Regulator approximation and fundamental unit computation for real-quadratic orders

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

Dissertation

zur Erlangung des akademischen Grades
Doctor rerum naturalium (Dr.rer.nat.)

von

Dipl.-Inform. Markus Hartmut Maurer

aus Püttlingen (Saarland)

Referenten:	Prof. Dr. J. Buchmann
	Prof. Dr. N.P. Smart
Tag der Einreichung:	25.09.2000
Tag der mündlichen Prüfung:	13.11.2000

Darmstadt 2000

D 17

Acknowledgements

Professor Dr. Johannes Buchmann for being an interested, inspiring, and promoting supervisor. For accompanying me through my academic education over many years, supporting and demanding.

Professor Dr. Nigel P. Smart for accepting the task of the second referee.

My colleagues, especially:
Dr. Mike Jacobson, for successful team work.
Dr. Thomas Papanikolaou, for his enthusiasm.
Thomas Pfahler, for the pleasant work atmosphere.
Marita Skrobic, for her kindness, helpfulness, and competence.
Professor Dr. Edlyn Teske, for many helpful and interesting conversations.

My friends, grandmother, and parents, for always giving me their support and love.

Thank you.

List of symbols and notation

Symbols

		page
$a_i(n)$	Bach's weights for the Euler product	111
$B(x, \chi_\Delta)$	truncated Euler product	111
$b(r)$	b-value of the real number r	25
$C(n)$	error bound, approximation of $L(1, \chi_\Delta)$	111
$d(\alpha)$	denominator of the quadratic number α	19
$d(A)$	denominator of the quadratic ideal A	20
ERH	extended Riemann hypothesis	22
$gcd(a, b)$	greatest common divisor of integers a and b	17
$H(\alpha)$	height of the quadratic number α	19
h	class number of O	21
$I \sim J$	equivalent ideals I and J	21
K	field of fractions of O	19
$L_\Delta[u, v]$	$\exp((v + o(1))(\log \Delta)^u (\log \log \Delta)^{1-u})$	
$L_{l,i}$	lower bound on the distance of minima	76
Ln	Lenstra logarithm	21
ln	natural logarithm	
log	logarithm to base 2	
$L(1, \chi_D)$	the Dirichlet L-function at 1 with Kronecker symbol	22
$l(A)$	smallest positive integer in the ideal A	20
$\ell(n, \Delta)$	approximation to $\ln L(1, \chi_\Delta)$	111
$M(n)$	bit complexity for multiplying two n-bit integers	17
$Min(I)$	set of minima of the ideal I	21
$\mathbb{N}$	set of positive integers $\{1, 2, \ldots\}$	
$N(\alpha)$	norm of the quadratic number α	19
$N(A)$	norm of the quadratic ideal A	20
$O(f)$	big-O notation	17
O	quadratic order, real if not explicitly stated otherwise	19
O^*	units of O	20
$\mathbb{Q}$	set of rational numbers	
$\mathbb{R}$	set of real numbers	
R	regulator of O	20
$r(A)$	smallest positive rational number in the ideal A	20
$sign(r)$	$r \in \mathbb{R}$, $sign(r) = 1$, if $r \geq 0$, $sign(r) = -1$, if $r < 0$	
size	bit size of an object	17
$U_{l,i}$	upper bound on the distance of minima	76
$\mathbb{Z}$	set of integers	

Quadratic Orders

Throughout this thesis Δ always denotes a quadratic discriminant, i.e., $\Delta \in \mathbb{Z}$ with $\Delta \equiv 0, 1 \bmod 4$, not a square in $\mathbb{Z}$. By O we denote the corresponding quadratic order $O = \mathbb{Z} + \mathbb{Z}(\Delta + \sqrt{\Delta})/2$, and by K we denote the field of fractions of O. If not explicitely stated otherwise, Δ is assumed to be positive, i.e., O is a real quadratic order and K a real quadratic number field.

Algorithms

The algorithms of this thesis are presented in a C++ like notation.

