Journal de Théorie des Nombres
de Bordeaux **16** (2004), 19–63

Topics in computational algebraic number theory

par Karim BELABAS

Résumé. Nous décrivons des algorithmes efficaces pour les opérations usuelles de la théorie algorithmique des corps de nombres, en vue d'applications à la théorie du corps de classes. En particulier, nous traitons l'arithmétique élémentaire, l'approximation et l'obtention d'uniformisantes, le problème du logarithme discret, et le calcul de corps de classes via un élément primitif. Tout ces algorithmes ont été implantés dans le système Pari/Gp.

Abstract. We describe practical algorithms from computational algebraic number theory, with applications to class field theory. These include basic arithmetic, approximation and uniformizers, discrete logarithms and computation of class fields. All algorithms have been implemented in the Pari/Gp system.

Contents

Manuscrit reçu le 20 décembre 2002.

1. Introduction and notations

Let K be a number field given by the minimal polynomial P of a primitive element, so that $K = \mathbb{Q}[X]/(P)$. Let $\mathcal{O}_K$ its ring of integers, $\mathfrak{f} = \mathfrak{f}_0\mathfrak{f}_\infty$ a modulus of K, where $\mathfrak{f}_0$ is an integral ideal and $\mathfrak{f}_\infty$ is a formal collection of real Archimedean places (we write $v \mid \mathfrak{f}_\infty$ for $v \in \mathfrak{f}_\infty$). Let $\mathrm{Cl}_{\mathfrak{f}}(K) = I_{\mathfrak{f}}(K)/P_{\mathfrak{f}}(K)$ denote the ray class group modulo $\mathfrak{f}$ of K; that is, the quotient group of non-zero fractional ideals coprime to $\mathfrak{f}_0$, by principal ideals (x) generated by $x \equiv 1 \bmod^* \mathfrak{f}$. The latter notation means that

- $v_{\mathfrak{p}}(x-1) \geqslant v_{\mathfrak{p}}(\mathfrak{f}_0)$ for all prime divisors $\mathfrak{p}$ of $\mathfrak{f}_0$.
- $\sigma(x) > 0$ for all $\sigma \mid \mathfrak{f}_\infty$.

The ordinary class group corresponds to $\mathfrak{f}_0 = \mathcal{O}_K$, $\mathfrak{f}_\infty = \emptyset$ and is denoted $\mathrm{Cl}(K)$.

Class field theory, in its classical form and modern computational incarnation[1], describes all finite abelian extensions of K in terms of the groups $\mathrm{Cl}_{\mathfrak{f}}(K)$. This description has a computational counterpart via Kummer theory, developed in particular by Cohen [10] and Fieker [17], relying heavily on efficient computation of the groups $\mathrm{Cl}_{\mathfrak{f}}(K)$ in the following sense:

Definition 1.1. a finite abelian group G is *known algorithmically* when its Smith Normal Form (SNF)

$$G = \bigoplus_{i=1}^{r} (\mathbb{Z}/d_i\mathbb{Z})\, g_i, \qquad \text{with } d_1 \mid \cdots \mid d_r \text{ in } \mathbb{Z}, \text{ and } g_i \in G,$$

is given, and we can solve the discrete logarithm problem in G. For $G = \mathrm{Cl}_{\mathfrak{f}}(K)$, this means writing any $\mathfrak{a} \in I_{\mathfrak{f}}(K)$ as $\mathfrak{a} = (\alpha)\prod_{i=1}^{r} g_i^{e_i}$, for some uniquely defined $(e_1, \ldots, e_r) \in \prod_{i=1}^{r}(\mathbb{Z}/d_i\mathbb{Z})$ and $(\alpha) \in P_{\mathfrak{f}}(K)$.

In this note, we give practical versions of most of the tools from computational algebraic number theory required to tackle these issues, with an emphasis on realistic problems *and* scalability. In particular, we point

[1] Other formulations in terms of class formations, idèle class groups and infinite Galois theory are not well suited to explicit computations, and are not treated here.

out possible precomputations, strive to prevent numerical instability and coefficient explosion, and to reduce memory usage. All our algorithms run in deterministic polynomial time and space, except 7.2, 7.7 (discrete log in $\mathrm{Cl}_{\mathfrak{f}}(K)$, which is at least as hard as the corresponding problem over finite fields) and 6.15 (randomized with expected polynomial running time). All of them are also efficient in practice, sometimes more so than well-known randomized variants. None of them is fundamentally new: many of these ideas have been used elsewhere, e.g. in the computer systems Kant/KASH [14] and PARI/GP [29]. But, to our knowledge, they do not appear in this form in the literature.

These techniques remove one bottleneck of computational class field theory, namely coefficient explosion. Two major difficulties remain. First, integer factorization, which is needed to compute the maximal order. This is in a sense a lesser concern, since fields of arithmetic significance often have smooth discriminants; or else their factorization may be known by construction. Namely, Buchmann and Lenstra [6] give an efficient algorithm to compute $\mathcal{O}_K$ given the factorization of its discriminant $\mathrm{disc}(K)$, in fact given its largest squarefree divisor. (The "obvious" algorithm requires the factorization of the discriminant of P.)

And second, the computation of $\mathrm{Cl}(K)$ and $\mathcal{O}_K^*$, for which one currently requires the truth of the Generalized Riemann Hypothesis (GRH) in order to obtain a practical randomized algorithm (see [9, §6.5]). The latter runs in expected subexponential time if K is imaginary quadratic (see Hafner-McCurley [22]); this holds for general K under further natural but unproven assumptions. Worse, should the GRH be wrong, no subexponential-time procedure is known, that would check the correctness of the result. Even then, this algorithm performs poorly on many families of number fields, and of course when $[K : \mathbb{Q}]$ is large, say 50 or more. This unfortunately occurs naturally, for instance when investigating class field towers, or higher class groups from algebraic K-theory [4].

The first three sections introduce some further notations and define fundamental concepts like Archimedean embeddings, the T_2 quadratic form and LLL reduction. Section §5 deals with mundane chores, implementing the basic arithmetic of K. Section §6 describes variations on the approximation theorem over K needed to implement efficient ideal arithmetic, in particular two-element representation for ideals, and a crucial ingredient in computations $\bmod^* \mathfrak{f}$. In Section §7, we introduce a representation of algebraic numbers as formal products, which are efficiently mapped to $(\mathcal{O}_K/\mathfrak{f})^*$ using the tools developed in the previous sections. We demonstrate our claims about coefficient explosion in the examples of this final section.

All timings given were obtained using the Pari library version 2.2.5 on a Pentium III (1GHz) architecture, running Linux-2.4.7; we allocate 10 MBytes RAM to the programs, unless mentioned otherwise.

Acknowledgements : It is hard to overestimate what we owe to Henri Cohen's books [9, 10], the state-of-the-art references on the subject. We shall constantly refer to them, supplying implementation details and algorithmic improvements as we go along. Neither would this paper exist without Igor Schein's insistence on computing "impossible" examples with the Pari/Gp system, and it is a pleasure to acknowledge his contribution. I also would like to thank Bill Allombert, Claus Fieker, Guillaume Hanrot and Jürgen Klüners for enlightening discussions and correspondences. Finally, it is a pleasure to thank an anonymous referee for a wealth of useful comments and the reference to [5].

2. Further notations and conventions

Let P a monic integral polynomial and $K = \mathbb{Q}[X]/(P) = \mathbb{Q}(\theta)$, where $\theta = X \pmod P$. We let $n = [K : \mathbb{Q}]$ the absolute degree, (r_1, r_2) the signature of K, and order the n embeddings of K in the usual way: σ_k is real for $1 \leqslant k \leqslant r_1$, and $\sigma_{k+r_2} = \overline{\sigma_k}$ for $r_1 < k \leqslant r_1 + r_2$.

Definition 2.1. The $\mathbb{R}$-algebra $E := K \otimes_{\mathbb{Q}} \mathbb{R}$, which is isomorphic to $\mathbb{R}^{r_1} \times \mathbb{C}^{r_2}$, has an involution $x \mapsto \overline{x}$ induced by complex conjugation. It is a Euclidean space when endowed with the positive definite quadratic form $T_2(x) := \mathrm{Tr}_{E/\mathbb{R}}(x\overline{x})$, with associated norm $\|x\| := \sqrt{T_2(x)}$. We say that $x \in K$ is small when $\|x\|$ is so.

If $x \in K$, we have explicitly

$$T_2(x) = \sum_{k=1}^{n} |\sigma_k(x)|^2 .$$

We write $d(\Lambda, q)$ for the determinant of a lattice (Λ, q); in particular, we have

$$d(\mathcal{O}_K, T_2)^2 = |\mathrm{disc}\, K| . \tag{1}$$

Given our class-field theoretic goals, knowing the maximal order $\mathcal{O}_K$ is a prerequisite, and will enable us not to worry about denominators[2]. In our present state of knowledge, obtaining the maximal order amounts to finding a full factorization of $\mathrm{disc}\, K$, hence writing $\mathrm{disc}\, P = f^2 \prod p_i^{e_i}$, for some integer f coprime to $\mathrm{disc}\, K$, and prime numbers p_i. In this situation, see [6, 20] for how to compute a basis. We shall fix a $\mathbb{Z}$-basis $(w_1, \ldots, w_n)$

[2]Low-level arithmetic in K could be handled using any order instead of $\mathcal{O}_K$, for instance if we only wanted to factor polynomials over K (see [3]). Computing $\mathcal{O}_K$ may be costly: as mentioned in the introduction, it requires finding the largest squarefree divisor of $\mathrm{disc}\, K$.

of the maximal order $\mathcal{O}_K$. Then we may identify K with $\mathbb{Q}^n$: an element $\sum_{i=1}^n x_i w_i$ in K is represented as the column vector $\mathbf{x} := (x_1, \ldots, x_n)$. In fact, we store and use such a vector as a pair $(d\mathbf{x}, d)$ where $d \in \mathbb{Z}_{>0}$ and $d\mathbf{x} \in \mathbb{Z}^n$. The minimal such d does not depend on the chosen basis (w_i), but is more costly to obtain, so we do not insist that the exact denominator be used, i.e. $d\mathbf{x}$ is not assumed to be primitive. For x in K, M_x denotes the n by n matrix giving multiplication by x with respect to the basis (w_i). For reasons of efficiency, we shall impose that

- $w_1 = 1$ (see §4.3),
- (w_i) is LLL-reduced for T_2, for some LLL parameter $1/4 < c < 1$ (see §4).

Our choice of coordinates over the representatives arising from $K = \mathbb{Q}[X]/(P)$ is justified in §5.1.

The letter p denotes a rational prime number, and $\mathfrak{p}/p$ is a prime ideal of $\mathcal{O}_K$ above p. We write $N\alpha$ and $\operatorname{Tr}\alpha$ respectively for the absolute norm and trace of $\alpha \in K$. Finally, for $x \in \mathbb{R}$, $\lceil x \rfloor := \lfloor x + 1/2 \rfloor$ is the integer nearest to x; we extend this operator coordinatewise to vectors and matrices.

3. Archimedean embeddings

Definition 3.1. Let $\sigma : K \to \mathbb{R}^{r_1} \times \mathbb{C}^{r_2}$ be the embeddings vector defined by

$$\sigma(x) := (\sigma_1(x), \ldots, \sigma_{r_1+r_2}(x)),$$

which fixes an isomorphism between $E = K \otimes_{\mathbb{Q}} \mathbb{R}$ and $\mathbb{R}^{r_1} \times \mathbb{C}^{r_2}$.

We also map E to $\mathbb{R}^n$ via one of the following $\mathbb{R}$-linear maps from $\mathbb{R}^{r_1} \times \mathbb{C}^{r_2}$ to $\mathbb{R}^{r_1} \times \mathbb{R}^{r_2} \times \mathbb{R}^{r_2} = \mathbb{R}^n$:

$$\begin{aligned} \phi : (\mathbf{x}, \mathbf{y}) &\mapsto (\mathbf{x}, \operatorname{Re}(\mathbf{y}), \operatorname{Im}(\mathbf{y})), \\ \psi : (\mathbf{x}, \mathbf{y}) &\mapsto (\mathbf{x}, \operatorname{Re}(\mathbf{y}) + \operatorname{Im}(\mathbf{y}), \operatorname{Re}(\mathbf{y}) - \operatorname{Im}(\mathbf{y})). \end{aligned}$$

The map ψ identifies the Euclidean spaces (E, T_2) and $(\mathbb{R}^n, \| \|_2^2)$, and is used in §4.2 to compute the LLL-reduced basis (w_i). The map ϕ is slightly less expensive to compute and is used in §3.2 to recognize algebraic integers from their embeddings (ψ could be used instead).

We extend ϕ and ψ : $\operatorname{Hom}(\mathbb{R}^n, \mathbb{R}^{r_1} \times \mathbb{C}^{r_2}) \to \operatorname{End}(\mathbb{R}^n)$ by composition, as well as to the associated matrix spaces.

3.1. Computation. Let $\sigma_i : K \to \mathbb{C}$ be one of the n embeddings of K and $\alpha \in K = \mathbb{Q}[X]/(P)$. Then $\sigma_i(\alpha)$ can be approximated by evaluating a polynomial representative of α at (floating point approximations of) the corresponding complex root of P, computed via a root-finding algorithm with guaranteed error terms, such as Gourdon-Schönhage [21], or Uspensky [30] for the real embeddings.

Assume that floating point approximations $(\hat{\sigma}(w_i))_{1\leqslant i\leqslant n}$ of the $(\sigma(w_i))_{1\leqslant i\leqslant n}$ have been computed in this way to high accuracy. (If higher accuracy is later required, refine the roots and cache the new values.) From this point on, the embeddings of an arbitrary $\alpha \in K$ are computed as integral linear combinations of the $(\hat{\sigma}(w_i))$, possibly divided by the denominator of α. In most applications (signatures, Shanks-Buchmann's distance), we can take $\alpha \in \mathcal{O}_K$ so no denominators arise. We shall note $\hat{\sigma}(\alpha)$ and $\hat{\sigma}_i(\alpha)$ the floating point approximations obtained in this way.

The second approach using precomputed embeddings is usually superior to the initial one using the polynomial representation, since the latter may involve unnecessary large denominators. A more subtle, and more important, reason is that the defining polynomial P might be badly skewed, with one large root for instance, whereas the LLL-reduced $\hat{\sigma}(w_i)$ (see §4.2) have comparable L^2 norm. Thus computations involving the $\hat{\sigma}(w_i)$ are more stable than evaluation at the roots of P. Finally, in the applications, $\|\alpha\|$ is usually small, hence α often has small coordinates. In general, coefficients in the polynomial representation are larger, making the latter computation slower and less stable.

In the absence of denominators, both approaches require n multiplications of floating point numbers by integers for a single embedding (and n floating point additions). Polynomial evaluation may be sped up by multipoint evaluation if multiple embeddings are needed and is asymptotically faster, since accuracy problems and larger bitsizes induce by denominators can be dealt with by increasing mantissa lengths by a bounded amount depending only on P.

3.2. Recognition of algebraic integers. Let $\alpha \in \mathcal{O}_K$, known through floating point approximations $\hat{\sigma}(\alpha)$ of its embeddings $\sigma(\alpha)$; we want to recover α. This situation occurs for instance when computing fundamental units [9, Algorithm 6.5.8], or in the discrete log problem for $\mathrm{Cl}(K)$, cf. Algorithm 7.2. In some situations, we are only interested in the characteristic polynomial χ_α of α, such as when using Fincke-Pohst enumeration [19] to find minimal primitive elements of K (α being primitive if and only if χ_α is squarefree). The case of absolute norms (the constant term of χ_α) is of particular importance and is treated in §5.2.

Let $Y = \sigma(a)$ and W the matrix whose columns are the $(\sigma(w_j))_{1\leqslant j\leqslant n}$; $\hat{Y}$ and $\hat{W}$ denote known floating point approximations of Y and W respectively. Provided $\hat{Y}$ is accurate enough, one recovers

$$\chi_\alpha = \prod_{i=1}^{n} \big(X - \sigma_i(\alpha)\big),$$

by computing an approximate characteristic polynomial

$$\widehat{\chi}_\alpha = \prod_{i=1}^{n} \big(X - \hat{\sigma}_i(\alpha)\big),$$

then rounding its coefficients to the nearest integers. This computation keeps to $\mathbb{R}$ by first pairing complex conjugate roots (followed by a divide and conquer product in $\mathbb{R}[X]$). We can do better and recover α itself: if $\alpha = \sum_{i=1}^{n} \alpha_i w_i$ is represented by the column vector $A = (\alpha_i) \in \mathbb{Z}^n$, we recover A from $WA = Y$ as $A = \lceil \phi(\hat{W})^{-1}\phi(\hat{Y})\rfloor$. Of course, it is crucial to have reliable error bounds in the above to guarantee proper rounding.

Remark 3.2. Using ϕ, we keep computations to $\mathbb{R}$ and disregard redundant information from conjugates, contrary to [9, Chapter 6], which inverts $\Omega := (\sigma_i(w_j))_{1\leqslant i,j\leqslant n}$ in $M_n(\mathbb{C})$. We could just as well use ψ instead, or more generally compose ϕ with any automorphism of $\mathbb{R}^n$. Using ψ would have the slight theoretical advantage that the columns of $\psi(\hat{W})$ are LLL-reduced for the L^2 norm (see §4.2).

Remark 3.3. The matrix inversion $\phi(\hat{W})^{-1}$ is performed only once, until the accuracy of $\hat{W}$ needs to be increased. The coordinates of α are then recovered by a mere matrix multiplication, the accuracy of which is determined by a priori estimates, using the known $\phi(\hat{W})^{-1}$ and $\phi(\hat{Y})$, or a preliminary low precision multiplication with proper attention paid to rounding so as to guarantee the upper bound. Since $\|\phi(Y)\|_2 \leqslant \|\psi(Y)\|_2 = \|\alpha\|$, the smaller $\|\alpha\|$, the better a priori estimates we get, and the easier it is to recognize α.

Remark 3.4. The coefficients of χ_α are bounded by $C\,\|\hat{Y}\|_\infty^n$, for some $C > 0$ depending only on K and (w_i), whereas the vector of coordinates A is bounded linearly in terms of $\hat{Y}$. So it may occur that $\hat{Y}$ is accurate enough to compute A, but not χ_α. In which case, one may use A for an algebraic resultant computation or to recompute $\hat{\sigma}(\alpha)$ to higher accuracy.

