# CLASSGROUP COMPUTATIONS IN LARGE DEGREE

CLAUS FIEKER

This is a draft version of May 28, 2018, 10:10a. It is still growing and under construction.

## 1. Overview

$K$ is a number field, viewed as $K\mathbb{Q}[t]/f$ where $f \in \mathbb{Z}[t]$ is monic and irreducible. Then $\mathcal{E} := \mathbb{Q}[t]/f$ is an order, the so called equation order.

Note: the equation order is non-canonical, it does depend on the chosen polynomial.

In $K$ there is a canonical order $\mathbb{Z}_K$ the maximal order, or ring of integers. It is the integral closure of $\mathbb{Z}$ in $K$, or the normalisation of $\mathcal{E}$. $\mathbb{Z}_K$ is a Dedekind domain, a free $\mathbb{Z}$-module of rank $n = \deg f = K : \mathbb{Q}$.

The (fractional) ideals in $\mathbb{Z}_K$ form a group under multiplication. In general orders, e.g. the equation order the invertable ideals still form a group, but there are non-invertable ideals as well. The quotient of the ideal group modulo the principal ideals is the ideal class group $\mathrm{Cl}_K$. This is a finite abelian group of size $h_K$ the class number of $K$ (or $\mathbb{Z}_K$).

The units in $\mathbb{Z}_K$ (or any order in $K$) are, modulo torsion, a free abelian group of rank $r = r_1 + r_2 - 1$ where $r_1$ is the number of real roots of $f$ (or the number of real embeddings of $K$ into $\mathbb{C}$, the field of complex numbers) and $2r_2$ is the number of non-real roots. The torsion subgroup is cyclic, generated by a root of unity.

We fix an ordering on the embeddings, $(.)^{(i)} : K \to \mathbb{R}$ for $1 \le i \le r_1$, $(.)^{(i+r_1)} = \overline{(.)^{(i+r_1+r_2)}} : K \to \mathbb{C}$ the pairs of complex embeddings. Orders give rise to lattices in essentially two ways:

- $x \mapsto \psi(x) = (x^{(1)}, \ldots, x^{(r_1)}, \Re(x^{(r_1+1)}), \Im(x^{(r_1+1)}), \ldots) \in \mathbb{R}^n$
- $x \mapsto L(x) = (\log|x^{(1)}|, \ldots, \log|x^{(r_1)}|, \log|x^{(r_1+1)})|, \ldots, \log|x^{(r_1+r_2)}|) \in \mathbb{R}^{r_1+r_2}$

Later we will also encounter variants.

## 2. Classgroup I

Since the class group is finite, we can fix a number $B$ and a set of prime ideals $\mathcal{F} := \{P | N(P) \le B\}$ called a factor base. If $B$ is large enough, then the elements of $\mathcal{F}$ generate the class group. We will mostly assume this.

A relation $\alpha \in K$ is an element s.th. $v_P(\alpha) = 0$ for all $P \notin \mathcal{F}$. Equivalently, $\langle\alpha\rangle = \alpha\mathbb{Z}_K = \prod_{P\in\mathcal{F}} P^{v_P(\alpha)}$ . Clearly, the set of all relations forms a $\mathbb{Z}$-module under multiplication. Let $\mathcal{R}$ be this set. Dirichlet's unit theorem implies that, up to torsion, $\mathcal{R}/T(\mathbb{Z}_K^*) \equiv \mathbb{Z}^{r+\#\mathcal{F}}$ the set of $\mathcal{F}$-units, then

$$1 \to \mathcal{R} \to \langle\mathcal{F}\rangle \to \mathrm{Cl}_K \to 1$$

Let $\phi : \mathcal{R} \to \mathbb{Z}^{\mathcal{F}} : \alpha \to (v_P(\alpha))_{P\in\mathcal{F}}$ we get

$$1 \to \ker\phi \to \mathbb{Z}^{\mathcal{F}} \to \mathrm{Cl}_K \to 1$$

Thus computation of the class group is "equivalent" to finding $\ker\phi$. As this is a submodule of $\mathbb{Z}^{\mathcal{F}}$, which is finitely generated, hence we need generators for $\ker\phi$. This is not quite enough, we will add conditions to this later.

Fixing relations $\alpha_i \in \mathcal{R}$ $(1 \le i \le k)$ we obtain the relation matrix $M = (v_P(\alpha_i))_{P\in\mathcal{F},i}$.

Computation of the class group means finding

- $\mathcal{F}$
- $\alpha_i$
- $M$

The difficult part is the 2nd step: finding (enough) relations. To emphasize again: computation of the class group is *not* just about the computation of the structure/ order of the finite abelian group $\mathrm{Cl}_K$, it is about the computation of data allowing further use.

Buchmann gave the complexity of the class group algorithm, under the GRH and "plausible assumptions" for families of fields of bounded degree as

$$L_d[1/2, O(1)]$$

where $d$ is the disriminant of the field and

$$L_d[\alpha, \beta] = \exp(\beta \log d^\alpha \log\log d^{1-\alpha} + o(1))$$

is the usual subexponential function.

## 3. Limits

We assume the maximal order to be known. There is the fundamental result by Chistov that the computation of the maximal order is polynomial time equivalent to finding the largest square-free divisor of the polynomial discriminant. From a complexity point of view, the latter is an "$L_d[1/3, O(1)]$" algorithm, the class group is mostly $L_d[1/2, O(1)]$. There is a range of possibilities for the actual computation of the maximal order, once the factorisation (or the critical primes) is known, but I won't deal with this.

Actually, one can run the class group algorithm directly on the non-maximal order to obtain essentially the Picard-group of the order, however, it seems to be better (read faster) to first compute the maximal order and then proceed with the computation.

