(m = 0)$ then $\Lambda := (\lambda_1 - \lambda_m) X_1 + \cdots + (\lambda_{n-1} - \lambda_m) X_{n-1}$
2. return `CharNSRec`(f, n - 1, $\Lambda$) $\oplus$ $(X - \lambda_m f_1)$
3. Compute the Newton representation $S(f, n)$
4. Recover $X_{\Lambda, \delta_m}$ from $S(X_{\Lambda, \delta_m}, \ell)$
5. return $X_{\Lambda, \delta_m}$

**Proposition 5.** Let $m \leq n$ and suppose that the characteristic $k$ is either zero or greater than $\delta_m$. Then Algorithm `CharNS` computes the characteristic polynomials of linear forms in $\mathbb{F}_m$ in time $O(M(\delta_m))$ if $m$ is bounded, $O(m m! M(\delta_m))$ if $m \leq n/2$ and $O(2^n M(\delta_m))$ in general.

**Proof.** Let $C(m, \ell)$ be the cost of `CharNSRec` on input $\Lambda \in \mathbb{F}_m$ and precision $\ell$. We use the abbreviation $C(m) := C(m, \delta_m)$, so that $C(1) = O(n)$. For $2 \leq m \leq n - 1$, Lemma 2 gives $C(m, \ell) = C(m) + O(M(\ell))$ for $\ell \geq \delta_m$, so we get

$$C(m) = mC(m - 1, \delta_m) + C(1, \delta_m) + O(m M(\delta_m)) = mC(m - 1) + O(M(\delta_m)) + O(M(\delta_m)) \\ \leq mC(m - 1) + O(M(\delta_m)).$$

Then, by unravelling the recurrence and using the super-linearity of the function $M$, we deduce

$$\frac{C(m)}{M(\delta_m)} \leq O\left(\frac{m + \cdots + \sum_{i=1}^{m} n! (n - 1)!}{(i - 1)! (n - i)!}\right) \leq O\left(\frac{n - 1}{(n - 1)!}\right).$$

When $m$ is bounded, the sum is bounded. If $m \leq n/2$, we derive the bound $C(m) = O(m M(\delta_m))$ from the remark $\binom{n - i}{i} \leq \binom{n}{i} \leq \binom{n}{i}$ for $1 \leq i \leq m$. For arbitrary $m \leq n - 1$, we get the cruder bound $C(m) = O(2^n M(\delta_m))$. In all these cases, the cost of Algorithm `CharNS` is the same, up to $O(M(\delta_m))$ for conversions. For $m = n$, let $\Lambda := (\lambda_1 - \lambda_m) X_1 + \cdots + (\lambda_{n-1} - \lambda_m) X_{n-1}$. Then, $f_1 = \sum \alpha_i$ implies $X_{\Lambda, \delta_m} = X_{\Lambda, \delta_m - 1} \oplus (X - \lambda_m f_1)$; the cost for $m = n$ is thus the same as for $m = n - 1$, up to $O(M(\delta_n))$ for the composed sum.

This proves the left-hand columns of the first two rows in Theorem 1. Using the discussion in Subsection 2.2, we can also compute a univariate representation of $\mathbb{F}_m$. After computing $X_{\Lambda, \delta_m}$, we test whether $\Lambda$ is primitive for $\mathbb{F}_m$; by testing whether $X_{\Lambda, \delta_m}$ is squarefree; this takes time $O(M(\delta_m) \log(\delta_m))$, which is $O(m M(\delta_m) \log(n))$. If the characteristic of $k$ is either zero, or at least equal to $2^6$, we expect to try finitely many $\Lambda$ before finding a primitive one. When this is the case, we can apply the procedure of Subsection 2.2 to obtain all parametrizations; this costs $m$ times as much as computing $X_{\Lambda, \delta_m}$. Considering the cases $m$ constant and $m \leq n/2$, this completes the proof of the first two points in our main theorem.

To conclude this section, we mention (without proof) some extensions.