Remark 3.5. In many applications, it is advantageous to use non-Archimedean embeddings $K \to K \otimes_{\mathbb{Q}} \mathbb{Q}_p = \oplus_{\mathfrak{p}|p} K_{\mathfrak{p}}$ which is isomorphic to $\mathbb{Q}_p^n$ as a $\mathbb{Q}_p$-vector space. This cancels rounding errors, as well as stability problems in the absence of divisions by p. In some applications (e.g., automorphisms [1], factorization of polynomials [3, 18]), a single embedding $K \to K_{\mathfrak{p}}$ is enough, provided an upper bound for $\|\alpha\|$ is available.

4. T_2 and LLL reduction

We refer to [26, 9] for the definition and properties of LLL-reduced bases, and the LLL reduction algorithm, simply called *reduced bases* and *reduction* in the sequel. In particular, reduction depends on a parameter $c \in]1/4, 1[$, which is used to check the Lovász condition and determines the frequency

of swaps in the LLL algorithm. A larger c means better guarantees for the output basis, but higher running time bounds. We call c the *LLL parameter* and $\alpha := 1/(c-1/4) > 4/3$ the *LLL constant*.

Proposition 4.1 ([9, Theorem 2.6.2])**.** *Let $(w_i)_{1\leqslant i\leqslant n}$ a reduced basis of a lattice (Λ, q) of rank n, for the LLL constant α. Let $(w_i^*)_{1\leqslant i\leqslant n}$ the associated orthogonalized Gram-Schmidt basis, and linearly independent vectors $(b_i)_{1\leqslant i\leqslant n}$ in Λ. For $2 \leqslant i \leqslant n$, we have $q(w_{i-1}^*) \leqslant \alpha q(w_i^*)$; for $1 \leqslant i \leqslant n$, we have*

$$q(w_i) \leqslant \alpha^{i-1} q(w_i^*), \quad \text{and} \quad q(w_i) \leqslant \alpha^{n-1} \max_{1\leqslant j\leqslant i} q(b_j).$$

4.1. T_2 and $\|\ \|$. It is algorithmically useful to fix a basis (w_i) which is small with respect to T_2. This ensures that an element with small coordinates with respect to (w_i) is small, and in particular has small absolute norm. More precisely, we have $|Nx|^{2/n} \leqslant T_2(x)/n$ by the arithmetic-geometric mean inequality and

$$n\,|Nx|^{2/n} \leqslant T_2\Big(\sum_{i=1}^n x_i w_i\Big) \leqslant \Big(\sum_{i=1}^n x_i^2\Big)\Big(\sum_{i=1}^n T_2(w_i)\Big). \tag{2}$$

If (w_i) is reduced, Proposition 4.1 ensures that picking another basis may improve the term $\sum_{i=1}^n T_2(w_i)$ at most by a factor $n\alpha^{n-1}$.

4.2. Integral versus floating point reduction. We first need to compute a reduced basis $(w_i)_{1\leqslant i\leqslant n}$ for $\mathcal{O}_K$, starting from an arbitrary basis $(b_i)_{1\leqslant i\leqslant n}$. When K is totally real, T_2 is the natural trace pairing, whose Gram matrix is integral and given by $(\mathrm{Tr}(b_i b_j))_{1\leqslant i,j\leqslant n}$; so we can use de Weger's integral reduction ([9, §2.6.3]). If K is not totally real, we have to reduce floating point approximations of the embeddings. In fact we reduce $(\psi\circ\hat{\sigma}(b_i))_{1\leqslant i\leqslant n}$ (see §3), which is a little faster and a lot stabler than using the Gram matrix in this case, since Gram-Schmidt orthogonalization can be replaced by Householder reflections or Givens rotations.

The LLL algorithm is better behaved and easier to control with exact inputs, so we now explain how to use an integral algorithm[3] to speed up all further reductions with respect to T_2. Let ${}^tRR$ be the Cholesky decomposition of the Gram matrix of $(T_2, (w_i))$. In other words,

$$R = \mathrm{diag}(\|w_1^*\|, \dots, \|w_n^*\|) \times (\mu_{i,j})_{1\leqslant i,j\leqslant n}$$

is upper triangular, where (w_i^*) is the orthogonalized basis associated to (w_i) and the $\mu_{i,j}$ are the Gram-Schmidt coefficients, both of which are by-products of the reduction yielding (w_i). Let

$$r := \min_{1\leqslant i\leqslant n} \|w_i^*\|,$$

[3]This does not prevent the implementation from using floating point numbers for efficiency. But the stability and complexity of LLL are better understood for exact inputs (see [26, 25, 31]).

which is the smallest diagonal entry of R. For $e \in \mathbb{Z}$ such that $2^e r > 1/2$, let $R^{(e)} := \lceil 2^e R \rfloor$. The condition on e ensures that $R^{(e)}$ has maximal rank. If $x = \sum_{i=1}^n x_i w_i \in K$ is represented by the column vector $X = (x_i)_{1 \leqslant i \leqslant n} \in \mathbb{Q}^n$, we have $\|x\| = \|RX\|_2$. Then $T_2^{(e)}(X) := \|R^{(e)}X\|_2^2$ is a convenient integral approximation to $2^{2e}T_2(X)$, which we substitute for T_2 whenever LLL reduction is called for. This is also applicable to the twisted variants of T_2 introduced in [9, Chapter 6] to randomize the search for smooth ideals in subexponential class group algorithms.

In general, this method produces a basis which is not reduced with respect to T_2, but it should be a "nice" basis. In most applications (class group algorithms, pseudo-reduction), we are only interested in the fact that the first basis vector is not too large:

Proposition 4.2. *Let Λ be a sublattice of $\mathcal{O}_K$ and let (a_i) (resp. (b_i)) a reduced basis for Λ with respect to $T_2^{(e)}$ (resp. T_2), with LLL constant α. The LLL bound states that*

$$\|b_1\| \leqslant B_{LLL} := \alpha^{(n-1)/2} d(\Lambda, T_2)^{1/n}.$$

Let $\|M\|_2 := (\sum_{i,j=1}^n |m_{i,j}|^2)^{1/2}$ for $M = (m_{i,j}) \in M_n(\mathbb{R})$. Let $S := (R^{(e)})^{-1}$, then

$$\|a_1\| / B_{LLL} \leqslant \left(\frac{\det R^{(e)}}{2^{ne} \det R}\right)^{1/n} \left(1 + \frac{\sqrt{n(n+1)}}{2\sqrt{2}} \|S\|_2\right). \tag{3}$$

Proof. Let $X \in \mathbb{Z}^n$ be the vector of coordinates of a_1 on (w_i) and $Y := R^{(e)}X$. Since

$$d(\Lambda, T_2^{(e)}) = [\mathcal{O}_K : \Lambda] \det R^{(e)} \quad \text{and} \quad d(\Lambda, T_2) = [\mathcal{O}_K : \Lambda] \det R,$$

the LLL bound applied to the $T_2^{(e)}$-reduced basis yields

$$\sqrt{T_2^{(e)}(X)} = \|Y\|_2 \leqslant \alpha^{(n-1)/2} d(\Lambda, T_2^{(e)})^{1/n} = 2^e B_{LLL} \left(\frac{\det R^{(e)}}{2^{ne} \det R}\right)^{1/n}.$$

We write $R^{(e)} = 2^e R + \varepsilon$, where $\varepsilon \in M_n(\mathbb{R})$ is upper triangular such that $\|\varepsilon\|_\infty \leqslant 1/2$, hence $\|\varepsilon\|_2 \leqslant \frac{1}{2}\sqrt{\frac{n(n+1)}{2}}$, and obtain $2^e RX = Y - \varepsilon SY$. Taking L^2 norms, we obtain

$$2^e \|a_1\| \leqslant (1 + \|\varepsilon S\|_2) \|Y\|_2,$$

and we bound $\|\varepsilon S\|_2 \leqslant \|\varepsilon\|_2 \|S\|_2$ by Cauchy-Schwarz. □

Corollary 4.3. *If $2^e r \geqslant 1$, then*

$$\|a_1\| / B_{LLL} \leqslant 1 + \frac{O_\alpha(1)^n}{2^e}.$$

Proof. For all $1 \leqslant i \leqslant n$, we have $\|w_i^*\| \geqslant \alpha^{(1-i)/2}\,\|w_i\|$ by the properties of reduced bases. Since $\|w_i\| \geqslant \sqrt{n}$ (with equality iff w_i is a root of unity), we obtain

$$r = \min_{1\leqslant i\leqslant n} \|w_i^*\| \geqslant \sqrt{n}\alpha^{(1-n)/2},$$

and $1/r = O_\alpha(1)^n$. Since R and $R^{(e)}$ are upper triangular one gets

$$\det R^{(e)} = \prod_{i=1}^{n} \lceil 2^e\,\|w_i^*\| \rfloor \leqslant \prod_{i=1}^{n} (2^e\,\|w_i^*\| + 1/2) \leqslant 2^{ne} \det R \left(1 + \frac{1}{2^{e+1}r}\right)^n.$$

Rewrite $R = D + N$ and $R^{(e)} = D^{(e)} + N^{(e)}$, where D, $D^{(e)}$ are diagonal and N, $N^{(e)}$ triangular nilpotent matrices. A non-zero entry n/d of ND^{-1}, where $d > 0$ is one of the diagonal coefficients of D, is an off-diagonal Gram-Schmidt coefficient of the size-reduced basis $(w_i)_{1\leqslant i\leqslant n}$, hence $|n/d| \leqslant 1/2$. Since $|n| \leqslant d/2$ and $1 \leqslant 2^e r \leqslant 2^e d$, the corresponding entry of $Z := N^{(e)}(D^{(e)})^{-1}$ satisfies

$$\frac{|\lceil 2^e n \rfloor|}{\lceil 2^e d \rfloor} \leqslant \frac{2^e\,|n| + 1/2}{2^e d - 1/2} \leqslant \frac{2^{e-1}d + 2^{e-1}d}{2^{e-1}d} = 2.$$

It follows that the coefficients of $(\mathrm{Id}_n + Z)^{-1} = \sum_{i=0}^{n-1} (-1)^{i-1} Z^i$ are $O(1)^n$. By analogous computations, coefficients of $(D^{(e)})^{-1}$ are $O(1/(r2^e))$. Since $R = D^{(e)}(\mathrm{Id}_n + Z)$, its inverse S is the product of the above two matrices, and we bound its norm by Cauchy-Schwarz: $\|S\|_2 = \frac{1}{2^e} \times O_\alpha(1)^n$. □

Qualitatively, this expresses the obvious fact that enough significant bits eventually give us a reduced basis. The point is that we get a bound for the quality of the reduction, at least with respect to the smallest vector, which is independent of the lattice being considered. In practice, we evaluate (3) exactly during the precomputations, increasing e as long as it is deemed unsatisfactory. When using $T_2^{(e)}$ as suggested above, we can always reduce the new basis with respect to T_2 later if maximal reduction is desired, expecting faster reduction and better stability due to the preprocessing step.

4.3. Hermite Normal Form (HNF) and setting $w_1 = 1$. We refer to [9, §2.4] for definitions and algorithms related to the HNF representation. For us, matrices in HNF are upper triangular, and "HNF of A modulo $z \in \mathbb{Z}$" means the HNF reduction of $(A \mid z\,\mathrm{Id}_n)$, *not* modulo a multiple of $\det(A)$ as in [9, Algorithm 2.4.8]. The algorithm is almost identical: simply remove the instruction $R \leftarrow R/d$ in Step 4.

In the basis (w_i), it is useful to impose that $w_1 = 1$, in particular to compute intersection of submodules of $\mathcal{O}_K$ with $\mathbb{Z}$, or as a prerequisite to the extended Euclidean Algorithm 5.4. One possibility is to start from the canonical basis (b_i) for $\mathcal{O}_K$ which is given in HNF with respect to the power

basis $(1, \theta, \ldots, \theta^{n-1})$; we have $b_1 = 1$. Then reduce (b_i) using a modified LLL routine which prevents size-reduction on the vector corresponding initially to b_1. Finally, put it back to the first position at the end of the LLL algorithm. This does not affect the quality of the basis, since

$$\|1\| = \sqrt{n} = \min_{x \in \mathcal{O}_K \setminus \{0\}} \|x\| .$$

Unfortunately, this basis is not necessarily reduced. Another approach is as follows:

Proposition 4.4. *Let (w_i) a basis of a lattice (Λ, q) such that w_1 is a shortest non-zero vector of Λ. Then performing LLL reduction on (w_i) leaves w_1 invariant provided the parameter c satisfies $1/4 < c \leqslant 1/2$.*

Proof. Let $\| \ \|$ be the norm associated to q. It is enough to prove that w_1 is never swapped with its size-reduced successor, say s. Let $w_1^* = w_1$ and s^* be the corresponding orthogonalized vectors. A swap occurs if $\|s^*\| < \|w_1\| \sqrt{c - \mu^2}$, where the Gram-Schmidt coefficient $\mu = \mu_{2,1}$ satisfies $|\mu| \leqslant 1/2$ (by definition of size-reduction) and $s^* = s - \mu w_1$. From the latter, we obtain

$$\|s^*\| = \|s\| - \|\mu w_1\| \geqslant \|w_1\| (1 - |\mu|)$$

since s is a non-zero vector of Λ. We get a contradiction if $(1-|\mu|)^2 \geqslant c-\mu^2$, which translates to $(2|\mu| - 1)^2 + (1 - 2c) \geqslant 0$. □

5. Working in K

5.1. Multiplication in $\mathcal{O}_K$. In this section and the next, we let $\mathsf{M}(B)$ be an upper bound for the time needed to multiply two B-bits integers and we assume $\mathsf{M}(B + o(B)) = \mathsf{M}(B)(1 + o(1))$. See [24, 32] for details about integer and polynomial arithmetic. In the rough estimates below we only take into account multiplication time. We deal with elements of $\mathcal{O}_K$, leaving to the reader the generalization to arbitrary elements represented as (equivalence classes of) pairs $(x, d) = x/d$, $x \in \mathcal{O}_K$, $d \in \mathbb{Z}_{>0}$.

5.1.1. *Polynomial representation.* The field K was defined as $\mathbb{Q}(\theta)$, for some $\theta \in \mathcal{O}_K$. In this representation, integral elements may have denominators, the largest possible denominator D being the exponent of the additive group $\mathcal{O}_K/\mathbb{Z}[\theta]$. To avoid rational arithmetic, we handle content and principal part separately.

Assume for the moment that $D = 1$. Then, $x, y \in \mathcal{O}_K$ are represented by integral polynomials. If $x, y, P \in \mathbb{Z}[X]$ have B-bits coefficients, then we compute xy in time $2n^2\mathsf{M}(B)$; and even $n^2\mathsf{M}(B)(1 + o(1))$ if $\log_2 \|P\|_\infty = o(B)$, so that Euclidean division by P is negligible. Divide

and conquer polynomial arithmetic reduce this to $O(n^{\log_2 3}\mathsf{M}(B))$. Assuming FFT-based integer multiplication, segmentation[4] further improves the theoretical estimates to $O(\mathsf{M}(2Bn + n\log n))$.

In general, one replaces B by $B + \log_2 D$ in the above estimates. In particular, they still hold provided $\log_2 D = o(B)$. Recall that D depends only on P, not on the multiplication operands.

5.1.2. *Multiplication table.* For $1 \leqslant i, j, k \leqslant n$, let $m_k^{(i,j)} \in \mathbb{Z}$ such that

$$w_i w_j = \sum_{k=1}^{n} m_k^{(i,j)} w_k, \tag{4}$$

giving the multiplication in K with respect to the basis (w_i). We call $M := (m_k^{(i,j)})_{i,j,k}$ the multiplication table over $\mathcal{O}_K$. This table is computed using the polynomial representation for elements in $K = \mathbb{Q}[X]/(P)$, or by multiplying Archimedean embeddings and recognizing the result (§3.1 and §3.2), which is much faster. Of course $m_k^{(i,j)} = m_k^{(j,i)}$, and $m_k^{(i,1)} = \delta_{i,k}$ since $w_1 = 1$, so only $n(n-1)/2$ products need be computed in any case. The matrix M has small integer entries, often single precision if (w_i) is reduced. In general, we have the following pessimistic bound:

Proposition 5.1. *If $(w_i)_{1\leqslant i\leqslant n}$ is reduced with respect to T_2 with LLL constant α, then*

$$T_2(w_i) \leqslant C_i = C_i(K,\alpha) := \left(n^{-(i-1)}\alpha^{n(n-1)/2}\left|\operatorname{disc} K\right|\right)^{1/(n-i+1)}.$$

Furthermore, for all $1 \leqslant i, j \leqslant n$ and $1 \leqslant k \leqslant n$, we have

$$\left|m_k^{(i,j)}\right| \leqslant \frac{\alpha^{n(n-1)/4}}{\sqrt{n}}\frac{C_i + C_j}{2} \leqslant \frac{\alpha^{3n(n-1)/4}}{n^{n-(1/2)}}\left|\operatorname{disc} K\right|.$$

Proof. The estimate C_i comes, on the one hand, from

$$\|w_i^*\|^{n-i}\prod_{k=1}^{i}\|w_k^*\| \geqslant (\alpha^{-(i-1)/2}\|w_i\|)^{n-i}\prod_{k=1}^{i}\alpha^{-(k-1)/2}\|w_k\|,$$

and on the other hand, from

$$\|w_i^*\|^{n-i}\prod_{k=1}^{i}\|w_k^*\| \leqslant \prod_{k=1}^{n-i}\alpha^{k/2}\prod_{k=1}^{n}\|w_k^*\|.$$

[4]Also known as "Kronecker's trick", namely evaluation of x, y at a large power R^k of the integer radix, integer multiplication, then reinterpretation of the result as $z(R^k)$, for some unique $z \in \mathbb{Z}[X]$.