## 4. Classgroup II: Biasse-Fieker

Buchmann's result (or the versions of Cohen, Diaz y Diaz and Oliver) were for a long time both theortically and practically the limit. Implementations in Pari/gp (=sage), Kant/KaSH = Magma both followed this approach, so lets look into more details.

Let $B$ as above be a bound for the prime ideals, we'll come back to the choice of $B$. Buchmann's strategy was essentially this: Let $\mathfrak{a}$ be a random product of ideals in $\mathcal{F}$. Compute $\mathfrak{b} = \mathfrak{a}^{-1}$ and let $b$ be the 1st basis vector of a LLL-reduced basis for $\mathfrak{b}$. If $b\mathfrak{a}$ is $\mathcal{F}$-smooth, then $b$ is a relation. If not, then take a new ideal $\mathfrak{a}$ and try again.

Buchmann "proved" that the probability of a random ideal of norm $N(\mathfrak{a}) < C$ to be $B$-smooth is essentially the "same" as that of an random integer bounded by $C$ to be $B$-smooth, hence

$$\mathrm{prob}_{\text{good ideal}}(C, B) = \psi(C, B)/C.$$

Assuming $B = C^{1/a}$ for $a \in \mathbb{Z}_{>0}$ we get via the Dickman-rho function:

$$\psi(C, C^{1/a}) \leq \rho(a) \leq 1/a!$$

So, what is $C$? Using the LLL theorem, one sees

$$C \leq 2^n\sqrt{d}$$

The $\sqrt{d}$ is the reason for the $L_d[1/2, O(1)]$ in Buchmann, while the $2^n$ is the source of the bounded degree: $2^n$ grows exponentially.

Thus the problem in the class group algorithm is to find (many) elements/ ideals of norm bounded by s.th. smaller than $\sqrt{d}$. So far all generic methods involve geometry of numbers, hence lattices and have the $\sqrt{d}$.

Biasse, in this PhD thesis, motivated by the analogy to plane curves in cryptography, found conditions on the polynomial defining the field to allow for many small elements: if the coefficients of the defining polynomial are sufficiently small relative to the discriminant, one get many small elements for free: any small linear combination of small powers of the primitive element will have small norm. He quantified all the "small" and was able to prove that there are families of fields where the class group can be computed in $L_d[1/3]$. But still of bounded (fixed) degree.

In order to remove the degree dependency, Biasse suggested using stronger lattice reduction: replacing LLL by BKZ with a suitable blocking factor, increased the runtime complexity of the lattice reduction from polynomial to sub-exponential, but also dropped the $2^n$ to s.th. sub-exponential. In total, this combined to a sub-exponential runtime for fields of unbounded degree.

Reality: is has been observed that the $2^n$ in the LLL is too pessimistic. Stehlé deomonstrated (and argued) that the constant, for random lattices, grows more like $1.02^n$ which means that for fields of degree 500 (which are totally out of reach), we still have only a facor of $20,000$ which is negliable. So, the Buchmann approach is fine in all fields of attainable degree. Currently.

Bigger problem: the "smallest" discriminant of a field of degree 500, using the GRH bounds of Martinet and others, is $> 2^{2730}$, hence, even with a LLL "constant" of 1, we will only find elements of norm $2^{1400}$. If we assume a factor basis of $10^6$ ideals (which we cannot handle), the largest prime, thus the bound $B$ is bounded by the $10^6$-th prime: $p = 15485863$. For the smoothness probability we need to write $B = C^(1/a)$ or $a = \log_p(C) = 58.6$, giving $\rho(a) \le 10^{-75}$. Actually computing $\rho(57)$ gives an even *much* smaller bound. This means we cannot possibly hope to find any relations this way.

For $a = 9$ we have $\rho(9) = 1.01 \cdot 10^{-9}$ which would give a $1 : 10^9$ chance of finding a relation. However, then $B = C^{1/9} = 2^{155}$ is too large for the universe, let alone my computer.

Summary: the Fieker-Biasse approach, while a theoretical breakthrough, is not practical.

Observation: so far, no-one is interested in generic, random, fields of degree $> 100$. All such fields of (current) interest are *special.*

- cyclotomic (or (maximal) subfields)
- multi-quadratic
- small, sparse, defining polynomials
- composita of *nice* fields.

## 5. Ideal Arithmetic

In order to perform any computation in $K$ or $\mathbb{Z}_K$ with either elements or ideals, we need to fix a presentation. Fundamentally, there are two canonical choices for elements of $K$:

- using $K = \mathbb{Q}[t]/f$, elements are represented as polynomials in $\mathbb{Q}[t]$ of degree bounded by $\deg K - 1$
- using the vector space structure, $K = \mathbb{Q}^n$, elements are vectors of length $n$.

(Later we might allow for sparse polynomials or sparse vectors, but lets focus on the dense case)

5.1. **Polynomial presentation.** Element addition/ substraction/ scalar multiplication can immediately be implemented using the correspoding operation for polynomials, hence at an arithmetic cost of $O(n)$.

Multiplication requires a polynomial multiplication, followed by a division with remainder by the defining polynomial. Using classical (school) methods, this comes at a cost of $O(n^2)$. Using asymptotically fast methods this drops to roughly $O(n \log n)$.

Division requires an extended euclidean algorithm, a gcd computation with Bezout coefficients. Again the arithmetic costs are $O(n^2)$ classically and $O(n \log^2 n)$ fast.

Norm computation can be achieved as a resultant of the element with the defining polynomial, hence again at cost $O(n^2)$ resp. $O(n \log n)$.

