Algorithms for the universal decomposition algebra

Let $\mathbb{k}$ be a field and let $f \in \mathbb{k}[T]$ be a polynomial of degree n. The *universal decomposition algebra* $\mathbb{A}$ is the quotient of $\mathbb{k}[X_1, \ldots, X_n]$ 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 $\mathbb{A}$. We use a univariate representation of $\mathbb{A}$, *i.e.* an isomorphism of the form $\mathbb{A} \simeq \mathbb{k}[T]/Q(T)$, since in this representation, arithmetic operations in $\mathbb{A}$ 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 $\mathbb{A}$.

7.1 Introduction

Let $\mathbb{k}$ be a field and let $f = X^n + \sum_{i=1}^{n} (-1)^i f_i X^{n-i}$ in $\mathbb{k}[X]$ 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 $\mathbb{k}$. The *ideal of symmetric relations* $\mathcal{I}_s$ is the ideal

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

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

$$\mathcal{X}_{P,\mathbb{A}}(T) = \prod_{\sigma \in \mathfrak{S}_n} (T - P(\alpha_{\sigma(1)}, \ldots, \alpha_{\sigma(n)})) \in \mathbb{k}[T]. \tag{7.1}$$

This polynomial is related to the *absolute Lagrange resolvent*

$$L_P(T) := \prod_{\mathrm{Stab}(P)\backslash\backslash\mathfrak{S}_n} (T - P(\alpha_{\sigma(1)}, \ldots, \alpha_{\sigma(n)})) \in \mathbb{k}[T],$$

where $\mathrm{Stab}(P)\backslash\backslash\mathfrak{S}_n$ are the left cosets of the stabilizer of P in the symmetric group $\mathfrak{S}_n$; indeed, these polynomials satisfy the relation $\mathcal{X}_{P,\mathbb{A}} = L_P^{\#\mathrm{Stab}(P)}$.

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 [Lag70, Soi81, Val89, AV94, AV97, Leh97, Yok97, RV99, AV00]; 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 δ^2 in the general case.

In some particular cases, though, it is known that resolvents can be computed in quasi-linear time [CM94]. 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 $\mathbb{k}$ they perform. Practically, this is well adapted to cases where $\mathbb{k}$ is a finite field; over $\mathbb{k} = \mathbb{Q}$, a combination of our results and modular techniques, such as in [Ren04] for resolvents, 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 $\mathbb{A}$. A commonly used representation is *triangular*. The *divided differences*, also known as *Cauchy modules* [Che50, RV99], are defined by $C_1(X_1) := f(X_1)$ and

$$C_{i+1} := \frac{C_i(X_1, ..., X_i) - C_i(X_1, ..., X_{i-1}, X_{i+1})}{X_i - X_{i+1}} \tag{7.2}$$

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

$$\begin{gathered}\mathbb{A}_1 := \mathbb{k}[X_1]/(C_1)\\ \vdots \\ \mathbb{A}_m := \mathbb{k}[X_1, ..., X_m]/(C_1, ..., C_m)\\ \vdots \\ \mathbb{A} = \mathbb{A}_n := \mathbb{k}[X_1, ..., X_n]/(C_1, ..., C_n).\end{gathered}$$

In this approach, elements of $\mathbb{A}$ are represented by means of multivariate polynomials reduced modulo $(C_1, ..., C_n)$. For all $m \leqslant n$, $\mathbb{A}_m$ has dimension $\delta_m := n!/(n-m)!$; its elements are represented as polynomials in $X_1, ..., X_m$.

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 $\mathbb{A}_m$. The characteristic polynomial of $P \in \mathbb{A}_m$ will be written $\mathcal{X}_{P,\mathbb{A}_m} \in \mathbb{k}[T]$; it has degree δ_m and admits the factorization

$$\mathcal{X}_{P,\mathbb{A}_m} = \prod_{\alpha_1, ..., \alpha_m \in \mathcal{R} \text{pairwise}} (T - P(\alpha_1, ..., \alpha_m)). \tag{7.3}$$

Divided differences are inexpensive to compute via their recursive formula, but it is difficult to make computations in $\mathbb{A}_m$ 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 $X_1 X_2 + X_3 X_4$, ...).

When m is fixed (say $m = 4$ in the above example) and $n \to \infty$, $\delta_m = n!/(n-m)!$ is equivalent to n^m. In this case, there exist algorithms of cost $\tilde{\mathcal{O}}(\delta_m) = \tilde{\mathcal{O}}(n^m)$ for multiplication and inversion (when possible) in $\mathbb{A}_m$ [DMMSX06, LMS09]. Here, and everywhere else in this chapter, the $\tilde{\mathcal{O}}$ notation indicates the omission of logarithmic factors. For characteristic polynomial computation, it is possible to deduce from [LMP09] a cost estimate of $\tilde{\mathcal{O}}(\delta_m n^2) = \tilde{\mathcal{O}}(n^{m+2})$.

However, all these algorithms hide exponential factors in m in their big-O, which makes them unsuitable for the case $m \simeq n$. For the case $m = n$, the paper [BCHS11] gives a multiplication algorithm of cost $\tilde{\mathcal{O}}(\delta_n)$, but this algorithm hides high degree logarithmic terms in the big-O. There is no known quasi-linear algorithm for inverting elements of $\mathbb{A}_n$.

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

$$\begin{array}{ccc} \mathbb{A}_m = \mathbb{k}[X_1, \ldots, X_m]/(C_1, \ldots, C_m) & \to & \mathbb{k}[T]/(Q) \\ X_1, \ldots, X_m & \mapsto & S_1, \ldots, S_m \\ \Lambda & \mapsto & T. \end{array}$$

A brief history of univariate representations and triangular sets can be found in Section 6.6.1.1.

When using univariate representations, the elements of $\mathbb{A}_m \simeq \mathbb{k}[T]/(Q)$ are then represented as univariate polynomials of degree less than δ_m. As usual, we will thus denote by $\mathsf{M}(n)$ the cost of polynomial multiplication for polynomials of degrees bounded by n, under the super-linearity assumptions of [GG03]. One can take $\mathsf{M}(n) = \mathcal{O}(n \log(n) \log(\log(n)))$ using Fast Fourier Transform [SS71, CK91].