Since $\|w_k\| \geqslant \sqrt{n}$ for $1 \leqslant k \leqslant n$, this yields

$$\begin{aligned} n^{(i-1)/2} \|w_i\|^{n-i+1} &\leqslant \prod_{k=1}^{i} \alpha^{(k-1)/2} \prod_{k=1}^{n-i} \alpha^{(k+i-1)/2} \times \prod_{k=1}^{n} \|w_k^*\| \\ &= \alpha^{n(n-1)/4} d(\mathcal{O}_K, T_2). \end{aligned}$$

Now, fix $1 \leqslant i, j \leqslant n$ and let $m_k := m_k^{(i,j)}$. For all $1 \leqslant l \leqslant n$, we write

$$\sum_{k=1}^{n} m_k \sigma_l(w_k) = \sigma_l(w_i w_j),$$

and solve the associated linear system $WX = Y$ in unknowns $X = (m_1, \ldots, m_n)$. Using Hadamard's lemma, the cofactor of the entry of index (l, k) of W is bounded by

$$\prod_{k=1, k\neq l}^{n} \|w_k\| \leqslant \frac{1}{\sqrt{n}} \alpha^{n(n-1)/4} \left|\det W\right|,$$

by the properties of reduced bases and the lower bound $\|w_l\| \geqslant \sqrt{n}$. Hence,

$$\max_{1\leqslant k\leqslant n} |m_k| \leqslant \frac{1}{\sqrt{n}} \alpha^{n(n-1)/4} \sum_{l=1}^{n} \left|\sigma_l(w_i w_j)\right|.$$

Using LLL estimates and (1), we obtain

$$\begin{aligned} \sum_{l=1}^{n} |\sigma_l(w_i w_j)| &\leqslant \frac{1}{2}(T_2(w_i) + T_2(w_j)) \\ &\leqslant \max_{1\leqslant k\leqslant n} T_2(w_k) \\ &\leqslant \frac{\prod_{k=1}^{n} T_2(w_k)}{(\min_{1\leqslant k\leqslant n} T_2(w_k))^{n-1}} \leqslant \frac{1}{n^{n-1}} \alpha^{n(n-1)/2} \left|\operatorname{disc} K\right|. \end{aligned}$$

A direct computation bounds C_i by the same quantity for $1 \leqslant i \leqslant n$: it reduces to $n \leqslant C_1$ which follows from the first part. □

For $x, y \in \mathcal{O}_K$, we use M to compute $xy = \sum_{k=1}^{n} z_k w_k$, where

$$z_k := \sum_{j=1}^{n} y_j \sum_{i=1}^{n} x_i m_k^{(i,j)},$$

in $n^3 + n^2$ multiplications as written. This can be slightly improved by taking into account that $w_1 = 1$; also, as usual, a rough factor 2 is gained for squarings.

Assuming the x_i, y_j, and $m_k^{(i,j)} x_i$ are B-bits integers, the multiplication table is an $n^3 \mathsf{M}(B)(1+o(1))$ algorithm. This goes down to $n^2 \mathsf{M}(B)(1+o(1))$

if $\log_2 \|M\|_\infty = o(B)$, since in this case the innermost sums have a negligible computational cost.

5.1.3. *Regular representation.* Recall that M_x is the matrix giving the multiplication by $x \in K$ with respect to (w_i). Since $w_1 = 1$, we recover x as the first column of M_x; also, $x \in \mathcal{O}_K$ if and only if M_x has integral entries. M_x is computed using the multiplication table M as above, then xy is computed as $M_x y$ in n^2 integer multiplications, for an arbitrary $y \in \mathcal{O}_K$. It is equivalent to precompute M_x then to obtain xy as $M_x y$, and to compute directly xy using M. (Strictly speaking, the former is slightly more expensive due to different flow control instructions and memory management.) So M_x comes for free when the need to compute xy arises and neither M_x nor M_y is cached.

Let x, y have B-bit coordinates. Provided $\log_2 \|M\|_\infty + \log_2 n = o(B)$, M_x has $B+o(B)$-bit entries, and the multiplication cost is $n^2\mathsf{M}(B)(1+o(1))$.

5.1.4. *What and where do we multiply?* In computational class field theory, a huge number of arithmetic operations over K are performed, so it is natural to allow expensive precomputations. We want a multiplication method adapted to the following setup:

- The maximal order $\mathcal{O}_K = \oplus_{i=1}^n \mathbb{Z} w_i$ is known.
- The basis (w_i) is reduced for T_2.
- We expect to mostly multiply small algebraic integers $x \in \mathcal{O}_K$, hence having small coordinates in the (w_i) basis.

This implies that algebraic integers in polynomial representation have in general larger bit complexity, due to the larger bit size of their components, and the presence of denominators. This would not be the case had we worked in other natural orders, like $\mathbb{Z}[X]/(P)$, or with unadapted bases, like the HNF representation over the power basis. In practice, $\mathcal{O}_K$ is easy to compute whenever $\mathrm{disc}(K)$ is smooth, which we will enforce in our experimental study. Note that fields of arithmetic significance, e.g., built from realistic ramification properties, usually satisfy this.

For a fair comparison, we assume that P ran through a polynomial reduction algorithm, such as [11]. This improves the polynomial representation $\mathbb{Q}[X]/(P)$ at a negligible initialization cost, given (w_i) as above (computing the minimal polynomials of a few small linear combinations of the w_i). Namely, a polynomial P of small height, means faster Euclidean division by P (alternatively, faster multiplication by a precomputed inverse).

5.1.5. *Experimental study.* We estimated the relative speed of the various multiplication methods in the Pari library, determined experimentally over

random integral elements

$$x = \sum_{i=1}^{n} x_i w_i, \quad y = \sum_{i=1}^{n} y_i w_i,$$

satisfying $|x_i|, |y_i| < 2^B$, in random number fields[5] K of degree n and smooth discriminant, for increasing values of n and B. Choosing elements with small coordinates, then converting to polynomial representation, e.g., instead of the other way round, introduces a bias in our test, but we contend that elements we want to multiply arise in this very way. Also, this section aims at giving a concrete idea of typical behaviour in a realistic situation; it is not a serious statistical study.

For each degree n, we generate 4 random fields $K = \mathbb{Q}[X]/(P)$; all numerical values given below are averaged over these 4 fields. Let D the denominator of $\mathcal{O}_K$ on the power basis, and M the multiplication table as above; we obtain:

$[K:\mathbb{Q}]$	$\log_2 \lvert\mathrm{disc}\, K\rvert$	$\log_2 D$	$\log_2 \lVert P\rVert_\infty$	$\log_2 \lVert M\rVert_\infty$
2	5.3	0	3.3	3.3
5	27.	2.2	5.5	4.4
10	73.8	0.50	4.7	5.4
20	192.	0.50	3.1	6.1
30	319.	533.6	40.	7.7
50	578.2	1459.	55.	7.9

So M has indeed very small entries, and we see that D gets quite large when we do not choose arbitrary random P (building the fields as compositum of fields of small discriminant, we restrict their ramification). Notice that M is relatively unaffected. Consider the following operations:

- A: compute xy as $M_x y$, assuming M_x is precomputed.
- TAB: compute xy using directly M.
- POL: compute xy from polynomial representations, omitting conversion time.
- PC: convert x from polynomial to coordinate representation.
- CP: convert x from coordinate to polynomial representation.

[5] When $n \leqslant 20$, the fields $K = \mathbb{Q}[X]/(P)$ are defined by random monic $P \in \mathbb{Z}[X]$, $\lVert P\rVert_\infty \leqslant 10$, constructed by picking small coefficients until P turns out to be irreducible. In addition we impose that $\mathrm{disc}(P)$ is relatively smooth: it can be written as $D_1 D_2$ with $p \mid D_1 \Rightarrow p < 5.10^5$ and $|D_2| < 10^{60}$, yielding an easy factorization of $\mathrm{disc}(P)$. For $n > 20$, we built the fields as compositum of random fields of smaller degree, which tends to produce large indices $[\mathcal{O}_K : \mathbb{Z}[X]/(P)]$ (small ramification, large degree). In all cases, we apply a reduction algorithm [11] to defining polynomials in order to minimize $T_2(\theta)$. This was allowed to increase $\lVert P\rVert_\infty$.

For each computation $X \in \{\text{TAB}, \text{POL}, \text{PC}, \text{CP}\}$, we give the relative time t_X/t_A:

	$B = 10$		$B = 100$		$B = 1000$		$B = 10000$	
n	TAB	POL	TAB	POL	TAB	POL	TAB	POL
2	1.0	2.7	1.0	2.4	1.1	1.2	1.1	1.0
5	2.7	2.2	2.3	1.9	1.3	1.2	1.2	1.0
10	4.8	1.9	3.7	1.6	1.4	0.86	1.2	0.79
20	8.9	1.6	6.1	1.3	1.7	0.68	1.4	0.61
30	10.	8.0	6.9	5.0	2.0	1.5	1.4	0.70
50	22.	24.	14.	14.	3.9	2.5	1.8	0.68

	$B = 10$		$B = 100$		$B = 1000$		$B = 10000$	
n	PC	CP	PC	CP	PC	CP	PC	CP
2	3.2	2.4	2.1	1.5	0.27	0.17	0.041	0.0069
5	1.6	1.0	1.0	0.67	0.14	0.074	0.019	0.0064
10	1.1	0.74	0.71	0.49	0.099	0.058	0.014	0.011
20	1.0	0.58	0.56	0.35	0.078	0.054	0.024	0.028
30	2.0	1.6	1.2	1.6	0.25	0.73	0.050	0.16
50	7.2	6.5	4.0	5.0	0.52	1.6	0.066	0.35

The general trends are plain, and consistent with the complexity estimates:

- For fields defined by random polynomials ($n \leqslant 20$), the denominator D is close to 1. Polynomial multiplication (POL) is roughly twice slower than the M_x method for small to moderate inputs, and needs large values of B to overcome it, when $\mathsf{M}(B)$ becomes so large that divide and conquer methods are used (the larger n, the earlier this occurs). The multiplication table (TAB) is roughly $n/2$ times slower when B is small, and about as fast when $B \gg 1$.
- In the compositums of large degree, D is large. This has a marked detrimental effect on polynomial multiplication, requiring huge values of $B \gg \log_2 D$ to make up for the increased coefficient size.

In short, converting to polynomial representation is the best option for a one-shot multiplication in moderately large degrees, say $n > 5$, when the bit size is large compared to $\log_2 D$. When D is large, the multiplication table becomes faster.

In any case, (A) is the preferred method of multiplication, when precomputations are possible (prime ideals and valuations, see §5.4.1), or more than about $[K : \mathbb{Q}]/2$ multiplications by the same M_x are needed, to amortize its computation (ideal multiplication, see §5.3.2).

We shall not report on further experiments with larger polynomials P. Suffice to say that, as expected, the polynomial representation becomes relatively more costly, since M is mostly insensitive to the size of P.

5.2. Norms. Let $x = \sum_{i=1}^{n} x_i w_i \in \mathcal{O}_K$, $(x_i) \in \mathbb{Z}^n$. If x has relatively small norm, the fastest practical way to compute Nx seems to multiply together the embeddings of x, pairing complex conjugates, then round the

result. This requires that the embeddings of the (w_i) be precomputed to an accuracy of C significant bits (cf.§3.1), with

$$C = O(\log Nx) = O(n \log \|x\|).$$

Note that the exact required accuracy is cheaply determined by computing, then bounding, Nx as a low accuracy floating point number. Note also that a non trivial factor $D > 1$ of Nx may be known by construction, for instance if x belongs to an ideal of known norm, as in §6.1.1 where $D = p^{f(\mathfrak{p}/p)}$. In this case $(Nx)/D$ can be computed instead, at lower accuracy $C - \log_2 D$, hence lower cost: we divide the approximation of Nx by D before rounding. If the embeddings of x are not already known, computing them has $O(n^2\mathsf{M}(C))$ bit complexity. Multiplying the n embeddings has bit complexity $O(n\mathsf{M}(C))$.

If $S(X)$ is a representative of x in $K = \mathbb{Q}[X]/(P)$, then $Nx = \mathrm{Res}_X(P, S)$. Computing a resultant over $\mathbb{Z}$ via a modular Euclidean algorithm using the same upper bound for Nx has a better theoretical complexity, especially if quadratic multiplication is used above, namely

$$O(n^2 C \log C + C^2),$$

using $O(C)$ primes and classical algorithms (as opposed to asymptotically fast ones). Nevertheless, it is usually slower if the x_i are small, in particular if a change of representation is necessary for x. In our implementations, the subresultant algorithm (and its variants, like Ducos's algorithm [15]) is even slower. If the embeddings are not known to sufficient accuracy, one can either refine the approximation or compute a modular resultant, depending on the context.

Remark 5.2. The referee suggested an interesting possibility, if one allows Monte-Carlo methods (possibly giving a wrong result, with small probability)[6]. In this situation, one can compute modulo small primes and use Chinese remainders without bounding a priori the necessary accuracy, i.e. without trying to evaluate C, but stopping as soon as the result stabilizes. It is also possible to compute M_x then $Nx = \det M_x$ modulo small primes and use Chinese remainders. This is an $O(n^3 C \log C + C^2)$ algorithm, which should be slower than a modular resultant if n gets large, but avoids switching to polynomial representation.

[6]For instance, when factoring elements of small norms in order to find relations in $\mathrm{Cl}(K)$ for the class group algorithm: if an incorrect norm is computed, then a relation may be missed, or an expensive factorization into prime ideals may be attempted in vain. None of these are practical concerns if errors occur with small probability.

5.3. Ideals.

5.3.1. *Representation.* An integral ideal $\mathfrak{a}$ is given by a matrix whose columns, viewed as elements of $\mathcal{O}_K$, generate $\mathfrak{a}$ as a $\mathbb{Z}$-module. We do not impose any special form for this matrix yet although, for efficiency reasons, it is preferable that it be a basis, and that $a \in \mathbb{N}$ such that $(a) = \mathfrak{a} \cap \mathbb{Z}$ be readily available, either from the matrix, or from separate storage.

This matrix is often produced by building a $\mathbb{Z}$-basis from larger generating sets, for instance when adding or multiplying ideals. An efficient way to do this is the HNF algorithm modulo a. It has the added benefit that the HNF representation is canonical, for a fixed (w_i), with entries bounded by a. A reduced basis is more expensive to produce, but has in general smaller entries, which is important for some applications, e.g pseudo-reduction, see §5.3.6. Using the techniques of this paper, it is a waste to systematically reduce ideal bases.

5.3.2. *Multiplication.* Let $\mathfrak{a}, \mathfrak{b} \in I(K)$ be integral ideals, given by HNF matrices A and B. We describe $\mathfrak{a}$ by a 2-element $\mathcal{O}_K$-generating set: $\mathfrak{a} = (a, \pi)$, with $(a) = \mathfrak{a} \cap \mathbb{Z}$ and a suitable π (see §6.3). Then the product $\mathfrak{ab}$ is computed as the HNF of the $2n \times n$ matrix $(aA \mid M_\pi B)$. If $(b) = \mathfrak{b} \cap \mathbb{Z}$, the HNF can be computed modulo $ab \in \mathfrak{ab}$. Note that $\mathfrak{a} \cap \mathbb{Z}$ is easily read off from A since $w_1 = 1$, namely $|a|$ is the upper left entry of the HNF matrix A. The generalization to fractional ideals represented by pairs $(\alpha, \mathfrak{a})$, $\alpha \in \mathbb{Q}$, $\mathfrak{a}$ integral, is straightforward.

One can determine $\mathfrak{ab}$ directly from the $\mathbb{Z}$-generators of $\mathfrak{a}$ and $\mathfrak{b}$, but we need to build, then HNF-reduce, an $n \times n^2$ matrix, and this is about $n/2$ times slower.

5.3.3. *Inversion.* As in [9, §4.8.4], our ideal inversion rests on the duality

$$\mathfrak{a}^{-1} = (\mathfrak{d}^{-1}\mathfrak{a})^* := \left\{x \in K, \mathrm{Tr}(x\mathfrak{d}^{-1}\mathfrak{a}) \subset \mathbb{Z}\right\},$$

where $\mathfrak{d}$ is the different of K and $\mathfrak{a}$ is a non-zero fractional ideal. In terms of the fixed basis (w_i), let $T = (\mathrm{Tr}(w_i w_j))_{1 \leqslant i,j \leqslant n}$, $X = (x_i)_{1 \leqslant i \leqslant n}$ representing $x = \sum_{i=1}^n x_i w_i \in K$, and M the matrix expressing a basis of a submodule $\mathcal{M}$ of K of rank n. Then the equation $\mathrm{Tr}(x\mathcal{M}) \subset \mathbb{Z}$ translates to $X \in \mathrm{Im}_{\mathbb{Z}}\, {}^tM^{-1}T^{-1}$. In particular $\mathfrak{d}^{-1}$ is generated by the elements associated to the columns of T^{-1}. The following is an improved version of [9, Algorithm 4.8.21] to compute the inverse of a general $\mathfrak{a}$, paying more attention to denominators, and trivializing the involved matrix inversion:

Algorithm 5.3 (inversion)
Input: A non-zero integral ideal $\mathfrak{a}$, $(a) = \mathfrak{a} \cap \mathbb{Z}$, $B = dT^{-1} \in M_n(\mathbb{Z})$ where d is the denominator of T^{-1}, and the integral ideal $\mathfrak{b} := d\mathfrak{d}^{-1}$ associated to B, given in two-element form.
Output: The integral ideal $a\mathfrak{a}^{-1}$.

(1) Compute $\mathfrak{c} = \mathfrak{a}\mathfrak{b}$, using the two-element form of $\mathfrak{b}$. The result is given by a matrix C in HNF.
(2) Compute $D := C^{-1}(aB) \in M_n(\mathbb{Z})$. Proceed as if back-substituting a linear system, using the fact that C is triangular and that all divisions are exact.
(3) Return the ideal represented by the transpose of D.

The extraneous factor d, introduced to ensure integrality, cancels when solving the linear system in Step (2). In the original algorithm, $|\operatorname{disc} K| = N\mathfrak{d}$ played the role of the exact denominator d, and $C^{-1}B$ was computed using the inverse of TC, which is not triangular. If $N\mathfrak{a} \ll d$, it is more efficient to reduce to two-element form $\mathfrak{a} = a\mathcal{O}_K + \alpha\mathcal{O}_K$ (§6.3) and use [10, Lemma 2.3.20] to compute $a\mathfrak{a}^{-1} = \mathcal{O}_K \cap a\alpha^{-1}\mathcal{O}_K$. The latter is done by computing the intersection of $\mathbb{Z}^n$ with the $\mathbb{Z}$-module generated by the columns of $M_{a\alpha^{-1}}$, via the HNF reduction of an $n \times n$ matrix (instead of the $2n \times 2n$ matrix associated to the intersection of two general ideals [10, Algorithm 1.5.1]).

5.3.4. *Reduction modulo an ideal.* Let $x \in \mathcal{O}_K$ and $\mathfrak{a}$ be an integral ideal, represented by the matrix of a $\mathbb{Z}$-basis A. We denote $x \pmod{\mathfrak{a}}$ the "small" representative $x - A\lceil A^{-1}x \rfloor$ of x modulo $\mathfrak{a}$. In practice, we choose A to be either

- HNF reduced: the reduction can be streamlined using the fact that A is upper triangular [10, Algorithm 1.4.12].
- reduced for the ordinary L^2 norm, yielding smaller representatives.

We usually perform many reductions modulo a given ideal. So, in both cases, data can be precomputed: in particular the initial reduction of A to HNF or reduced form, and its inverse. So the fact that LLL is slower than HNF modulo $\mathfrak{a} \cap \mathbb{Z}$ should not deter us. But the reduction itself is expensive: it performs n^2 (resp. $n^2/2$) multiplications using the reduced (resp. HNF) representation.