We give an example. The class `quadratic_number_standard` has the member variables x, y, z, p, where x, y, z are integers and p is a pointer to an object of type `quadratic_order`. The class `quadratic_order` has a member function `discriminant()` that returns the discriminant of the quadratic order it represents.

In C++ the member function `multiply` of the class `quadratic_number_standard` that multiplies two objects a_i, $i = 1, 2$ of type `quadratic_number_standard` and stores the result into the object for which the function is called would be written as follows:

```
void
quadratic_number_standard::multiply( quadratic_number_standard a_1,
                                     quadratic_number_standard a_2)
 {
   bigint h = a_1.x * a_2.x + a_1.y * a_2.y * a_1.p->discriminant();

   y = a_1.x * a_2.y + a_1.y * a_2.x;
   x = h;
   z = a_1.z * a_2.z;

   this->normalize();
 }
```

When explaining algorithms or when proving their correctness we find it useful to have a name for the object whose method is called. Therefore, when describing an algorithm, we give a name, e.g. a, to the object, and we replace the class name reference, i.e. "`quadratic_number_standard::`", by the name of the object followed by a dot, i.e. "`a.`".

Using this convention the above function would be written as follows: Let a be an object of type `quadratic_number_standard`.

```
void a.multiply( quadratic_number_standard a_1,
                 quadratic_number_standard a_2)
 {
```

```
  bigint h = a_1.x * a_2.x + a_1.y * a_2.y * a_1.p.discriminant();

  y = a_1.x * a_2.y + a_1.y * a_2.x;
  x = h;
  z = a_1.z * a_2.z;

  a.normalize();
}
```

Note that we also use the name of the object inside the function instead of the `this` pointer. We also dereference a pointer using a dot instead of `->`. For example, we write `a.normalize()` instead of `this->normalize()`.

But as in C++, the variables which are not declared inside the function are the member variables of the object whose method is called. For example, x, y, and z used in this function are the variables from the representation (x, y, z, p) of a.

Contents

Introduction

In this thesis we study computational problems in a real quadratic order O. In particular, we study algorithms for computing the regulator R and the fundamental unit of O, for deciding equivalence of O-ideals, and for determining generators of principal O-ideals.

Those are some of the most difficult and important problems in computational number theory. They are closely related to the problem of solving the Pell equation and, more generally, the diophantine equation $aX^2 + bXY + cY^2 = n$ (see [**Bue89**], [**Hua82**], [**Lag80**]). Recently, the difficulty of solving those problems has also been used as the basis of the security of cryptographic protocols (see [**SBW91**], [**SBW94**], [**BBT94**], [**BMT96**], [**BMM00**], [**HP00**]).

The first algorithm for solving our problems was invented by Legendre, Lagrange, and Gauss ([**Gau86**]). It is based on the continued fraction algorithm but is rather inefficient. This method requires time $R\Delta^{o(1)}$ for computing the fundamental unit and an approximation of fixed precision to the regulator R, where Δ is the discriminant of O. In 1972 Shanks ([**Sha73**]) presented a more efficient algorithm. In the version of Biehl and Buchmann ([**BB94**]) this algorithm has running time $R^{1/2}\Delta^{o(1)}$. Experiments show that the regulator is very often of the order of magnitude $\Delta^{1/2}$ ([**Coh95**]). Then the algorithm takes time $\Delta^{(1/4)+o(1)}$. Lenstra and Schoof ([**Len82**], [**Sch82**]) presented an algorithm, whose running time is $\Delta^{(1/5)+o(1)}$ assuming the extended Riemann hypothesis (ERH). It is still exponential in the binary length of the discriminant. A subexponential algorithm was suggested by Buchmann, Abel, and Vollmer ([**Buc90**], [**Abe94**], [**Vol00**]). Its running time is $\exp((5\sqrt{3}/6 + o(1))(\log\Delta)^{1/2}(\log\log\Delta)^{1/2})$ assuming the ERH.