Then, multiplications and inversions (when possible) in $\mathbb{A}_m$ cost respectively $\mathcal{O}(\mathsf{M}(\delta_m))$ and $\mathcal{O}(\mathsf{M}(\delta_m) \log(\delta_m))$. For characteristic polynomial, the situation is not as good, as no quasi-linear algorithm is known: the best known result [Sho94] is $\mathcal{O}(\mathsf{M}(\delta_m)\, \delta_m^{1/2} + \delta_m^{(\omega+1)/2})$. Here, ω is so that we can multiply $n \times n$ matrices within $\mathcal{O}(n^\omega)$ ring operations on any ring R. The best known bound on ω is $\omega \leqslant 2.3727$ [CW90, Sto10, VW11], resulting in a $\mathcal{O}(\delta_m^{1.69})$ characteristic polynomial algorithm.

Computing a univariate representation for $\mathbb{A}_m$ is expensive: for $m = n$, starting from any defining equations of $\mathcal{I}_s$, it takes time $\tilde{\mathcal{O}}(\delta_n^2)$ with the geometric resolution algorithm [GLS01]. Starting from the divided differences, the RUR algorithm [Rou99] or the FGLM algorithm [FGLM93] take time $\mathcal{O}(\delta_n^3)$; a recent improvement of the latter [FM11] 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 [BLMM01] or [PS11]; the latter performs the conversion for any m in subquadratic time $\tilde{\mathcal{O}}(\mathsf{M}(\delta_m)\, \delta_m^{1/2} + \delta_m^{(\omega+1)/2})$, which is $\tilde{\mathcal{O}}(\delta_m^{1.69})$.

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 chapter is to show how to circumvent the downsides of univariate representations, by providing fast algorithms for their construction (for $\mathbb{A}_n$ 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 $f(m,n) = \mathcal{O}(g(m,n))$ means that there exists K such that $f(m,n) \leqslant K\, g(m,n)$ holds for all m,n. For univariate representations, our algorithms are Las Vegas: we give *expected* running times.

Theorem 7.1. *Let $m \leqslant n$ and suppose that the characteristic of $\mathbb{k}$ is zero, or at least $2\,\delta_m^2$. Then we can compute characteristic polynomials and univariate representations in $\mathbb{A}_m$ with costs as specified in the following table.*

	$\mathcal{X}_{P,\mathbb{A}_m}$	*univ. representation (expected time)*
m fixed	$\mathcal{O}(\mathsf{M}(\delta_m))$ *for P linear*	$\mathcal{O}(\mathsf{M}(\delta_m)\log(n))$
$m \leqslant n/2$	$\mathcal{O}(n\, m\, \mathsf{M}(\delta_m))$ *for P linear*	$\mathcal{O}(n\, m^2\, \mathsf{M}(\delta_m))$
any m	$\mathcal{O}(n^{(\omega+1)/2}\, m\, \mathsf{M}(\delta_m))$	$\mathcal{O}(n^{(\omega+1)/2}\, m\, \mathsf{M}(\delta_m))$

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 $\tilde{\mathcal{O}}(n^{m+1.69})$, 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 7.3) computes characteristic polynomials by means of their Newton sums, following previous work of [Val89, AV94, CM94], 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 7.4) relies on iterated resultants [Lag70, Soi81, Leh97, RV99] 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 this chapter, our focus is on computing characteristic polynomials rather than resolvents. From this, one can deduce resolvents by root extraction, but it is of course preferable to compute the resolvent 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 [Leh97, RV99, AV12] discuss the resultant-based approach. However, quantifying the complexity gains of this improvement is beyond the scope of this chapter. 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.

7.2 Preliminaries

7.2.1 The Newton representation

Let g be monic of degree n in $\mathbb{k}[X]$, and let β_1, ..., β_n its roots in an algebraic closure of $\mathbb{k}$. For $i \in \mathbb{N}$, we let $S_i(g) \in \mathbb{k}$ be the ith *Newton sum* of g, defined by $S_i(g) := \sum_{\ell=1}^{n} \beta_\ell^i$, and for $m \in \mathbb{N}$ we write $S(g, m) := (S_i(g))_{0 \leqslant i \leqslant m}$.

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 [Sch82]; see [Bos03] for references and a more detailed exposition, including the proofs of the results we state below.

Lemma 7.2. *Let g be a monic polynomial of degree n in $\mathbb{k}[X]$. Then, for $m \in \mathbb{N}$, one can compute $S(g, m)$ in time $\mathcal{O}(\mathsf{M}(m))$. If the characteristic of $\mathbb{k}$ is either zero or greater than n, one can recover g from $S(g, n)$ in time $\mathcal{O}(\mathsf{M}(n))$.*

In particular, knowing $S(g, n)$, we can compute $S(g, n')$ for any $n' \geqslant n$ in time $\mathcal{O}(\mathsf{M}(n'))$.

The Newton representation is useful to speed up certain polynomial operations, such as multiplication and exact division, since $S_i(g h) = S_i(g) + S_i(h)$ for all $i \in \mathbb{N}$. Other improved operations include the *composed sum* and *composed product* of g and another polynomial h, with roots $\gamma_1, ..., \gamma_m$; they are defined by

$$\begin{aligned} g \oplus h &:= \prod_{i=1\ldots n, j=1\ldots m} (X - (\beta_i + \gamma_j)), \\ g \otimes h &:= \prod_{i=1\ldots n, j=1\ldots m} (X - (\beta_i \gamma_j)). \end{aligned}$$

Lemma 7.3. ([Bos03, section 7.3]) *Let g, h be monic polynomials in $\mathbb{k}[X]$, and suppose that $S(g, r)$ and $S(h, r)$ are known. Then one can compute $S(g \otimes h, r)$ in time $\mathcal{O}(r)$; if the characteristic of $\mathbb{k}$ is either zero or greater than r, one can compute $S(g \oplus h, r)$ in time $\mathcal{O}(\mathsf{M}(r))$.*

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

7.2.2 Univariate representations

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

By Zippel-Schwartz lemma [Zip79, Sch80], for $K \in \mathbb{N}_{>0}$, a random linear form Λ will be primitive for $\mathbb{A}_m$ with probability greater than $1 - 1/(2K)$ if its coefficients are taken in a set of cardinality $K\delta_m^2$; 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 δ_m [CG10].