The special case $\mathfrak{a} = (z)$, $z \in \mathbb{Z}_{>0}$ is of particular importance; we can take $A = z\,\mathrm{Id}$, and $x \pmod{z}$ is obtained by reducing modulo z the coordinates of x (symmetric residue system), involving only n arithmetic operations. To prevent coefficient explosion in the course of a computation, one should reduce modulo $\mathfrak{a} \cap \mathbb{Z}$ and only use reduction modulo $\mathfrak{a}$ on the final result, if at all.

5.3.5. *Extended Euclidean algorithm.* The following is an improved variant of [10, Algorithm 1.3.2], which is crucial in our approximation algorithms, and more generally to algorithms over Dedekind domains (Chapter 1, *loc. cit.*). In this section we use the following notations: for a matrix X, we write X_j its j-th column and $x_{i,j}$ its (i,j)-th entry; we denote by E_j the j-th column of the $n \times n$ identity matrix.

Algorithm 5.4 (Extended Gcd)
Input: $\mathfrak{a}$ and $\mathfrak{b}$ two coprime integral ideals, given by matrices A and B in HNF. We specifically assume that $w_1 = 1$.
Output: $\alpha \in \mathfrak{a}$ such that $(1 - \alpha) \in \mathfrak{b}$.

(1) Let $z_{\mathfrak{a}}$ and $z_{\mathfrak{b}}$ be positive generators of $\mathfrak{a} \cap \mathbb{Z}$ and $\mathfrak{b} \cap \mathbb{Z}$ respectively.
(2) *[Handle trivial case].* If $z_{\mathfrak{b}} = 0$, return 1 if $\mathfrak{a} = \mathcal{O}_K$. Otherwise, output an error message stating that $\mathfrak{a} + \mathfrak{b} \neq \mathcal{O}_K$ and abort the algorithm.
(3) For $j = 1, 2, \ldots, n$, we construct incrementally two matrices C and U, defined by their columns C_j, U_j; columns C_{j+1} and U_{j+1} are accumulators, discarded at the end of the loop body:
   (a) *[Initialize].* Let $(C_j, C_{j+1}) := (A_j, B_j)$ and $(U_j, U_{j+1}) := (E_j, 0)$. The last $n - j$ entries of C_j and C_{j+1} are 0.
   (b) *[Zero out C_{j+1}].* For $k = j, \ldots, 2, 1$, perform Subalgorithm 5.5. During this step, the entries of C and U may be reduced modulo $z_{\mathfrak{b}}$ at will.
   (c) *[Restore correct $c_{1,1}$ if $j \neq 1$].* If $j > 1$, set $k := 1$, $C_{j+1} := B_1$, $U_{j+1} := 0$, and perform Subalgorithm 5.5.
   (d) If $c_{1,1} = 1$, exit the loop and go to Step (5).
(4) Output an error message stating that $\mathfrak{a} + \mathfrak{b} \neq \mathcal{O}_K$ and abort the algorithm.
(5) Return $\alpha := AU_1 \pmod{\operatorname{lcm}(z_{\mathfrak{a}}, z_{\mathfrak{b}})}$. Note that $\operatorname{lcm}(z_{\mathfrak{a}}, z_{\mathfrak{b}}) \in \mathfrak{a} \cup \mathfrak{b}$.

Sub-Algorithm 5.5 (Euclidean step)

(1) Using Euclid's extended algorithm compute (u, v, d) such that
$$uc_{k,j+1} + vc_{k,k} = d = \gcd(c_{k,j+1}, c_{k,k}),$$
and $|u|$, $|v|$ minimal. Let $a := c_{k,j+1}/d$, and $b := c_{k,k}/d$.
(2) Let $(C_k, C_{j+1}) := (uC_{j+1} + vC_k, aC_{j+1} - bC_k)$. This replaces $c_{k,k}$ by d and $c_{k,j+1}$ by 0.
(3) Let $(U_k, U_{j+1}) := (uU_{j+1} + vU_k, aU_{j+1} - bU_k)$.

Proof. This is essentially the naive HNF algorithm using Gaussian elimination via Euclidean steps, applied to $(A \mid B)$. There are four differences: first, we consider columns in a specific order, so that columns known to have fewer non-zero entries, due to A and B being upper triangular, are treated first. Second, we skip the final reduction phase that would ensure that $c_{k,k} > c_{k,j}$ for $j > k$. Third, the matrix U is the upper part of the base change matrix that would normally be produced, only keeping track of operations on A: at any time, all columns C_j can be written as $\alpha_j + \beta_j$, with $(\alpha_j, \beta_j) \in \mathfrak{a} \times \mathfrak{b}$, such that $\alpha_j = AU_j$. Here we use the fact that $\mathfrak{b}$ is an $\mathcal{O}_K$-module, so that $z_{\mathfrak{b}} w_i \in \mathfrak{b}$ for any $1 \leqslant i \leqslant n$. Fourth, we allow reducing C or U modulo $z_{\mathfrak{b}}$, which only changes the β_j.

We only need to prove that if $(\mathfrak{a}, \mathfrak{b}) = 1$, then the condition in Step (3d) is eventually satisfied, justifying the error message if it is not. By abuse of notation, call A_i (resp. B_i) the generator of $\mathfrak{a}$ (resp. $\mathfrak{b}$) corresponding to the i-th column of A (resp. B). After Step (3b), $c_{1,1}$ and $z_{\mathfrak{b}}$ generate the ideal $I_j := (A_1, \ldots, A_j, B_1, \ldots, B_j) \cap \mathbb{Z}$. Hence, so does $c_{1,1}$ after Step (3c). Since $(\mathfrak{a}, \mathfrak{b}) = 1$, we see that $I_n = \mathbb{Z}$ and we are done. □

Cohen's algorithm HNF-reduces the concatenation of A and B, obtaining a matrix $U \in \mathrm{GL}(2n, \mathbb{Z})$, such that $(A \mid B)U = (\mathrm{Id}_n \mid 0)$. It then splits the first column of U as $(u_A \mid u_B)$ to obtain $\alpha = Au_A$. Our variant computes only part of the HNF (until 1 is found in $\mathfrak{a} + \mathfrak{b}$, in Step (3d)), considers smaller matrices, and prevents coefficient explosion. For a concrete example, take K the 7-th cyclotomic field, and $\mathfrak{a}$, $\mathfrak{b}$ the two prime ideals above 2. Then Algorithm 5.4 experimentally performs 22 times faster than the original algorithm, even though coefficient explosion does not occur.

Remark 5.6. This algorithm generalizes Cohen's remark that if $(z_{\mathfrak{a}}, z_{\mathfrak{b}}) = 1$, then the extended Euclidean algorithm over $\mathbb{Z}$ immediately yields the result. Our algorithm succeeds during the j-th loop if and only if 1 belongs to the $\mathbb{Z}$-module spanned by the first j generators of $\mathfrak{a}$ and $\mathfrak{b}$. In some of our applications, we never have $(z_{\mathfrak{a}}, z_{\mathfrak{b}}) = 1$; for instance in Algorithm 6.3, this gcd is the prime p.

Remarks 5.7.

(1) In Step (3c), the Euclidean step can be simplified since C_{j+1}, U_{j+1} do not need to be updated.
(2) We could reduce the result modulo $\mathfrak{ab}$, but computing the product would already dominate the running time, for a minor size gain.
(3) As most modular algorithms, Algorithm 5.4 is faster if we do not perform reductions modulo $z_{\mathfrak{b}}$ systematically, but only reduce entries which grow larger than a given threshold.

5.3.6. *LLL pseudo-reduction.* This notion was introduced by Buchmann [7], and Cohen *et al.* [12]. Let $\mathfrak{A}$ an integral ideal, and $\alpha \in \mathfrak{A}$ be the first element of a reduced basis of the lattice $(\mathfrak{A}, T_2)$. By Proposition 4.1 and (2), $\|\alpha\|$ and $N\alpha$ are small; the latter is nevertheless a multiple of $N\mathfrak{A}$. We rewrite $\mathfrak{A} = \alpha(\mathfrak{A}/\alpha)$ where $(\mathfrak{A}/\alpha)$ is a fractional ideal, pseudo-reduced in the terminology of [9]. Extracting the content of $\mathfrak{A}/\alpha$, we obtain finally $\mathfrak{A} = a\alpha\mathfrak{a}$, where $a \in \mathbb{Q}$, $\alpha \in \mathcal{O}_K$ and $\mathfrak{a} \subset \mathcal{O}_K$ are both integral and primitive. Assume $\mathfrak{A}$ is given by a matrix of $\mathbb{Z}$-generators $A \in M_n(\mathbb{Z})$. The reduction is done in two steps:

- Reduce A in place with respect to the L^2 norm.
- Reduce the result A' with respect to an approximate T_2 form as defined is §4.2, that is reduce $R^{(e)}A'$ with respect to the L^2 norm, for a suitably chosen e.

We define a pseudo reduction map by $\mathrm{red}(\mathfrak{A}) = \mathrm{red}_e(\mathfrak{A}) := (a, \alpha, \mathfrak{a})$. This is a purely algorithmic definition, depending on the precise way in which α is found: none of the three components is intrinsically defined. This construction generalizes in the obvious way to fractional ideals.

If we want $N\mathfrak{a}$ to be as small as possible[7], then e is chosen relatively large, and the LLL parameter $c \in]1/4, 1[$ is chosen close to 1, for optimal reduction. In our applications, however, we are primarily interested in preventing coefficient explosion, so we may skip the second step altogether for the sake of speed. From (2), the corresponding α already has a relatively small norm. In fact it is easy to prove that $|N\alpha| / N\mathfrak{A}$ is bounded by a constant depending only on $(w_i)_{1 \leqslant i \leqslant n}$ and the LLL parameter c.

5.4. Prime ideals.

5.4.1. *Uniformizers.* Let $\mathfrak{p}/p$ be a prime ideal. It is desirable to describe $\mathfrak{p}$ as

$$\mathfrak{p} = p\mathcal{O}_K + \pi\mathcal{O}_K,$$

and we shall see below that it is useful to impose $v_{\mathfrak{p}}(\pi) = 1$. This condition is automatically satisfied if $\mathfrak{p}/p$ is ramified; both $\pi - p$ and $\pi + p$ satisfy it if π does not. Such a π is called a *p-uniformizer* for $\mathfrak{p}$. More generally:

Definition 5.8. Let $\mathfrak{f} \in I(K)$, and $\mathfrak{p}$ a prime ideal.

- An integer $\pi \in \mathcal{O}_K$ is an *$\mathfrak{f}$-uniformizer* for $\mathfrak{p}$, if $v_{\mathfrak{p}}(\pi) = 1$ and $v_{\mathfrak{q}}(\pi) = 0$, for all $\mathfrak{q} \mid \mathfrak{f}$, $\mathfrak{q} \neq \mathfrak{p}$. (The ideal $\mathfrak{f}$ may or may not be divisible by $\mathfrak{p}$.)
- Let $\mathcal{O}_{K,\mathfrak{p}} := \{x \in K, v_{\mathfrak{q}}(x) \geqslant 0, \forall \mathfrak{q} \neq \mathfrak{p}\}$ be the ring of $\mathfrak{p}$-integers. A $\mathfrak{p}$-integer $\tau \in \mathcal{O}_{K,\mathfrak{p}}$ is an *anti-uniformizer* for $\mathfrak{p}$, if $v_{\mathfrak{p}}(\tau) = -1$.

We shall see in §6.1 how to find a uniformizer. Anti-uniformizers are used to invert $\mathfrak{p}$ (see §5.4.2) and to compute valuations at $\mathfrak{p}$ (see §5.4.3).

Proposition 5.9. *Let $\mathfrak{p}/p$ a prime ideal, π a p-uniformizer for $\mathfrak{p}$, and $\tau_0 \in \mathcal{O}_K$ such that $\pi\tau_0 \equiv 0 \pmod{p}$, and $p \nmid \tau_0$. Then $\tau = \tau_0/p$ is an anti-uniformizer.*

Proof. (simplifies [9, §4.8.3]) The conditions on τ_0 are equivalent to $v_{\mathfrak{p}}(\tau_0) = e(\mathfrak{p}/p) - 1$, and $v_{\mathfrak{q}}(\tau_0) \geqslant e(\mathfrak{q}/p)$ for other prime divisors $\mathfrak{q}$ of p. The result follows. □

Given π, we compute such a τ_0 as a lift of any non-zero solution of $M_\pi X = 0$ over $\mathbb{F}_p^n$.

[7] In particular when we want $\mathfrak{a}$ to be smooth with respect to a factor base $\{\mathfrak{p}, N\mathfrak{p} < y\}$. In this case and if $K/\mathbb{Q}$ is not Galois, consider rather the original $(\mathfrak{A}/\alpha)$, which is often more friable than its primitive part $\mathfrak{a}$. Namely, let $\mathfrak{p}/p$ be a prime ideal; $\mathfrak{p}^{-1}$ is smooth if $N\mathfrak{p} < y$, but $p\mathfrak{p}^{-1}$ is not whenever there exists $\mathfrak{q} \mid p$ with $N\mathfrak{q} \geqslant y$.

Remark 5.10. If we are allowed precomputations associated to $\mathfrak{p}$, the algorithmic data we store is $(p, e(\mathfrak{p}/p), f(\mathfrak{p}/p), M_\pi, M_{\tau_0})$, where $e(\mathfrak{p}/p)$ and $f(\mathfrak{p}/p)$ are the inertia and residue degree respectively, π is a p-uniformizer, and $\tau_0 = p\tau \in \mathcal{O}_K$ where τ is an anti-uniformizer for $\mathfrak{p}$. Note that π and τ_0 are essentially defined modulo p, hence their coordinates can be taken in $]-p/2, p/2]$, except that the condition $v_\mathfrak{p}(\pi) = 1$ requires that we extend the range of the first coordinate of π to $]-p, p]$ if $e(\mathfrak{p}/p) = 1$. The entries of M_π and M_{τ_0} are correspondingly small.

5.4.2. *Multiplication by* $\mathfrak{p}^n$. It is easy to compute $\mathfrak{p}^n$, $n \in \mathbb{Z}$, from the above data; see [10, Proposition 2.3.15], which treats the case $n \geqslant 0$. The general case is an exercise: let $\mathfrak{p} = (p, \pi)$ and $n_0 = \lceil |n| / e(\mathfrak{p}/p) \rceil$.

- If $n \geqslant 0$, then $\mathfrak{p}^n = (p^{n_0}, \pi^n)$
- If $n < 0$, then $p^{n_0}\mathfrak{p}^n = (p^{n_0}, \tau_0^{|n|}/p^{|n|-n_0})$, where the division is exact and both sides are integral.

As a consequence, it is simpler to multiply by $\mathfrak{p}^n$ than by a general ideal, since the two-element representation is readily available. It is even simpler to multiply by $\mathfrak{p}^{\pm 1}$ since M_π and M_{τ_0} are precomputed.

5.4.3. *Valuation of* $x \in K^*$. For a fixed choice of anti-uniformizer $\tau = \tau_0/p$, we define the $\mathfrak{p}$-coprime part[8] of $x \in K^*$ as $cp_\mathfrak{p}(x) := x\tau^{v_\mathfrak{p}(x)}$. First we assume that $x \in \mathcal{O}_K$:

Algorithm 5.11 (valuation and coprime part)
Input: A prime ideal $\mathfrak{p}/p$, $x \in \mathcal{O}_K \setminus \{0\}$.
Output: $v := v_\mathfrak{p}(x)$ and $y := cp_\mathfrak{p}(x) \in \mathcal{O}_K$.

(1) *[Important special case].* If $x \in \mathbb{Z}$, return $v := e(\mathfrak{p}/p)w$ and $y := (xp^{-w})cp_\mathfrak{p}(p)^w$, where $w := v_p(x)$. Note that $cp_\mathfrak{p}(p) = p\tau^{e(\mathfrak{p}/p)}$ can be precomputed.
(2) Let $v := 0$, $y := x$.
(3) *[Multiply].* Let $y' := y\tau_0 \in \mathcal{O}_K$, computed as $M_{\tau_0}y$.
(4) *[Test].* If $y' \not\equiv 0 \pmod p$, abort the algorithm and return v and y.
(5) Set $y := y'/p$.
(6) Set $v := v + 1$ and go to Step (3).

The general case $x = y/d \in K^*$, $(y, d) \in \mathcal{O}_K \times \mathbb{Z}$ is straightforward: x has valuation $v_\mathfrak{p}(y) - v_\mathfrak{p}(d)$, and coprime part $cp_\mathfrak{p}(y)/cp_\mathfrak{p}(d)$.

Remarks 5.12.

(1) The multiplication, divisibility test and actual division by p in Step (3), (4) and (5) are done simultaneously: each coordinate of y' is tested in turn, right after it is computed.

[8] We shall use that definition in §7.4. It is actually tailored for $x \in \mathcal{O}_K$: in this case, τ is raised to a non-negative power, and $cp_\mathfrak{p}(x) \in \mathcal{O}_K$; using $\pi^{-v_\mathfrak{p}(x)}$ for a uniformizer π would also yield a $\mathfrak{p}$-unit, but may introduce denominators.

(2) One can bound $v_{\mathfrak{p}}(x) \leqslant v_p(Nx)/f(\mathfrak{p}/p)$, hoping for instance to notice that $v_{\mathfrak{p}}(x) = 0$. This is in general pointless since the norm computation is more expensive than multiplication by M_{τ_0} and division by p, unless $p \gg Nx$, see §5.2. On the other hand if Nx is known, or we want $v_{\mathfrak{p}}(x)$ for many different primes, this test is useful.

This algorithm is suitable for small valuations, which is the case in our applications, since we prevent coefficient explosion. If one expects v to be large, Bernstein's elegant binary search [5, Algorithm E] is more indicated, applied as reduce(τ_0, p, x):

Algorithm 5.13 (reduce)
Input: $(t \in \mathcal{O}_K, q \in \mathbb{N}_{>0}, w \in \mathcal{O}_K \setminus \{0\})$, such that $t/q \notin \mathcal{O}_K$.
Output: $(v, w(t/q)^v)$, where $v \geqslant 0$ is maximal such that $w(t/q)^v \in \mathcal{O}_K$.

(1) If wt is not divisible by q, print $(0, w)$ and stop.
(2) Let $(v, y) :=$ reduce$(t^2, q^2, wt/q)$.
(3) If yt is divisible by q, print $(2v+2, yt/q)$, otherwise print $(2v+1, y)$.