There is one serious problem with most of the algorithms mentioned above. Since the regulator is a transcendental number, they all use approximations to real numbers. However, the analysis of the algorithms does not determine the precision of approximation necessary for the algorithms to be correct. Therefore, the proofs of the correctness and the estimates for the running times of the algorithms are incomplete.

In this thesis we give complete descriptions, correctness proofs, and complexity analyses of the important algorithms for approximating regulators, computing fundamental units, deciding ideal equivalence, and computing generators of principal ideals of quadratic orders. We describe improvements for several algorithms. We also present an object oriented implementation of all algorithms including experimental results. Some of our algorithms have been used to implement cryptographic protocols ([**BMM00**], [**HP00**]).

We begin with the background material on the complexity of integer operations, continued fractions, and quadratic number fields in Chapter 1.

In Chapter 2 we develop a framework, the `xbigfloat` model, for dealing with roundoff errors in number theoretic computations. When using approximations to real numbers in number theoretic computations it is necessary to know exactly what the error of the approximation is. Unfortunately, the methods from numerical analysis cannot be used since they only determine the order of magnitude of the errors.

In Chapter 3 we describe details of the implementation of the `xbigfloat` model. We present an analysis of the implemented algorithms which shows how accurate intermediate results must be approximated such that the output is an approximation of a prescribed precision. For the known algorithms (e.g. [**Wil66**], [**Kog60**], [**Bre76**]) there is no such analysis.

In Chapter 4 we describe three different representations for elements of $\mathbb{Q}(\sqrt{\Delta})$. The first is the standard representation $(x + y\sqrt{\Delta})/z$ with integers x, y, z. This representation is not appropriate for the fundamental unit of O. For example, Lagarias [**Lag80**] shows that there is an infinite set of quadratic orders, such that the binary length of the fundamental unit is exponential in $\log \Delta$. Therefore, if we use the standard representation no polynomial time algorithm for computing the fundamental unit can exist. To circumvent this problem we follow Buchmann, Thiel, and Williams ([**BTW95**]) and introduce a power product representation, where the base elements are in standard representation and the exponents are integers. (We explain how to find a power product representation of the fundamental unit whose size is polynomial in $\log \Delta$ in Chapter 7.) We also describe a logarithm representation, which represents an element by its logarithm and the ideal, that is generated by the element.

In Chapter 5 we explain how to compute with fractional O-ideals. In particular we present an algorithm that given the fundamental unit and some generator of a principal O-ideal determines the smallest generator of that ideal which is greater than 1.

In Chapter 6 we present a technique that is important for the computation of the fundamental unit and for deciding equivalence of ideals: the computation of a minimum (see page 21 for a definition of minimum) of an ideal, whose logarithm is close to a given distance.

In Chapter 7 we use the algorithms of Chapter 6 to compute the fundamental unit from a regulator approximation. We prove that this can be done in time $O(M(\log \Delta) \log\log \Delta\, (\log \Delta)^2)$ (Proposition 7.2.1), where $M(n)$ is the time for multiplying two n-bit integers.

In Chapter 8 we describe another fundamental technique: the approximation of the series $L(1, \chi_\Delta)$ (see Section 1.5 for a definition). It is an important invariant of a quadratic order. For example, it is used in the subexponential algorithm to decide, whether the algorithm can terminate. Based on the ideas of Bach [**Bac95**] we develop an algorithm that computes an approximation L to $L(1, \chi_\Delta)$, such that $|L/L(1, \chi_\Delta) - 1| < 2^{-k}$ for an a priori given positive integer k. Assuming the ERH we prove that the running time of the algorithm is $O(4^k M(k + \log|\Delta|) \log(k + \log|\Delta|) \log^2|\Delta|)$ (Theorem 8.4.1). At the end of the chapter we describe how that algorithm and the analytic class number formula (see page 23) can be used to approximate the product of class number and regulator.

The problem of deciding equivalence of ideals and of computing a generator of a principal ideal is treated in Chapter 9. Here, we deal with the more general problem of computing a relative generator α of two ideals A and B, i.e. $A = \alpha B$, if such an α exists whose logarithm is below a given bound. An algorithm for solving this problem has been presented by Biehl and Buchmann [**BB94**]. We extend their work and describe an efficient implementation of that algorithm.