Application of an automorphism corresponds to a modular composition, again at cost $O(n^2)$ or $O(n \log n)$.

On the other hand, membership testing, ie. $x \in \mathbb{Z}_K$ requires a base-change, hence always quadratic time.

5.2. **Vector presentation.** Addition and scalar multiplication is no problem here.

In order to multiply we need additional information, either a multiplication table, ie. an array containing products of the basis elements, or a base-change to convert to the polynomials for multiplication. The costs work out to be $O(n^3)$ for the multiplication table vs. $O(n^2)$ for the base-change. However, here things are not what they seem: Using $\omega_i\omega_j = \sum \Gamma_{i,j,k}\omega_k$ we get

$$(\sum a_i\omega_i)(\sum b_j\omega_j) = \sum_{i,j,k} \Gamma_{i,j,k} a_i b_j \omega_k$$

Here, $\Gamma_{i,j,k}$ is fix (fixed by the basis $(\omega_i)_i$, while the coefficients are arbitray, so we have $n^2$ products of arbitrarily large numbers and $n^3$ products of large times small, thus the effective cost is, for large elements, only $O(n^2)$ again.

Division can be done by base-change or by solving a linear system: Let $M_\beta$ be the representation matrix of $\beta$, ie. the matrix where the rows contain the coefficients if $\beta\omega_i$ and let $1 = \sum o_i\omega_i$ then $xM_\beta = 1$ yields the coefficients $x_i$ of the inverse of $\beta$. The costs are $O(n^3)$ arithmetic operations for the linear algebra.

Norms of elements are here be obtained via the determinant of the representation matrix, again at cost $O(n^3)$.

5.3. **Comparison.** Clearly, off hand, there is no point in using the vector version as every operation is more expensive than in the polynomial case. However, this is not quite true in reality: coefficient size matters. Asymptotically, the coefficient size of a single element in the different representations is the "same" ie. the difference is bounded (by a costant coming from the basis used). In any fixed application, in particularly when the field degree is only moderate, this can make a dramatic difference! As we are intersted in large degree (mainly) we ignore this subtle problem and assume the polynomial presentation hence forth.

More representations are known in the literature:

- factored form: elements are power products
- complex approximations and minimal polynomials
- straight-line programmes
- representation matrices

They are all useful - in the correct situation.

5.4. **Ideals.** Ideals are more interesting. Here we have a few "natural" presentations:

- a principal ideals - we we happen to know a generator
- as a $\mathbb{Z}$-module using a basis
- as a $\mathbb{Z}_K$-module using 2-generators.

Fine, but what operations do we need to support? In the class group algorithm we need

- products, quotients
- valuations
- small elements

In general, we'd also like gcd's aka sums of ideals.

5.5. **$\mathbb{Z}$-Modules.** Ideals, in particular integral ideals are free $\mathbb{Z}$-modules of rank $n$ and thus, after fixing a basis of the order/ field, can be represented via an integral matrix:

$$\mathfrak{A} = \sum \alpha_i \mathbb{Z}, \quad \alpha_i = \sum a_{i,j}\omega_j$$

and $M_A = (a_{i,j})_{i,j}$.

To compute $\mathfrak{A}\mathfrak{B}$ given via $M_A$, $M_B$ we have to compute all $n^2$ products of basis elements:

$$\mathfrak{A}\mathfrak{B} = \sum \alpha_i \beta_j \mathbb{Z}$$

thus need $n^2$ products and the HNF reduction of an $n^2 \times n$ matrix. The algebraic complexity is thus $O(n^4)$ (for the linear algebra: the products can be evaluated using representation matrices making the linear algebra the most expensive part).

If one ideal is given via 2-generators and the other via a basis, the linear algebra reduces to $2n \times n$, the complexity to $O(n^3)$. Note that if both ideals are given via 2-generators, we need a $4n \times n$ reduction which is worse.

For division, there are at least two methods: a direct reduction to a $n^2 \times n$ system, or, via inversion of the trace matrix to a clever ideal product (which is typically a $n^2 \times n$ linear problem)

However, it should be noted that

- the presentation can easily be made unique (fixing a hnf)
- to test membership, or to find small elements, a basis is currently needed.

5.6. **2-Element.** In Dedekind domains, every ideal can be generated using 2 elements only. To be more precise:

**Lemma 1.** *Let $0 \neq a \in \mathfrak{A}$ be arbitrary, then one can find $\alpha \in \mathfrak{A}$ s.th. $\langle a, \alpha\rangle = \mathfrak{A}$.*

*If $a \in \mathbb{N}$ and $\alpha$ is chosen uniformly at random in $\mathfrak{A}/a\mathfrak{A}$, then the probability of $\alpha$ working is at least*

$$\prod_{p|a}\prod_{\mathfrak{P}|p} 1 - 1/N(\mathfrak{P})$$

Thus unless small primes $p|a$ are involved that are highly split (e.g. 2 totally split), this probability is typically large: fiding second generators is trivial. Testing is not. Also, there are almost no algorithms for ideal operations via general 2-element presentation - apart from powering: we always have

$$\langle \alpha_i\rangle^l = \langle \alpha_i^l\rangle$$

Pohst and Zassenhaus refined the 2-element presentation to the $S$-normal presentation for any finite set $S$ of prime numbers: Let $S(\mathbb{Z}_K) = \{\mathfrak{P}|\mathfrak{P}|p \in S\}$. Then $(a, \alpha) \in \mathbb{N} \times \mathbb{Z}_K$ is a $S$-normal set of generators for $\mathfrak{A}$ iff

