Fast arithmetic for the algebraic closure of finite fields

Luca De Feo
Laboratoire PRiSM
Université de Versailles
luca.de-feo@uvsq.fr

Javad Doliskani
Computer Science
Department
Western University
jdoliska@uwo.ca

Éric Schost
Computer Science
Department
Western University
eschost@uwo.ca

ABSTRACT

We present algorithms to construct and do arithmetic operations in the algebraic closure of the finite field $\mathbb{F}_p$. Our approach is inspired by algorithms for constructing irreducible polynomials, which first reduce to prime power degrees, then use composita techniques. We use similar ideas to give efficient algorithms for embeddings and isomorphisms.

Categories and Subject Descriptors

F.2.1 [**Theory of computation**]: Analysis of algorithms and problem complexity—*Computations in finite fields*; G.4 [**Mathematics of computing**]: Mathematical software

General Terms

Algorithms,Theory

Keywords

Finite fields, irreducible polynomials, extensions.

1. INTRODUCTION

Several computer algebra systems or libraries, such as Magma [3], Sage [39], NTL [37], PARI [31] or Flint [25], offer built-in features to build and compute in arbitrary finite fields. At the core of these designs, one finds algorithms for building irreducible polynomials and algorithms to compute embeddings and isomorphisms. The system used in Magma (one of the most complete we know of) is described in [4].

Previous algorithms typically rely on linear algebra techniques, for instance to describe embeddings or isomorphisms (this is the case for the algorithms in [4], but also for those in [29, 1]). Unfortunately, linear algebra techniques have cost at least quadratic in the degree of the extensions we consider, and (usually) quadratic memory requirements. Our goal here is to replace linear algebra by polynomial arithmetic, exploiting fast polynomial multiplication to obtain algorithms of quasi-linear complexity. As we will see, we meet this goal for several, but not all, operations.

ISSAC'14, July 23–25, 2014, Kobe, Japan.

http://dx.doi.org/10.1145/2608628.2608672 ...$15.00.

Setup. Let p be a prime (that will be fixed throughout this paper). We are interested in describing extensions $\mathbb{F}_{p^n}$ of $\mathbb{F}_p$; such an extension has dimension n over $\mathbb{F}_p$, so representing an element in it involves n base field elements.

It is customary to use polynomial arithmetic to describe these extensions (but not necessary: Lenstra's algorithm [29] uses a multiplication tensor). For an extension degree n, a first step is to construct an irreducible polynomial Q_n of degree n in $\mathbb{F}_p[x]$. Identifying $\mathbb{F}_{p^n}$ with $\mathbb{F}_p[x]/\langle Q_n\rangle$, operations $(+, \times, \div)$ in $\mathbb{F}_p[x]/\langle Q_n\rangle$ all take quasi-linear time in n.

However, this is not sufficient: we also want mechanisms for e.g. field embeddings. Given irreducible polynomials Q_m and Q_n over $\mathbb{F}_p$, with $\deg(Q_m) = m$ dividing $\deg(Q_n) = n$, there exist algorithms to embed $\mathbb{F}_p[x]/\langle Q_m\rangle$ in $\mathbb{F}_p[x]/\langle Q_n\rangle$ (for the system to be consistent, these embeddings must be *compatible* [4]). However, most algorithms use linear algebra techniques.

To bypass these issues, we use an approach inspired by Shoup's algorithm for computing irreducible polynomials [35, 36] (see also [16, 30]): first reduce to the case of prime power degrees, then use composita techniques, in a manner that ensures compatibility of the embeddings automatically.

Background: towers. Suppose that for any prime ℓ, an *ℓ-adic tower* over $\mathbb{F}_p$ is available. By this, we mean a family of polynomials $(T_{\ell,i})_{i\geq 1}$, with $T_{\ell,i} \in \mathbb{F}_p[x_1, \ldots, x_i]$, monic of degree ℓ in x_i, such that for all i the ideal $\langle T_{\ell,1}, \ldots, T_{\ell,i}\rangle$ is maximal in $\mathbb{F}_p[x_1, \ldots, x_i]$. Our model of the field with p^{ℓ^i} elements could then be $\mathbb{K}_{\ell^i} = \mathbb{F}_p[x_1, \ldots, x_i]/\langle T_{\ell,1}, \ldots, T_{\ell,i}\rangle$, but we prefer to work with univariate polynomials (the cost of arithmetic operations is higher in the multivariate basis).

For $1 \leq i \leq n$, let then $Q_{\ell,i}$ be the minimal polynomial of x_i in the extension $\mathbb{K}_{\ell^n}/\mathbb{F}_p$. This polynomial does not depend on n, but only on i; it is monic, irreducible of degree ℓ^i in $\mathbb{F}_p[x_i]$ and allows us to define $\mathbb{F}_{p^{\ell^i}}$ as $\mathbb{F}_p[x_i]/\langle Q_{\ell,i}\rangle$. For $1 \leq i \leq j \leq n$, let further $Q_{\ell,i,j-i}$ be the minimal polynomial of x_j in the extension $\mathbb{F}_p[x_i]/\langle Q_{\ell,i}\rangle \hookrightarrow \mathbb{K}_{\ell^n}$ (as above, it does not depend on n). This polynomial is monic, irreducible of degree ℓ^{j-i} in $\mathbb{F}_{p^{\ell^i}}[x_j] = \mathbb{F}_p[x_i]/\langle Q_{\ell,i}\rangle[x_j]$.

Thus, $\mathbb{F}_p[x_j]/\langle Q_{\ell,j}\rangle$ and $\mathbb{F}_p[x_i, x_j]/\langle Q_{\ell,i}, Q_{\ell,i,j-i}\rangle$ are two models for $\mathbb{F}_{p^{\ell^j}}$. Provided conversion algorithms between these representations are available, we can perform embeddings (that will necessarily be compatible) between different levels of the ℓ-adic tower, i.e. extensions of degrees $(\ell^i)_{i\geq 1}$.

Such towers, together with efficient conversion algorithms, were constructed in the cases $\ell = p$ in [13, 15, 19], $\ell = 2$ in [21], and for other values of ℓ in [18]. Thus, it remains to give algorithms to "glue" towers defined for different values of ℓ. This is the purpose of this paper.

Our contribution. The algorithms used to construct towers are inspired by those used in [35, 36, 16] to build irreducible polynomials. Also used in these references is the following idea: let $Q_m(x)$ and $Q_n(y)$ be irreducible polynomials over $\mathbb{F}_p$, with coprime degrees $m, n > 1$, and having respectively $(a_i)_{1\le i\le m}$ and $(b_j)_{1\le j\le n}$ as roots in an algebraic closure of $\mathbb{F}_p$. Then their *composed product* $Q_{mn} = \prod_{1\le i\le m, 1\le j\le n}(z - a_i b_j)$ is irreducible of degree mn in $\mathbb{F}_p[z]$.

In this paper, we use an *algebraic complexity model*, where the cost of an algorithm is counted in terms of the number of operations $(+, \times, \div)$ in $\mathbb{F}_p$. If the goal is building irreducible polynomials, then computing Q_{mn} is enough: an algorithm given in [6] has quasi-linear cost in mn. Our goal here is to give algorithms for further operations: computing embeddings of the form $\varphi_x : \mathbb{F}_p[x]/\langle Q_m\rangle \to \mathbb{F}_p[z]/\langle Q_{mn}\rangle$ or $\varphi_y : \mathbb{F}_p[y]/\langle Q_n\rangle \to \mathbb{F}_p[z]/\langle Q_{mn}\rangle$, and the isomorphism $\Phi : \mathbb{F}_p[x,y]/\langle Q_m, Q_n\rangle \to \mathbb{F}_p[z]/\langle Q_{mn}\rangle$ or its inverse.

Standard solutions to these questions exist, using *modular composition* techniques: once the image $S = \Phi(x)$ is known, computing $\varphi_x(a)$ amounts to computing $a(S) \bmod Q_{mn}$; similarly, computing $\Phi(b)$, for b in $\mathbb{F}_p[x,y]/\langle Q_m, Q_n\rangle$, amounts to computing $b(S,T) \bmod Q_{mn}$, with $T = \Phi(y)$. This can be done using the Brent and Kung algorithm [11]: the resulting cost is $O(mn^{(\omega+1)/2}) \subset O(mn^{1.69})$ for φ_x (see the analysis in [36]) and $O((mn)^{(\omega+1)/2}) \subset O(m^{1.69}n^{1.69})$ for Φ or its inverse [33]. Here, we denote by ω a constant in $(2,3]$ such that one can multiply matrices of size m over any ring A using $O(m^\omega)$ operations $(+,\times)$ in A; using the algorithms of [14, 40], we can take $\omega \le 2.38$.