First, it is possible to adapt this algorithm to exploit symmetries of $P$, since they are known to create multiplicities in $X_{P, \delta_m}$; we can accordingly reduce the number of Newton sums we need (thus, one can compute resolvents directly in this manner). This is useful in practice, but we were not able to quantify the gains in terms of complexity.

Another remark is that an analogue to Proposition 4 holds for $P(X_1, \ldots, X_m) := Q(X_1, \ldots, X_{m-1}) \times R(X_m)$, replacing the operation $\oplus$ by $\otimes$. As an application, consider the case $P := X_1 X_2 X_3 + X_4$, so that $Q := X_1 X_2 X_3 X_4$ and $R := X_4$. To compute $X_{P, \delta_m}$, we are led to deal with $Q$, $P_1 := (1 + X_3 X_4) X_1$, $P_2 := (1 + X_1 X_3) X_2$ and $P_3 := (1 + X_1 X_2) X_3$ in $\mathbb{F}_4$. By symmetry, it is enough to consider $Q$ and $P_3$. For $Q$, we can continue the recursion all the way down to univariate polynomials, using the multiplicative version of the previous proposition. For $P_3$, however, we cannot. Writing $P_3$ as $(1 + X_1 X_2) X_3$, the recursive call lead us in particular to compute the characteristic polynomial of $(1 + X_1 X_2) X_3$, which does not satisfy the assumptions of the proposition.

Similar (but slightly more complicated) results hold as well when $P$ can be written as $P(X_1, \ldots, X_l) \otimes R(S_1, \ldots, S_m)$, with $\otimes \in \{+,-\}$. Taking for instance $P := X_1 X_2 X_3 X_4$, we are led recursively to compute the characteristic polynomials of $X_1 X_2$ and $P_1 := X_1 (X_2 + X_3)$. However, the case of $P_1$ reduces to that of $X_2 (X_2 + X_3)$, which does not satisfy the assumptions of the proposition. We will discuss these examples again in the next section.

### 4. Resultant Techniques

Resultant methods to compute characteristic polynomials in $\mathbb{F}_m$ go back to Lagrange’s elimination method (similar to today’s resultant) to compute resolvents [27]. This idea was developed in [39, 28, 33].

The basic idea is simple. Let again $C_1, \ldots, C_m$ be the divided differences associated to $f$. For $P \in k[X_1, \ldots, X_m]$, define recursively the resultants

$$G_m := T - P(X_1, \ldots, X_m) \in k[X_1, \ldots, X_m],$$

$$G_i := \text{Res}_{X_{i+1}}(C_{i+1}, G_{i+1}) \in k[X_1, \ldots, X_i],$$

for $i = m - 1, \ldots, 0$, so that $X_{P, \delta_m} = G_0 \in k[T]$. In order to
avoid an exponential growth of the degrees in the intermediate $G_i$'s, we need to compute the resultant $\text{Res}_{X_i}(C_i, G_i)$ over the coefficient ring $\mathbb{K}_{a_i-1}[T]$.

However, we mentioned that arithmetic in $\mathbb{K}_{a_i-1}$ is rather slow; univariate computations are faster. We give below a general framework that relies on both triangular and univariate representations to compute efficiently such results. Recall that a family of polynomials $T = (T_1, \ldots, T_m)$ in $k[X_1, \ldots, X_m]$ is a triangular set if the following holds for all $i \leq m$: $T_i$ is in $k[X_1, \ldots, X_i]$, $T_i$ is monic in $X_i$, and $T_i$ is reduced with respect to $(T_1, \ldots, T_{i-1})$. Our main idea holds for general triangular families of polynomials, but it is only for the special case of divided difference that it will lead to an efficient algorithm (see Corollary 11 below).

### 4.1 General algorithms

In this section, we describe a general approach to compute characteristic polynomials modulo a triangular set. Following [18, 32], our main idea is to introduce mixed representations, that allow one to convert from triangular to bivariate representations, and back, one variable at a time.