- $v_{\mathfrak{P}}(a) \geq v_{\mathfrak{P}}(\mathfrak{A})$ for all $\mathfrak{P} \in S(\mathbb{Z}_K$, $v_{\mathfrak{P}}(a) = 0$ otherwise
- $v_{\mathfrak{P}}(\alpha) = v_{\mathfrak{P}}(\mathfrak{A})$ for all $\mathfrak{P} \in S(\mathbb{Z}_K)$

($a$ has too large valuation in $S$ - but zero outside, $\alpha$ has the correct valuation in $S$ - but more outside, $a$ is "to high", $\alpha$ to broad)

Here too we have

**Lemma 2.** *If $a \in \mathfrak{A} \cap \mathbb{N}$, $S$ containing all primes in $a$, and $\alpha$ is chosen uniformly at random in $\mathfrak{A}/a^2\mathfrak{A}$, then the probability of $\alpha$ working is* *exactly*

$$\prod_{p|a}\prod_{\mathfrak{P}|p} 1 - 1/N(\mathfrak{P})$$

And here we have an effective test:

**Theorem 1** (Zassenhaus). *Let $m = \min\langle\alpha\rangle \cap \mathbb{N}$, then $(a, \alpha)$ is $S$-normal for $S$ containing all primes in $a$ iff*

$$\gcd(a, a/\gcd(a, m)) = 1$$

.

*Furthermore, $\langle a, \alpha\rangle = \mathfrak{A}$ iff* $\gcd(a^n, N(\alpha)) = N(\mathfrak{A})$

Together with

$$\min\langle\alpha\rangle \cap \mathbb{N} = \mathrm{den}(1/\alpha)$$

This allows for a fast test (at cost of one inversion - and some integers)

The importance here is that in this presentation we have efficient algorithms:

If $\langle a, \alpha\rangle$ and $\langle b, \beta\rangle$ are $S$-normal for the same set $S$, then $\langle ab, \alpha\beta\rangle$ is a $S$-normal presentation for the product - at cost of a single element mutiplication!

Furthermore, $N(\langle a, \alpha) = \gcd(a^n, N(\alpha))$ again, at the cost of a single element norm and finally

Let $\mathrm{den}(1/\alpha) = st$ s.th. $\gcd(t, a) = 1$ and all primes dividing $s$ also divide $a$, then $\langle 1, t/\alpha\rangle$ is a $S$-normal presentation for $\langle a.\alpha\rangle^{-1}$

To use this for the product, we need to be able to change the set $S$:

**Lemma 3** (Sircana). *Let $\langle a, \alpha\rangle$ be $S$-normal and $\langle b, \beta\rangle$ $T$-normal. Then define $s = \prod S \setminus T$ and $t = \prod T \setminus S$, then $\langle a, t\alpha + a^2\rangle$ and $\langle b, s\beta + b^2\rangle$ are both $S \cup T$-normal for the same ideals.*

Note: in the implementation the sets $S$ and $T$ are only implicit: they are defined as the (unknown) primes dividing some integer. Here $s$ and $t$ will have to be computed using some integer gcd operations.

Using this, the (general) ideal multiplication becomes essentially the same cost as a single element multiplication!

The core problems here are

- the presentation is non-unique, ie. comparison has to be done via $\mathfrak{A}\mathfrak{B}^{-1} = \mathbb{Z}_K$
- the presentation is non-unique, ie. sets via hashing, or, in general hashing cannot be done this way
- the presentation is non-unique, in particular the 1st generator tends to grow (unneccessarily) over the time and thus the 2nd one as well

- we know the norm, but not the minimum
- we cannot (yet) compute sums (gcd)

But there are also benefits:

- almost all prime ideals are naturally given in a $\{p\}$-normal presentation
- to compute valuations, an anti-uniformizer is handy an element of valuation $-1$ at the critical prime, 0 at all conjugate primes and non-negative elsewhere. The 2nd generator of the inverse doees this
- until the elements are too large, this is *fast*

To size-reduce a $S$-normal presentation, we can try to find the minimum $m$ as $\gcd(a, \operatorname{den}(1/\alpha))$, then set $S$ to the divisors of $m$ and reduce $\alpha$ modulo $m^2$. But here is the twist: computing $1/\alpha$ takes about $O(n \log n)$ or $O(n^2)$ operations in $\mathbb{Z}$, but, generically, using integers of size $O(n \log a)$ as well. So the total cost is $O(n^2 \log n)$ or, classically $O(n^3 \log a)$ Using linear algebra (hnf) to find the minimum can be done in $O(n^3 \log a)$ as well - so the advantage is gone (or harder to realize).

However, one can see $\operatorname{den}(1/\alpha) = \operatorname{rres}(\alpha, f)$ and one can try to compute the resultant directly in $\mathbb{Z}/a\mathbb{Z}$. which generically is hard. We only have

**Lemma 4** (KL). *Let $\alpha, f \in \mathbb{Z}/a\mathbb{Z}[t]$ and assume $f$ monic. Then we can compute*

- $\operatorname{res}(\alpha, f)$, $\operatorname{rres}(\alpha, f)$ *($= \operatorname{den}(1/\alpha)$)*
- *the Bezaut coefficients*
- $\gcd(a^n, N(\alpha))$

*In time $\tilde{O}(n^2 \log a)$*

Unfortunately, we don't have generically fast ($O(n \log n)$) methods here. Also, one polynomial needs to be monic.

At least, we're beating the linear algebra again. For large degree fields, this makes a huge difference - and the methods, apart from the resultant, are easy to implement.

There are complications for index divisors, but those are not hard to overcome. If the minimum is close to the norm, then the old, direct, approach is faster by a logarithmic factor. If, as generically, the norm is larger by a size-factor of $n$, we win.