When Λ is primitive, in the univariate representation $\mathfrak{P} = (Q, S_1, \ldots, S_n)$ corresponding to Λ, we obtain Q as $Q = \mathcal{X}_{\Lambda,\mathbb{A}_m}$. The polynomials S_i are called *parametrizations* because they are the images of the variables X_i by the isomorphism $\mathbb{A}_m \simeq \Bbbk[T]/(Q)$. 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 $\Bbbk$ to $\Bbbk' := \Bbbk(L_1, \ldots, L_m)$, where L_i are new indeterminates. Let $\mathbb{A}'_m := \mathbb{A}_m \otimes_{\Bbbk} \Bbbk'$ be obtained by adding $L_1, \ldots, L_m$ to the ground field in $\mathbb{A}_m$, and let finally $\mathcal{X}_{L,\mathbb{A}'_m} \in \Bbbk'[T]$ be the characteristic polynomial of $L := L_1 X_1 + \cdots + L_n X_m$. Then, the following holds:

$$S_i = -\frac{\partial \mathcal{X}_{L,\mathbb{A}'_m}}{\partial L_i} \Big/ \frac{\partial \mathcal{X}_{L,\mathbb{A}'_m}}{\partial T} \bmod \mathcal{X}_{L,\mathbb{A}'_m}\bigg|_{L_1,\ldots,L_m = \lambda_1,\ldots,\lambda_m};$$

see for instance [Kro82, Kön03, Mac16, HKP+00, GLS01, DL08].

We can avoid working with m-variate rational function coefficients, as the formula above implies that we can obtain S_i as follows. Let $\Bbbk_\varepsilon := \Bbbk[\varepsilon]/(\varepsilon^2)$. For a given Λ, and for $i \leqslant m$, let $\mathcal{X}_{\Lambda_i}$ be the characteristic polynomial of $\Lambda_i := \Lambda + \varepsilon X_i$, computed over $\Bbbk_\varepsilon$. Then, $\mathcal{X}_{\Lambda_i}$ takes the form $\mathcal{X}_{\Lambda_i} = \mathcal{X}_{\Lambda,\mathbb{A}_m} + \varepsilon R_i$, and we obtain S_i as $S_i = R_i / \mathcal{X}'_{\Lambda,\mathbb{A}_m} \bmod \mathcal{X}_{\Lambda,\mathbb{A}_m}$.

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

7.3 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 7.5 below. Our approach relies on Newton sums computations, following [Lag70, Val89, AV94, CM94]; an analogue of the following result can be found in [CM94] for the special cases $P = X_1 + \cdots + X_m$ and $P = X_1 \cdots X_m$. See also [BFSS06] for similar considerations in the bivariate case.

Proposition 7.4. *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 $1 \leqslant i \leqslant m-1$, define

$$P_i := Q(X_1, \ldots, X_{m-1}) + R(X_i) \in \mathbb{A}_{m-1},$$

and let $R_1 := R(X_1) \in \mathbb{A}_1$. Then the following equality holds:

$$\mathcal{X}_{P,\mathbb{A}_m} = \frac{\mathcal{X}_{Q,\mathbb{A}_{m-1}} \oplus \mathcal{X}_{R_1,\mathbb{A}_1}}{\prod_{i=1}^{m-1} \mathcal{X}_{P_i,\mathbb{A}_{m-1}}}. \tag{7.4}$$

Proof. Let $\mathcal{R} = \{\alpha_1, \ldots, \alpha_n\}$ be the roots of f and note that $\mathcal{X}_{R_1,\mathbb{A}_1} = \prod_{i=1}^n (T - R(\alpha_i))$. We rewrite (7.3) as

$$\mathcal{X}_{Q,\mathbb{A}_{m-1}} = \prod_{\alpha_1, \ldots, \alpha_{m-1} \in \mathcal{R} \text{pairwise}} (T - Q(\alpha_1, \ldots, \alpha_{m-1})).$$

Thus, $\mathcal{X}_{Q,\mathbb{A}_{m-1}} \oplus \mathcal{X}_{R_1,\mathbb{A}_1}$ equals

$$\prod_{\alpha_1, \ldots, \alpha_{m-1} \in \mathcal{R} \text{pairwise}, \alpha_m \in \mathcal{R}} (T - P(\alpha_1, \ldots, \alpha_m)).$$

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

$$\prod_{\alpha_1, \ldots, \alpha_{m-1} \in \mathcal{R} \text{pairwise}} (T - P(\alpha_1, \ldots, \alpha_{m-1}, \alpha_i)),$$

that is, $\mathcal{X}_{P_i,\mathbb{A}_{m-1}}$. Formula (7.4) follows. □

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 λX_1 is $f(\lambda T)$. We deduce our recursive algorithm CharNSRec, together with the top-level function CharNS; they compute $\mathcal{X}_{\Lambda,\mathbb{A}_m}$, for $\Lambda = \lambda_1 X_1 + \cdots + \lambda_m X_m$.

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 ℓ, that is, the number of Newton sums we need. As in the previous proposition, we write $\Lambda_0 := \lambda_1 X_1 + \cdots + \lambda_{m-1} X_{m-1}$ and, for $i \leqslant m$, $\Lambda_i := \lambda_1 X_1 + \cdots + \lambda_{m-1} X_{m-1} + \lambda_m X_i$.

```
Algorithm CharNSRec

Input: $S(f,n)$, $m$, $\Lambda$, the precision $\ell$.
Output: $S(\mathcal{X}_{\Lambda,\mathbb{A}_m},\ell)$.
 1. $\ell' := \min(\ell,\delta_m)$
 2. if $(m=1)$ then out $:= (S_i(f)\,\lambda_1^i)_{0\leqslant i\leqslant n}$ else
     a. out $:=$ CharNSRec$(S(f,n), m-1, \Lambda_0, \ell')$
     b. out $:= \oplus_{\mathrm{NS}}($out$,$ CharNSRec$(S(f,n), 1, \lambda_m X_1, \ell'), \ell')$
     c. for $i$ from $1$ to $m-1$
            out := out $-$ CharNSRec$(S(f,n), m-1, \Lambda_i, \ell')$
 3. if $(\ell' < \ell)$ then Extend the series "out" up to precision $\ell$
 4. return out
```

The minus in step 2.c corresponds to a division in Formula (7.4). The main algorithm follows; it uses a trick when $m=n$ to reduce the depth of the recursion by one unit.

```
Algorithm CharNS

Input: $f$, $m$, $\Lambda$.
Output: $\mathcal{X}_{\Lambda,\mathbb{A}_m}$.
 1. if $(m=n)$ then
     a. $\bar{\Lambda} := (\lambda_1-\lambda_n)X_1+\cdots+(\lambda_{n-1}-\lambda_n)X_{n-1}$
     b. return CharNS$(f, n-1, \bar{\Lambda}) \oplus (X-\lambda_n f_1)$
 2. Compute the Newton representation $S(f,n)$
 3. $S(\mathcal{X}_{\Lambda,\mathbb{A}_m},\delta_m) :=$ CharNSRec$(S(f,n), m, \Lambda, \delta_m)$
 4. Recover $\mathcal{X}_{\Lambda,\mathbb{A}_m}$ from $S(\mathcal{X}_{\Lambda,\mathbb{A}_m},\delta_m)$
 5. return $\mathcal{X}_{\Lambda,\mathbb{A}_m}$
```