Our main result improves on these former ones. We denote by $\mathsf{M} : \mathbb{N} \to \mathbb{N}$ a function such that for any ring A, polynomials in $A[x]$ of degree at most n can be multiplied in $\mathsf{M}(n)$ operations $(+,\times)$ in A, and we make the usual superlinearity assumptions on M [22, Chapter 8].

THEOREM 1. *One can apply φ_x (resp. φ_y) to an element of $\mathbb{F}_p[x]/\langle Q_m\rangle$ (resp. $\mathbb{F}_p[x]/\langle Q_n\rangle$), or invert it on its image, using $O(n\mathsf{M}(m) + m\mathsf{M}(n))$ operations in $\mathbb{F}_p$.*

Suppose that $m \le n$. Then one can apply Φ to an element of $\mathbb{F}_p[x,y]/\langle Q_m, Q_n\rangle$ or invert it using either $O(m^2\mathsf{M}(n))$ or $O(\mathsf{M}(mn)n^{1/2} + \mathsf{M}(m)n^{(\omega+1)/2})$ operations in $\mathbb{F}_p$.

Using the $O^{\tilde{}}$ notation to neglect polylogarithmic factors, we can take $\mathsf{M}(n) \in O^{\tilde{}}(n)$. Our algorithm for embeddings and their inverses has quasi-linear cost $O^{\tilde{}}(mn)$. Those for Φ or Φ^{-1} have respective costs $O^{\tilde{}}(m^2n)$ and $O^{\tilde{}}(mn^{(\omega+1)/2})$; the minimum of the two is in $O^{\tilde{}}((mn)^{2\omega/(\omega+1)})$; for $\omega \in (2,3]$, the resulting exponent is in $(1.333\ldots, 1.5]$.

If $S = \Phi(x)$ and $T = \Phi(y)$ are known, a result by Kedlaya and Umans [26] for modular composition, and its extension in [32], yield an algorithm with *bit complexity* essentially linear in mn and $\log(p)$ on a RAM. Unfortunately, making these algorithms competitive in practice is challenging; we are not aware of any implementation of them. It is also worth noting that our algorithms apply in a more general setting than finite fields (mild assumptions are required).

Outline. Section 2 presents basic algorithms for polynomials and their transposes. Section 3 introduces the main idea behind our algorithms: the trace induces a duality on algebras of the form $\mathbb{F}_p[x]/\langle Q\rangle$, and some conversion algorithms are straightforward in dual bases; the algorithms are detailed in Section 4. Section 5 explains how the results in this paper can be used in order to construct the algebraic closure of $\mathbb{F}_p$. We conclude with experimental results.

2. PRELIMINARIES

We recall first previous results concerning polynomial arithmetic and transposition of algorithms. In all this section, a ground field k, not necessarily finite, is fixed. For integers m, n, we denote by $k[x]_m$ (resp. $k[x,y]_{m,n}$) the set of polynomials P in $k[x]$ with $\deg(P) < m$ (resp. P in $k[x,y]$ with $\deg(P,x) < m$ and $\deg(P,y) < n$).

2.1 Polynomial multiplication and remainder

We start with some classical algorithms and their complexity. For all the algorithms that follow, all polynomials are written on the canonical monomial basis (this is innocuous for the moment, but other bases will be discussed below).

The product of two polynomials of respective degrees at most m and n can be computed in $\mathsf{M}(\max(m,n))$ operations in k. If P is a monic polynomial of degree m in $k[x]$, for $n \ge 1$, we let $\mathsf{rem}(.,P,n)$ be the operator

$$\begin{array}{rccc} \mathsf{rem}(.,P,n): & k[x]_n & \to & k[x]_m \\ & a & \mapsto & a \bmod P. \end{array}$$

For $n \le m$, this is free of cost. For $n > m$, this can be computed in time $O(n\mathsf{M}(m)/m)$ using the Cook-Sieveking-Kung algorithm and blocking techniques [5, Ch. 5.1.3]. Defining $A = k[x]/\langle P\rangle$, and choosing a fixed $b \in A$, we can then define the mapping $\mathsf{mulmod}(.,b,P)$, which maps $a \in A$ to $ab \bmod P$; it can be computed in time $O(\mathsf{M}(m))$. Finally, given an integer m, the reversal operator in length m is

$$\begin{array}{rccc} \mathsf{rev}(.,m): & k[x]_m & \to & k[x]_m \\ & a & \mapsto & x^{m-1}a(1/x). \end{array}$$

2.2 Duality and the transposition principle

The *transposition principle* is an algorithmic result which states that, given an algorithm that performs a matrix-vector product $u \mapsto Mu$, one can deduce an algorithm with essentially the same cost which performs the transposed matrix-vector product $v \mapsto M^t v$ [12, Ch. 13].

Following [20], we give here a more abstract presentation of the transposition principle, using the algebraic theory of duality (see [9, Ch. IX.1.8]). The added level of abstraction will pay off by greatly simplifying the proofs of the next sections.

Let E and F be k-vector spaces, with $\dim(E) = \dim(F) < \infty$, and suppose that $\langle .,. \rangle : E \times F \to k$ is a non-degenerate bilinear form. Then, to any vector space basis $\boldsymbol{\xi} = (\xi_i)_i$ of E, we can associate a unique *dual basis* $\boldsymbol{\xi}^* = (\xi_i^*)_i$ of F such that $\langle \xi_i, \xi_j^* \rangle = \delta_{i,j}$ (the Kronecker symbol). In other words, given a in F, the coefficients (a_i) of a on the basis $\boldsymbol{\xi}^*$ are given by $a_i = \langle \xi_i, a\rangle$.

For example, denote by E^* the dual space of E, i.e. the k-linear forms on E. The bilinear form on $E \times E^*$ defined by with $\langle v, \ell\rangle = \ell(v)$ for all $v \in E$ and $\ell \in E^*$ is non-degenerate. This is indeed the canonical example, and any non-degenerate form, is isomorphic to this one. We will see in the next section another family of examples, with $E = F$.

Let E', F' be two further vector spaces, with $\dim(E') = \dim(F') < \infty$ and let $\langle .,. \rangle'$ be a bilinear form $E' \times F' \to k$. Then, to any linear mapping $u : E \to E'$, one associates its *dual* (with respect to $\langle .,. \rangle$ and $\langle .,. \rangle'$), which is a linear mapping $u^t : F' \to F$ characterized by the equality $\langle u(a), b'\rangle' = \langle a, u^t(b')\rangle$, for all $a \in E$ and $b' \in F'$.

Let as above $\boldsymbol{\xi}$ be a basis of E, and let $\boldsymbol{\xi}^*$ be the dual basis of F; consider as well a basis $\boldsymbol{\upsilon}$ of E' and its dual basis

$\boldsymbol{\upsilon}^*$ of F'. If M is the matrix of u in the bases $(\boldsymbol{\xi}, \boldsymbol{\upsilon})$, the matrix of u^t in the bases $(\boldsymbol{\upsilon}^*, \boldsymbol{\xi}^*)$ is the transpose of M.

As presented in [8, 20], the transposition principle is an algorithmic technique that, given an algorithm to compute $u : E \to E'$ in the bases $(\boldsymbol{\xi}, \boldsymbol{\upsilon})$, yields an algorithm for the dual map $u^t : F' \to F$ in the bases $(\boldsymbol{\upsilon}^*, \boldsymbol{\xi}^*)$. The two algorithms have same cost, up to $O(\dim(E) + \dim(E'))$. In a nutshell, starting from an algorithm relying on a few basic operations (such as polynomial or matrix multiplication), its transpose is obtained by transposing each basic subroutine, then reversing their order.

Let us briefly review the transposes of operations described in the previous subsection. The transpose of polynomial multiplication is described in [8]; it is closely related to the *middle product* [24]. Let next P be monic of degree m, and define $A = k[x]/\langle P\rangle$. As shown in [8], the dual map of rem

$$\mathsf{rem}^t(., P, n): \quad A^* \quad \to \quad k[x]_n^*$$

is equivalent to *linear sequence extension*: it takes as input the initial m values of a linear recurring sequence of minimal polynomial P, and outputs its first n values. The transposed version of the Cook-Sieveking-Kung fast Euclidean division algorithm yields an algorithm with cost $O(n\mathsf{M}(m)/m)$ operations in k [23, 38].

For a fixed $b \in A$, the transpose of mulmod is the map

$$\begin{array}{rccc} \mathsf{mulmod}^t(., b, P): & A^* & \to & A^* \\ & \ell & \mapsto & b \cdot \ell, \end{array}$$

where $b \cdot \ell$ is defined by $(b \cdot \ell)(a) = \ell(ab)$. Algorithms for mulmod^t have been subject to much research (for instance, Berlekamp's *bit serial multiplication* [2] is a popular arithmetic circuit for mulmod^t in the case $k = \mathbb{F}_2$); algorithms of cost $O(\mathsf{M}(m))$ are given in [38, 8].