## 6. Classgroups III

We need (at least):

- parameters (e.g. factor basis bound)

- many (good) (small) elements
- recognize smoothess
- factor smooth elements
- the image of the relation matrix
- kernel (elements) of the relation matrix
- dependencies among units
- termination

6.1. **Parameters.** The total runtime is roughly the time to find relations plus the processing time (linear algebra). The linear algebra is cubic, possibly only quadratic, in the size of the factor base, so question: how hard is it to find relations? The assumption is that we're finding random elements of bounded norm, typically of norm bounded by $\sqrt{D}$ (or a constant multiple of it). The $\psi$ and $\rho$ functions tell us what the expected yield should be, but experimentally, the succeessrate is much higher! Good! But why?

**Heuristic 1** (Donnelly)**.** *The elements are randomly distributed in a bounded (by coordinates/ conjugates) region. The norm however, is not uniform, it follows the hyperbolic volume: the number elements of norm $\leq b$ should be proportional to*

$$\operatorname{vol}\{\prod x_i \leq b | 0 < x_i \leq 1\}$$

This volume can easily be computed and seems to explain the distribution. The theoretical optimal parameters are such that the relation search and the linear algebra take the same time. Practically, however, the best possible examples depend on the largest size the linear algebra can handle: the relation search is easy to parallelize, the linear algebra not.

6.2. **Relations.** The classical Buchmann (Cohen, Diaz y Diaz, Olivier) method tries a random product $\mathfrak{A}$ of factor basis elements, then finds a/some small elements $\alpha$ in $\mathfrak{A}^{-1}$ and checks if $\alpha\mathfrak{A}$ is smooth over the factor base.

Assuming various things (product large enough, elements not in subfields, ideals uniformly distributed over the class group), one can prove an expected runtime. However in praxis this is too slow. All implementations try to find elements of small norm (to increase the smoothness probability). Controlling the norm directly is difficult, so one settles for $T_2$-small, ie. small in the lattice. Problem is that this controls the norm only badly, ie. in Donnelly's sense. In ideals of small norm, there are frequently many more small norm elements.

Unfortunately, they

- produce dependend rows in the relation matrix
- tend to fall in subfields

In total: they can not completely generate the class group. Only if the norm of the ideal is "sufficiently large" will the relations behave as Buchmann wants them. But then, the ideals are large and slow.

The generic idea here is to not compute the ideals from scratch at every step, but change them slowly: remove a factor (at random), insert another factor, to simulate a random walk through the class group. As "sufficiently large" is hard to quantify, periodic checks are performed to see if enough "progress" is made. If not, the product parameters are increased.

For the choice of elements in $\mathfrak{A}$ there are also differnt choices being used:

- LLL-basis elements only
- (small) (random) combinations of LLL-basis elements
- systematic enumeration of small elements (Fincke-Pohst)

Various people have had various success with either of those methods.

A different approch is to use *sieving* as in the factorisation (NFS) or discrete logarithm computation. So far, this works fine in small degree (and large discriminant) fields, but not for large degree fields. There are some ideas to try, but so far, no-one successfully reported sieving in fields of degree $> 10$.

6.3. **Smoothness.** Usually, the first step here is to see if the norm is smooth, so in particular to compute the norm.

Task: given $\alpha \in \mathfrak{A}$ of norm $N(\alpha) = dN(\mathfrak{A})$ and $d \le \sqrt{D}$, find $d$. The reason for this complicated set-up is that finding $d$ with the additional information is easier than the generic norm.

To compute the norm we have many methods. Focussing on the faster ones:

- product of the conjugates
- resultant

we can use the fact that we want $d = N(\alpha)/N(\mathfrak{A})$ which is much smaller than $N(\alpha)$ directly. This means we can easily bound the (real) precision necessary in the 1st method or limit the number of primes used in the modular method employed in the 2nd. We also note that the occaisonly wrong norm (because it was unexpectedly large) is harmless: a large norm is less likely to yield a relation and we don't require completeness, only to find enough elements.

Also, given that we need many norm computations with, say, the same primes involved in the modular resultant, we can save even more time.

Next step: given $d$ or $dN(\mathfrak{A})$, test smoothness. Again, there are non-trivial savings to be had here:

- using product trees, individual tests are sped up
- using batch-smoothness tests, many integers are tested at the same time!

It should be noted that both improvements require fast arithmetic as they do a tradeof involving large integers. The core idea is, instead of testing (and removing) all primes $p_i$ individualy, to compute $P = \prod p_i$ and then do $d \to d/\gcd(d, P)$ until the gcd is 1. Properly balanced this is much faster than individual tests. Extensions of this will also identify the primes involved in $d$.

Next step: we have all (integer) primes dividing $dN(\mathfrak{A}) = N(\alpha)$ and need the prime ideals $\mathfrak{P}$ (and the exponents) that are involved. Naively, one checks for all prime ideals above a critical prime and computes the valuation. However, for large degree fields this is deadly slow.

Assume, for now, that the equation order is $p$-maximal and that $p$ is unramified. In most examples this covers most ideals, hence most of the time. Also, assume $\alpha \in \mathbb{Z}[t]/f$. Dedekind: $f \equiv \prod f_i \bmod p$ results in prime ideals $\mathfrak{P}_i = \langle p, f_i \rangle$ and

$$\mathfrak{P}_i | \alpha \quad \text{iff} \quad f_i | \underset{\mathbb{F}_p}{\gcd}(f, \alpha)$$

So the critical primes can be found using product trees and (this time) fast polynomial arithmetic (over small finite fields)!

A similar approach can be used for the large prime variant: if, after all factors of the factor base are removed, we are left with an ideal of prime norm, then $\gcd(\alpha, f)$ over the corresponding finite field yields a unique, canonical representation of the ideal, hence a basis for hashing.