Proposition 7.5. *Let $m\leqslant n$ and suppose that the characteristic $\Bbbk$ is either zero or greater than δ_m. Then Algorithm* CharNS *computes the characteristic polynomials of linear forms in $\mathbb{A}_m$ in time $\mathcal{O}(\mathsf{M}(\delta_m))$ if m is bounded, $\mathcal{O}(m\,n\mathsf{M}(\delta_m))$ if $m\leqslant n/2$ and $\mathcal{O}(2^n\,\mathsf{M}(\delta_m))$ in general.*

Proof. Let be $\mathcal{C}(m,\ell)$ be the cost of CharNSRec on input $\Lambda\in\mathbb{A}_m$ and precision ℓ. We use the abbreviation $\mathcal{C}(m):=\mathcal{C}(m,\delta_m)$, so that $\mathcal{C}(1)=\mathcal{O}(n)$. For $2\leqslant m\leqslant n-1$, Lemma 7.2 gives $\mathcal{C}(m,\ell)=\mathcal{C}(m)+\mathcal{O}(\mathsf{M}(\ell))$ for $\ell\geqslant\delta_m$, so we get

$$\begin{aligned}\mathcal{C}(m) &= m\,\mathcal{C}(m-1,\delta_m)+\mathcal{C}(1,\delta_m)+\mathcal{O}(m\,\mathsf{M}(\delta_m))\\ &= m\,(\mathcal{C}(m-1)+\mathcal{O}(\mathsf{M}(\delta_m)))+\mathcal{O}(m\,\mathsf{M}(\delta_m))\\ &\leqslant m\,\mathcal{C}(m-1)+\mathcal{O}(m\,\mathsf{M}(\delta_m)).\end{aligned}$$

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

$$\begin{aligned}\frac{\mathcal{C}(m)}{\mathsf{M}(\delta_m)} &\leqslant \mathcal{O}\left(m+m\,(m-1)\,\frac{\delta_{m-1}}{\delta_m}+\cdots+m!\,\frac{\delta_1}{\delta_m}\right)\\ &\leqslant \mathcal{O}\left(\sum_{i=1}^{m}\frac{m!}{(i-1)!}\,\frac{(n-m)!}{(n-i)!}\right)\\ &\leqslant \mathcal{O}\left(\frac{n}{\binom{n}{m}}\sum_{i=1}^{m}\binom{n-1}{i-1}\right).\end{aligned}$$

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

This proves the left-hand columns of the first two rows in Theorem 7.1. Using the discussion in Subsection 7.2.2, we can also compute a univariate representation of $\mathbb{A}_m$. After computing $\mathcal{X}_{\Lambda,\mathbb{A}_m}$, we test whether Λ is primitive for $\mathbb{A}_m$, by testing whether $\mathcal{X}_{\Lambda,\mathbb{A}_m}$ is squarefree; this takes time $\mathcal{O}(\mathsf{M}(\delta_m)\,\log\,(\delta_m))$, which is $\mathcal{O}(m\,\mathsf{M}(\delta_m)\log(n))$. If the characteristic of $\mathbb{k}$ is either zero, or at least equal to $2\,\delta_m^2$, we expect to try finitely many Λ before finding a primitive one. When this is the case, we can apply the procedure of Subsection 7.2.2 to obtain all parametrizations; this costs m times as much as computing $\mathcal{X}_{\Lambda,\mathbb{A}_m}$. Considering the cases m constant and $m\leqslant n/2$, 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 $\mathcal{X}_{P,\mathbb{A}_m}$: 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 7.4 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 $Q:=X_1\,X_2\,X_3$ and $R:=X_4$. To compute $\mathcal{X}_{P,\mathbb{A}_4}$, we are led to deal with Q, $P_1:=(1+X_2\,X_3)\,X_1$, $P_2:=(1+X_1\,X_3)\,X_2$ and $P_3:=(1+X_1\,X_2)\,X_3$ in $\mathbb{A}_3$. 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 $(1+X_1\,X_2)\times X_3$, the recursive call lead us in particular to compute the characteristic polynomial of $(1+X_1\,X_2)\times X_2$, which does not satisfy the assumptions of the proposition.

Similar (but slightly more complicated) results hold when $P(X_1,\ldots,X_m)$ can be written as $Q(X_1,\ldots,X_\ell)\operatorname{op}R(X_{\ell+1},\ldots,X_m)$, with $\operatorname{op}\in\{+,\times\}$. 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.

7.4 Resultant techniques

Resultant methods to compute characteristic polynomials in $\mathbb{A}_m$ go back to Lagrange's elimination method (similar to today's resultant) to compute resolvents [Lag70]. This idea was developed in [Soi81, Leh97, RV99].

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

$$\begin{aligned} G_m &:= T - P(X_1, \ldots, X_m) \in \Bbbk[X_1, \ldots, X_m, T], \\ G_i &:= \operatorname{Res}_{X_{i+1}}(C_{i+1}, G_{i+1}) \in \Bbbk[X_1, \ldots, X_i, T], \end{aligned}$$

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

However, we mentioned that arithmetic in $\mathbb{A}_{i-1}$ 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 resultants. Recall that a family of polynomials $\boldsymbol{T} = (T_1, \ldots, T_m)$ in $\Bbbk[X_1, \ldots, X_m]$ is a *triangular set* if the following holds for all $i \leqslant m$: T_i is in $\Bbbk[X_1, \ldots, X_i]$, 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 7.11 below).

7.4.1 General algorithms

In this section, we describe a general approach to compute characteristic polynomials modulo a triangular set. Following [DFS09, PS11], 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 $\boldsymbol{T} = (T_1, \ldots, T_m)$ be a triangular set in $\Bbbk[X_1, \ldots, X_m]$. For $i \leqslant m$, let $d_i := \deg(T_i, X_i)$, $\mu_i := d_1 \cdots d_i$ and $\mu_i' := d_{i+1} \cdots d_m$. We write $R_{\boldsymbol{T}} := \Bbbk[X_1, \ldots, X_m]/(T_1, \ldots, T_m)$; this is a $\Bbbk$-algebra of dimension $\mu_m = d_1 \cdots d_m$. More generally, for $i \leqslant m$, we write $R_{\boldsymbol{T},i} := \Bbbk[X_1, \ldots, X_i]/(T_1, \ldots, T_i)$; this is a $\Bbbk$-algebra of dimension μ_i.