Let $T = (T_1, \ldots, T_m)$ be a triangular set in $k[X_1, \ldots, X_m]$. For $i \leq m$, let $d_i := \deg(T_i, X_i)$. We write $R_T := k[X_1, \ldots, X_m]/(T_1, \ldots, T_m)$; this is a $k$-algebra of dimension $\mu_m = d_1 \cdots d_m$. More generally, for $i \leq m$, we write $R_{T,i} := k[X_1, \ldots, X_i]/(T_1, \ldots, T_i)$; this is a $k$-algebra of dimension $\mu_i$.

Generalizing the notation used up to now, for $P$ in $R_T$, we write $X_P, R_P$, for its characteristic polynomial in $R_T$, that is, the characteristic polynomial of the multiplication-by-$P$ endomorphism of $R_T$. To compute $X_P, R_P$, we will use the ‘iterated resultant’ techniques sketched in the preamble.

Since computing modulo triangular sets is difficult, our work around is to introduce a family of univariate representations $\mathfrak{P}_1, \ldots, \mathfrak{P}_m$ of respectively $R_{T,1}, \ldots, R_{T,m-1}$: in the introduction, we only defined univariate representations for the algebras $\mathbb{K}_i$, but the definition carries over unchanged to this slightly more general context [23, 32]. For $i \leq m-1$, $\mathfrak{P}_i$ has the form $\Phi_i = (Q_i, S_{i,1}, \ldots, S_{i,i})$, with all polynomials in $k[Z_i]$ and with associated linear form $\Lambda_i := \lambda_{i,1}X_1 + \cdots + \lambda_{i,i}X_i$. For $i = 1$, we add w.l.o.g. the mild restriction that $\Lambda_1 = X_1$, so that $Q_1 = T_1$.

We first show how to use these objects to perform conversions between multivariate and bivariate representations, going one variable at a time. For $i \leq m-1$, we know that $Q_i$ has degree $\mu_i$ and that we have the $k$-algebra isomorphism

$$R_{T,i} \xrightarrow{\varphi_i} k[Z_i]/(Q_i)$$

with $\Lambda_i \mapsto Z_i$.

We extend $\varphi_i$ to another isomorphism

$$\Phi_i : R_{T,i}(X_{i+1}, \ldots, X_m) \xrightarrow{\psi_i} k[Z_i]/(Q_i)[X_{i+1}, \ldots, X_m],$$

where $\psi_i$ acts coefficientwise, and we define $Q_{i,j} = \Phi_i(T_j)$ for $1 \leq j \leq m$.

Let us see $Q_{i+1,i}, \ldots, Q_{i,m}$ in $k[Z_i, X_{i+1}, \ldots, X_m]$, by taking their canonical preimages. Then, $(Q_i, Q_{i+1,i}, \ldots, Q_{i,m})$ form a triangular set in $k[Z_i, X_{i+1}, \ldots, X_m]$, such that $\deg(Q_{i,j}, X_j) = \deg(T_{j}, X_j)$ for $i+1 \leq j \leq m$. For $i \leq m-1$ and $i \leq j \leq m$, we will write

$$R_{i,j} = k[Z_i, X_{i+1}, \ldots, X_j]/(Q_i, Q_{i+1,i}, \ldots, Q_{i,j}).$$

Then, still acting coefficientwise in $X_{i+1}, \ldots, X_j$, $\varphi_i$ extends to an isomorphism $\Phi_{i,j} : R_{T,j} \xrightarrow{\psi_{i,j}} R_{i,j}$.

Two operations will be needed to convert between the various induced representations: lift-up and push-down [18, 32]. For $i \leq m-2$ and $i+1 \leq j \leq m$, we call lift-up the change of basis $u_{i,j} := \Phi_{i,j} \circ \Phi_{i+1,j}^{-1}$. This is thus an isomorphism $R_{i,j} \xrightarrow{id} R_{i+1,j}$, with

$$R_{i,j} = k[Z_i, X_{i+1}, \ldots, X_j]/(Q_i, Q_{i+1,i}, \ldots, Q_{i,j}),$$

$$R_{i+1,j} = k[Z_{i+1}, X_{i+2}, \ldots, X_j]/(Q_{i+1}, Q_{i+1,i+2}, \ldots, Q_{i,j}).$$

In particular, with $j = i + 1$, we write $u_i$ instead of $u_{i,i+1}$; thus, it is the bivariate-to-univariate conversion given by

$$u_i : R_{i,i+1} = k[Z_i, X_{i+1}]/(Q_i, Q_{i+1,i+1}) \xrightarrow{\Psi_i} R_{i+1,i+1} = k[Z_{i+1}]/(Q_{i+1}).$$

Conversely, we call push-down the inverse change of basis; as above, for $j = i + 1$, we write $v_{i,j} = v_{i+1,j}$.