A twist however comes from the fact that is two partial relations are merged, the resulting full relation is no longer integral so that the smoothness test is more involved (but can still be done on the polynomials).

In the last step, the actual valuation has to be computed. For almost all ideals involved this will be trivial: the valuation will be 1 and the valuation of the norm is also 1. For the small prime ideals, the valuation can be larger, here one has to be more careful.

6.4. **Image.** The image computation is mostly done in two steps:

- in a suitable finite field (or modulo an auxilliary prime $p$)

- once the matrix has full rank, over $\mathbb{Z}$

The 1st step is done to avoid coefficient explosion - while the finite result is mostly small, the intermediate results (can) have a bit-length proportional to the size of the matrix. Apart from testing for full rank, we can also identify the "missing" primes and change the search strategy to force them in.

When using sieving, a different approach is used: the factor basis is much larger and many more relations are found. Then in the linear algebra, many rows (relations) and columns (ideals) are removed in order to make the size manageable. If enough survives, this is fine.

As the relation matrix is extremely sparse, sparse linear algebra is used here.

6.5. **Kernel.** The kernel, or more precisely, kernel elements, correspond, via the power product of the relations, to units. Hence we need kernel elements. Problem: the bit-size of each coordinate of a kernel element is proportional to the size of the matrix, and, to make things worse, its dense. That means for typical sizes of relation matrices, we cannot simply compute the kernel.

Two approaches are currently used:

- take a (small) (dependend) subset of the rows and the kernel elements from those rows
- compute a straight-line representation of the transformations done in the previous step. Ie. remember the sequence of elementary transformations. In comparison to the direct kernel, this takes $O(nm)$ storage, while the full kernel would be $O((m-n)m^2)$ due to the size of the entries.

In either case, we usually do not compute (or use) the full kernel, but just some elements. Individual kernel elements can also be found using iterative methods (Lanczos, Wiedemann) like in the NFS.

A problem here, in particular with the 1st approach is that the system of units thus constructed, even if one uses all subsets of fixed size, will have a non-trivial (large) index.

6.6. **Units.** Given units $u_i$, given as (huge) products of (many) (small) elements with (huge) exponents, we need to find dependencies, ie. find $n_i$ s.th. $\prod u_i^{n_i} = 1$. Those dependencies are then used to construct a basis for the group generated by all those units from the last step.

The (usual) methods here are using

- (real) logarithms
- $p$-adic logarithms

In either case, $\prod u_i^{n_i} = 1$ is transformed to $\sum n_i \log u_i = 0$. In both cases the complications arise from the need to use approximations: we will never see/ be able to prove an exact zero. Using real-logarithms, the precision neccessary is at least the bit-length of the exponents, hence proportional to the size of the relation matrix.

Using $p$-adic logarithms, the precision will depend on the size of the (unknown) output, which is potentially much smaller. However, it also depends o Leopold's conjecture, hence all relations found need verification - using real-logarithms. Combining the information from various $p$-adic computations using CRT, we could parallelize this.

6.7. **Termination.** At this point we have a (conjectured) class group, in particular a class number $h$ and a (conectured) regulator $r$. Both have, under the assumption that the factor basis was large enough, the property that they can (only) be too large, if they are, they are too large by an integer factor of at least 2.

**Theorem 2.** *There is an explicit constant $c$ coming from the residue of the zeta-function of the number field s.th. $c = hr$.*

This is supplemented by

**Theorem 3** (GRH: Bach, Belabas)**.** *There is a polynomial time algorithm that will compute $\tilde{c}$ s.th. $\tilde{c}/\sqrt{2} \le c \le \tilde{c}\sqrt{2}$*

The information needed here is essentially the factor base: all prime ideals of norm bounded by $O(\log^2 D)$.

So: if $\tilde{c}/\sqrt{2} \le hr \le \tilde{c}\sqrt{2}$ we are done.

If not, we need more relations as either the image (class group) or the kernel (units) are too small.

6.8. **Saturation.** In the run of the algorithm, close to completion, we will be off by a small index, typically a factor of 2 or maybe 3. At this point the relation search is struggeling to find the missing bit. Also, in other applications we are given a a subgroup $V = \langle \alpha_i | 1 \le i \le l \rangle$ of some group of $S$-units and need to enlarge to a $p$-maximal over group

$$W := \{x \in U_S | x^{p^k} \in V\}$$

This task split into 2 problems: identify vectors $k_i \in \mathbb{Z}$ s.th.

$$\prod \alpha_i^{k_i} \in (K^*)^p$$

and second to compute the $p$th-root.

**Lemma 5.** *Let $p$ be a prime and $\alpha \in \mathbb{Z}_K$. Then*

- *$t^p - \alpha \in K[t]$ is irreducible iff it has no roots.*

- *if $t^p - \alpha$ is irreducible, then there are infinitely prime ideals $\mathfrak{P}$ sth. $t^p - \alpha$ is irreducible over the residue field $\mathbb{F}_{\mathfrak{P}}$*

This is repeatedly used: for (many) prime ideals $\mathfrak{P}$ s.th. $p | N(\mathfrak{P}) - 1 = \mathbb{F}_{\mathfrak{P}}^*$ and define

$$\phi_{\mathfrak{P}} : C_p^l = V \to \mathbb{F}_{\mathfrak{P}}^*/()^p = C_p$$

The intersection of the kernels of $\phi_{\mathfrak{P}}$ for many $\mathfrak{P}$, in particular the none-zero elements (or a basis for the intersection modulo $p$-th powers i $V$) are candidates for being $p$th powers. That need testing. Idealy, if enough ideals are used, those elements are $p$th powers.