Generalizing the notation used up to now, for P in $R_{\boldsymbol{T}}$, we write $\mathcal{X}_{P,R_{\boldsymbol{T}}}$ for its characteristic polynomial in $R_{\boldsymbol{T}}$, that is, the characteristic polynomial of the multiplication-by-P endomorphism of $R_{\boldsymbol{T}}$. To compute $\mathcal{X}_{P,R_{\boldsymbol{T}}}$, we will use the "iterated resultant" techniques sketched in the preamble.

Since computing modulo triangular sets is difficult, our workaround is to introduce a family of univariate representations $\mathfrak{P}_1, \ldots, \mathfrak{P}_{m-1}$ of respectively $R_{\boldsymbol{T},1}, \ldots, R_{\boldsymbol{T},m-1}$; in the introduction, we only defined univariate representations for the algebras $\mathbb{A}_i$, but the definition carries over unchanged to this slightly more general context [GLS01, PS11]. For $i \leqslant m-1$, $\mathfrak{P}_i$ has the form $\mathfrak{P}_i = (Q_i, S_{i,1}, \ldots, S_{i,i})$, with all polynomials in $\Bbbk[Z_i]$ 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 $i \leqslant m-1$, we know that Q_i has degree μ_i and that we have the $\mathbb{k}$-algebra isomorphism

$$\begin{array}{rcl} R_{\boldsymbol{T},i} & \longrightarrow & \mathbb{k}[Z_i]/(Q_i) \\ \varphi_i\colon\ X_1, \dots, X_i & \longmapsto & S_{1,i}, \dots, S_{i,i} \\ \Lambda_i & \longmapsto & Z_i. \end{array}$$

We extend φ_i to another isomorphism

$$\Phi_i\colon \quad R_{\boldsymbol{T},i}[X_{i+1}, \dots, X_m] \longrightarrow \mathbb{k}[Z_i]/(Q_i)[X_{i+1}, \dots, X_m],$$

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

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

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

Then, still acting coefficientwise in $X_{i+1}, \dots, X_j$, φ_i extends to an isomorphism $\Phi_{i,j}$: $R_{\boldsymbol{T},j} \to R_{i,j}$.

Two operations will be needed to convert between the various induced representations: *lift-up* and *push-down* [DFS09, PS11]. For $i \leqslant m-2$ and $i+1 \leqslant j \leqslant m$, we call *lift-up* the change of basis $\mathsf{up}_{i,j} := \Phi_{i+1,j} \circ \Phi_{i,j}^{-1}$. This is thus an isomorphism $R_{i,j} \to R_{i+1,j}$, with

$$\begin{array}{rcl} R_{i,j} & = & \mathbb{k}[Z_i, X_{i+1}, \dots, X_j]/(Q_i, Q_{i,i+1}, \dots, Q_{i,j}), \\ R_{i+1,j} & = & \mathbb{k}[Z_{i+1}, X_{i+2}, \dots, X_j]/(Q_{i+1}, Q_{i+1,i+2}, \dots, Q_{i+1,j}). \end{array}$$

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

$$\mathsf{up}_i\colon \quad \begin{array}{c} R_{i,i+1} = \mathbb{k}[Z_i, X_{i+1}]/(Q_i, Q_{i,i+1}) \\ \downarrow \\ R_{i+1,i+1} = \mathbb{k}[Z_{i+1}]/(Q_{i+1}). \end{array}$$

Conversely, we call *push-down* the inverse change of basis; as above, for $j = i+1$, we write $\mathsf{down}_i = \mathsf{down}_{i,i+1}$. The operations up_i and down_i are crucial, since all $\mathsf{up}_{i,j}$ (resp. $\mathsf{down}_{i,j}$), for $j \geqslant i+2$, are obtained by applying up_i (resp. down_i) coefficientwise. We do not discuss here how to implement them in general (see [PS11]); we will give a better solution in the case of divided differences below. For the moment, we simply record the following straightforward result.

Lemma 7.6. *For $i \leqslant m-2$, suppose that one can apply up_i (resp. down_i) using u_i (resp. v_i) operations in $\mathbb{k}$. Then, one can apply $\mathsf{up}_{i,m}$ using $u_i \mu'_{i+1}$ operations in $\mathbb{k}$ (resp. one can apply $\mathsf{down}_{i,m}$ using $v_i \mu'_{i+1}$ operations in $\mathbb{k}$).*

Finally, we define $\mathsf{Up}_m = \mathsf{up}_{m-2,m} \circ \cdots \circ \mathsf{up}_{1,m}$ and $\mathsf{Down}_m = \mathsf{Up}_m^{-1}$ so that we have

$$\begin{tikzcd} R_{m-1,m} = \Bbbk[Z_{m-1}, Z_m]/(Q_{m-1}, Q_{m-1,m}) \arrow[d, shift right, "\mathrm{Down_m}"'] \\ R_{\boldsymbol{T}} = \Bbbk[X_1, \ldots, X_m]/(T_1, \ldots, T_m). \arrow[u, shift right, "\mathrm{Up_m}"'] \end{tikzcd}$$

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 7.6.

Lemma 7.7. *For $i \leqslant m-2$, suppose that one can apply up_i (resp. down_i) using u_i (resp. v_i) operations in $\Bbbk$. Then one can apply Up_m (resp. Down_m) in respective times*

$$\sum_{i=1}^{m-2} u_i \, \mu'_{i+1} \quad \text{and} \quad \sum_{i=1}^{m-2} v_i \, \mu'_{i+1}.$$

Now we explain how to compute $G := \mathcal{X}_{P,R_{\boldsymbol{T}}} \in \Bbbk[Y]$ for any P in $R_{\boldsymbol{T}}$. Let $\Bbbk' := \Bbbk[Y]$; then, $\boldsymbol{T}$ is also a triangular set in $\Bbbk'[X_1, \ldots, X_m]$, and we define, for $i \leqslant m$,

$$R'_{\boldsymbol{T},i} := \Bbbk'[X_1, \ldots, X_i]/(T_1, \ldots, T_i) = R_{\boldsymbol{T},i}[Y].$$