The operations $u_i$ and $v_i$ are crucial, since all $u_{i,j}$ (resp. $v_{i,j}$), for $j \geq i+2$, are obtained by applying $u_i$ (resp. $v_i$) coefficientwise. We do not discuss here how to implement them in general (see [32]); we will give a better solution in the case of divided differences below. For the moment, we simply record the following straightforward result.

**Lemma 6.** For $i \leq m-2$, suppose that one can apply $u_i$ (resp. $v_i$) operations in $k$. Then, one can apply $u_{i,m}$ using $u_i, u_{i+1}$ operations in $k$ (resp. one can apply $v_{i,m}$ using $v_i, v_{i+1}$ operations in $k$).

Finally, we define $u_{m-1} = u_{m-2,m} \circ \cdots \circ u_{1,m}$ and $v_{m} = u_1^{-1}$ so that we have

$$R_{m-1,m} = k[Z_{m-1}, Z_m]/(Q_{m-1}, Q_{m-1,m})$$

$$R_T \xrightarrow{\Psi_m} R_{m-1,m} \xleftarrow{\uparrow u_{m-1}}$$

We could want to go all the way down to univariate polynomials instead of bivariate, but it would not be useful: the algorithm below uses bivariate polynomials. In terms of complexity, the following is a direct consequence of Lemma 6.

**Lemma 7.** For $i \leq m-2$, suppose that one can apply up, (resp. down) using $u_i$ (resp. $v_i$) operations in $k$. Then one can apply $u_{m}$ (resp. $v_{m}$) in $R_T$ in respective times

$$\sum_{i=1}^{m-2} u_i \mu_{i+2} \quad \text{and} \quad \sum_{i=1}^{m-2} v_i \mu_{i+1}.$$

Now we explain how to compute $G := X_P, R_P \in k[Y]$ for any $P$ in $R_T$. Let $k' := k[Y]$; then, $T$ is also a triangular set in $k'[X_1, \ldots, X_m]$, and we define, for $i \leq m$,

$$R_{T,i} := k'[X_1, \ldots, X_i]/(T_1, \ldots, T_i) \in R_{T,i}[Y].$$

As explained in the preamble of this section, we start by defining $G_m := Y - P \in R_{T,m}$. For $i = m-1, \ldots, 0$, suppose that we know $G_{i+1} \in R_{T,i+1}$. Seeing $R_{T,i+1}$ as $R_{T,i+1}(X_{i+1})/(T_{i+1})$, we define

$$G_i := \text{Res}_{X_{i+1}}(T_{i+1}, G_{i+1}) \in R_{T,i}.$$

Standard properties of resultants (see e.g. [11, § 12.2]) show that $G_0 = G$. By induction, we prove that $\deg(G_i, Y) = \mu_i$; in particular, $\deg(G_0(Y)) = \mu$, as it should be.

We are going to compute $G_i, i = 0, \ldots, m$ assuming that we know the univariate representations $\mathfrak{P}_1, \ldots, \mathfrak{P}_{m-1}$, and use
univariate arithmetic as much as possible. For $1 \leq i \leq m-1$ and $i \leq j \leq m$, $R_{i,j}$ is well defined and isomorphic to $R_{j,i}$ because $R_{i,j} = R_{j,i}[Y]$ and $R_{j,i} = R_{i,j}[Y]$. Besides, lift-up and push-down are still defined; they are written respectively $up_i': R_{i,i+1} \rightarrow R_{i+1,i+1}$ and $down_i'$. 