5.4.4. *Valuation of ideals.* The valuation of an ideal is computed as the minimum of the valuations of its generators. Algorithm 5.11 is run in parallel on all generators, and the computation stops as soon as one divisibility test fails. We can operate modulo a suitable power of the underlying prime:

Algorithm 5.14 (valuation of $\mathfrak{a} \subset \mathcal{O}_K$)
Input: A prime ideal $\mathfrak{p}/p$, a non-zero primitive integral ideal $\mathfrak{a}$, given as a $\mathbb{Z}$-module by the matrix $A \in M_n(\mathbb{Z})$. For $X \in M_n(\mathbb{Z})$ we denote X_j the j-th column of X for $1 \leqslant j \leqslant n$, identified with an element of $\mathcal{O}_K$.
Output: $v_{\mathfrak{p}}(\mathfrak{a})$.

(1) Compute $v_{\max} := v_{\mathfrak{p}}(\mathfrak{a} \cap \mathbb{Z})$. If $v_{\max} = 0$, abort the algorithm and return 0.
(2) If $N\mathfrak{a}$ is known or cheap to compute, e.g., A is in HNF, let

$$v_{\max} := \min(v_{\max},\, v_p(N\mathfrak{a})/f(\mathfrak{p}/p)).$$

(3) Set $v := 0$, $B := A$. While $v < v_{\max}$, do the following:
  (a) Let $u := \lceil (v_{\max} - v)/e(\mathfrak{p}/p) \rceil$.
  (b) For $j = 1, 2, \ldots, n$:
    (i) Let $y' := M_{\tau_0}(B_j \bmod p^u)$.
    (ii) If $y' \not\equiv 0 \pmod{p}$, go to Step (4).
    (iii) Set $B_j := y'/p$ (multiplication by M_{τ_0}, test and division are done simultaneously).
  (c) Set $v := v + 1$.
(4) Return v.

Proof. Obviously, $v_{\mathfrak{p}}(\mathfrak{a}) \leqslant v_{\max}$. So we can stop once we determine that $v_{\mathfrak{p}}(A_j) \geqslant v_{\max}$ for all $1 \leqslant j \leqslant n$. We now prove that in Step (3(b)i), we can in fact reduce modulo $\mathfrak{b}_v := \mathfrak{p}^{v_{\max}-v}$, not only modulo $\mathfrak{b}_v \cap \mathbb{Z}$ which is (p^u) by §5.4.2:

- If $v_{\mathfrak{p}}(B_j) \geqslant v_{\mathfrak{p}}(\mathfrak{b}_v) = v_{\max} - v$, then $v_{\mathfrak{p}}(B_j \bmod \mathfrak{b}_v) \geqslant v_{\max} - v$.
- If $v_{\mathfrak{p}}(B_j) < v_{\mathfrak{p}}(\mathfrak{b}_v)$, then $v_{\mathfrak{p}}(B_j \bmod \mathfrak{b}_v) = v_{\mathfrak{p}}(B_j)$.

By induction, $v_{\mathfrak{p}}(A_j) < v_{\max}$ implies $v_{\mathfrak{p}}(B_j) = v_{\mathfrak{p}}(A_j) - v = v_{\mathfrak{p}}(\tau^v A_j)$. □

A general non-zero ideal $\mathfrak{a}$ is uniquely written $c\mathfrak{b}$, $\mathfrak{b}$ integral and primitive, $c \in \mathbb{Q}$; its valuation is given by $v_{\mathfrak{p}}(\mathfrak{a}) = v_{\mathfrak{p}}(c) + v_{\mathfrak{p}}(\mathfrak{b})$.

Remarks 5.15.

(1) If there is a single prime $\mathfrak{p}$ above p, computing $v_{\mathfrak{p}}(\mathfrak{a}) = v_p(N\mathfrak{a})/f(\mathfrak{p}/p)$ is faster. If it is in fact inert, $v_{\mathfrak{p}}(\mathfrak{a}) = v_p(\mathfrak{a} \cap \mathbb{Z})$ is even faster.

(2) If $A = (a_{i,j})_{1 \leqslant i,j \leqslant n}$ is in HNF, $j = 1$ is omitted from the main loop since $v_{\mathfrak{p}}(A_1) = v_{\mathfrak{p}}(\mathfrak{a} \cap \mathbb{Z})$ is already known. Also, $N\mathfrak{a}$ needs not be computed since

$$v_p(N\mathfrak{a}) = \sum_{i=1}^{n} v_p(a_{i,i}).$$

As above, $v_p(a_{1,1})$ is already known.

(3) One can replace the columns of A by their primitive parts before starting the main loop. The algorithm is more involved since individual valuations need to be maintained, in case p divides some of the contents. We leave the details to the reader.

(4) As shown in the proof, we could reduce modulo $\mathfrak{p}^{v_{\max}-v}$ in Step (3(b)i), but this would be exceedingly expensive. The last remark of §5.3.5 about the advisability of systematic reduction also applies here.

(5) [9, Algorithm 4.8.17] is more complicated and less efficient, since it computes an HNF at each step, and uses no reduction.

(6) If a two-element form $\mathfrak{a} = a\mathcal{O}_K + b\mathcal{O}_K$ is available, we compute $\min(v_{\mathfrak{p}}(a), v_{\mathfrak{p}}(b))$ instead, which is especially attractive if $a \in \mathbb{Z}$. It is tempting to compute such a two-element form with $a \in \mathbb{Z}$ in any case, using Algorithm 6.13, if a does not have many small prime ideal divisors (using Algorithm 6.15 for $y > 2$ requires computing valuations). This may be worthwhile when $v = v_{\mathfrak{p}}(\mathfrak{a})$ is not too small: the expected cost is

$$1/\prod_{v_{\mathfrak{p}}(a) > v} (1 - 1/N\mathfrak{p})$$

$n \times n$ HNF reductions modulo a, followed by the valuation of a single element, compared to the valuation of $n-1$ elements as above. For an explicit example, take K the 11-th cyclotomic field, and $\mathfrak{p}$ a prime above 3, then Algorithm 5.14 applied to $\mathfrak{p}^v$ is faster than the reduction to two element form for $v \leqslant 6$.

(7) A more efficient, and deterministic, approach is to compute $\mathfrak{b} := (\mathfrak{a}, \mathfrak{p}^{v_{\max}}) = \mathfrak{p}^v$, then $v = v_p(N\mathfrak{b})/f(\mathfrak{p}/p)$. Let π be a p-uniformizer for $\mathfrak{p}$, $v_{\max} = v_{\mathfrak{p}}(\mathfrak{a} \cap \mathbb{Z})$ be as above and $u := \lceil v_{\max}/e(\mathfrak{p}/p) \rceil$. Compute $y := \pi^{v_{\max}} \pmod{p^u}$, then $\mathfrak{b}$ is given by the HNF modulo p^u of $(\mathfrak{a} \mid M_y)$. If many valuations are needed y, and in fact the HNF of M_y modulo p^u, can be precomputed for small values of $v_{\max}$. Experimentally, if we assume the HNF of M_y is precomputed, this method is faster than Algorithm 5.14 whenever $v > 0$ and there is more than one prime above p; if the HNF has to be computed, the method is comparable to the reduction to two element form.

6. Approximation and two-element representation for ideals

6.1. Prime ideals and uniformizers. Let p be a prime number, factoring into prime ideals as

$$p\mathcal{O}_K = \prod_{i=1}^{g} \mathfrak{p}_i^{e_i}.$$

If p does not divide the index $[\mathcal{O}_K : \mathbb{Z}[\theta]]$, Kummer's criterion applies and we easily compute the prime divisors of p in the form $\mathfrak{p} = p\mathcal{O}_K + \pi\mathcal{O}_K$, where $\pi = T(\theta)$ and T is (a lift to $\mathbb{Z}[X]$ of) an irreducible factor of P over $\mathbb{F}_p[X]$. If π is not a p-uniformizer, then $e(\mathfrak{p}/p) = 1$ and $\pi \pm p$ is a p-uniformizer.

If p divides the index, the situation is more interesting. If a p-maximal order was computed using the Round 4 algorithm, then the above information about prime divisors of p is obtained as a by-product (see Ford et al. [20, §6]). Otherwise, an alternative is Buchmann-Lenstra's variant of Berlekamp's algorithm, which in principle yields the same information [9, §6.2.2]. But its most popular version [9, §6.2.4] skips squarefree factorization in order to avoid ideal arithmetic, and does not look for random primitive elements. This variant does not readily produce uniformizers.

More precisely, let $I_p = \prod_{i=1}^{g} \mathfrak{p}_i$ be the radical of $p\mathcal{O}_K$. This radical is computed as the $\mathbb{Z}$-module generated by p and a lift of $I_p/(p) = \mathrm{Ker}(x \to x^{p^k}) \subset \mathcal{O}_K/(p)$ for any k such that $p^k \geqslant [K : \mathbb{Q}]$. Alternatively, for $p > [K : \mathbb{Q}]$, $I_p/(p)$ is the p-trace-radical, i.e the kernel of the $\mathbb{F}_p$-linear map

$$\begin{array}{rcl} \mathcal{O}_K/(p) & \to & \mathrm{Hom}(\mathcal{O}_K/(p), \mathbb{F}_p) \\ x & \mapsto & (y \mapsto \mathrm{Tr}(xy) \pmod{p}). \end{array}$$

Berlekamp's algorithm splits the separable algebra $\mathcal{O}_K/I_p$ given an oracle computing roots of totally split polynomials in $\mathbb{F}_p[X]$. From the computation of the simple factors $\mathcal{O}_K/\mathfrak{p}_i$, it produces the $\mathfrak{p}_i/I_p$ as $\mathbb{F}_p$-subspaces of $\mathcal{O}_K/I_p$.

6.1.1. *Naïve randomized algorithm.* Let $\mathfrak{p}$ be one of the $\mathfrak{p}_i$ given as above by an $\mathbb{F}_p$-basis of $\mathfrak{p}/I_p \subset \mathcal{O}_K/I_p$. In particular, the residue degree $f(\mathfrak{p}/p)$ is known: it is the codimension of $\mathfrak{p}/I_p$. On the other hand, $e(\mathfrak{p}/p)$ is still unknown at this point[9]. From that data, [9, Algorithm 4.7.10] proposes to find a p-uniformizer for $\mathfrak{p}$ by picking random elements x in $\mathfrak{p}$ until Nx is not divisible by $p^{f(\mathfrak{p}/p)+1}$; then $\pi = x$ is a p-uniformizer. This is sensible assuming either p is not too small or that it has few prime divisors:

Lemma 6.1. *An element chosen uniformly at random in* $\mathfrak{p}$ *is a* p*-uniformizer with probability* $\prod_{i=1}^{g}(1 - 1/N\mathfrak{p}_i)$.

Proof. This follows from the inclusion-exclusion principle applied to the sets $A_i := (\mathfrak{p}\mathfrak{p}_i)/I_p$, which satisfy

$$\#(\cup_{i\in S} A_i) = \#(\mathfrak{p}\prod_{i\in S}\mathfrak{p}_i/I_p) = \#(\mathfrak{p}/I_p)/\prod_{i\in S} N\mathfrak{p}_i$$

for any $S \subset [1, g]$. In fact, $\pi \in \mathfrak{p}$ is a p-uniformizer if and only if $\pi \notin \cup_i A_i$. □

In the worst case, p is totally split and the probability becomes $(1-1/p)^n$, which is still relatively large assuming p is not too small; for instance if $p \geqslant n$, this is greater than $\exp(-1 - 1/n) \geqslant \exp(-3/2)$ (see also Lemma 6.16).

Remark 6.2. One should check whether p divides $Nx/p^{f(\mathfrak{p}/p)}$, since the latter should be easier to compute than Nx, see §5.2.

6.1.2. *A practical deterministic algorithm.* Cohen [10, Proposition 1.3.10] gives a deterministic polynomial time procedure relying on the general approximation theorem, to compute a p-uniformizer. But this algorithm is not very practical: it requires many ideal multiplications, each of which requires computing, then HNF-reducing, $n \times n^2$ matrices, as well as computing a base change matrix (realizing an HNF reduction) of dimension ng, which may be as large as n^2. Here is a practical variant:

Algorithm 6.3 (p-uniformizer, deterministic)
Input: $\{\mathfrak{p}_i/I_p, 1 \leqslant i \leqslant g\}$, given by $\mathbb{F}_p$-bases.
Output: a p-uniformizer for $\mathfrak{p}$.

(1) *[Compute* $V_{\mathfrak{p}} = \prod_{\mathfrak{p}_i \neq \mathfrak{p}} \mathfrak{p}_i$ *as* $\mathbb{Z}$*-module].*

[9] We may later compute it as $v_{\mathfrak{p}}(p)$ using Algorithm 5.11, which requires an anti-uniformizer τ, obtained from a p-uniformizer via Proposition 5.9.

(a) Compute $\overline{\overline{V}} := \prod_{\mathfrak{p}_i \neq \mathfrak{p}} \mathfrak{p}_i/I_p$ as the intersection of the $\mathbb{F}_p$ vector spaces $\mathfrak{p}_i/I_p$ (not as a product of ideals: we cannot quickly compute two-element representations).
(b) Let $\overline{V} \subset \mathcal{O}_K/(p)$ be the $\mathbb{F}_p$-subspace generated by $I_p/(p)$ and lifts of an $\mathbb{F}_p$-basis of $\overline{\overline{V}}$.
(c) Let $V_{\mathfrak{p}} \subset \mathcal{O}_K$ be the $\mathbb{Z}$-submodule generated by $p\mathcal{O}_K$ and lifts of generators of $\overline{V}$ (HNF reduction modulo p).

(2) *[Compute $\mathfrak{p}^2$ as $\mathbb{Z}$-module].*
(a) Compute a lift $\overline{\mathfrak{p}}$ of $(\mathfrak{p}/I_p)$ to $\mathcal{O}_K/(p)$ as above. Let $(\gamma_1, \dots, \gamma_k) \in \mathcal{O}_K^k$ be a lift of a basis of $\overline{\mathfrak{p}}$; here $k = g - f(\mathfrak{p}/p)$.
(b) Compute $\mathfrak{p}^2$, which is the $\mathbb{Z}$-module generated by $p^2\mathcal{O}_K$ and the $k(k+1)/2$ products $\gamma_i\gamma_j$, $i \leqslant j \leqslant k$ (the HNF reduction is done modulo p^2).

(3) Compute $u \in \mathfrak{p}^2$ and $v \in V_{\mathfrak{p}}$ such that $u + v = 1$ using Algorithm 5.4.
(4) Find $\tau \in \mathfrak{p} \setminus \mathfrak{p}^2$.
(5) Let $\pi := v\tau + u \pmod p$. If $\pi \in \mathfrak{p}^2$, set $\pi := \pi + p$. Return π.

Note that $\mathfrak{p}/p$ is ramified if and only if $\mathfrak{p}^2 \cap \mathbb{Z} = (p\mathbb{Z})$ in Step (2b). If $\mathfrak{p}/p$ is unramified, then we can take $\pi = p$ in Step (4); otherwise, at least one of the γ_i does not belong to $\mathfrak{p}^2$, and this is easy to test since the HNF for $\mathfrak{p}^2$ is known. The reduction modulo p in the last step ensures that a small element is returned, and the test $\pi \in \mathfrak{p}^2$ is only needed when $e(\mathfrak{p}/p) = 1$.

The most expensive part of Algorithm 6.3 is the computation of $\mathfrak{p}^2$ in Step (2b). It requires $O(n^2)$ multiplication in $\mathcal{O}_K/(p^2)$, for a bit complexity of $O(n^4 \log^2 p)$, followed by an HNF reduction $n \times n(n+1)/2$ modulo p^2, for the same bit complexity. The computation of $\overline{\overline{V}}$ has the same cost in the worst case (n is replaced by $\dim \mathcal{O}_K/I_p = \sum_{i=1}^{g} f(\mathfrak{p}_i/p) \leqslant n$), but is in practice much faster, and is amortized over several $\mathfrak{p}$. Namely, the set of all $V_{\mathfrak{p}_i}$ is computed as follows: for $i = 1, \dots, g-1$, let $a_i = \mathfrak{p}_1 \dots \mathfrak{p}_i$ and $b_i = \mathfrak{p}_i \dots \mathfrak{p}_g$, then $V_{\mathfrak{p}_i}$ is generated by $p\mathcal{O}_K$ and $a_{i-1}b_{i+1}$, for a total of $3g - 4$ multiplications (intersections) in $\mathcal{O}_K/I_p$, instead of the straightforward $g(g-1)$. Note that the obvious computation of $V_{\mathfrak{p}}$ as $I_p\mathfrak{p}^{-1}$ is more expensive since ideal multiplication cannot make use of the two-element representation.

6.1.3. *A better deterministic algorithm.* We favor a second variant, which is still deterministic, but both simpler and faster than Algorithm 6.3. We use the notations from the previous section.

Algorithm 6.4 (p-uniformizer, final version)
Input: $\{\mathfrak{p}_i/I_p, 1 \leqslant i \leqslant g\}$, and $\{\overline{V}_{\mathfrak{p}_i}, 1 \leqslant i \leqslant g\}$, all of them subspaces of $\mathcal{O}_K/(p)$, given by $\mathbb{F}_p$-bases.
Output: a p-uniformizer for $\mathfrak{p}$.

(1) *[Compute $(u, v) \in \mathfrak{p} \times V_{\mathfrak{p}}$ such that $u + v = 1 \pmod p$].*
   (a) Let $(A \mid B)$ be the concatenation of the matrices giving the $\mathbb{F}_p$-bases of $\overline{\mathfrak{p}}$ and $\overline{V}_{\mathfrak{p}}$.
   (b) Compute an inverse image $\binom{a}{b}$ of 1 by $(A \mid B)$, using $\mathbb{F}_p$-linear algebra in $\mathcal{O}_K/(p)$.
   (c) Let u, v be lifts to $\mathcal{O}_K$ of Aa, Bb respectively (take symmetric lifts modulo p).

(2) *[Try $\tau = p$].* At this point, we have $v_{\mathfrak{p}_i}(u) = 0$, for all $\mathfrak{p}_i \neq \mathfrak{p}$, and $v_{\mathfrak{p}}(u) \geqslant 1$.
   (a) Let $x := u$. If $p^{f(\mathfrak{p}/p)+1} \nmid Nx$, return x.
   (b) Let $x := u + p$ or $u - p$, whichever has first coordinate in $]-p, p]$. If $p^{f(\mathfrak{p}/p)+1} \nmid Nx$, return x.

(3) *[$\mathfrak{p}/p$ ramified, $v_{\mathfrak{p}}(u) \geqslant 2$. Try $\tau = \gamma_i$].* For $\tau = \gamma_1, \ldots, \gamma_{k-1}$
   (a) Let $x := v\tau + u \pmod p$.
   (b) If $p^{f(\mathfrak{p}/p)+1} \nmid Nx$, terminate the algorithm and return x.