As explained in the preamble of this section, we start by defining $G_m := Y - P \in R'_{\boldsymbol{T},m}$. For $i = m-1, \ldots, 0$, suppose that we know $G_{i+1} \in R'_{\boldsymbol{T},i+1}$. Seeing $R'_{\boldsymbol{T},i+1}$ as $R'_{\boldsymbol{T},i+1} = R'_{\boldsymbol{T},i}[X_{i+1}]/(T_{i+1})$, we define

$$G_i := \mathrm{Res}_{X_{i+1}}(T_{i+1}, G_{i+1}) \in R'_{\boldsymbol{T},i}.$$

Standard properties of resultants (see e.g. [Bou73, § 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 $G_{m-1}, \ldots, G_0$ assuming that we know the univariate representations $\mathfrak{P}_1, \ldots, \mathfrak{P}_{m-1}$, and use univariate arithmetic as much as possible. For $1 \leqslant i \leqslant m-1$ and $i \leqslant j \leqslant m$, $R'_{i,j}$ is well defined and isomorphic to $R'_{\boldsymbol{T},j}$ because $R'_{i,j} = R_{i,j}[Y]$ and $R'_{\boldsymbol{T},j} = R_{\boldsymbol{T},j}[Y]$. Besides, lift-up and push-down are still defined; they are written respectively $\mathsf{up}'_i\colon R'_{i,i+1} \to R'_{i+1,i+1}$ and down'_i.

Lemma 7.8. *For $i \leqslant m-2$, suppose that one can apply up_i (resp. down_i) using u_i (resp. v_i) operations in $\Bbbk$. Then, for F in $R'_{i,i+1}$, with $d := \deg(F, Y)$, we can compute $\mathsf{up}'_i(F) \in R'_{i+1,i+1}$ using $\mathcal{O}(d\, u_i)$ operations in $\Bbbk$. For F in $R'_{i+1,i+1}$, with $d := \deg(F, Y)$, we can compute $\mathsf{down}'_i(F) \in R'_{i,i+1}$ using $\mathcal{O}(d\, v_i)$ operations in $\Bbbk$.*

This leads to our algorithm for characteristic polynomials. For convenience, we let $R_{0,1} := R_1$, 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 CharResultant

Input: $P$ in $R_{\boldsymbol{T}}$.
Output: $\mathcal{X}_{P,R_{\boldsymbol{T}}}$.