Now we have $\beta = \prod \alpha_i^{n_i}$, $0 \le n_i < p$ and need to either compute a $p$th root - or show the non-existence. The key problem to any direct approach is the the sheer size of $\beta$: During the class group computation, if applied directly, $k$ will be of size $|\mathcal{F}|$, thus even if the exponents are bounded $\beta$ will have huge coefficients. When spplied to the unit group only, then the $\alpha_i$ are themselves given as power products of the relations, so

$$\alpha_i = \prod r_{i,j}^{m_{i,j}}$$

where the number of terms is $O(|\mathcal{F}|)$ and, expected, $\log |m_{i,j}| = O(|\mathcal{F}|)$ as well. Brauer-Siegel, in conjuction with LLL shows that we will have $\log |\alpha_i| = O(\frac{1}{r}\sqrt{D})$, hence cannot have small (or even manageable sized) coefficients.

We will use the compact presentation to transform

$$\beta = \prod \alpha_i^{n_i} = \prod_{i=0}^{l} \gamma_i^{p^i}$$

where $l = O(\log |\beta|)$ and $T_2(\gamma_i) = O(\sqrt{D})^p$. (Not quite, the size of the prime ideals dividing $\beta$ also comes in here) Clearly, $\beta$ is a $p$th power iff $\gamma_0$ is and $\gamma_0$ is small enough for direct computation. In fact, the expectation is that generically at this point $\gamma_0 = 1$.

6.9. **Compact Presentation.** Given $\beta$ huge and $\langle \beta \rangle = \prod \mathfrak{P}_i^{k_i}$, will can find $\gamma_i$ s.th.

$$\beta = \prod \gamma_i^{p^i}$$

Note, that

- in the class group context all elements are build from relations, hence the prime ideals are explicitly known
- generically, a coprime factorisation is sufficient, thus can be computed in polynomial time (using coprime bases over the integers and then for ideals)

The method splits into 2 parts: support reduction and then size reduction.

To start: $\mathfrak{a}_i = \prod \mathfrak{P}_j^{n_{i,j}}$ and $n_{i,j} = \lfloor k_i/p^i \rfloor \bmod p$, so $0 \le n_{i,j} < p$, and $\langle \beta \rangle = \prod \mathfrak{a}_i^{p^i}$. Gradually, each ideal $\mathfrak{a}_i$ is now reduced: find an (LLL-small) element $\mu \in \mathfrak{a}_i{-}1$ thus $\tilde{\mathfrak{a}_i} = \mu_i \mathfrak{a}_i$ is an integral ideal of bounded norm, $\langle \beta \rangle = \prod \mu_i^{p^i} \prod \tilde{\mathfrak{a}_i}^{p^i}$ with small ideals $\tilde{\mathfrak{a}_i}$. The size of $\mu_i$ will at least depend on the size of the primes dividing $\mathfrak{P}_i$.

The second step appoximates $\tilde{\beta} = \beta \prod \mu_i^{-p^i}$: for $l = \lfloor \log |\tilde{\beta}| \rfloor$ down to 1 do the following $\eta = \frac{1}{p^l} \log |\tilde{\beta}|$ and find $\gamma_l \in \mathfrak{b}_l^{-1}$ which is LLL small for $T_{2,\eta}$. As a result, $\gamma_l^{p^l} \approx \tilde{\beta}$ and $\tilde{\beta}\gamma_l^{-p^l}$ is smaller. Next, we update the $\mathfrak{b}_i$ and continue.

The final $\gamma_0$ is defined as the last $\tilde{\beta}$. Since this is "smallit can explicitly be computed using modular evaluation.

## 7. Special Fields

The techniques above still do not solve the problems inherent in large degree fields, however they are successfully be utilized in "special" fields. So far, essentially no-one is interested in generic large fields, large fields, in crypto or other applications are carefully constructed for various applications. This special properties might be used to help the class group computation:

### 7.1. **Cyclotomics.**

The security of current incarnations of NTRU depends on not being able to solve the PID problem in (large degree) cyclotimic field, mostly power-of-two fields. The Genntry-Sy? algorithm reduced the PID problem to one in the maximal real subfield, hence to a field of half degree.

Alexandre Gelin, in his recent thesis, managed to use special features of the real subfield to solve this in a (totally real) field of degree 128: One can show that $\zeta_{2^n}^i + \zeta_{2^n}^{-i}$ up to ordering are a LLL reduced basis for the maximal order. Gelin used a bound of $25,000$ for the factorbase and tried as relations random sums of up to 6 LLL-basis elements. Those relations were then supplemented by their Galois orbits. In his experiment, he needed little more than 6 hours to obtain a relation matrix of full rank. In fact, this can be sped up to just under one hour.

However, proving anything in this situation seems to be out of reach.

### 7.2. **Multiquadratic.**

Last year, in a long paper, a team suggested to use multiquadratic fields, in particular fields of the form $\mathbb{Q}[\sqrt{p}_1, \ldots, ]$ for (small) distinct primes $p_i$. They showed that in those fields

- fast multiplcation is possible

- determination of the maximal order is "easy" (trivial outside 2)
- the unit group, up to powers of 2 is generated by the units of all (quadratic) subfiields. The maximal power is known as well
- for elements, explicitly given in the canonical basis. fast extraction of square roots is possible.
- the PID problem can also be solved effectively.

In fact, for all those problems, they could avoid any lattice techniques and work in the canonical base using polynomials only.

They reported very good running times for a field of degree 256. They also reported fundamental difficulties if large primes are encountered: in this case, the units of the quadratic field are of size exponential in $p$, not $\log p$, thus the units "require" the compact presentation, hence lattice techniques.