**Lemma 8.** For $i \leq m-2$, suppose that one can apply $up_i$ (resp. $down_i$) using $u_i$ (resp. $v_i$) operations in $k$. Then, for $F$ in $R_{i,i+1}$, with $d := \deg(F,Y)$, we can compute $up_i'(F) \in R_{i+1,i+1}$ using $O(du_i)$ operations in $k$. For $F$ in $R_{i+1,i+1}$, with $d := \deg(F,Y)$, we can compute $down_i'(F) \in R_{i,i+1}$ using $O(dv_i)$ operations in $k$.

This leads to our algorithm for characteristic polynomials. For convenience, we let $R_{0,1} := R_0$, and we let $down_0$ be the identity map. For the moment, we assume that all polynomials $Q_i,i+1$ needed below are already known.

**Algorithm Characterize**

**Input:** $P \in R_T$.

**Output:** $X_{P,R_T}$.

1. $P' := up_m(P)$
2. $G_m := Y - P'$
3. for $i = m - 1, \ldots, 1$ do
   a. $G'_i := \text{Res}_{X_{i+1}}(Q_{i+1}, G_{i+1})$
   b. $G_i := down_i'(G'_i)$
4. return $G_0 = \text{Res}_{X_1}(G_1, Q_1)$.

To analyze this algorithm, we remark that over any ring $R$, resultants of polynomials of degree $d$ in $R[X]$ can be computed in $O(d^{(d+1)/2})$ ring operations, provided one of these polynomials is monic, and $1, \ldots, d$ are units in $R$. Indeed, the resultant $\text{Res}_X(A, B)$, with $A$ monic of degree $d$ and $\deg(B, X) < d$ is the constant term of the characteristic polynomial of $B$ modulo $A$. This whole polynomial can be computed in time $O(d^{(d+1)/2})$ by an algorithm of Shoup [38] which performs no zero-test and only divisions by 1, \ldots, $d$.

**Proposition 9.** Suppose that one can apply $up_i$ (resp. $down_i$) using $u_i$ (resp. $v_i$) operations in $k$, and that $k$ has characteristic either zero, or at least $\mu_k$. Then Algorithm Characterize computes $X_{P,R_T}$ in time

$$O \left( \sum_{i=0}^{m-2} (u_i + v_i) \mu_i' + \sum_{i=0}^{m-1} d_{i+1}^{(i+1)/2} M(\mu_i') \right).$$

**Proof.** We have seen that Step 1 takes time $\sum_{i=1}^{m-2} u_i \mu_i'$. For $i = m - 1, \ldots, 1$, $G'_i$ has degree $\mu_i'$ in $Y$, so Step 3.b takes time $v_i - \mu_i'$ by Lemma 8.

In Step 3.a, we compute $G_i$ by evaluation / interpolation in the variable $Y$, using evaluation points in geometric progression [9]; such points exist by assumption on the characteristic of $k$. Both $G_{i+1}$ and $Q_{i+1}$ have degree at most $d_{i+1}$ in $X_{i+1}$, and $\deg(G'_i, Y) = \mu_i$. Thus, the cost is $O(d_{i+1}M(\mu_i'))$ operations in $R_{i,i}$ for all evaluations / interpolations, since all points are in geometric progression [9]. Since the evaluation points are in $k$, evaluation and interpolation are $k$-linear operations, so each of them uses $\mu_i$ operations in $k$.

The cost for all individual resultants is $O(\mu_i d_{i+1}^{(i+1)/2})$ ring operations in $R_{i,i}$, each of which takes $O(M(\mu_i'))$ operations in $k$. The conclusion follows using the inequalities $\mu_i M(\mu_i') \leq M(\mu_i')$ and $M(\mu_i') \leq M(\mu_i')$. \hfill $\square$

### 4.2 The case of divided differences

We now apply the former results to the triangular set of divided differences. Fix $m \in \mathbb{N}$ such that $m \leq n$, and take $T = (C_1, \ldots, C_m)$ in $k[X_1, \ldots, X_m]$. Note that $d := \deg(C_1, X_1)$ is equal to $n + 1 - i \leq n$, and that $R_{i,i}$ becomes $A_i$ for $1 \leq i \leq m$. We also have $\mu_i = \delta_i$ and $\mu_i' = \delta_i / \delta_i$.