Lastly, the reversal operator on $k[x]_m$ is its own transpose.

3. TRACE AND DUALITY

Next, we discuss some classical facts about the trace form, and give algorithms to change between monomial bases and their duals. In all this section, k is a perfect field. General references for the following are [27, 17].

Traces in reduced algebras. Let s be a positive integer and I a zero dimensional radical ideal in $k[x_1, \ldots, x_s]$. Thus, $A = k[x_1, \ldots, x_s]/I$ is a reduced k-algebra of finite dimension d, where d is the cardinality of $V = V(I) \subset \overline{k}^s$ (in general, A is not a field).

Let a be in A. As we did in the case of one variable, we associate to a the endomorphism of multiplication-by-a $M_a : A \to A$ given by $M_a(b) = ab$. Even though A may not be a field, we still define the *minimal polynomial* of a as the minimal polynomial of M_a; since I is radical, this polynomial is squarefree, with roots $a(x)$, for x in V. Similarly, the *trace* of a is the trace of M_a, and denote it by $\tau_I(a)$. Because I is radical, the trace defines a non-degenerate bilinear form on $A \times A$, given by $\langle a, b\rangle_I = \tau_I(ab)$.

Thus, to any basis $\boldsymbol{\xi} = (\xi_i)_{0\le i<d}$ of A, one can associate a dual basis $\boldsymbol{\xi}^* = (\xi_i^*)_{0\le i<d}$, such that $\langle \xi_i, \xi_j^*\rangle_I = \delta_{i,j}$ for all i, j. It will be useful to keep in mind that for $a \in A$, its expression on the dual basis $\boldsymbol{\xi}^*$ is $a = \sum_{0\le i<d} \langle a, \xi_i\rangle_I \xi_i^*$.

We now describe algorithms for converting between the monomial basis and its dual, in two particular cases, involving respectively univariate and bivariate polynomials. In both cases, our conclusion will be that such conversions have quasi-linear complexity.

Univariate conversion. Let P be monic of degree m and squarefree in $k[x]$, and define $A = k[x]/\langle P\rangle$. We denote by P' its derivative and by τ_P the trace modulo the ideal $\langle P\rangle$.

The k-algebra A is endowed with the canonical monomial basis $\boldsymbol{\xi} = (x^i)_{0\le i<m}$. In view of what was said in the previous subsection, the coefficients of an element $a \in A$ on the dual basis $\boldsymbol{\xi}^*$ are the traces $\tau_P(ax^i)_{0\le i<m}$. The following lemma shows that the generating series of these traces is rational, with a known denominator; this will be the key to the conversion algorithm. This is a restatement of well-known results, see for instance the proof of [34, Theorem 3.1].

LEMMA 1. *For a in A, the following holds in $k[[x]]$:*

$$\sum_{i\ge 0} \tau_P(ax^i)x^i = \frac{\mathsf{rev}(P'a \bmod P, m)}{\mathsf{rev}(P, m+1)}.$$

Some well-known algorithms to convert between $\boldsymbol{\xi}$ and $\boldsymbol{\xi}^*$ follow easily. In these algorithms, and all that follows, input and output are vectors (written in sans serif font).

Algorithm 1: **MonomialToDual**(a, P)

Input $\mathsf{a} = (a_i)_{0\le i<m} \in k^m$,
P monic squarefree in $k[x]$ of degree m
Output $(\tau_P(ax^i))_{0\le i<m}$, with $a = \sum_{0\le i<m} a_i x^i$
1. $T = 1/\mathsf{rev}(P, m+1) \bmod x^m$
2. $b = \mathsf{rev}(P' \sum_{0\le i<m} a_i x^i \bmod P, m)\, T \bmod x^m$
3. **return** $(\mathsf{coefficient}(b, x^i))_{0\le i<m}$

Algorithm 2: **DualToMonomial**(b, P)

Input $\mathsf{b} = (b_i)_{0\le i<m} \in k^m$,
P monic squarefree in $k[x]$ of degree m
Output $(a_i)_{0\le i<m}$ such that $\tau_P(\sum_{0\le i<m} a_i x^{i+j}) = b_j$ for all j
1. $S = 1/P' \bmod P$
2. $b = \mathsf{rev}(P, m+1) \sum_{0\le i<m} b_i x^i \bmod x^m$
3. $c = \mathsf{rev}(b, m)$
4. $d = c\, S \bmod P$
5. **return** $(\mathsf{coefficient}(d, x^i))_{0\le i<m}$

LEMMA 2. *Algorithms 1 and 2 are correct. The former uses $O(\mathsf{M}(m))$ operations in k, the latter $O(\mathsf{M}(m)\log(m))$. If the polynomial $S = 1/P' \bmod P$ is known, the running time of Algorithm 2 drops to $O(\mathsf{M}(m))$.*

PROOF. Correctness follows from Lemma 1. Once we point out that power series inversion modulo x^m can be done in time $O(\mathsf{M}(m))$, the running time analysis of the former is straightforward. For Algorithm 2, the dominant part is the computation of S, which takes time $O(\mathsf{M}(m)\log(m))$ by fast XGCD; all other steps take $O(\mathsf{M}(m))$ operations in k. □

Bivariate conversions. Now we consider two monic squarefree polynomials P in $k[x]$ of degree m, and Q in $k[y]$ of degree n. We define $A = k[x, y]/I$, with $I = \langle P, Q\rangle$, then A has the canonical monomial basis $(x^i y^j)_{0\le i<m, 0\le j<n}$. We denote by τ_I the trace modulo I, and by τ_P and τ_Q the traces modulo respectively $\langle P\rangle$ and $\langle Q\rangle$.

In addition to its monomial basis, A can be endowed with a total of four natural bases, which are described as follows. Let $\boldsymbol{\xi} = (x^i)_{0\le i<m}$ and $\boldsymbol{\upsilon} = (y^i)_{0\le j<n}$ be the monomial bases of respectively $k[x]/\langle P\rangle$ and $k[y]/\langle Q\rangle$; let $\boldsymbol{\xi}^*$ and $\boldsymbol{\upsilon}^*$ be their respective dual bases, with respect to τ_P and τ_Q.

The monomial basis seen above is $\boldsymbol{\xi} \otimes \boldsymbol{\upsilon}$; the other combinations $\boldsymbol{\xi}^* \otimes \boldsymbol{\upsilon}$, $\boldsymbol{\xi} \otimes \boldsymbol{\upsilon}^*$ and $\boldsymbol{\xi}^* \otimes \boldsymbol{\upsilon}^*$ are bases of A as well. After a precomputation of cost $O(\mathsf{M}(m)\log(m)+\mathsf{M}(n)\log(n))$, Lemma 2 shows that conversions between any pair of these bases can be done using $O(n\mathsf{M}(m) + m\mathsf{M}(n))$ operations in k (by applying the univariate conversion algorithms n times x-wise and / or m times y-wise). Using fast multiplication, this is quasi-linear in the dimension mn of A.

The following easy lemma will help us exhibit the duality relationships between these bases; it follows from the fact that A is the tensor product of $k[x]/\langle P\rangle$ and $k[y]/\langle Q\rangle$.

LEMMA 3. *Let b be in $k[x]/\langle P\rangle$ and c in $k[y]/\langle Q\rangle$. Then we have $\tau_I(bc) = \tau_P(b)\ \tau_Q(c)$.*

This lemma implies that $\boldsymbol{\xi} \otimes \boldsymbol{\upsilon}$ and $\boldsymbol{\xi}^* \otimes \boldsymbol{\upsilon}^*$ are dual to one another with respect to $\langle .,. \rangle_I$, as are $\boldsymbol{\xi}^* \otimes \boldsymbol{\upsilon}$ and $\boldsymbol{\xi} \otimes \boldsymbol{\upsilon}^*$.

4. EMBEDDING AND ISOMORPHISM

This section contains the main algorithms of this paper. We consider two squarefree polynomials $P(x)$ and $Q(y)$ of respective degrees m and n, with coefficients in a perfect field k. Let us then set $A = k[x,y]/I$, where I is the ideal $\langle P(x), Q(y)\rangle$ in $k[x,y]$. In all this section, *we assume that xy is a generator of A as a k-algebra.*

The main example we have in mind is the following: k is a finite field and both P and Q are irreducible, with $\gcd(m,n) = 1$. Then our assumption is satisfied and in addition A is a field, namely, the *compositum* of the fields $k[x]/\langle P\rangle$ and $k[y]/\langle Q\rangle$, see [10]. More generally, if we let $(r_i)_{i<m}$ be the roots of P in an algebraic closure of k, and let $(s_j)_{j<n}$ be the roots of Q, then as soon as the products $r_i s_j$ are pairwise distinct, xy generates A as a k-algebra.