However, when restricted to small primes, the method was proven to run in polynomial time, hence gave the first example of large fields with fast class group not depending on quantum computers.

Classically, this has also been used for normal fields of degree 8 and Galois group $D_4$.

### 7.3. **Verification.**

## 8. Julia/ Hecke

Julia is a strongly typed, JIT compiled language. It is not object oriented, but instead polymorphic: there can be many functions with the same name, selection is done based on the types of all arguments. In order to make (maximal) use of the JIT compilation, care must be taken to have functions being type stable, ie. the output type should only depend on the input type, not the actual values.

In julia, do

```
julia> using Hecke
```

and you are ready to go.

Similar to Magma, Gap and Sage, in order to use non-trivial objects, you need to create the parents, usually some rings.

```
julia> Qx, x = FlintQQ["x"]
```

creates both $\mathbb{Q}[x]$ as a ring and the polynomial $x$.

```
julia> (x+1)^4
```

To create a number field, you need to have a polynomial

```
julia> K, a = number_field(x^5+1)
julia> ZK = maximal_order(K)
julia> C, mC = class_group(ZK)
```

Now try it again: (using `<cursor-up>`: julia remembers your command history even between restarts)

```
julia> C, mC = class_group(ZK)
```

instantenous. But cheating: the class group of $\mathbb{Z}_K$ is already computed so the last one is returned.

```
julia> C, mC = class_group(ZK, redo = true)
```

will actually recompute it again.

Still much faster than the 1st time as now the program does not need to be compiled again. In order to time commands, you can prefix them with `@time`, e.g.

```
julia> @time C, mC = class_group(ZK, redo = true)
```

Note, adding a semicolon `;` to the end of a line suppresses the output.

The class group computation returned 2 values: the class group, as an abelian group (try `typeof(C)`) and a map $mC : C \to I$ where $I$ is the set of fractional ideals. This map can be used to

- obtain an ideal $\mathfrak{a}$ representing an element in $C$
- given an ideal $\mathfrak{a}$, solve the discrete logarithm problem, ie. find the element in $C$

This pattern in repeated everywhere in Hecke: abstract objects (groups) are returned in 2 values, the abstract presentation and a map linking it to the data.

## 8.1. **Fast Algorithms.**

Try `2^100` and compare (and explain) it to `fmpz(2)^100` or `BigInt(2)^100`.

Now compare

```
julia> function factorial(n::Int)
         p = fmpz(1)
         for i=2:n
           p *= i
         end
         return p
       end

julia> @time factorial(100)
julia> @time factorial(100)
julia> @time factorial(100);
julia> @time factorial(50000);
julia> @time prod(fmpz(i) for i=1:50000)
julia> @time prod([fmpz(i) for i=1:50000)]
```

Can you explain this? Can you do better? Can you compete with

```
julia> @time Hecke.my_prod([fmpz(i) for i=1:50000]);
```

(you'll have to run it twice for the compilation)

Doing `@which bla(1,2)` shows you what function will be called and where to find it.

The same ideas can be used for different rings. Whenever we have a product that is asymptotically faster than "quadratic", this technique wins. Use `include("your file")` to store/ load more complex examples.

8.2. **Class Group.** Try to find the possibilities and limits for class group computation in Hecke. Obviously, this depends on the time, but lets restrict to 1min max. Try class groups of "random" small number fields. You migh want to use `rand(fmpz(1):fmpz(100))` or even `x^6 + rand(Qx, 1:5, -100:100)`

Can you find non-quadratic fields with non-trivial class group? Can you find a family? (Evil question: can you actually find a family of fields with *bounded* class number? Nevertheless, explicitly having non-trivial class groups is fun.)

You can follow the progress by doing `set_verbose_level(:ClassGroup, 1)` or even 2. Beware, the larger the number the more output is produced (mostly incomprehensible) - and the slower the programm runs.

Also `\tt ?class_group` shows that there are many/several options not yet explored. In particular, `bound` can be used to manually select the bound used for the factor base in the class group computation.

8.3. **Unit group.** Given one of the above fields, or possible `\tt t^2 -p` for (large) primes $p$ (you can use `next_prime` or `PrimesSet` to get primes).

Eg. s.th. smallish

```
julia> K, a = number_field(x^2-1000003)
julia> ZK = maximal_order(K)
julia> u, mu = unit_group(ZK)
julia> mu(u[2])
julia> u, mu = unit_group_fac_elem(ZK)
julia> mu(u[2])
```

`u[2]` gives you acess to the second generator of the group.

Compare to $x^2 - 100001$ and try larger examples. When do you *have* to use the factored form?

8.4. **Other.** The julia packages are installed into your home directory (in general) in

```
~/.julia/v0.6/Hecke
```

Have a look at the `examples` directory to get some ideas. Can you create multi-quadratic fields? By hand? using the example code? Explore the non-simple fields and compare the results. Saturation and compact presentation are both installed. Can you get it to work?

Can you find good/optimal parameters for the class group? Try varying the bound remembering to use `redo`.

8.5. **Installing.** In order to install julia, one checks `julialang.org/downloads` and best obtains a binary. After starting julia, at the prompt, try

```
julia> Pkg.add("Hecke")
```

which should install

- AbstractAlgebra
- Nemo, including the c-written projects
  - mpir (a gmp-replacement)
  - mpfr
  - flint2
  - antic
  - arb
- Hecke

For ease of use, you should also do

```
julia> Pkg.add("Revise")
```

at least if you want to develop/ change any of the above packages.

Then, you need to

`using Revise` if you want, and `using Hecke` once after starting julia. This will make everything in AbstractAlgebra, Nemo and Hecke available.