We are going to study lift-up and push-down for divided differences, with the objective to give estimates on the quantities $u_i$ and $v_i$ defined above. Thus, we start from univariate representations $\mathcal{P}_1, \ldots, \mathcal{P}_m$ for $A_1, \ldots, A_{m-1}$; for the moment, they are part of the input.

We impose a further restriction on $\mathcal{P}_1, \ldots, \mathcal{P}_m$, assuming that for all $i < m-1$, $A_{i+1} = A_i + \lambda_{i+1}X_i$, for some $\lambda_{i+1}$ in $k$. When this is the case, we call $\mathcal{P}_1, \ldots, \mathcal{P}_m$ compatible. Then, we have $A_i = X_1 + \lambda_2X_2 + \cdots + \lambda_iX_i$, since by assumption $A_1 = X_1$. Thus, compatible univariate representations are associated to a $(m-2)$-tuple $(A_2, \ldots, A_{m-1})$ in $k^{m-2}$, with the condition that every $X_1 + \lambda_2X_2 + \cdots + \lambda_iX_i$ is a primitive element of $A_i$ for all $i \leq m - 1$. Under this condition, we now study the cost of lift-up and push-down. Indeed, in this case, we can deduce the explicit form of $up_i$ and $down_i$:

$$k[Z, X_{i+1}] / (Q_i, Q_{i+1}) \rightarrow k[Z_{i+1}] / (Q_i, Q_{i+1})$$

$$up_i : X_{i+1} \rightarrow Z_{i+1} - \lambda_{i+1}X_{i+1}$$

$$down_i : Z_{i+1} \rightarrow Z_i + \lambda_{i+1}X_{i+1}.$$
1. Compute $H^* (Z - \lambda X, X)$ modulo $f$ using the shift algorithm of [1] (which is possible under our assumption on the characteristic of $k$) with coefficients in $k[X]/(f)$.

2. Evaluate previous result at $X = S$ using Horner scheme.

Step 1 takes time $O(M(n)M(ab))$; the next step uses $n$ multiplications modulo $W$, for a total of $O(nM(ab))$.

**Computing down.** Conversely, for $G \in k[Z]/(W)$, we show how to compute $H := \rho(G)$.

Let $G^*$ be the canonical preimage of $G$ in $k[Z]$, so that $H = G(Z + \lambda X)$ modulo $(A, B)$.

We obtain $H$ as follows:

1. Compute $G(Z + \lambda X)$ modulo $f$, using again the shift algorithm of [1] with coefficients in $k[X]/(f)$.

2. Reduce previous result modulo $(A, B)$.

Step 1 takes time $O(M(n)M(ab))$, then the reduction takes time $O(M(n)M(ab))$ by fast Euclidean division.

**Conclusion.** By the former discussion, given $A = Q_1, B = Q_{i+1}, up$ and $W = Q_{i+1}, up$, and $down$, can be computed in time $u_i = O(M(n)M(\delta_{i+1}))$.

First, though we have to compute $Q_{i+1}$. Supposing that $Q_{i+1}$ is known, we can compute $Q_{i+1}$ by adjusting Formula (2), writing

$$Q_{i+1} = up_{i+1} - Q_{i+1} = \left(\frac{Q_{i+1} \cdot (Z_{i+1} \cdot X_{i+1}) - Q_{i+1} \cdot (Z_{i+1} \cdot X_{i+1})}{X_{i+1} \cdot X_i}\right).$$

The quotient can be computed in $O(\delta_{i+1}^2)$. Next we apply $up_{i+1}$ coefficientwise on a polynomial of degree $d_{i+1}$ in $Z_{i+1}$ — this is possible, since we know $Q_{i+1}$, so this costs $O(M(n)M(\delta_{i+1}))$.