1. $P' := \mathsf{Up}_m(P)$                                   $P' \in R_{m-1,m}$
2. $G_m := Y - P'$                                          $G'_m \in R'_{m-1,m}$
3. for $i = m-1, \ldots, 1$ do
    a. $G'_i := \mathrm{Res}_{X_{i+1}}(Q_{i,i+1}, G_{i+1})$   $G'_i \in R'_{i,i}$
    b. $G_i := \mathsf{down}'_{i-1}(G'_i)$                   $G_i \in R'_{i-1,i}$
4. return $G_0 = \mathrm{Res}_{X_1}(G_1, Q_1)$.               $G_0 \in R'$
```

To analyze this algorithm, we remark that over any ring R, resultants of polynomials of degree d in $R[X]$ can be computed in $\mathcal{O}(d^{(\omega+1)/2})$ ring operations, provided one of these polynomials is monic, and 1, ..., d are units in R. Indeed, the resultant $\mathrm{Res}_X(A, B)$, 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 $\mathcal{O}(d^{(\omega+1)/2})$ by an algorithm of Shoup [Sho94] which performs no zero-test and only divisions by $1, \ldots, d$.

Proposition 7.9. *Suppose that one can apply* up_i *(resp.* down_i*) using* u_i *(resp.* v_i*) operations in* $\Bbbk$*, and that* $\Bbbk$ *has characteristic either zero, or at least* μ_m*. Then Algorithm* CharResultant *computes* $\mathcal{X}_{P,R_{\boldsymbol{T}}}$ *in time*

$$\mathcal{O}\left(\sum_{i=1}^{m-2} (u_i + v_i)\, \mu'_{i+1} + \sum_{i=0}^{m-1} d_{i+1}^{(\omega+1)/2}\, \mathsf{M}(\mu_m)\right).$$

Proof. We have seen that Step 1 takes time $\sum_{i=1}^{m-2} u_i\, \mu'_{i+1}$. For $i = m-1, \ldots, 1$, G'_i has degree μ'_i in Y, so Step 3.b takes time $v_{i-1}\, \mu'_i$ by Lemma 7.8.

In Step 3.a, we compute G_i by evaluation / interpolation in the variable Y, using evaluation points in geometric progression [BS05]; such points exist by assumption on the characteristic of $\Bbbk$. Both G_{i+1} and $Q_{i,i+1}$ have degree at most d_{i+1} in X_{i+1}, and deg $(G'_i, Y) = \mu'_i$. Thus, the cost is $\mathcal{O}(d_{i+1}\, \mathsf{M}(\mu'_i))$ operations in $R_{i,i}$ for all evaluations / interpolations. Since the evaluation points are in $\Bbbk$, evaluation and interpolation are $\Bbbk$-linear operations, so each of them uses μ_i operations in $\Bbbk$.

The cost for all individual resultants is $\mathcal{O}(\mu'_i\, d_{i+1}^{(\omega+1)/2})$ ring operations in $R_{i,i}$, each of which takes $\mathcal{O}(\mathsf{M}(\mu_i))$ operations in $\Bbbk$. The conclusion follows using the inequalities $\mu_i\, \mathsf{M}(\mu'_i) \leqslant \mathsf{M}(\mu_m)$ and $\mathsf{M}(\mu_i)\, \mu'_i \leqslant \mathsf{M}(\mu_m)$. □

7.4.2 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 $m \leqslant n$, and take $\boldsymbol{T} = (C_1, \ldots, C_m)$ in $\Bbbk[X_1, \ldots, X_m]$. Note that $d_i := \deg(C_i, X_i)$ is equal to $n + 1 - i \leqslant n$, and that $R_{\boldsymbol{T},i}$ becomes $\mathbb{A}_i$ for $1 \leqslant i \leqslant m$. We also have $\mu_i = \delta_i$ and $\mu'_i = \delta_m / \delta_i$.

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 $\mathfrak{P}_1, \ldots, \mathfrak{P}_{m-1}$ for $\mathbb{A}_1, \ldots, \mathbb{A}_{m-1}$; for the moment, they are part of the input.

We impose a further restriction on $\mathfrak{P}_1, \ldots, \mathfrak{P}_{m-1}$, assuming that for all $i < m-1$, $\Lambda_{i+1} = \Lambda_i + \lambda_{i+1} X_{i+1}$ for some λ_{i+1} in $\mathbb{k}$. When this is the case, we call $\mathfrak{P}_1$, ..., $\mathfrak{P}_{m-1}$ *compatible*. Then, we have $\Lambda_i = X_1 + \lambda_2 X_2 + \cdots + \lambda_i X_i$, since by assumption $\Lambda_1 = X_1$. Thus, compatible univariate representations are associated to a $(m-2)$-uple $(\lambda_2, \ldots, \lambda_{m-1}) \in \mathbb{k}^{m-2}$, with the condition that every $X_1 + \lambda_2 X_2 + \cdots + \lambda_i X_i$ is a primitive element of $\mathbb{A}_i$ for all $i \leqslant m-1$. 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:

$$\mathsf{up}_i: \begin{array}{ccc} \mathbb{k}[Z_i, X_{i+1}]/(Q_i, Q_{i,i+1}) & \longrightarrow & \mathbb{k}[Z_{i+1}]/(Q_{i+1}) \\ Z_i & \longmapsto & Z_{i+1} - \lambda_{i+1} S_{i+1,i+1} \\ X_{i+1} & \longmapsto & S_{i+1,i+1} \\ Z_i + \lambda_{i+1} X_{i+1} & \longmapsto & Z_{i+1}. \end{array}$$

The key for the following algorithms is then the remark that $f(X_{i+1}) = 0$ in $\mathbb{A}_{i+1}$; we will exploit the fact that the polynomial f is a small degree, *univariate* polynomial. To analyze its cost, we will use the following bounds: for $\ell \geqslant 1$, consider the sum $S(m, n, \ell) := \sum_{1 \leqslant i \leqslant m} i^\ell \mathsf{M}(\delta_i)$. Then we claim that the following holds:

$$S(m, n, \ell) \leqslant \exp(1)\, m^\ell \mathsf{M}(\delta_m) = \mathcal{O}(m^\ell \mathsf{M}(\delta_m)).$$

Indeed,the super-linearity of the function M implies

$$\frac{S(m,n,\ell)}{\mathsf{M}(\delta_m)} \leqslant \sum_{1 \leqslant i \leqslant m} i^\ell \frac{\delta_i}{\delta_m} \leqslant m^\ell \sum_{1 \leqslant i \leqslant m} \frac{\delta_i}{\delta_m} \leqslant \sum_{i \in \mathbb{N}} \frac{1}{n!}.$$

Proposition 7.10. *Suppose that $\mathfrak{P}_1$, ..., $\mathfrak{P}_{m-1}$ are known and compatible. If the characteristic of $\mathbb{k}$ is either zero or at least δ_{m-1}, then for $1 \leqslant i \leqslant m-2$, up_i and down_i can be computed in time $u_i = \mathcal{O}(\mathsf{M}(n)\, \mathsf{M}(\delta_{i+1}))$ and $v_i = \mathcal{O}(\mathsf{M}(n)\, \mathsf{M}(\delta_{i+1}))$.*

Proof. First, we study the following simplified problem: given $\lambda \in \mathbb{k}$, some polynomials $A \in \mathbb{k}[Z]$, $B \in \mathbb{k}[Z, X]$ monic in X, and W, S in $\mathbb{k}[Z]$, compute the mapping

$$\mathsf{up}: \begin{array}{ccc} \mathbb{k}[Z, X]/(A, B) & \longrightarrow & \mathbb{k}[Z]/(W) \\ Z & \longmapsto & Z - \lambda S \\ X & \longmapsto & S \\ Z + \lambda X & \longmapsto & Z, \end{array}$$

and its inverse down, assuming up is well-defined and invertible. We write $a := \deg(A)$ and $b := \deg(B, X)$, so that $\deg(W) = a\, b$. We also assume that $f(X) = 0$ in $\mathbb{k}[Z, X]/(A, B)$, for some monic polynomial $f \in \mathbb{k}[X]$ of degree $n \geqslant b$. Finally, the characteristic of $\mathbb{k}$ is supposed to be either 0 or at least $a\, b$. Then, we show that both directions take time $\mathcal{O}(\mathsf{M}(n)\, \mathsf{M}(a\, b))$.

COMPUTING up. Given $H \in \Bbbk[Z, X]/(A, B)$, we first show how to compute $G := \mathsf{up}(H)$. Let $H^\star$ be the canonical preimage of H in $\Bbbk[Z, X]$, so that $G = H^\star(Z - \lambda S, S) \bmod W$. Then, we obtain G as follows:

1. Compute $H^\star\,(Z - \lambda X, X)$ modulo f using the shift algorithm of [ASU75] (which is possible under our assumption on the characteristic of $\Bbbk$) with coefficients in $\Bbbk[X]/(f)$
2. Evaluate previous result at $X = S$ using Horner scheme.

Step 1 takes time $\mathcal{O}(\mathsf{M}(n)\,\mathsf{M}(a))$; the next step uses n multiplications modulo W, for a total of $\mathcal{O}(n\,\mathsf{M}(a\,b))$.