(4) Return $x := v\gamma_k + u \pmod p$.

Here, we produce u without computing $\mathfrak{p}^2$. Also we use the fact that u, v are essentially defined modulo p to compute them using $\mathbb{F}_p$-linear algebra instead of HNF reductions; we could also use an adapted version of Algorithm 5.4.

Using a random $\tau \in \mathfrak{p}$ in Step (3a) yields a uniformizer iff $\tau \notin \mathfrak{p}^2$, hence with probability $1 - N\mathfrak{p}^{-1} \geqslant 1/2$. Our variant is deterministic, requiring at most two norm computations when $\mathfrak{p}/p$ is unramified, and at most $k+1 \leqslant [K : \mathbb{Q}]$ otherwise. As previously mentioned, knowing that $\mathfrak{p}/p$ is ramified in Step (3) enables us to reduce x modulo p, without losing the p-uniformizer property. If we know in advance that $\mathfrak{p}/p$ is unramified, for instance if $p \nmid \operatorname{disc} K$, the norm computation in Step (2b) can be skipped, since x is necessarily a uniformizer at this point.

6.1.4. *Comparison.* This computation needs to be done at most once for each of the prime divisors of the index $[\mathcal{O}_K : \mathbb{Z}[\theta]]$, so the exact threshold between the competing deterministic algorithms is unimportant. On the other hand, it is crucial not to rely solely on the naive randomized algorithm, as shown by the following example. Consider the field generated by a root of

$$x^{30} - x^{29} + x^{28} - x^{27} + x^{26} + 743x^{25} - 1363x^{24} - 3597x^{23} - 22009x^{22} + 458737x^{21} + 2608403x^{20} + 6374653x^{19} - 1890565x^{18} - 112632611x^{17} - 467834081x^{16} - 1365580319x^{15} - 1188283908x^{14} + 3831279180x^{13} + 28661663584x^{12} + 89106335984x^{11} + 226912479680x^{10} + 443487548480x^{9} + 719797891328x^{8} + 946994403328x^{7} + 1015828094976x^{6} + 878645952512x^{5} + 555353440256x^{4} + 124983967744x^{3} + 67515711488x^{2} - 5234491392x + 400505700352$$

which is the abelian field K of smallest conductor (namely 341) such that 2 splits completely in K and such that $[K : \mathbb{Q}] \geqslant 20$. We allow for precomputations: maximal order, roots of P and embeddings of (w_i) to a relative accuracy of 160 digits, T_2-reduction of the maximal order, preconditioning multiplication in K. This takes 8.5s altogether, 65% of which are spent in the LLL reduction, which is in fact not needed for this specific application.

Algorithm 6.4 computes all 30 prime divisors of 2 in less than 0.6s: 0.3s are spent in Buchmann-Lenstra's splitting, and 0.2s to find all 30 uniformizers, half of which spent computing the $V_{\mathfrak{p}}$. We used the embeddings to compute the required norms (negligible time); using a modular resultant, the norm computations now dominate the running time, which roughly doubles to 1s. Using Algorithm 6.3, already about 8s are needed to compute the various $\mathfrak{p}_i^2$. Using the naive randomized algorithm, the expected number of trials to compute a *single* uniformizer is 2^{30}, and takes a few days in practice. In smaller degrees, the speedup is less extreme: take the fixed field of K by the Frobenius over 7, defined by the more decent-looking polynomial

$$x^{10} - x^9 + 2x^8 + 326x^7 - 1559x^6 - 7801x^5 + 22580x^4 \\ - 47436x^3 + 234144x^2 + 2013120x + 3406336.$$

Then Algorithm 6.4 (experimentally, on average) still splits 2 about 30 times faster than the randomized algorithm (2^{10} expected trials); at least 300 times faster if the latter uses a resultant algorithm to compute norms (we tried Collins's and Ducos's subresultants and a modular resultant, as implemented in the PARI library). So we would better not ignore this issue: the same phenomenon would slow down all ideal multiplications whenever small primes have many divisors in K. We shall give a general solution in §6.3, once we have seen how to solve approximation problems.

6.2. Approximation.

6.2.1. *Uniformizers.* We first write explicitly algorithms to obtain suitable uniformizers for prime ideals:

Algorithm 6.5 (ƒ-uniformizer)
Input: an integral ideal $\mathfrak{f}$ and a prime ideal $\mathfrak{p}/p$.
Output: an $\mathfrak{f}$-uniformizer for $\mathfrak{p}$.

(1) Compute $\mathfrak{a} := \mathfrak{f}\mathfrak{p}^{-v_{\mathfrak{p}}(\mathfrak{f})}$ and $\mathfrak{p}^2$.
(2) Compute $(u, v) \in \mathfrak{p}^2 \times \mathfrak{a}$ such that $u + v = 1$, using Algorithm 5.4.
(3) Let π be a p-uniformizer for $\mathfrak{p}$. Return $v\pi + u \pmod{\mathfrak{a}\mathfrak{p}^2}$.

The following variant is faster than Algorithm 6.5, but produces larger uniformizers:

Algorithm 6.6 (ƒ-uniformizer, using $\mathfrak{f} \cap \mathbb{Z}$)
Input: an integral ideal $\mathfrak{f}$, and a prime ideal $\mathfrak{p}/p$; $(f) = \mathfrak{f} \cap \mathbb{Z}$.
Output: an f-uniformizer, hence an $\mathfrak{f}$-uniformizer, for $\mathfrak{p}$

(1) Let $a := fp^{-v_p(f)}$, $m := p$ if $e(\mathfrak{p}/p) > 1$ and $m := p^2$ otherwise.

(2) Using the extended Euclidean algorithm over $\mathbb{Z}$, find $(u', v') \in \mathbb{Z}$ of minimal absolute value, such that $u'm + v'a = 1$. Let $u := u'm$ and $v := v'a$.
(3) Let π be a p-uniformizer for $\mathfrak{p}$. Return $v\pi + u \pmod{am}$.

Remark 6.7. In many applications, the factorization of $\mathfrak{f}$, hence of $\mathfrak{f} \cap \mathbb{Z}$, is known. In this case, one may replace the definition of $\mathfrak{a}$ (resp. a) in Step (1) by the coprime square free kernel

$$\mathfrak{a} := \prod_{\substack{\mathfrak{q} \mid \mathfrak{f},\ \mathfrak{q} \neq \mathfrak{p} \\ \mathfrak{q} \text{ prime}}} \mathfrak{q}, \quad \text{resp.} \quad a := \prod_{\substack{q \mid f,\ q \neq p \\ q \text{ prime}}} q.$$

The algorithm then operates on smaller objects and produces smaller uniformizers. The extra ideal multiplications needed to compute $\mathfrak{a}$ make this costly for Algorithm 6.5, but it is profitably used in Algorithm 6.6.

6.2.2. *Approximation.* We finally give improved algorithms for the Chinese remainder and approximation theorems (see [10, §1.3.2]):

Algorithm 6.8 (Approximation theorem (weak form))
Input: a set S of prime ideals, and $\{e_{\mathfrak{p}}\} \in \mathbb{Z}^S$.
Output: $a \in K$ such that $v_{\mathfrak{p}}(a) = e_{\mathfrak{p}}$ for $\mathfrak{p} \in S$, $v_{\mathfrak{p}}(a) \geqslant 0$ for $\mathfrak{p} \notin S$.

(1) Let $S_{\mathbb{Q}} := \{\ p \text{ rational prime } :\ \exists\, \mathfrak{p} \in S \text{, such that } \mathfrak{p} \mid p\ \}$, and $f := \prod_{p \in S_{\mathbb{Q}}} p$.
(2) For all $\mathfrak{p} \in S$, compute an f-uniformizer $\pi_{\mathfrak{p}}$ for $\mathfrak{p}$, using Algorithm 6.6.
(3) Set $z := \prod_{\mathfrak{p} \in S} \pi_{\mathfrak{p}}^{e_{\mathfrak{p}}} \in K$. Let d be the denominator of z ($d = 1$ unless $e_{\mathfrak{p}} < 0$ for some $\mathfrak{p}$). Write $d = d_1 d_2$, $d_i \in \mathbb{N}$, where $(d_2, f) = 1$, and d_2 is maximal.
(4) Return $zd_2 \pmod{\prod_{\mathfrak{p} \in S} \mathfrak{p}^{e_{\mathfrak{p}}+1}}$.

Proof. The valuation of z at any prime $\mathfrak{q}$ dividing f is $e_{\mathfrak{q}}$ if $\mathfrak{q} \in S$ and 0 otherwise. The same holds for the valuations of zd_2, since $(d_2, f) = 1$. If $v_{\mathfrak{q}}(zd_2) < 0$, then $\mathfrak{q} \mid d_1$; all prime divisors of d_1 divide f by the maximality of d_2, hence $\mathfrak{q} \mid f$. Finally $\mathfrak{q} \in S$, as was to be shown. □

Remarks 6.9.

(1) The final reduction is included to somewhat prevent coefficient explosion in applications where the product z must be fully evaluated. But the computation of the modulus is expensive compared to the other steps. If acceptable, it is preferable to return the element $d_2 \prod_{\mathfrak{p} \in S} \pi_{\mathfrak{p}}^{e_{\mathfrak{p}}}$ in $\mathbb{Z}[\mathcal{O}_K]$, as an unevaluated product (see §7.1).

(2) We can replace Step (1) by the following: start with $\mathfrak{f} := \prod_{\mathfrak{p}\in S}\mathfrak{p}$; then, for all $\mathfrak{p} \in S$ such that $e_{\mathfrak{p}} < 0$, multiply $\mathfrak{f}$ by

$$\prod_{\substack{\mathfrak{q}\mid p,\ \mathfrak{q}\nmid\mathfrak{f}\\ \mathfrak{q}\text{ prime}}}\mathfrak{q}.$$

Then we compute $\mathfrak{f}$-uniformizers using Algorithm 6.5, and leave the other steps unchanged. The algorithm operates with smaller uniformizers since $\mathfrak{f} \mid f$, at the expense of more complicated ideal operations. The original version is faster and has comparable output size, provided we do not remove the last reduction step.

Algorithm 6.10 (Chinese remainder theorem)
Input: a set S of pairwise coprime integral ideals and $\{x_{\mathfrak{a}}\} \in \mathcal{O}_K^S$.
Output: $a \in \mathcal{O}_K$ such that $a \equiv x_{\mathfrak{a}} \pmod{\mathfrak{a}}$ for all $\mathfrak{a} \in S$.

(1) Compute $\mathfrak{f} := \prod_{\mathfrak{a}\in S}\mathfrak{a}$.
(2) For all $\mathfrak{a} \in S$, find $(u_{\mathfrak{a}}, v_{\mathfrak{a}}) \in \mathfrak{a} \times \mathfrak{f}/\mathfrak{a}$ such that $u_{\mathfrak{a}} + v_{\mathfrak{a}} = 1$, using Algorithm 5.4.
(3) Return $\sum_{\mathfrak{a}\in S} v_{\mathfrak{a}}x_{\mathfrak{a}} \pmod{\mathfrak{f}}$.

Remarks 6.11.

(1) Recall that it is simpler to multiply by $\mathfrak{p}^n$, $n \in \mathbb{Z}$ than by a general ideal (§5.4.2). In most applications, the elements of S are powers of prime ideals.
(2) [10, Proposition 1.3.8] computes the HNF of the concatenation of the matrices associated to all ideals $\mathfrak{f}/\mathfrak{a}$. This produces all $v_{\mathfrak{a}}$ with a single HNF reduction instead of $\#S$ reductions in our variant. But the latter are modular, operate on smaller matrices ($n \times n$ instead of $n\#S \times n\#S$), and allow for early abort. Hence using Algorithm 5.4 as above is more efficient, in time and space.

Algorithm 6.12 (Approximation theorem (strong form))
Input: a set S of prime ideals, $\{e_{\mathfrak{p}}\} \in \mathbb{Z}^S$, and $\{x_{\mathfrak{p}}\} \in K^S$
Output: $y \in K$ such that $v_{\mathfrak{p}}(y - x_{\mathfrak{p}}) \geqslant e_{\mathfrak{p}}$ for $\mathfrak{p} \in S$, $v_{\mathfrak{p}}(y) \geqslant 0$ for $\mathfrak{p} \notin S$.

(1) For $\mathfrak{p} \in S$, $\mathfrak{p} \mid p$, make sure the denominator d of $x_{\mathfrak{p}}$ is a power of p: write $d = d_0p^k$, $(d_0, p) = 1$ and replace $x_{\mathfrak{p}}$ by $(d_0^{-1} \bmod \mathfrak{p}^{e_{\mathfrak{p}}+v_{\mathfrak{p}}(d)}) \times (d_0x_{\mathfrak{p}})$.
(2) Let d be the common denominator of all $x_{\mathfrak{p}}$ (the lcm of the p^k above). For all $\mathfrak{p} \in S$ replace $x_{\mathfrak{p}}$ by $dx_{\mathfrak{p}}$, $e_{\mathfrak{p}}$ by $e_{\mathfrak{p}} + v_{\mathfrak{p}}(d)$.
(3) If any $e_{\mathfrak{p}}$ is $\leqslant 0$, remove $\mathfrak{p}$ from S.
(4) Add to S the $\mathfrak{p} \mid d$, $\mathfrak{p} \notin S$. For all such $\mathfrak{p}$, set $e_{\mathfrak{p}} := v_{\mathfrak{p}}(d)$, $x_{\mathfrak{p}} := 0$.
(5) Using Algorithm 6.10, find y solving the Chinese Remainder problem given by $\{\mathfrak{p}^{e_{\mathfrak{p}}}\}$ and $\{x_{\mathfrak{p}}\}$.
(6) Return y/d.

6.3. Two-element representation. Obtaining two-element $\mathcal{O}_K$-generating sets for an ideal $\mathfrak{a}$ is crucial to efficiently multiply by $\mathfrak{a}$ as explained in §5.3.2. We have already seen in the case of prime ideals that the naive randomized method, although in general fast, is not suitable over arbitrary number fields. It is straightforward to deduce an efficient deterministic procedure from Algorithm 6.8, *provided* $\mathfrak{a}$ is easily factored.

6.3.1. *Practical randomized variant.* The naive randomized algorithm (cf. [10, Algorithm 1.3.15]) is as follows:

Algorithm 6.13 (Two-element form for general ideals, naive version)
Input: an integral ideal $\mathfrak{A}$, given by an HNF matrix A, $(a) = \mathfrak{A} \cap \mathbb{Z}$.
Output: π such that $\mathfrak{A} = a\mathcal{O}_K + \pi\mathcal{O}_K$, or FAIL.

(1) Pick π uniformly at random in $\mathfrak{A}/a\mathcal{O}_K$.
(2) If the HNF of M_π modulo a is equal to A (it is enough to check the diagonal elements), return π. Otherwise return FAIL.

Lemma 6.14. *This algorithms succeeds with probability*

$$\prod_{\mathfrak{p}:\, v_\mathfrak{p}(a) > v_\mathfrak{p}(\mathfrak{A})} (1 - 1/N\mathfrak{p}) \geqslant \prod_{\mathfrak{p} \mid a} (1 - 1/N\mathfrak{p}).$$

Proof. Analogous to Lemma 6.1. □

We modified [10, Algorithm 1.3.15] in two respects. We removed its first step (LLL-reduce the HNF basis) since we only want random elements in the ideal: their size does not play any role. Second, we pick random elements in $\mathfrak{A}/a\mathcal{O}_K$ instead of enforcing arbitrary limits on their coordinates, which ruins the probabilistic analysis without motivation. Reduction modulo a takes care of size increases introduced by these modifications. Combining this with our previous work, we obtain a robust general-purpose method:

Algorithm 6.15 (Two-element form for general ideals)
Input: a primitive integral ideal $\mathfrak{A}$, a real parameter $y \geqslant 2$.
Output: π such that $\mathfrak{A} = a\mathcal{O}_K + \pi\mathcal{O}_K$, $(a) = \mathfrak{A} \cap \mathbb{Z}$.

(1) *[partial split].* Let $a \in \mathbb{N}$ such that $(a) = \mathfrak{A} \cap \mathbb{Z}$.
   (a) Let

$$S_\mathbb{Q} := \{p < y \colon p \mid a\}, \text{and}$$
$$S := \{\text{prime ideals dividing of } p \in S_\mathbb{Q}\}.$$

   (b) Let

$$a_0 := \prod_{p \in S_\mathbb{Q}} p^{v_p(a)}, \quad \text{and} \quad a_1 := a/a_0.$$

   By definition, $(a_0, a_1) = 1$.

(c) Write $\mathfrak{A} = \mathfrak{A}_0\mathfrak{A}_1$, where

$$\mathfrak{A}_0 := \prod_{\mathfrak{p}\in S} \mathfrak{p}^{v_\mathfrak{p}(\mathfrak{A})} = (\mathfrak{A}, a_0) \quad \text{and} \quad \mathfrak{A}_1 := \mathfrak{A} \prod_{\mathfrak{p}\in S} \mathfrak{p}^{-v_\mathfrak{p}(\mathfrak{A})} = (\mathfrak{A}, a_1),$$

and only $\mathfrak{A}_1$ needs to be explicitly computed, as the given gcd: a mere HNF reduction modulo a_1. By construction, $(\mathfrak{A}_0, \mathfrak{A}_1) = 1$ and $(a_i) = \mathfrak{A}_i \cap \mathbb{Z}$.

(2) *[find partial uniformizers].*

(a) Find $\pi_0 \in \mathcal{O}_K$ such that $(a_0, \pi_0) = \mathfrak{A}_0$. Algorithm 6.13 succeeds with probability

$$\prod_{\substack{\mathfrak{p}\in S \\ v_\mathfrak{p}(a_0) > v_\mathfrak{p}(\mathfrak{A}_0)}} (1 - 1/N\mathfrak{p})$$

If this is too small, use Algorithm 6.8 to find $\pi_0 \in \mathbb{Z}[\mathcal{O}_K]$ such that $v_\mathfrak{p}(\pi_0) = v_\mathfrak{p}(\mathfrak{A}) = v_\mathfrak{p}(\mathfrak{A}_0)$ for all $\mathfrak{p} \in S$, then evaluate π_0 modulo a_0.

(b) Using Algorithm 6.13, find π_1 such that $(a_1, \pi_1) = \mathfrak{A}_1$.

(3) Using the extended Euclidean algorithm over $\mathbb{Z}$, find $v_0, v_1 \in \mathbb{Z}$ solving the Bezout equation $a_0v_0 + a_1v_1 = 1$. Set $u_i := a_iv_i \in \mathfrak{A}_i \cap \mathbb{Z}$.