Let $R \in k[z]$ be the minimal polynomial of xy in the extension A/k (equivalently, the roots of R are the products $r_i s_j$); this polynomial is known as the *composed product* of P and Q, and we will denote it $R = P \odot Q$. As k-algebras, we have $A \simeq k[x]/\langle R\rangle$, so there exist embeddings φ_x, φ_y, and an isomorphism Φ of the form

$$\begin{array}{rlccc} & \varphi_x : & k[x]/\langle P\rangle & \to & k[z]/\langle R\rangle \\ & \varphi_y : & k[y]/\langle Q\rangle & \to & k[z]/\langle R\rangle \\ \text{and} & \Phi : & A = k[x,y]/\langle P,Q\rangle & \to & k[z]/\langle R\rangle \\ & & xy & \mapsto & z. \end{array}$$

In this section, we give algorithms for computing R, applying φ_x, φ_y and their sections, and finally Φ and its inverse. Except from the computation of R, these are all linear algebra problems. If R and the images $S = \Phi(x), T = \Phi(y)$ are known, then as was explained in the introduction, direct solutions are available for both φ_x (or φ_y) and Φ – modular composition – but none of these approaches have a quasi-linear running time.

We take a different path. Our algorithms have quasi-linear running time for φ_x and φ_y and improve on the Brent-Kung algorithm for Φ. Put together, Lemmas 5 to 9 below prove Theorem 1. One of the key aspects of these algorithms is that some are written in the usual monomial bases, whereas others are naturally expressed in the corresponding dual bases. From the complexity point of view, this is not an issue, since we saw that all change-of-bases can be done in quasi-linear time.

In what follows, we write $\tau_P, \tau_Q, \tau_R, \tau_I$ for the traces modulo the ideals $\langle P\rangle \subset k[x]$, $\langle Q\rangle \subset k[y]$, $\langle R\rangle \subset k[z]$ and $I = \langle P,Q\rangle \subset k[x,y]$; the corresponding bilinear forms are denoted by $\langle .,. \rangle_P, \ldots$

We let $\boldsymbol{\xi} = (x^i)_{0\le i<m}$, $\boldsymbol{\upsilon} = (y^i)_{0\le j<n}$ and $\boldsymbol{\zeta} = (z^i)_{0\le i<mn}$ be the monomial bases of respectively $k[x]/\langle P\rangle$, $k[y]/\langle Q\rangle$ and $k[z]/\langle R\rangle$. We also let $\boldsymbol{\xi}^* = (\xi_i^*)_{0\le i<m}$, $\boldsymbol{\upsilon}^* = (\upsilon_i^*)_{0\le i<n}$ and $\boldsymbol{\zeta}^* = (\zeta_i^*)_{0\le i<mn}$ be the dual bases, with respect to respectively $\langle .,. \rangle_P$, $\langle .,. \rangle_Q$ and $\langle .,. \rangle_R$.

Finally, we denote by $\mathsf{u}_P \in k^m$ the vector of the coordinates of $1 \in k[x]/\langle P\rangle$ on the dual basis $\boldsymbol{\xi}^*$; the vector u_Q is defined similarly. These vectors can both be computed in quasi-linear time, since we have, for instance, $\mathsf{u}_P = \mathsf{MonomialToDual}((1,0,\ldots,0),P)$. Thus, in what follows, we assume that these vectors are known.

4.1 Embedding and computing R

We first show how to compute the embeddings φ_x and φ_y, and their inverses in quasi-linear time in mn. We actually give a slightly more general algorithm, which computes the restriction of Φ to the set

$$\Pi = \{bc \mid b \in k[x]/\langle P\rangle,\ c \in k[y]/\langle Q\rangle\} \subset k[x,y]/\langle P,Q\rangle.$$

We will use the following lemma, which results from the base independence of the trace (the second equality is Lemma 3).

LEMMA 4. *Let b be in $k[x]/\langle P\rangle$ and c in $k[y]/\langle Q\rangle$. Then we have $\tau_R(\Phi(bc)) = \tau_I(bc) = \tau_P(b)\ \tau_Q(c)$.*

An easy consequence is that $\tau_R(z^i) = \tau_P(x^i)\tau_Q(y^i)$. From this lemma, we also immediately deduce Algorithm 3, which computes the image in $k[z]/\langle R\rangle$ of any element of Π, with inputs and outputs written on dual bases.

Algorithm 3: **Embed**$(\mathsf{b},\mathsf{c},r)$

Input $\mathsf{b} = (b_i)_{0\le i<m} \in k^m$, $\mathsf{c} = (c_i)_{0\le i<n} \in k^n$
an optional integer $r \ge mn$ set to $r = mn$ by default
Output $\mathsf{a} = (a_i)_{0\le i<r} \in k^r$
1. $(t_i)_{0\le i<r} = \mathsf{rem}^t(\mathsf{b}, P, r)$
2. $(u_i)_{0\le i<r} = \mathsf{rem}^t(\mathsf{c}, Q, r)$
3. **return** $(t_i u_i)_{0\le i<r}$

LEMMA 5. *Let $b \in k[x]/\langle P\rangle$ and $c \in k[y]/\langle Q\rangle$. Given the coefficients b and c of respectively b and c in the bases $\boldsymbol{\xi}^*$ and $\boldsymbol{\upsilon}^*$, $\mathsf{Embed}(\mathsf{b},\mathsf{c},r)$ computes $a_i = \tau_R\left(\Phi(bc)z^i\right)$ for $0 \le i < r$ in time $O(r(\mathsf{M}(m)/m + \mathsf{M}(n)/n))$. If $r = mn$, $(a_i)_{0\le i<mn}$ are the coefficients of $\Phi(bc)$ in the basis $\boldsymbol{\zeta}^*$.*

PROOF. Recall that for $0 \le i < m$, $b_i = \tau_P(bx^i)$, and that for $0 \le i < n$, $c_i = \tau_Q(cy^i)$. By definition of rem^t, the sequences (t_i) and (u_i) encode the same traces, but up to index r. By Lemma 4, the algorithm correctly computes

$$\left(\tau_P(bx^i)\tau_Q(cy^i)\right)_{i<r} = \left(\tau_R(\Phi(bc)z^i)\right)_{i<r}.$$

For $r = mn$, this is indeed the representation of $\Phi(bc)$ on the dual basis $\boldsymbol{\zeta}^*$ of $k[z]/\langle R\rangle$. The cost of the calls to rem^t is in Section 2.2; the last step takes r multiplications in k. □

In particular, the map φ_x is computed as $\mathsf{Embed}(\cdot,\mathsf{u}_Q)$, and the map φ_y as $\mathsf{Embed}(\mathsf{u}_P,\cdot)$. Another interesting consequence is that, when A is known to be a field, Embed allows us to compute R, using the Berlekamp-Massey algorithm.

Algorithm 4: **Compute**$R(P,Q)$

Input P in $k[x]$, Q in $k[y]$
Output R in $k[z]$
1. $(t_i)_{0\le i<2mn} = \mathsf{Embed}(\mathsf{u}_P, \mathsf{u}_Q, 2mn)$,
2. **return** $\mathsf{BerlekampMassey}((t_i)_{0\le i<2mn})$

Algorithm 5: **Project**(a)

Input $\mathsf{a} = (a_i)_{0\le i<mn} \in k^{mn}$
Output $\mathsf{b} = (b_i)_{0\le i<m} \in k^m$
1. $\mathsf{c} = (1, 0, \ldots, 0)$
2. $(u_i)_{0\le i<mn} = \mathsf{rem}^t(\mathsf{c}, Q, mn)$
3. $d = \sum_{i=0}^{mn-1} a_i u_i x^i \bmod P$
4. **return** $(\mathsf{coefficient}(d, i))_{0\le i<m}$

Indeed, in this case, $\mathsf{Embed}(\mathsf{u}_P, \mathsf{u}_Q, 2mn)$ computes the sequence $(\tau_R(z^i))_{0\le i<2mn}$. If we know that A is a field, R is irreducible, so the minimal polynomial of this sequence (which is computed by the Berlekamp-Massey algorithm) is precisely R; the running time is $O(\mathsf{M}(mn)\log(mn))$ operations in k. This algorithm for computing R is well-known; see for instance [6] for a variant using power series exponentials instead of Berlekamp-Massey's algorithm (that applies in large enough characteristic) and [7] for the specific case of finite fields of small characteristic.