To summarize, we can compute $Q_{i+1}$ from $Q_{i+1}$ in time $O(M(n)M(\delta_{i+1}))$. By the discussion on the function $S(m, n, \ell)$, with here $\ell = 0$, the total cost from $Q_{i+1}$ to $Q_{i+1}$ is $O(M(n)M(\delta_{i+1}))$. □

**Corollary 11.** Suppose that $\Psi_1, \ldots, \Psi_m$ are known and compatible. If the characteristic of $k$ is either 0 or at least $\delta_1$, then for any $P \in S_m$, we can compute $X_{P,A_m}$ in time $O(n^{(\omega+1)^2} m M(\delta_{m+1}))$.

If $P = \Lambda$ is a primitive linear form in $m$, compatible with the previous ones, we can compute the corresponding parametrizations in the same expected amount of time.

**Proof.** The first part is obvious, as the dominant term from Proposition 9 comes from Step 3.a.

When $P = \Lambda$ is primitive, we write as usual $Q_m$ instead of $X_{P,A_m}$. Using the discussion in Subsection 2.2, we can compute $Q_m$ and the last parametrization $S_{m,1}$ of $\Psi_m$ in the same cost. The other parametrizations are obtained from $\Psi_{m-1}$ by $S_{m,j} = up_{m-1} (S_{m-1,j})$ for $j < m$. This is done using Proposition 10, since all that is required for algorithm $up_{m-1}$ are $Q_m$ and $S_{m,m}$. So all other parametrizations cost $O(nM(n)M(\delta_{m+1}))$, which is not dominant. □

**Proof of Theorem 1.** We will give here the complexity estimate for computing $\Psi_1, \ldots, \Psi_{m}$ once they are known, computing the characteristic polynomial of an arbitrary $P$ is done using the corollary above.

We need to pick $\Lambda := 1 + \lambda_2 X_2 + \cdots + \lambda_m X_m \in \mathbb{A}_m$ primitive such that its restrictions $\Lambda_i := 1 + \lambda_2 X_2 + \cdots + \lambda_i X_i$ to fewer variables are still primitive. As per the assumption on the characteristic of $k$, we pick the coefficients $\lambda_2, \ldots, \lambda_m$ in $\{1, \ldots, 2\delta_m^2\}$. By the remark in Subsection 2.2, for $2 \leq i \leq m$, $\Lambda_i$ is not primitive for $\Lambda$, with probability at most $\delta_i^2/\delta_m$. Because of the inequality

$$\sum_{2 \leq i \leq m} \frac{\delta_i^2}{\delta_m} \leq \sum_{i=2}^{\infty} \frac{1}{(n!)^2} = 2.5,$$

the probability of all $\Lambda_i$ being primitive is at least 0.375. Thus, on average, we have to pick a finite number of $\Lambda$.

Our algorithm first picks $\Lambda$ as explained above. We assumed in Subsection 4.2 that the representation $\Psi_i$ ought to be associated to $A_1 = X_1$, so that $\Psi_i = (Z_i, Z_i)$. Assume now that $\Psi_1, \ldots, \Psi_{m+1}$ are known. Using the first point in the previous corollary, we compute $X_{\Lambda_{m+1}}$, and we test whether this polynomial is squarefree. If not, we start all over from a new $\Lambda$. Otherwise, we continue with the second point in the corollary, to deduce $\Psi_i$.

The dominant cost comes from applying the corollary.

Since we expect to pick finitely many $\Lambda$, the expected cost is $O(\sum_{i=2}^{\infty} \frac{1}{(n!)^2} M(\delta_i))$. This is $O(n^{(\omega+1)^2} m M(\delta_{m+1}))$, in view of our discussion on the function $S(m, n, \ell)$, with here $\ell = 1$. This concludes the proof of our main theorem.

**Improvements.** Improvements given in [28, 33] to take into account predictable multiplicities in the successive results can be applied here as well; however, it is unclear to us how they would impact the complexity analysis.

Our last remark concerns examples from the previous section. We mentioned there some issues with the application of Proposition 4 (and its multiplicative version) to the polynomial $X_1 X_2 X_3 + X_4$, as we could not apply that proposition recursively to the polynomial $(1 + X_1 X_2) \times X_2$. The result above shows that we can compute the characteristic polynomial of $(1 + X_1 X_2) \times X_2$ in time $O(n^{(\omega+1)^2} M(\delta_{m})) = O(M(\delta_{m}))$. As a result, we are thus able to complete the whole computation for $P$ in quasi-linear time $O(M(\delta_{m}))$ as well. The same holds for $X_1 X_2 X_3 + X_4$.