(4) Let $\pi_0' := \pi_0u_1 + u_0$, $\pi_1' := \pi_1u_0 + u_1$ and return $\pi_0'\pi_1'$ modulo a.

Proof. Since $\pi_i' \equiv \pi_i \pmod{a_i\mathcal{O}_K}$ and $\pi_i' \equiv 1 \pmod{a_{1-i}\mathcal{O}_K}$, we have $\mathfrak{A}_i = a_i\mathcal{O}_K + \pi_i'\mathcal{O}_K$ and $(\pi_i', \mathfrak{A}_{1-i}) = 1$ for $i \in \{0, 1\}$. We already know that $(a_0, \mathfrak{A}_1) = (a_1, \mathfrak{A}_0) = 1$. It follows that $a_0a_1\mathcal{O}_K + \pi_0'\pi_1'\mathcal{O}_K = \mathfrak{A}_0\mathfrak{A}_1 = \mathfrak{A}$. □

Lemma 6.16. *Let $C := n\frac{\log a}{y \log y}$. In Step (2b), a random element $\pi_1 \in \mathfrak{A}_1/a_1\mathcal{O}_K$ satisfies $\mathfrak{A}_1 = a_1\mathcal{O}_K + \pi_1\mathcal{O}_K$ with probability greater than $\exp(-C - C/y) \geqslant \exp(-3C/2)$.*

Proof. Let $C_1 := n\frac{\log a_1}{y \log y} \leqslant C$. We use the lower bound from Lemma 6.14 and

$$\prod_{\mathfrak{p}|a_1} (1 - 1/N\mathfrak{p}) \geqslant (1 - 1/y)^{n \log_y a_1} \geqslant \exp(-C_1 - C_1/y) \geqslant \exp(-C - C/y).$$

The middle inequality follows from the rough bound $(1-x) \geqslant \exp(-x-x^2)$, valid for all $0 \leqslant x \leqslant 1/2$ for instance. □

Remarks 6.17.

(1) Choosing $y = 2$ amounts to using the naive algorithm. Picking a larger y means we use the safe deterministic algorithm to handle the dangerous part $\mathfrak{A}_0$ of $\mathfrak{A}$, which is easy to factor, and the fast randomized one to avoid factoring $\mathfrak{A}_1$ completely, or wasting too much time computing valuations. The initial steps are cheap, provided small primes and their prime divisors are precomputed.

(2) We can fix some $C > 0$ and choose y such that $Cy \log y = n \log a$, which bounds from below the probability of success in Step (2b) by a positive constant by Lemma 6.16. Since this y is polynomially bounded in terms of n and $\log a$, all the extra steps linked to $\mathfrak{A}_0$ are done in polynomial time. Since each iteration of Algorithm 6.13 is in polynomial time, the resulting algorithm has expected polynomial running time.

(3) This choice of y assumes that all small primes split completely, which is quite pessimistic. A practical alternative is to run the naive algorithm for a few trials. If it does not succeed, we split the computation using an incremental criterion instead of a worst case bound: factor out a set S of prime divisors of $\mathfrak{A} \cap \mathbb{Z}$ until $\prod_{\mathfrak{p} \in S}(1 - 1/N\mathfrak{p}) < 1/2$, say, defining ideals $\mathfrak{A}_0$ and $\mathfrak{A}_1$ as above. Then recursively apply the same strategy to $\mathfrak{A}_1$, improving our chances by a factor greater than 2 after each splitting. To avoid spending exponential time in the trial division, we can compute a worst case y as above and abort trial division once all rational primes less than y have been removed.

(4) Proposition 1.3.10 (*loc. cit.*), based on the approximation theorem, is impractical: it requires the full factorization of $\mathfrak{A}$, costly ideal multiplications (via HNF reduction of $n \times n^2$ matrices), as well as computing a base change matrix of dimension $n\#S$ where S is the set of primes dividing $\mathfrak{A}$.

(5) The usual form of the two-element problem is: given $\mathfrak{A}$ and an arbitrary $a \in \mathfrak{A}$, find π such that $\mathfrak{A} = a\mathcal{O}_K + \pi\mathcal{O}_K$. The above form is the one needed in practice and we leave this generalization to the reader.

6.3.2. *Deterministic polynomial time algorithm.* From the preceding discussion, Algorithm 6.15 can be modified to run in deterministic polynomial time if the factorization of $\mathfrak{A}$ is known. As it stands, because of our use of Algorithm 6.13, it is randomized, with expected polynomial running time. As suggested in [10, p. 23], it can be modified to run in deterministic polynomial time using factor refinement (see Bach, Driscoll and Shallit [2], Buchmann, Eisenbrand [8], and Bernstein [5]) as follows: given $a \in \mathfrak{A}$, factor (a) and $\mathfrak{A}$ into pairwise coprime ideals. One can refine these factorizations in deterministic polynomial time, so as to find a possibly larger finite coprime base $\{\mathfrak{q}_i, i \in I\}$ such that

$$\mathfrak{A} = \prod_{i \in I} \mathfrak{q}_i^{e_i}, \quad (a) = \prod_{i \in I} \mathfrak{q}_i^{f_i},$$

as well as elements $x_i \in \mathfrak{q}_i^{e_i}$ such that $(x_i/\mathfrak{q}_i^{e_i}, \mathfrak{q}_i) = 1$. Namely, if $\mathfrak{q}^k$ is a term in the factorization of $\mathfrak{A}$, given by a $\mathbb{Z}$-basis, one of the n generators of $\mathfrak{q}^k$ does not belong to $\mathfrak{q}^{k+1}$, say x. Either $(x/\mathfrak{q}^k, \mathfrak{q}) = 1$ and x is a suitable

element, or this splits $\mathfrak{q}$ and we may refine the factorization. Let π be a solution of the Chinese remainder problem associated to $\{\mathfrak{q}_i^{e_i+1}, i \in I\}$ and $\{x_i, i \in I\}$, found using Algorithm 6.10; then $\mathfrak{A} = a\mathcal{O}_K + \pi\mathcal{O}_K$.

We can improve this algorithm by picking $x_i = a$ whenever $e_i = f_i$, but it remains impractical if $\mathfrak{A}$ cannot be factored: it requires many ideal multiplications, using only $\mathbb{Z}$-generators (or solving recursively two-element-form subproblems). We would expect each of these to be slower than the whole run of Algorithm 6.13 on $\mathfrak{A}_1$ in Algorithm 6.15.

7. Another representation for ideals and applications

7.1. The group ring representation. One of our goals is to compute ray class groups, as in [10, Algorithm 4.3.1]. As it stands, if $\mathrm{Cl}(K) = \oplus_{i=1}^{r}(\mathbb{Z}/d_i\mathbb{Z})g_i$, this algorithm requires computing $\alpha_i \in K^*$ such that $(\alpha_i) = g_i^{d_i}$ for all $1 \leqslant i \leqslant r$, and this is quite wasteful if the class group has large exponent.

It is natural to represent $I(K)$ as $\mathbb{Q}^* \times I(K)/\mathbb{Q}^*$, as we have done, where (c, I) represents the ideal cI and I is primitive. Since it is less expensive to multiply principal ideals than general ideals, we could go one step further and represent $I(K)$ as $(K^*/U(K)) \times \mathrm{Cl}(K)$ for some fixed choice of representatives for K^* modulo units and for classes of $\mathrm{Cl}(K)$; unfortunately, obtaining this representation is much harder than ordinary multiplication. Besides, principal ideals also become large, when raised to a large power.

To improve on these aspects, we use the following representation for ideals:

$$\begin{aligned} I(K) &= (\mathbb{Z}[\mathcal{O}_K] \times I(K)) \,/\sim \\ &= \left\{ \Big(\prod_i \alpha_i^{e_i}, \mathfrak{a} \Big), \alpha_i \in \mathcal{O}_K, e_i \in \mathbb{Z}, \mathfrak{a} \in I(K) \right\} / \sim, \end{aligned}$$

where the first component is a formal product, and $\sim$ is the obvious equivalence relation: $(\prod_i \alpha_i^{e_i}, \mathfrak{a})$ represents the ideal which is the product of the two components. None of the individual components are well defined, only their product is. For efficiency reasons, we shall always choose $\mathfrak{a}$ integral and primitive. Multiplication is trivial:

Algorithm 7.1

Input: two ideals $(\prod_i \alpha_i^{e_i}, \mathfrak{a})$ and $(\prod_j \beta_j^{f_j}, \mathfrak{b})$

Output: their product

(1) Compute $(c, \gamma, \mathfrak{c}) :=$ red$(\mathfrak{ab})$, such that $\mathfrak{ab} = c\gamma\mathfrak{c}$, and where $c = x/y$, $x, y \in \mathbb{Z}$.

(2) Output $(xy^{-1}\gamma \prod_i \alpha_i^{e_i} \prod_j \beta_j^{f_j}, \mathfrak{c})$, where the first component is a mere concatenation of the factors.

The precise pseudo-reduction variant used above is irrelevant. Inversion, hence division, is equally easy. The main advantage of this representation is that it is easy to compute large products of ideals, without discarding the principal part, or suffering from coefficient explosion: we always map $\mathbb{Z}[\mathcal{O}_K]$ to a sensible domain before evaluating the formal component. The catch is that testing for equality in a deterministic way becomes non trivial, but we shall never need it.

7.2. Discrete logarithms in $\mathrm{Cl}(K)$**.** We want to compute discrete logarithms in the class group $\mathrm{Cl}(K)$, as in [9, §6.5.5]. It is easier, and sufficient for most applications as we will shortly see, to obtain the principal part as an element in the group ring $\mathbb{Z}[\mathcal{O}_K]$ as follows:

Algorithm 7.2 (Discrete log in $\mathrm{Cl}(K)$)
Input: An ideal $I \in I(K)$, possibly given as a product of ideals. We are given $\mathrm{Cl}(K) = \oplus_{i=1}^{r}(\mathbb{Z}/d_i\mathbb{Z})g_i$.
Output: (f_j) and $\beta \in \mathbb{Z}[\mathcal{O}_K]$, such that $I = \beta \prod_j g_j^{f_j}$.

(1) Compute I as $(\alpha, \mathfrak{a})$, $\alpha \in \mathbb{Z}[\mathcal{O}_K]$, $\mathfrak{a} \in I(K)$, using repeatedly Algorithm 7.1, if I was given in factored form.
(2) Solve the discrete logarithm problem for the *small* ideal $\mathfrak{a}$ in $\mathrm{Cl}(K)$ as $\mathfrak{a} = (\tau)\prod_i g_i^{e_i}$, for some yet unknown principal ideal (τ). (This is done by multiplying $\mathfrak{a}$ by random products of prime ideals in the factor base used to compute the class group, then pseudo-reducing as explained in §5.3.6, until the ideal component of the reduction is smooth.)
(3) Using again Algorithm 7.1, compute $\mathfrak{a}(\prod_i g_i^{e_i})^{-1}$, as $(\beta, \mathfrak{b})$.
(4) Realize the *small* principal ideal $\mathfrak{b}$ as (x), using the same method as in Step (2), but this time computing logarithmic distance components, which yield approximate Archimedean embeddings of x, from which x can be recovered algebraically (see [9, Algorithm 6.5.10]).
(5) Output (e_i) and $\tau := (\alpha\beta x) \in \mathbb{Z}[\mathcal{O}_K]$.

Remarks 7.3.

(1) If the generators (g_i) are smooth with respect to the factor base, then it is not necessary to "smooth out" $\mathfrak{b}$ in Step (4), since the result of Step (2) can be re-used. The original generators produced by Buchmann's algorithm satisfy this property, but arbitrary reduced representatives may not.
(2) Since x is defined modulo units, one may reduce the logarithmic Archimedean components of x modulo the logarithmic embeddings for the units as in Algorithm 7.8, Step (3) (setting $\ell = 1$) in order to get a possibly smaller representative in Step (4).
(3) [9, Algorithm 6.5.10] adds logarithmic Archimedean components instead of accumulating algebraic elements in the group ring. The above variant is more practical when the class number is large. In

particular, we do not need to recognize algebraic integers of huge height at the end of the computation, which can then be done at a lower precision (determined using a fast preliminary floating point computation, which is in general enough to determine x directly).

(4) When the generator is large, the $\mathbb{Z}[\mathcal{O}_K]$ representation is more compact than the expanded form. Let h be the class number; the number of terms in the group ring representation for the principal part of $\prod_{j=1}^{N} I_j^{f_j}$ arising from the above algorithm is

$$O(\log h + \sum_{j=1}^{N} \log(f_j)) = O(\log|\mathrm{disc}(K)| + \sum_{j=1}^{N} \log(f_j)).$$

(5) Even if we are interested in computations in $\mathrm{Cl}_{\mathfrak{f}}(K)$ it makes sense to compute as above instead of reducing $\bmod^* \mathfrak{f}$, which would also prevent coefficient explosion. Indeed, we may need to vary the modulus, for instance when computing the conductor of an abelian extension using [10, Algorithm 4.4.2], or to study various class fields over K.

We now use the group ring representation of elements for the basic operations of computational class field theory.

7.3. Signatures.

Definition 7.4. Let $\mathfrak{f} = \mathfrak{f}_0\mathfrak{f}_\infty$ be a modulus and $x \in K^*$. The *signature of x with respect to $\mathfrak{f}$* is

$$s(x, \mathfrak{f}) := \{\mathrm{sign}(v(x)), v \mid \mathfrak{f}_\infty\} \in \{-1, 1\}^{\mathfrak{f}_\infty}.$$

For $x = \prod_i \alpha_i^{e_i} \in \mathbb{Z}[\mathcal{O}_K]$, this is computed as

$$s(x, \mathfrak{f}) = \prod_{i:\, e_i \text{ odd}} s(\alpha_i, \mathfrak{f}),$$

where $s(\alpha, \mathfrak{f})$ is obtained as in §3.1 for $\alpha \in \mathcal{O}_K$, using precomputed floating point approximations of the $v(w_i)$, $v \mid \mathfrak{f}_\infty$. Note that x often originates from a binary powering algorithm, in which case most e_i are even. Also the height of each individual α_i is a priori much smaller than the height of their product, hence their signature is evaluated in lower precision. More precisely, low precision approximations are used first, increasing the precision of the computation until signs can be reliably decided, or until the precision of the cached data is reached, in which case it is computed to a higher accuracy.

Remark 7.5. It is possible to compute the signature of $\alpha \in \mathcal{O}_K$ algebraically, assuming we are given elements $(\beta_w) \in \mathcal{O}_K^{\mathfrak{f}_\infty}$ such that $\mathrm{sign}(v(\beta_w)) = (-1)^{\delta_{v,w}}$, for all $v, w \mid \mathfrak{f}_\infty$, for instance from [10, Algorithm 4.2.20] with $\mathfrak{m}_0 = \mathcal{O}_K$. Namely, Sturm or Uspensky's algorithm (see [30]) compute the

number N of real roots of the characteristic polynomial of α in $[0,\infty)$. The characteristic polynomial of $\alpha\beta_v$ has $N \pm 1$ real roots in $[0,\infty)$ according to whether $\mathrm{sign}(v(\alpha)) = \mp 1$.

7.4. Finding representatives coprime to $\mathfrak{f}$. In order to compute discrete logarithms in $\mathrm{Cl}_{\mathfrak{f}}(K)$, we assume that $\mathfrak{f}_0 = \prod_{\mathfrak{p}} \mathfrak{p}^{n_{\mathfrak{p}}}$, the finite part of the modulus, is given in factored form, and that the discrete log problem is solved in all residue fields $\mathcal{O}_K/\mathfrak{p}$ for $\mathfrak{p} \mid \mathfrak{f}$, which is the case if $N\mathfrak{p} - 1$ is smooth for instance.

We need to map $x \in \mathbb{Z}[\mathcal{O}_K]$ to $(\mathcal{O}_K/\mathfrak{f}_0)^*$. This is not completely obvious since nothing guarantees that the individual components in our representation $(\alpha, \mathfrak{a})$ are coprime to $\mathfrak{f}_0$, even though they originate from products of elements in $I_{\mathfrak{f}}(K)$. It is quite easy to recover that situation, using the uniformizers from §6.1, associated to the prime divisors of $\mathfrak{f}_0$. Let $a = \prod_i \alpha_i^{e_i} \in \mathbb{Z}[\mathcal{O}_K]$ represent an element of K coprime to $\mathfrak{f}$. To compute its image in $(\mathcal{O}_K/\mathfrak{f})^*$ it is enough to compute it in each $(\mathcal{O}_K/\mathfrak{p}^{n_{\mathfrak{p}}})^*$. So consider $\mathfrak{p}$ as above and τ an anti-uniformizer for $\mathfrak{p}$. Recall that we define $cp_{\mathfrak{p}}(x) := x\tau^{v_{\mathfrak{p}}(x)}$, which is integral if $x \in \mathcal{O}_K$, and computed using Algorithm 5.11. Since $v_{\mathfrak{p}}(a) = 0$, one has

$$\prod_i cp_{\mathfrak{p}}(\alpha_i)^{e_i} = \tau^{v_{\mathfrak{p}}(a)} \prod_i \alpha_i^{e_i} = a \quad \text{and} \quad (cp_{\mathfrak{p}}(\alpha_i), \mathfrak{p}) = 1, \forall i.$$

Now the reduction can be computed in the obvious way, reducing each e_i modulo the exponent of $(\mathcal{O}_K/\mathfrak{p}^{n_{\mathfrak{p}}})^*$ first. If this exponent is small, we use non-negative residues so that no modular inversion is needed. Otherwise, we use a symmetric residue system and split the product into positive and negative powers, so that at most one inversion is needed. Of course, we reduce modulo $\mathfrak{p}^{n_{\mathfrak{p}}}$, or rather $\mathfrak{p}^{n_{\mathfrak{p}}} \cap \mathbb{Z}$, along the way to prevent coefficient explosion.

Remarks 7.6.

(1) Although we may need to vary $\mathfrak{f}$ as mentioned in §7 [Remark 5], its prime divisors lie in a fixed set given with the problem, for instance the divisors of the modulus used to define the extension in the first place. Hence most of the needed data can be precomputed.

(2) $\alpha \in \mathbb{Z}$ is a frequent special case e.g., arising from denominators. Let $v = v_p(\alpha)$; the above algorithm replaces α by $(\alpha p^{-v}) cp_{\mathfrak{p}}(p)^v$, where both factors are integral and coprime to $\mathfrak{p}$. We regroup all powers of $cp_{\mathfrak{p}}(p)$, and include them as a whole.