COMPUTING down. Conversely, for $G \in \Bbbk[Z]/(W)$, we show how to compute $H := \mathsf{down}(G)$. Let $G^\star$ be the canonical preimage of G in $\Bbbk[Z]$, so that $H = G(Z + \lambda X) \bmod (A, B)$. We obtain H as follows:

1. Compute $G\,(Z + \lambda X)$ modulo f, using again the shift algorithm of [ASU75] with coefficients in $\Bbbk[X]/(f)$.
2. Reduce previous result modulo (A, B).

Step 1 takes time $\mathcal{O}(\mathsf{M}(n)\,\mathsf{M}(a\,b))$, then the reduction takes time $\mathcal{O}(\mathsf{M}(n)\,\mathsf{M}(a\,b))$ by fast Euclidean division.

CONCLUSION. By the former discussion, given $A = Q_i$, $B = Q_{i,i+1}$ and $W = Q_{i+1}$, up_i and down_i can be computed in time $u_i = \mathcal{O}(\mathsf{M}(n)\,\mathsf{M}(\delta_{i+1}))$.

First, though, we have to compute $Q_{i,i+1}$. Supposing that $Q_{i-1,i}$ is known, we can compute $Q_{i,i+1}$ by adjusting Formula (7.2), writing

$$Q_{i,i+1} = \mathsf{up}_{i-1,i+1}\left(\frac{Q_{i-1,i}(Z_{i-1}, X_{i+1}) - Q_{i-1,i}(Z_{i-1}, X_i)}{X_{i+1} - X_i}\right).$$

The quotient can be computed in $\mathcal{O}(\delta_{i-1}\,d_{i+1}^2)$. Next we apply up$_{\text{i-1}}$ coefficientwise on a polynomial of degree d_{i+1} in Z_{i+1} — this is possible, since we know $Q_{i-1,i}$, so this costs $\mathcal{O}(\mathsf{M}(n)\,\mathsf{M}(\delta_i)\,d_{i+1})$. To summarize, we can compute $Q_{i,i+1}$ from $Q_{i-1,i}$ in time $\mathcal{O}(\mathsf{M}(n)\,\mathsf{M}(\delta_{i+1}))$. By the discussion on the function $S(m, n, \ell)$, with here $\ell = 0$, the *total* cost from $Q_{0,1} = Q_1$ to $Q_{i,i+1}$ is $\mathcal{O}(\mathsf{M}(n)\,\mathsf{M}(\delta_{i+1}))$. □

Corollary 7.11. *Suppose that $\mathfrak{P}_1$, ..., $\mathfrak{P}_{m-1}$ are known and compatible. If the characteristic of $\Bbbk$ is either 0 or at least δ_m, then for any $P \in \mathbb{A}_m$, we can compute $\mathcal{X}_{P,\mathbb{A}_m}$ in time $\mathcal{O}(n^{(\omega+1)/2}\,m\,\mathsf{M}(\delta_m))$.*

If $P = \Lambda$ is a primitive linear form in $\mathbb{A}_m$, 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 7.9 comes from Step 3.a.

When $P=\Lambda$ is primitive, we will write as usual Q_m instead of $\mathcal{X}_{P,\mathbb{A}_m}$. Using the discussion in Subsection 7.2.2, we can compute Q_m *and* the last parametrization $S_{m,m}$ of $\mathfrak{P}_m$ in the same cost. The other parametrizations are obtained from $\mathfrak{P}_{m-1}$by $S_{m,j}=\mathsf{up}_{m-1}(S_{m-1,j})$ for $j<m$. This is done using Proposition 7.10, since all that is required for algorithm up_{m-1} are Q_m and $S_{m,m}$. So all other parametrizations cost $\mathcal{O}(m\,\mathsf{M}(n)\,\mathsf{M}(\delta_m))$, which is not dominant. □

Proof. (of Theorem 7.1) We will give here the complexity estimate for computing $\mathfrak{P}_1, \ldots, \mathfrak{P}_m$ – 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 $\mathbb{k}$, we pick the coefficients $\lambda_2, \ldots, \lambda_m$ in $\{1, \ldots, 2\,\delta_m^2\}$. By the remark in Subsection 7.2.2, for $2 \leqslant i \leqslant m$, Λ_i is not primitive for $\mathbb{A}_i$ with probability at most $\delta_i^2/4\,\delta_m^2$. Because of the inequality

$$\sum_{2\leqslant i\leqslant m} \frac{\delta_i^2}{\delta_m^2} \leqslant \sum_{i\in\mathbb{N}} \frac{1}{(n!)^2} < 2.5,$$

the probability of *all* Λ_i being primitive is at least 0.375. Thus, on average, we have to pick a finite number of Λ.

Our algorithm first picks Λ as explained above. We assumed in Subsection 7.4.2 that the representation $\mathfrak{P}_1$ ought to be associated to $\Lambda_1 = X_1$, so that $\mathfrak{P}_1 = (f(Z_1), Z_1)$. Assume now that $\mathfrak{P}_1, \ldots, \mathfrak{P}_{i-1}$ are known. Using the first point in the previous corollary, we compute $\mathcal{X}_{\Lambda_i,\mathbb{A}_i}$ and we test whether this polynomial is squarefree. If not, we start all over from a new Λ. Otherwise, we continue with the second point in the corollary, to deduce $\mathfrak{P}_i$.

The dominant cost comes from applying the corollary. Since we expect to pick finitely many Λ, the expected cost is $\mathcal{O}(\sum_{i\leqslant m} n^{(\omega+1)/2}\ i\ \mathsf{M}(\delta_i))$. This is $\mathcal{O}(n^{(\omega+1)/2}\ m\ \mathsf{M}(\delta_m))$, 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 given in [Leh97, RV99] to take into account predictable multiplicities in the successive resultants 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 7.4 (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 $\mathcal{O}(n^{(\omega+1)/2}\,\mathsf{M}(\delta_2)) = \mathcal{O}(\mathsf{M}(\delta_4))$. As a result, we are thus able to complete the whole computation for P in quasi-linear time $\mathcal{O}(\mathsf{M}(\delta_4))$ as well. The same holds for $X_1\,X_2 + X_3\,X_4$.

7.5 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 $\mathbb{A}_n$. Timings were measured on one core of a Intel Xeon 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 Λ has a large stabilizer, as explained in Section 7.2, we can reduce the number of Newton sums we need to compute its characteristic polynomial. We usually attempt to pick favorable Λ: a good strategy is to take $\Lambda=\sum_{1\leqslant i\leqslant n-1} i\, X_{n-i}$, for which the linear forms over $\mathbb{A}_{n-2}$ (which are the most expensive) have repeated coefficients.

In the following table, we take $\mathbb{k}=\mathbb{F}_p$, 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 C code. Despite an extra 2^n factor in the cost analysis, algorithm CharNS performs very well for this computation.

n		4	5	6	7	8
Time	Gröbner	0.001	0.03	5.8	1500	>6h
(sec)	CharNS	0.005	0.05	0.52	6.8	100

Table 7.1. Timings of computation of univariate representations

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.

n		5	6	7	8
Time (sec)	Up	0.008	0.1	2	40
	Down	0.01	0.1	1.4	25
	Univ. $\times$	$40\mu s$	0.0005	0.006	0.06
	Univ. $\div$	0.002	0.028	0.29	4.5
	Magma $\times$	0.003	0.085	4	170
	Magma $\div$	0.1	28	>30min	>6h

Table 7.2. Timings of arithmetic in $\mathbb{A}_n$

Finally, we focus computing $\mathcal{X}_{P,\mathbb{A}_n}$, for a generic polynomial P. The best alternative we could find comes from [Sho94] 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 $\mathbb{A}_n$, and the input must be written on the corresponding univariate basis. The complexity of “Shoup” is higher than that of CharResultant, but the algorithm is simpler and relies on fast built-in Magma code; as a result, it outperforms CharResultant. 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.

n		4	5	6	7	8
Time	Shoup	0.001	0.01	0.23	6.8	200
(sec)	CharResultant	0.03	0.24	2.6	45	1100

Table 7.3. Timings of computation of $\mathcal{X}_{P,\mathbb{A}_n}$ for generic polynomials P