For the inverse of say φ_x, we take a in $k[z]/\langle R\rangle$ of the form $a = \varphi_x(b)$, and compute b. Using the equality of Lemma 4 in the form $\tau_P(bx^i) = \tau_R(az^i)/\tau_Q(y^i)$ would lead to a simple algorithm, but some traces $\tau_Q(y^i)$ may vanish.

We take a different path. Let c be a fixed element in $k[y]/\langle Q\rangle$ such that $\tau_Q(c) = 1$; we will take for c the first element υ_0^* of the dual basis of $k[y]/\langle Q\rangle$, but this is not necessary. Let us denote by $\epsilon : k[x]/\langle P\rangle \to k[z]/\langle R\rangle$ the mapping defined by $\epsilon(b) = \Phi(bc)$, and let $\epsilon^t : k[z]/\langle R\rangle \to k[x]/\langle P\rangle$ be its dual map with respect to the bilinear forms $\langle .,.\rangle_P$ and $\langle .,.\rangle_R$. Then, for b and b' in $k[x]/\langle P\rangle$, we have

$$\begin{aligned}\langle b, b'\rangle_P &= \tau_P(bb') = \tau_P(bb')\tau_Q(c) = \tau_R(\Phi(bb'c))\\ &= \langle \epsilon(b), \Phi(b')\rangle_R = \langle b, \epsilon^t(\Phi(b'))\rangle_P,\end{aligned}$$

where the third equality comes from Lemma 4. Using the non-degeneracy of $\langle .,.\rangle_P$, we get $\epsilon^t(\Phi(b')) = b'$, that is, $\epsilon^t(\varphi_x(b')) = b'$. Thus, ϵ^t is an inverse of φ_x on its image.

Writing $\mathsf{c} = (1, 0, \ldots, 0)$, we remark that $\mathsf{Embed}(., \mathsf{c})$ precisely computes the mapping $b \mapsto \epsilon(b)$. Since **Embed** is written in the dual bases, the discussion of Section 2.2 shows that transposing this algorithm (with respect to b) yields an algorithm for ϵ^t written in the monomial bases.

LEMMA 6. *Let $b \in k[x]/\langle P\rangle$ and $a = \varphi_x(b)$. Given the coefficients* a *of a in the basis $\boldsymbol{\zeta} = (z^i)_{0\le i<mn}$,* Project(a) *computes the coefficients of b in the basis $\boldsymbol{\xi} = (x^i)_{0\le i<m}$ using $O(n\mathsf{M}(m) + n\mathsf{M}(n))$ operations in k.*

PROOF. We show correctness using transposition techniques as in [8]. For fixed c, $\mathsf{Embed}(\mathsf{b}, \mathsf{c})$ is linear in b and can be written as $\pi_{\mathsf{c}} \circ \mathsf{rem}^t$, where π_{c} is the map that multiplies a vector in k^{mn} coefficient-wise by $(\tau_Q(cy^i))_{i<mn}$, for $c = \sum_{0\le i<n} c_i \upsilon_i^*$; hence, its transpose is $\mathsf{rem} \circ \pi_{\mathsf{c}}^t$. It is evident that $\pi_{\mathsf{c}}^t = \pi_{\mathsf{c}}$ (since π_{c} is a diagonal map), whereas rem is just reduction modulo P. These correspond to steps 3 and 4. The discussion above now proves that the output is $\epsilon^t(a)$. The cost analysis is similar to the one in Lemma 5. □

4.2 Isomorphism

We are not able to give an algorithm for Φ that would be as efficient as those for embedding; instead, we provide two algorithms, with different domains of applicability. In what follows, without loss of generality, *we assume that $m \le n$.*

Recall that $\boldsymbol{\xi} \otimes \boldsymbol{\upsilon}$, $\boldsymbol{\xi}^* \otimes \boldsymbol{\upsilon}$, $\boldsymbol{\xi} \otimes \boldsymbol{\upsilon}^*$ and $\boldsymbol{\xi}^* \otimes \boldsymbol{\upsilon}^*$ are four bases of A, with $(\boldsymbol{\xi} \otimes \boldsymbol{\upsilon}, \boldsymbol{\xi}^* \otimes \boldsymbol{\upsilon}^*)$ and $(\boldsymbol{\xi}^* \otimes \boldsymbol{\upsilon}, \boldsymbol{\xi} \otimes \boldsymbol{\upsilon}^*)$ being

Algorithm 6: **Phi1**(b)

Input $\mathsf{b} = (b_{i,j})_{0\le i<m, 0\le j<n} \in k^{m\times n}$
Output $\mathsf{a} = (a_i)_{0\le i<mn} \in k^{mn}$
1. $(u_i)_{0\le i<m(n+1)-1} = \mathsf{rem}^t(\mathsf{u}_P, P, m(n+1)-1)$
2. $(a_i)_{0\le i<mn} = (0, \ldots, 0)$
3. **for** $0 \le i < m$
4. $\quad (t_j)_{0\le j<mn} = \mathsf{rem}^t((b_{i,j})_{0\le j<n}, Q, mn)$
5. $\quad (a_j)_{0\le j<mn} = (a_j + t_j u_{i+j})_{0\le j<mn}$
6. **return** $(a_i)_{0\le i<mn}$

Algorithm 7: **InversePhi1**(a)

Input $\mathsf{a} = (a_i)_{0\le i<mn} \in k^{mn}$
Output $\mathsf{b} = (b_{i,j})_{0\le i<m, 0\le j<n} \in k^{m\times n}$
1. $(u_i)_{0\le i<m(n+1)-1} = \mathsf{rem}^t(\mathsf{u}_P, P, m(n+1)-1)$
2. **for** $i = m-1, \ldots, 0$
3. $\quad d = \sum_{0\le j<mn} a_j u_{i+j} y^j \bmod Q$
4. $\quad (b_{i,j})_{0\le j<n} = (\mathsf{coefficient}(d, j))_{0\le j<n}$
5. **return** $(b_{i,j})_{0\le i<m, 0\le j<n}$

two pairs of dual bases with respect to $\langle .,.\rangle_I$. Our algorithms will exploit all these bases; this is harmless, since conversions between these bases have quasi-linear complexity.

Before giving the details of the algorithms, we make an observation similar to the one we did regarding the transpose of **Embed**. Let Φ^t be the dual map of Φ with respect to $\langle .,.\rangle_I$ and $\langle .,.\rangle_R$. Then, for any $b, b' \in k[z]/\langle R\rangle$, we have:

$$\begin{aligned}\langle b, b'\rangle_I &= \tau_I(bb') = \tau_R(\Phi(bb'))\\ &= \langle \Phi(b), \Phi(b')\rangle_R = \langle b, \Phi^t(\Phi(b'))\rangle_I;\end{aligned}$$

hence, $\Phi^t = \Phi^{-1}$. If $\mathbf{b}$ and $\mathbf{b}^*$ are two bases of $A = k[x,y]/I$, dual with respect to $\langle .,.\rangle_I$ (such as the ones seen above) and if $\mathbf{c}$ and $\mathbf{c}^*$ are two bases of $k[z]/\langle R\rangle$, dual with respect to $\langle .,.\rangle_R$, the previous equality, together with the transposition principle, shows the following: if we have an algorithm for Φ, expressed in the bases $(\mathbf{b}, \mathbf{c})$, transposing it yields an algorithm for Φ^{-1}, expressed in the bases $(\mathbf{c}^*, \mathbf{b}^*)$.

First case: m is small. We start by a direct application of the results in the previous subsection, which is well-suited to situations where m is small compared to n.

Let b be in $k[x,y]/I$ and let $a = \Phi(b)$. Writing $b = \sum_{0\le i<m} b_i x^i$, with all b_i in $k[y]/\langle Q\rangle$, we obtain a straightforward algorithm to compute a: compute all $\Phi(b_i x^i)$ using **Embed**, then sum. Since **Embed** takes its inputs written on the dual bases, the algorithm requires that all b_i be written on the dual basis of $k[y]/\langle Q\rangle$ (equivalently, the input is given on the basis $\boldsymbol{\xi} \otimes \boldsymbol{\upsilon}^*$ of A). We also use the fact that the expression of x^i on the dual basis $\boldsymbol{\xi}^*$ is u_P shifted by i positions to give a more compact algorithm, called **Phi1**.

Transposing this algorithm then gives an algorithm for Φ^{-1}. Its input is given on the monomial basis $(z^i)_{0\le i<mn}$ of $k[z]/\langle R\rangle$; the output is written on the basis $\boldsymbol{\xi}^* \otimes \boldsymbol{\upsilon}$ of A.