(3) This procedure solves trivially the problem of mapping $x \in K^*$, $(x, \mathfrak{p}) = 1$ to $(\mathcal{O}_K/\mathfrak{p}^n)^*$, by writing $x = (xd)d^{-1}$, if d is the denominator of x. Compared to [10, Algorithm 4.2.22], this local variant is simpler and efficiently prevents coefficient explosion.

The above method can be directly applied to $(\mathcal{O}_K/\mathfrak{f})^*$, if we make the anti-uniformizers coprime to $\mathfrak{f}$ using the techniques of §6.2.1. This is slower than the local algorithm, but turns an arbitrary $(\alpha, \mathfrak{a}) \in \mathbb{Z}[\mathcal{O}_K] \times I(K)$, representing an ideal coprime to $\mathfrak{f}$, into $(\beta, \mathfrak{b}) \equiv (\alpha, \mathfrak{a}) \bmod^* \mathfrak{f}$, where all components are integral and coprime to $\mathfrak{f}$. The details are left to the reader.

7.5. Discrete logarithms in $\mathrm{Cl}_\mathfrak{f}(K)$. We adapt [10, Algorithms 4.3.1 & 4.3.2], noting in passing that concerns about generators size have evaporated, so that the techniques of [10, §4.3.2] are not needed anymore in our context.

Algorithm 7.7 (Discrete logs in $\mathrm{Cl}_\mathfrak{f}(K)$)
Input: An ideal $(\alpha, \mathfrak{a})$, where $(\alpha\mathfrak{a}, \mathfrak{f}) = 1$. We are given

$$\mathrm{Cl}(K) = \oplus_{i=1}^{r}(\mathbb{Z}/d_i\mathbb{Z})g_i, \quad \text{and} \quad \mathrm{Cl}_\mathfrak{f}(K) = \oplus_{j=1}^{r_\mathfrak{f}}(\mathbb{Z}/D_j\mathbb{Z})G_j,$$

as well as elements $\gamma_i \in \mathbb{Z}[\mathcal{O}_K]$ such that $(\gamma_i g_i, \mathfrak{f}) = 1$, computed using Algorithm 6.8 (without final reduction).

Output: (f_j) and $\beta \in \mathbb{Z}[\mathcal{O}_K]$, $\beta = 1 \bmod^* \mathfrak{f}$, such that $\alpha\mathfrak{a} = \beta \prod_{j=1}^{r_\mathfrak{f}} G_j^{f_j}$.

(1) *[Work in $\mathrm{Cl}(K)$].* Using Algorithm 7.2, write $\mathfrak{a} = \tau \prod_{i=1}^{r} g_i^{e_i}$, where $\tau \in \mathbb{Z}[\mathcal{O}_K]$.

(2) *[Work in $(\mathcal{O}_K/\mathfrak{f})^*$].* Map $\alpha\tau \prod_{i=1}^{r} \gamma_i^{-e_i}$, which would be coprime to $\mathfrak{f}$ in expanded form, to each $(\mathcal{O}_K/\mathfrak{p}^{n_\mathfrak{p}})^*$ for $\mathfrak{p}^{n_\mathfrak{p}} \,||\, \mathfrak{f}$, and compute its discrete log in $(\mathcal{O}_K/\mathfrak{f})^*$.

(3) Glue the above results to get the discrete log in $\mathrm{Cl}_\mathfrak{f}(K)$ as in [10, Algorithm 4.3.2]. As usual, we do not evaluate the principal part ($\beta \equiv 1 \bmod^* \mathfrak{f}$) of the discrete logarithm, and give it in $\mathbb{Z}[\mathcal{O}_K]$.

The data linked to the γ_i is precomputed; this includes their signatures, and the $cp_\mathfrak{p}(\gamma_i)$ from the previous section for each $\mathfrak{p} \mid \mathfrak{f}$.

7.6. Computing class fields. Cohen [10, Chapter 5] explains how to use Kummer theory in order to compute the class field associated to a given subgroup of $\mathrm{Cl}_\mathfrak{f}(K)$. Using a theorem of Hecke on ramification in Kummer extensions of prime degree ℓ, he restricts to a small list of S-units, among which the defining element lies. This method only applies to extensions whose degree is square free over K: general extensions have to be built in successive steps. Fieker's algorithm [17] also uses Kummer theory, but in a more elegant way, exploiting properties of the Artin map, and does not restrict the relative degree of the extension.

Both methods let $\mathrm{Gal}(K(\zeta_\ell)/K)$ operate on various objects (S-units, ideal classes) to eventually generate defining elements for class fields. All computations can be done using the $\mathbb{Z}[\mathcal{O}_K]$ representation, in particular, the Galois actions are computed componentwise. Now the generating element we obtain in $\mathbb{Z}[\mathcal{O}_K]$ needs to be completely evaluated to produce the required defining polynomial. We make explicit this final evaluation,

using the fact that these elements are defined modulo ℓ-th powers to avoid coefficient explosion:

Algorithm 7.8 (Reduction modulo $(K^*)^\ell$)
Input: $\gamma = \prod_i \gamma_i^{e_i} \in \mathbb{Z}[\mathcal{O}_K]$, $\ell \geqslant 2$ an integer.
Output: $\beta = \gamma \bmod (K^*)^\ell$, $\beta \in \mathcal{O}_K$.

(1) *[exponent reduction].* Reduce all e_i modulo ℓ (to $[0, \ell-1]$), and remove the components with 0 exponent. Let γ' be the resulting group ring element.

(2) *[reduce approximate ℓ-th root ideal].*
   (a) Partially factor each γ_i into prime ideals (factor $N\gamma_i$ by trial division up to some bound) and write $(\gamma') = \prod_I I^{e_I}$, where I is prime or has large norm.
   (b) Compute $(\gamma, J) := \prod_I I^{\lfloor e_I/\ell \rfloor}$ using repeatedly Algorithm 7.1.
   (c) Let $\gamma'' := \gamma'\gamma^{-\ell} \in \mathbb{Z}[\mathcal{O}_K]$: its expansion would be in $\mathcal{O}_K$ and should be relatively small.

(3) *[reduce mod units].* Let $r := r_1 + r_2 - 1$, $(\eta_i)_{1\leqslant i\leqslant r}$ a system of fundamental units in $\mathcal{O}_K^*$, and let $\Lambda(x) := (\log |x|_{\sigma_k})_{1\leqslant k\leqslant r+1} \in \mathbb{R}^{r+1}$ denote the Dirichlet embeddings of $x \in K$. For the LLL constant α, LLL-reduce the matrix

$$\begin{pmatrix} 0 & \dots & 0 & C \\ \ell\Lambda(\eta_1) & \dots & \ell\Lambda(\eta_r) & \Lambda(\gamma'') \end{pmatrix},$$

   where C is a large enough constant:

$$C > \alpha^{r/2} M, \quad \text{with } M := \ell \max_{1\leqslant i\leqslant r} \|\Lambda(\eta_i)\|_2.$$

(4) The last vector in the LLL base change matrix has the form $u = (u_1, \dots, u_r, \pm 1)^t$. If its last coordinate is negative, negate u.

(5) Expand $\beta := \gamma'' \prod_i \eta_i^{\ell u_i} \in \mathcal{O}_K$, either by direct recognition from its embeddings if the accuracy is sufficient, or using modular techniques and chinese remaindering together with the bound on the embeddings obtained from the floating point computation.

Proof. The only non-obvious part is the assertion in Step (4). Let $(b_i)_{1\leqslant i\leqslant r+1}$ be the reduced basis obtained in Step (3). The first r vectors of the original basis of our rank $r+1$ lattice are smaller than M, hence $\|b_i\|_2 \leqslant \alpha^{r/2} M < C$ for $1 \leqslant i \leqslant r$ by Proposition 4.1. This implies that the first coordinate of b_i is 0 for $i \leqslant r$, and the assertion follows. □

Note that $(\Lambda(\eta_1), \dots, \Lambda(\eta_r))$ is a by-product of the class group algorithm ([9, §6.5]), and is reduced once and for all. Step (2) is similar to Montgomery's square root for the Number Field Sieve [27, 16], generalized to

ℓ-th powers and inexact root extraction[10]. We then make explicit use of our knowledge of the maximal order and its units.

Remark 7.9. In the context of class field computations using Kummer theory, the γ_i are completely factored in Step (2a), since all components of γ are S-units for an explicit set S contained in the set of prime divisors of $\ell\mathfrak{f}$. In Step (2c), we then have

$$\left|N\gamma''\right| \leqslant NJ \prod_{\mathfrak{p}|\ell\mathfrak{f}} (N\mathfrak{p})^{\ell-1} \leqslant NJ \cdot N(\ell\mathfrak{f})^{\ell-1},$$

where NJ is bounded be a constant depending only on K. Nevertheless, $\|\gamma''\|$ may still be large.

Remark 7.10. We could further borrow from Montgomery the idea of allowing *negative* exponents in Step (1) so as to foster cancellations if the support of γ is small, as is the case in both NFS and class field computations (see Nguyen [28] for various such strategies). Since the support of γ is so much smaller in our case than in NFS, it does not seem worth the effort.

7.7. Examples. We implemented the methods explained in this section in the PARI library, as the routine `rnfkummer`, which uses Cohen's method[11].

7.7.1. *A simple example.* For the very simple example in [10, §5.6.2], using fully evaluated elements yields an absolute equation with "rather large coefficients (typically 15 digits), and very large discriminant (typically 2000 decimal digits)"[*loc. cit.*]. Our implementation using formal products produces in 2s the already nice-looking relative equation

$$X^3 + (6z^4 - 18z^3 + 6z^2 - 18z - 12)X + (4z^5 - 30z^4 - 32z^2 - 24z - 4),$$

without applying any reduction algorithm besides the reduction modulo third powers from Algorithm 7.8; the absolute norm of its discriminant has 22 decimal digits. The corresponding absolute equation has L^2 norm $\approx 3.10^6$, a discriminant of 178 decimal digits and is trivial to reduce using [11], since the field discriminant is completely factored.

7.7.2. *A difficult example.* When the class group of $K(\zeta_\ell)$ is large, computations using fully evaluated elements are impossible due to coefficient explosion. We shall see they are extremely fast using the factored representation, once the class group and units of $K(\zeta_\ell)$ are computed. The latter remains the bottleneck of all methods using Kummer theory.

[10] If, as in NFS, we want to compute an exact ℓ-th root, we accumulate separately the ℓ-th powers discarded above. In NFS, one is interested in an ℓ-th root modulo a fixed integer and coefficient explosion does not occur when expanding this result.

[11] This code is included in PARI/GP version 2.2.4 and onward.

The following problem was contributed by Schein: compute the Hilbert class field of $K = \mathbb{Q}(\sqrt{181433})$, which is a degree $\ell = 5$ extension. Alas, the computed class group[12] of $K(\zeta_\ell)$ has type $\mathbb{Z}/3620\mathbb{Z} \times \mathbb{Z}/20\mathbb{Z}$, so fully evaluated algebraic numbers are useless here: many of the ones we need to manipulate incorporate 3620-th (or worse) powers. Working with floating point embeddings in discrete log computations implementing [9, Algorithm 6.5.10] requires about 10^5 decimal digits of accuracy. This is impractical.

Computing tentative class group and units for $K(\zeta_\ell)$, a randomized process, takes between 40s and 2 min depending on the chosen random seed. Using the techniques of this section, manipulating the same huge elements in a different form, we quickly produce a relative polynomial $P \in K[X]$, supposedly defining the Hilbert class field of K (15 *seconds*). This P is still large: $N_{K/\mathbb{Q}}(\mathrm{disc}(P))$ has 2628 decimal digits. We compute the absolute extension (< 10ms), use a polynomial reduction algorithm (1min 45s) and search for subfields [23] (< 10ms) to eventually produce the polynomial

$$X^5 - X^4 - 77X^3 - 71X^2 + 360X - 169$$

which is easily seen to define the required unramified extension of K. For instance, it is enough to notice that the quintic field it generates is totally real and has discriminant $\mathrm{disc}(K)^2$.

The Stark units algorithm of Cohen-Roblot [13] produces a relative polynomial of comparable size (the norm of its discriminant has 2485 decimal digits), but is more cumbersome: it requires about 45 minutes computational time, using 600 MBytes RAM in the Pari implementation (one can dispense with precomputations and reduce memory usage to our default 10 MBytes, roughly tripling running times).

Remark 7.11. To compute a class field of relative degree $\prod_p p^{e_p}$, the methods of Cohen and Fieker both spend most of their time determining tentative class groups and units for the $K(\zeta_{p^{e_p}})$. In addition, Cohen's method also needs to compute the invariants of the $K(\zeta_{p^i})$ for $1 \leqslant i < e_p$, but these are smaller degree fields, a priori easier to handle. So it might still be competitive in the general case.

[12] This part of the computation uses heuristic bounds and does not yield a proven result, even assuming the GRH. For this reason, our class field algorithms are of Monte-Carlo type (randomized, with possibly wrong result). This is harmless in practice since the final defining polynomial is easy to check.

References

[1] B. ALLOMBERT, *An efficient algorithm for the computation of Galois automorphisms*. Math. Comp. To appear.

[2] E. BACH, J. DRISCOLL, & J. SHALLIT, *Factor refinement*. J. Algorithms **15** (1993), no. 2, 199–222.

[3] K. BELABAS, *A relative van Hoeij algorithm over number fields*. J. Symbolic Comput. To appear.

[4] K. BELABAS & H. GANGL, *Generators and relations for $K_2\mathcal{O}_F$*. K-Theory. To appear.

[5] D. J. BERNSTEIN, *Factoring into coprimes in essentially linear time*. J. Algorithms. To appear.

[6] J. BUCHMANN & H. W. LENSTRA, JR., *Approximating rings of integers in number fields*. J. Théor. Nombres Bordeaux **6** (1994), no. 2, 221–260.

[7] J. BUCHMANN, *A subexponential algorithm for the determination of class groups and regulators of algebraic number fields*. Séminaire de Théorie des Nombres, Paris 1988–1989, Progr. Math., vol. **91**, Birkhäuser, 1990, 27–41.

[8] J. BUCHMANN & F. EISENBRAND, *On factor refinement in number fields*. Math. Comp. **68** (1999), no. 225, 345–350.

[9] H. COHEN, *A course in computational algebraic number theory*, third ed., Springer-Verlag, 1996.

[10] H. COHEN, *Advanced topics in computational number theory*, Springer-Verlag, 2000.

[11] H. COHEN & F. DIAZ Y DIAZ, *A polynomial reduction algorithm*. Sém. Théor. Nombres Bordeaux (2) **3** (1991), no. 2, 351–360.

[12] H. COHEN, F. DIAZ Y DIAZ, & M. OLIVIER, *Subexponential algorithms for class group and unit computations*. J. Symbolic Comput. **24** (1997), no. 3-4, 433–441, Computational algebra and number theory (London, 1993).

[13] H. COHEN & X.-F. ROBLOT, *Computing the Hilbert class field of real quadratic fields* Math. Comp. **69** (2000), no. 231, 1229–1244.

[14] M. DABERKOW, C. FIEKER, J. KLÜNERS, M. POHST, K. ROEGNER, M. SCHÖRNIG, & K. WILDANGER, KANT V4, J. Symbolic Comput. **24** (1997), no. 3-4, 267–283, Computational algebra and number theory (London, 1993).

[15] L. DUCOS, *Optimizations of the subresultant algorithm*. J. Pure Appl. Algebra **145** (2000), no. 2, 149–163.

[16] M. ELKENBRACHT-HUIZING, *An implementation of the number field sieve* Experiment. Math. **5** (1996), no. 3, 231–253.

[17] C. FIEKER, *Computing class fields via the Artin map*. Math. Comp. **70** (2001), no. 235, 1293–1303.

[18] C. FIEKER & C. FRIEDRICHS, *On reconstruction of algebraic numbers*. Algorithmic number theory (Leiden, 2000), LNCS, vol. **1838**, Springer, 2000, 285–296.

[19] U. FINCKE & M. POHST, *Improved methods for calculating vectors of short length in a lattice, including a complexity analysis*. Math. Comp. **44** (1985), no. 170, 463–471.

[20] D. FORD, S. PAULI, & X.-F. ROBLOT, *A fast algorithm for polynomial factorization over $\mathbb{Q}_p$*. J. Théor. Nombres Bordeaux **14** (2002), no. 1, 151–169.

[21] X. GOURDON, *Algorithmique du théorème fondamental de l'algèbre*. Rapport de recherche **1852**, INRIA, 1993.

[22] J. L. HAFNER & K. S. MCCURLEY, *A rigorous subexponential algorithm for computation of class groups*. J. Amer. Math. Soc. **2** (1989), no. 4, 837–850.

[23] J. KLÜNERS, *On computing subfields. A detailed description of the algorithm*. J. Théor. Nombres Bordeaux **10** (1998), no. 2, 243–271.

[24] D. E. KNUTH, *The art of computer programming*. Vol. **2**, second ed., Addison-Wesley Publishing Co., Reading, Mass., 1981, Seminumerical algorithms, Addison-Wesley Series in Computer Science and Information Processing.

[25] H. KOY & C.-P. SCHNORR, *Segment LLL-Reduction with Floating Point Orthogonalization*. CaLC, LNCS, vol. **2146**, Springer, 2001, 81–96.

[26] A. K. Lenstra, H. W. Lenstra, Jr., & L. Lovász, *Factoring polynomials with rational coefficients*. Math. Ann. **261** (1982), no. 4, 515–534.
[27] P. L. Montgomery, *Square roots of products of algebraic numbers*. Mathematics of Computation 1943–1993: a half-century of computational mathematics (Vancouver, BC, 1993). Proc. Sympos. Appl. Math., vol. **48**, Amer. Math. Soc., 1994, 567–571.
[28] P. Nguyen, *A Montgomery-like square root for the number field sieve. Algorithmic number theory (Portland, OR, 1998)*, Lecture Notes in Comput. Sci., vol. **1423**, Springer, 1998, 151–168.
[29] PARI/GP, version `2.1.5`, Bordeaux, 2003, `http://pari.math.u-bordeaux.fr/`.
[30] F. Rouillier & P. Zimmermann, *Efficient isolation of a polynomial real roots*. Journal of Computational and Applied Mathematics. To appear.
[31] C.-P. Schnorr & M. Euchner, *Lattice basis reduction: improved practical algorithms and solving subset sum problems*. Math. Programming **66** (1994), no. 2, Ser. A, 181–199.
[32] J. von zur Gathen & J. Gerhard, *Modern computer algebra*, Cambridge University Press, New York, 1999.

Karim Belabas
Mathématiques–Bâtiment 425
Université Paris–Sud
F–91405 Orsay Cedex
E-mail : `Karim.Belabas@math.u-psud.fr`