5. IMPLEMENTATION AND TIMINGS

Our algorithms were implemented in Magma 2.17.1; we report here on some experiments dedicated to computations in the case $m = n$, that is, in $\mathbb{A}_m$. Timings were measured on an Intel Xeon 16 cores at 2.27GHz with 74Gb RAM.

When $m = n$, although the complexity of CharNS is not quasi-linear (due to a 2^n overhead), it usually does better than algorithm CharResultant. A first reason is that for the former, the constant in the big-O is mild (we do only a few multiplications at each step). Besides, some other ideas are used in our code. Different recursive calls have often computations in common, so we use memoization. We also make use of symmetries: if $\Lambda$ has a large stabilizer, as explained in Section 2, we can reduce the number of Newton sums we need to compute its characteristic polynomial. We usually attempt to pick favorable $\Lambda$: a good strategy is to take $\Lambda = \sum_{1 \leq i \leq n-1} i X_{n-i}$, for which the linear forms over $\mathbb{A}_{n-2}$ (which are the most expensive) have repeated coefficients.

In the following table, we take $k = \mathbb{F}_p$, with $p$ a 28 bit prime; we give timings to compute a univariate representation of $\mathbb{A}_n$. We are not aware of other available implementations for this problem in Magma, so we compared our algorithm with the Magma Gröbner basis functions. Our algorithm is tailored for computations in $\mathbb{A}_n$, so it is at an advantage compared to generalist functions; on the other hand, Magma’s Gröbner basis functions use highly optimized $\mathbb{C}$ code. Despite an extra 2^n factor in the cost analysis, algorithm CharNS performs very well for this computation.
Next, we discuss the cost of basic arithmetic in $\mathbb{A}_n$, comparing in particular univariate operations to arithmetic modulo the Cauchy modules. Several Magma constructions exist for this purpose; we report on the most efficient solutions we found. As a conclusion, for an operation such as inversion, even with the overhead of lift-up and push-down, it pays off to convert to a univariate representation.

Finally, we focus computing $X_P, A_n$, for a generic polynomial $P$. The best alternative we could find comes from \cite{Shoup} and is written “Shoup” in the table. This algorithm uses univariate arithmetic; for it to be applicable, we must already know a univariate representation of $A_n$, and the input must be written on the corresponding univariate basis. The complexity of that algorithm higher than that of CharResultant, but the algorithm is simpler and relies on fast built-in Magma code; as a result, it outperforms CharNS. If the input $P$ is a linear form in $X_1, \ldots, X_n$, CharNS is actually faster than both, as showed in the first table.

### 6. REFERENCES

42. J. von zur Gathen and J. Gerhard. Modern Computer Algebra

<table>
<thead>
<tr>
<th>n</th>
<th>Time</th>
<th>CharResultant</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.001</td>
<td>0.03</td>
</tr>
<tr>
<td>5</td>
<td>0.005</td>
<td>0.05</td>
</tr>
<tr>
<td>6</td>
<td>0.52</td>
<td>6.8</td>
</tr>
<tr>
<td>7</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>&gt;6h</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>Up</th>
<th>Down</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.008</td>
<td>0.1</td>
</tr>
<tr>
<td>6</td>
<td>0.01</td>
<td>1.4</td>
</tr>
<tr>
<td>7</td>
<td>0.002</td>
<td>0.29</td>
</tr>
<tr>
<td>8</td>
<td>0.003</td>
<td>4.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Univ. ×</th>
<th>Univ. ÷</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>0.005</td>
<td>0.002</td>
</tr>
<tr>
<td>25</td>
<td>0.006</td>
<td>0.29</td>
</tr>
<tr>
<td>170</td>
<td>4</td>
<td>30min</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Univ. ÷</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>28</td>
</tr>
<tr>
<td>&gt;30min</td>
<td>&gt;6h</td>
</tr>
</tbody>
</table>