LEMMA 7. *Let $b \in k[x,y]/I$. Given the coefficients* b *of b in the basis $\boldsymbol{\xi} \otimes \boldsymbol{\upsilon}^*$,* Phi1(b) *computes the coefficients of $\Phi(b)$ in the basis $\boldsymbol{\zeta}^*$ using $O(m^2\mathsf{M}(n))$ operations in k.*

Let $a \in k[z]/\langle R\rangle$. Given the coefficients a *of a in the basis $\boldsymbol{\zeta} = (z^i)_{0\le i<mn}$,* InversePhi1(a) *computes the coefficients of $\Phi^{-1}(a)$ in the basis $\boldsymbol{\xi} \otimes \boldsymbol{\upsilon}^*$ using $O(m^2\mathsf{M}(n))$ operations in k.*

PROOF. Correctness of **Phi1** follows from the previous discussion; the most expensive step is m calls to rem^t, for a cumulated cost of $O(m^2\mathsf{M}(n))$.

The correctness of the transposed algorithm is proved as in Lemma 6, observing that it consists of the line-by-line

transposition of Phi1. The running time analysis is straightforward: the dominant cost is that of m remainders, each of which costs $O(m\mathsf{M}(n))$. □

Second case: m is not small. The previous algorithms are most efficient when m is small; now, we propose an alternative solution that does better when m and n are of the same order of magnitude (with still $m \le n$).

This approach is based on baby steps / giant steps techniques, as in Brent and Kung's modular composition algorithm, but uses the fact that $z = \Phi(xy)$ to reduce the cost. Given b in $A = k[x,y]/\langle P, Q\rangle$, let us write

$$b = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} b_{i,j}x^i y^j = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} b_{i,j}x^i y^i y^{j-i}$$
$$= \sum_{h=-m+1}^{n-1}\sum_{i=0}^{m-1} b_{i,i+h}(xy)^i y^h = \frac{1}{y^{m-1}}\sum_{h=0}^{m+n-2} c_h(xy)y^h,$$

with $c_h(z) = \sum_{0\le i<m} b_{i,i+h-m+1}z^i$ for all h (undefined indices are set to zero). Hence $a = \Phi(b)$ has the form

$$a = \frac{1}{T^{m-1}}\widetilde{a} \mod R \quad \text{with} \quad \widetilde{a} = \sum_{h=0}^{m+n-2} c_h T^h,$$

where $T = \Phi(y)$. We use baby steps / giant steps techniques from [28] (inspired by Brent and Kung's algorithm) to compute a, reducing the problem to polynomial matrix multiplication. Let $n' = m+n-1$, $p = \lceil\sqrt{n'}\rceil$ and $q = \lceil n'/p\rceil$, so that $n \le n' \le 2n-1$ and $p \simeq q \simeq \sqrt{n}$. For baby steps, we compute the polynomials $T_i = T^i \bmod R$, which have degree at most $mn-1$; we write $T_i = \sum_{0\le j<n} T'_{i,j}z^{jm}$, with $T'_{i,j}$ of degree less than m, and build the polynomial matrix $M_{T'}$ with entries $T'_{i,j}$. We define the matrix $M_C = [c_{iq+j}]_{0\le i<p, 0\le j<q}$ containing the polynomials c_h organized in a row-major fashion, and compute the product $M_V = M_C M_T$. We can then construct polynomials from the rows of M_V, and conclude with giant steps using Horner's scheme.

The previous discussion leads to Algorithm 8. Remark that input *and* output are written on the monomial bases.

Lemma 8. *Let $b \in k[x,y]/I$. Given the coefficients* b *of b in the basis $\boldsymbol{\xi}\otimes\boldsymbol{\upsilon} = (x^i y^j)_{0\le i<m, 0\le j<n}$,* Phi2(b) *computes the coefficients of $\Phi(b)$ in the basis $\boldsymbol{\zeta} = (z^i)_{0\le i<mn}$ in $O(\mathsf{M}(mn)n^{1/2} + \mathsf{M}(m)n^{(\omega+1)/2})$ operations in k.*

Proof. Correctness follows from the discussion prior to the algorithm. As to the cost analysis, remark first that $n' = O(n)$, and that p and q are both $O(\sqrt{n})$. Steps 4 and 14 cost $O(\mathsf{M}(mn)\log(mn))$ operations. Steps 5 (the baby steps) and the loop at Step 12 (the giant steps) cost $O(\sqrt{n}\mathsf{M}(mn))$. The dominant cost is the matrix product at Step 8, which involves matrices of size $O(\sqrt{n})\times O(\sqrt{n})$ and $O(\sqrt{n})\times O(n)$, with polynomial entries of degree m: using block matrix multiplication in size $O(\sqrt{n})$, this takes $O(\mathsf{M}(m)n^{(\omega+1)/2})$ operations in k. □

As before, writing the transpose of this algorithm gives us an algorithm for Φ^{-1}, this time written in the dual bases. The process is the same for the previous transposed algorithms we saw, involving line-by-line transposition. The only point that deserves mention is Step 13, where we transpose polynomial matrix multiplication; it becomes a similar matrix product, but this time involving transposed polynomial multiplications (with degree parameters $m-1$ and m). The cost then remains the same, and leads to Lemma 9.

Algorithm 8: **Phi2**(b)

Input b $= (b_{i,j})_{0\le i<m, 0\le j<n} \in k^{m\times n}$
Output a $= (a_i)_{0\le i<mn} \in k^{mn}$

1. $n' = m+n-1$, $p = \lceil\sqrt{n'}\rceil$, $q = \lceil n'/p\rceil$
2. y = MonomialToDual$((0,1,0,\ldots,0), Q)$
3. T = DualToMonomial(Embed(u_P, y), R)
4. $U = 1/T \bmod R$
5. $T' = [T^i \bmod R]_{0\le i\le q}$
6. $M_{T'} = [T'_{i,j}]_{0\le i<q, 0, \le j<n}$ — $T'_{i,j}$ are defined in the text
7. $M_C = [c_{iq+j}]_{0\le i<p, 0\le j<q}$ — c_h are defined in the text
8. $M_V = M_C M_{T'}$
9. $V = [\sum_{0\le j<n} M_{V\,i,j} z^{jm}]_{0\le i<p}$
10. $V' = [V_i \bmod R]_{0\le i<p}$
11. $a = 0$
12. **for** $i = p-1, \ldots, 0$
13. $\quad a = T'_q a + V'_i \bmod R$
14. $a = a\,U^{m-1} \bmod R$
15. **return** (coefficient$(a,i))_{0\le i<mn}$

Algorithm 9: **InversePhi2**(a)

Input a $= (a_i)_{0\le i<mn} \in k^{mn}$
Output b $= (b_{i,j})_{0\le i<m, 0\le j<n} \in k^{m\times n}$

1. $n' = m+n-1$, $p = \lceil\sqrt{n'}\rceil$, $q = \lceil n'/p\rceil$
2. y = MonomialToDual$((0,1,0,\ldots,0), Q)$
3. T = DualToMonomial(Embed(u_P, y), R)
4. $U = 1/T \bmod R$
5. $T' = [T^i \bmod R]_{0\le i\le q}$
6. $M_{T'} = [T'_{i,j}]_{0\le i<q, 0, \le j<n}$ — $T'_{i,j}$ as defined above
7. a = mulmod^t(a, U^{m-1}, R)
8. **for** $i = 0, \ldots, p-1$
9. $\quad V'_i$ = a
10. $\quad$ a = mulmod^t(a, T'_q, R)
11. $V = [\mathsf{rem}^t(V'_i, R, mn+m-1)]_{0\le i<p}$
12. $M_V = [(V_i)_{jm,\ldots,jm+2m-2}]_{0\le i<p, 0\le j<n}$
13. $M_C = \mathsf{mul}^t(M_V, M_{T'}, m-1, m)$
14. $c = [M_{C\,0,0}, \ldots, M_{C\,0,q-1}, \ldots, M_{C\,p-1,q-1}]$
15. **return** [coefficient$(c_{i-j+m-1}, i)]_{0\le i<m, 0\le j<n}$

Lemma 9. *Let $a \in k[z]/\langle R\rangle$. Given the coefficients* a *of a in the basis $\boldsymbol{\zeta}^*$,* InversePhi2(a) *computes the coefficients of $\Phi^{-1}(a)$ in the basis $\boldsymbol{\xi}^*\otimes\boldsymbol{\upsilon}^*$ in $O(\mathsf{M}(mn)n^{1/2} + \mathsf{M}(m)n^{(\omega+1)/2})$ operations in k.*

5. THE ALGEBRAIC CLOSURE OF $\mathbb{F}_p$

In this section, we explain how the algorithms of Section 4 can be used in order to construct and work in arbitrary extensions of $\mathbb{F}_p$, when used in conjunction with algorithms for *ℓ-adic towers* over $\mathbb{F}_p$. Space constraints prevent us from giving detailed algorithms, so we only outline the construction. We reuse definitions given in the introduction relative to ℓ-adic towers: polynomials $T_{\ell,i}$, $Q_{\ell,i}$ and $Q_{\ell,i,j-i}$ and fields $\mathbb{K}_{\ell^i} = \mathbb{F}_p[x_1,\ldots,x_i]/\langle T_{\ell,1},\ldots,T_{\ell,i}\rangle$. We also assume that algorithms for embeddings or change of basis in ℓ-adic towers are available (as in [18] and references therein).

Setup. For ℓ prime and $i \ge 1$, the residue class of x_i in $\mathbb{K}_{\ell^i}$ will be written x_{ℓ^i}. For a positive integer $m = \ell_1^{e_1}\cdots\ell_r^{e_r}$, with ℓ_i pairwise distinct primes and e_i positive integers, $\mathbb{K}_m$ denotes the tensor product $\mathbb{K}_{\ell_1^{e_1}}\otimes\cdots\otimes\mathbb{K}_{\ell_r^{e_r}}$; this is a field with p^m elements. If m divides n, then $\mathbb{K}_m$ embeds in $\mathbb{K}_n$. Taking the direct limit of all $\mathbb{K}_m$ under such embeddings, we get an algebraic closure $\mathbb{K}$ of $\mathbb{F}_p$. The residue classes written x_{ℓ^e} in $\mathbb{K}_{\ell^e}$ all lie in $\mathbb{K}$ and are still written x_{ℓ^e}.

For any integer m of the form $m = \ell_1^{e_1}\cdots\ell_r^{e_r}$ with ℓ_i's pairwise distinct primes, we write $x_m = x_{\ell_1^{e_1}}\cdots x_{\ell_r^{e_r}} \in \mathbb{K}$.

Minimal polynomials. We discuss first minimal polynomials of monomials in $\mathbb{K}$ over $\mathbb{F}_p$.

Take x_{ℓ^e} in $\mathbb{K}$, with ℓ prime. By construction, its minimal polynomial over $\mathbb{F}_p$ is $Q_{\ell,e}$, irreducible of degree ℓ^e in (say) $\mathbb{F}_p[z]$. Next, consider a term x_m, with $m = \ell_1^{e_1} \cdots \ell_r^{e_r}$, with ℓ_i's pairwise distinct primes. It equals $x_{\ell_1^{e_1}} \cdots x_{\ell_r^{e_r}}$, so it is a root of the composed product $Q_m = Q_{\ell_1,e_1} \odot \cdots \odot Q_{\ell_r,e_r}$. In Section 4, we pointed out that Q_m is irreducible of degree $m = \ell_1^{e_1} \cdots \ell_r^{e_r}$ in $\mathbb{F}_p[z]$, so it must be the minimal polynomial of x_m over $\mathbb{F}_p$. In particular, this implies that $\mathbb{F}_p(x_m)$ is a field with p^m elements, and that if we consider terms x_m and x_n, with m dividing n, then x_m is in $\mathbb{F}_p(x_n)$.

Note that this process of constructing irreducible polynomials over $\mathbb{F}_p$ is already in [35, 36, 16].

Embedding and change of basis. Consider a sequence $e = (e_1, \ldots, e_t)$ of positive integers, and let $n = e_1 \cdots e_t$. The set

$$B_e = \{x_{e_1}^{a_1} x_{e_1 e_2}^{a_2} \cdots x_{e_1 \cdots e_t}^{a_t} \mid 0 \le a_i < e_i \text{ for all } i\}$$

is a basis of $\mathbb{F}_p(x_n)$. Important examples are sequences of the form $e = (e_1)$, with thus $n = e_1$, for which B_e is the univariate basis $(x_n^i)_{0 \le i < n}$. Also useful for us are sequences $e = (e_1, e_2)$; letting $m = e_1$ and $n = e_1 e_2$, B_e is the bivariate basis $(x_m^i x_n^j)_{0 \le i < m, 0 \le j < n/m}$.

Consider sequences $d = (d_1, \ldots, d_s)$ and $e = (e_1, \ldots, e_t)$, with $m = d_1 \cdots d_s$ and $n = e_1 \cdots e_t$, and suppose that m divides n. The linear mapping $\mathbb{F}_p^m \to \mathbb{F}_p^n$ that describes the embedding $\mathbb{F}_{p^m} \to \mathbb{F}_{p^n}$ in the bases B_d and B_e is denoted by $\Phi_{e,d}$; when $m = n$, it is an isomorphism, with inverse $\Phi_{d,e}$. More generally, as soon as this expression makes sense, we have $\Phi_{f,d} = \Phi_{f,e} \circ \Phi_{e,d}$, so these mappings are compatible.

To conclude this section, we describe how the algorithms of this paper can be used in this framework to realize some particular cases of mappings $\Phi_{d,e}$ (more general examples can be deduced readily).

Embedding. Consider two integers m, n with m dividing n. We describe here how to embed $\mathbb{F}_p(x_m)$ in $\mathbb{F}_p(x_n)$, that is, how to compute $\Phi_{(n),(m)}$. Without loss of generality, we may assume that $n = m\ell$, with ℓ prime.

Assume first that $\gcd(m, \ell) = 1$. Since then $x_n = x_m x_\ell$, and we have access to the polynomials Q_m, Q_ℓ and Q_n (see above), we just apply the embedding algorithm of Section 4.

Suppose now that ℓ divides m, so $m = m'\ell^k$, with m', ℓ coprime. Using one of the inverse isomorphism algorithms of Section 4, we can rewrite an element given on the basis $(x_m^i)_{0 \le i < m}$ on the basis $(x_{m'}^i x_{\ell^k}^j)_{0 \le i < m', 0 \le j < \ell^k}$. Using an algorithm for embeddings in the ℓ-adic tower, we can then embed on the basis $(x_{m'}^i x_{\ell^{k+1}}^j)_{0 \le i < m', 0 \le j < \ell^{k+1}}$; applying our isomorphism algorithm, we end up on the basis $(x_{m\ell}^i)_{0 \le i < m\ell}$, since $x_{m\ell} = x_{m'} x_{\ell^{k+1}}$.

Further operations. Without entering into details, let us mention that further operations are feasible, in the same spirit as the embedding algorithm we just described.

For instance, for arbitrary integers m and n, it is possible to compute the relative minimal polynomial of x_{mn} over $\mathbb{F}_p(x_m)$; it is obtained as a composed product, with factors deduced from the decomposition of m and n into primes.

As another example, we can compute $\Phi_{(m,n),(mn)}$, that is, go from the univariate basis $(x_{mn}^i)_{0 \le i < mn}$ to the bivariate basis $(x_m^i x_{mn}^j)_{0 \le i < m, 0 \le j < n}$. This can be used to compute for instance relative traces, norms or minimal polynomials of arbitrary elements of $\mathbb{F}_{p^{mn}}$ over $\mathbb{F}_{p^m}$.

Figure 1: Timings in seconds, $p = 5$, $n = m + 1$

6. IMPLEMENTATION

To demonstrate the practicality of our algorithms, we made a C implementation and compared it to various ways of constructing the same fields in Magma. All timings in this section are obtained on an Intel Xeon E5620 CPU at 2.40GHz, using Magma V2.18-12, Flint 2.4.1 and Sage 6.

Our implementation is limited to finite fields of word-sized characteristic. It is based on the C library Flint [25], and we make it available as a Sage module in an experimental fork at `https://github.com/defeo/sage/tree/ff_compositum`. We plan to make it available as a standard Sage module, as well as a separate C library, when the code has stabilized.

Based on the observation that algorithms Embed and Project are simpler than conversion algorithms between monomial and dual bases, we chose to implement a *lazy change of basis* strategy. By this we mean that our Sage module (rather than the C library itself) represents elements on either the monomial or the dual basis, with one representation computed from the other only when needed. For example, two elements of the same field can be summed if both have a monomial or if both have a dual representation. Similarly, two elements can be multiplied using standard multiplication if both have a monomial representation, or using transposed multiplication if one of the two has a monomial representation. In all other cases, the required representation is computed and stored when the user input prompts it. To implement this strategy efficiently, our Sage module is written in the compiled language Cython.

We focus our benchmarks on the setting of Section 4: P and Q are two irreducible polynomials of coprime degrees m and n, and $R = P \odot Q$. We fix the base field $\mathbb{F}_p$ and make m and n grow together with $n = m+1$. We measure the time to compute R, to apply the algorithms Embed, Phi1, etc., and to compute the changes of bases. We noticed no major difference between different characteristics, so we chose $p = 5$ for our demonstration. As shown in Figure 1, the dominating phase is the computation of R (line labeled R). Surprisingly, transposed modular multiplication is slightly faster than ordinary modular multiplication. The cost of Embed is about the same as that of multiplication, while DualToMonomial is about 50% slower. Project and MonomialToDual have, respectively, similar performances (only slightly faster) hence they are not reported on the graph. This justifies our design choice of *lazy change of basis*.

Unsurprisingly, the isomorphism algorithms take significantly more time than the computation of R; for our choices of degrees, Phi2 is asymptotically faster than Phi1 and the crossover between them happens around $m = 70$.

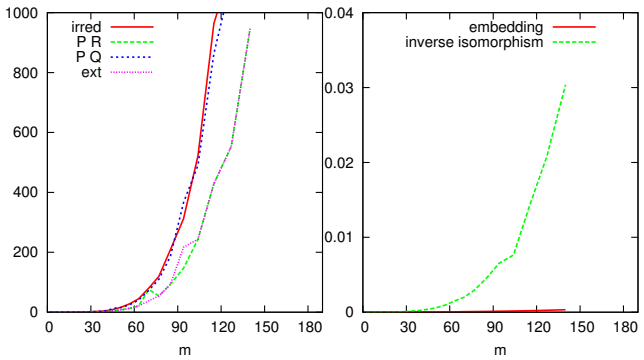

Figure 2: Magma timings in seconds, $p = 5$, $n = m+1$

We compare our implementation to four different strategies available in Magma. For each of them we measure the time to construct the finite fields and embedding data, as well as the time to do operations equivalent to Embed, resp. inverse isomorphism.

Figure 2 reports on the following experiments. In irred, we supply directly P, Q and R to Magma's finite field constructor, then we call the `Embed` routine to compute the embedding data. In P R, we use Magma's default constructor to compute P and R (Magma chooses its own polynomials), then we call the `Embed` routine to compute the embedding. In P Q, we use Magma's default constructor to compute P and Q (Magma chooses its own polynomials), then use the `CommonOverfield` routine to compute R, then `Embed` to compute the embedding data. In ext, we use Magma's default constructor to compute P, then the `ext` operator to compute an extension of degree n of $\mathbb{F}_p[x]/\langle P\rangle$ (Magma chooses its own polynomials).

Timings for constructing the extension and the embedding vary from one method to the other; once this is done, timings for applying embeddings or (inverse) isomorphisms are the same across these methods.

The Magma implementation cannot construct the embedding data in large cases ($m = 150$) in less than 1000 seconds, while our code takes a few seconds. Once the embedding data is known, Magma can apply the embeddings or isomorphisms extremely fast; in our case, one may do the same, using our algorithms to compute the matrices of Φ and Φ^{-1}, when precomputation time and memory are not a concern.

Acknowledgements. We would like to thank the referees for their insightful remarks. Part of this work was financed by NSERC, the CRC program and the ANR project ECLIPSES (ANR-09-VERS-018).

7. REFERENCES

[1] B. Allombert. Explicit computation of isomorphisms between finite fields. *Finite Fields Appl.*, 8(3):332 – 342, 2002.

[2] E. R. Berlekamp. Bit-serial Reed-Solomon encoders. *IEEE Trans. Inf. Theory*, 28(6):869–874, 1982.

[3] W. Bosma, J. Cannon, and C. Playoust. The MAGMA algebra system I: the user language. *J. Symb. Comput.*, 24(3-4):235–265, 1997.

[4] W. Bosma, J. Cannon, and A. Steel. Lattices of compatibly embedded finite fields. *J. Symb. Comput.*, 24(3-4):351–369, 1997.

[5] A. Bostan. *Algorithmes rapides pour les polynômes, séries formelles et matrices*, volume 1 of *Les cours du CIRM*. 2010.

[6] A. Bostan, P. Flajolet, B. Salvy, and É. Schost. Fast computation of special resultants. *J. Symb. Comput.*, 41(1):1–29, 2006.

[7] A. Bostan, L. González-Vega, H. Perdry, and É. Schost. From Newton sums to coefficients: complexity issues in characteristic p. In *MEGA'05*, 2005.

[8] A. Bostan, G. Lecerf, and É. Schost. Tellegen's principle into practice. In *ISSAC'03*, pages 37–44. ACM, 2003.

[9] N. Bourbaki. *Éléments de mathématique*. Springer, 2007. Algèbre. Chapitre 9.

[10] J. V. Brawley and L. Carlitz. Irreducibles and the composed product for polynomials over a finite field. *Discrete Math.*, 65(2):115–139, 1987.

[11] R. P. Brent and H.-T. Kung. Fast algorithms for manipulating formal power series. *Journal of the ACM*, 25(4):581–595, 1978.

[12] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic Complexity Theory*. Springer, February 1997.

[13] D. G. Cantor. On arithmetical algorithms over finite fields. *J. Combin. Theory Ser. A*, 50(2):285–300, 1989.

[14] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990.

[15] J.-M. Couveignes. Isomorphisms between Artin-Schreier towers. *Math. Comp.*, 69(232):1625–1631, 2000.

[16] J.-M. Couveignes and R. Lercier. Fast construction of irreducible polynomials over finite fields. *Israel J. Math.*, 194(1):77–105, 2013.

[17] D. A. Cox, J. Little, and D. O'Shea. *Using Algebraic Geometry*. Springer-Verlag, 2005.

[18] L. De Feo, J. Doliskani, and É. Schost. Fast algorithms for ℓ-adic towers over finite fields. In *ISSAC'13*, pages 165–172. ACM, 2013.

[19] L. De Feo and É. Schost. Fast arithmetics in Artin-Schreier towers over finite fields. *J. Symb. Comput.*, 47(7):771–792, 2012.

[20] Luca De Feo. *Algorithmes Rapides pour les Tours de Corps Finis et les Isogénies*. PhD thesis, École Polytechnique X, December 2010.

[21] J. Doliskani and É. Schost. Computing in degree 2^k-extensions of finite fields of odd characteristic. *Des. Codes Cryptogr.*, to appear.

[22] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 1999.

[23] J. von zur Gathen and V. Shoup. Computing Frobenius maps and factoring polynomials. *Comput. Complexity*, 2:187–224, 1992.

[24] G. Hanrot, M. Quercia, and P. Zimmermann. The middle product algorithm I. *Appl. Algebra Engrg. Comm. Comput.*, 14(6):415–438, 2004.

[25] William Hart. Fast library for number theory: an introduction. *Mathematical Software-ICMS 2010*, pages 88–91, 2010.

[26] K. S. Kedlaya and C. Umans. Fast polynomial factorization and modular composition. *SICOMP*, 40(6):1767–1802, 2011.

[27] E. Kunz. *Kähler differentials*. Friedr. Vieweg & Sohn, 1986.

[28] R. Lebreton, E. Mehrabi, and É. Schost. On the complexity of solving bivariate systems: The case of non-singular solutions. In *ISSAC'13*, pages 251–258. ACM, 2013.

[29] H. W. Lenstra Jr. Finding isomorphisms between finite fields. *Math. Comp.*, 56(193):329–347, 1991.

[30] H. W. Lenstra Jr. and B. De Smit. Standard models for finite fields: the definition, 2008.

[31] The PARI Group, Bordeaux. *PARI/GP, version* `2.7.0`, 2014.

[32] A. Poteaux and É. Schost. Modular composition modulo triangular sets and applications. *Comput. Complexity*, 22(3):463–516, 2013.

[33] A. Poteaux and É. Schost. On the complexity of computing with zero-dimensional triangular sets. *J. Symb. Comput.*, 50:110–138, 2013.

[34] F. Rouillier. Solving Zero-Dimensional systems through the Rational Univariate Representation. *Appl. Algebra Engrg. Comm. Comput.*, 9(5):433–461, 1999.

[35] V. Shoup. New algorithms for finding irreducible polynomials over finite fields. *Math. Comp.*, 54:435–447, 1990.

[36] V. Shoup. Fast construction of irreducible polynomials over finite fields. *J. Symb. Comput.*, 17(5):371–391, 1994.

[37] Victor Shoup. NTL: A library for doing number theory. `http://www.shoup.net/ntl`.

[38] Victor Shoup. Efficient computation of minimal polynomials in algebraic extensions of finite fields. In *ISSAC'99*, pages 53–58. ACM, 1999.

[39] William A. Stein and Others. *Sage Mathematics Software (Version 5.5)*. The Sage Development Team, 2013.

[40] V. Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *STOC'12*, pages 887–898. ACM, 2012.