In Chapter 10 we show how to compute an approximation to the regulator, if an approximation to an integer multiple of the regulator is given.

Complete descriptions with correctness proofs of the continued fraction method ([**Gau86**]), of the algorithm of Shanks ([**Sha73**]) and the variant of Biehl and Buchmann ([**BB94**]), and of the method of Lenstra and Schoof ([**Len82**], [**Sch82**]) for approximating the regulator are presented in Chapter 11.

The subexponential method for approximating the regulator ([**Buc90**], [**Abe94**]) is treated in Chapter 12. The implementation of the algorithm is joint work with Michael Jacobson [**Jac99**]. One of our contributions is an algorithm for computing a unit, that generates the subgroup which is generated by a given set of units.

In the appendix we present running times and statistical data for the algorithms developed in this thesis. The algorithms have been implemented in LiDIA, a library for computational number theory ([**LiD**],[**Pap97**]).

CHAPTER 1

Background

1.1. The complexity of integer operations

For any two functions $f, g : \mathbb{N} \mapsto \mathbb{R}_{\geq 0}$, we write

$$f = O(g) \Leftrightarrow \exists c, n_0 \in \mathbb{N} : f(n) \leq cg(n) \forall n \geq n_0.$$

For an integer a, we set

$$\text{size}(a) = \lfloor \log |a| \rfloor + 2,$$

where log denotes the logarithm to base 2. We call size(a) the bit size of a. Let $a, b \in \mathbb{Z}$, and set $n = max\{\text{size}(a), \text{size}(b)\}$. Addition and subtraction of a and b can be done in time $O(n)$.

We assume, that the running time for the algorithm that multiplies a and b is $O(M(n))$. If we use the standard school method, then $M(n) = n^2$. With the algorithm of Schönhage and Strassen [**SS71**] we have $M(n) = n \log n \log \log n$. We assume that $M(n)$ satisfies

$$M(\log n) \log \log n \leq cM(n) \tag{1.1.1}$$

for some constant $c \in \mathbb{N}$ and and for all $n \geq n_0$ for some constant $n_0 \in \mathbb{N}$. We also assume that addition of a and b can be performed in time $O(M(n))$. In chapter 2 we will use algorithms described by Brent [**Bre76**]. Therefore, we also assume (see [**Bre76**][(1.1)])

$$M(\alpha n) \leq \beta M(n)$$

for some $0 < \alpha, \beta < 1$ and all sufficiently large n.

The greatest positive integer that divides a and b is called the greatest common divisor of a and b. It is denoted by $gcd(a, b)$. Schönhage has shown in [**Sch71**] that the $gcd(a, b)$ and a representation of it, i.e., integers $x, y \in \mathbb{Z}$ with

$$gcd(a, b) = xa + yb,$$

can be computed in time $O(M(n) \log n)$, where x and y satisfy

$$|x| \leq |b|, \quad |y| \leq |a|.$$

If we use quadratic time multiplication, $M(n) = n^2$, then the extended euclidean algorithm computes the gcd of a and b and a representation of it in time $O(n^2)$. The bounds on x and y are also valid in that case. See [**Buc99**] and [**BS96**] for details.

1.2. Continued fractions

Let $q_0, \dots, q_n \in \mathbb{Z}$ with $q_1, \dots, q_n \geq 1$, and $q_n \geq 2$, if $n > 0$. We call $[q_0, q_1, \dots, q_n]$ a *regular continued fraction.* It represents the rational number

$$q_0 + \cfrac{1}{q_1 + \cfrac{1}{\ddots \, \cfrac{1}{q_{n-1} + \cfrac{1}{q_n}}}}$$

For $0 \leq i \leq n$, the rational number represented by $[q_0, \dots, q_i]$ is called the i-th *convergent* of $[q_0, \dots, q_n]$. For $a, b \in \mathbb{Z}$, $b \neq 0$, there exists exactly one regular continued fraction, that represents a/b ([**BS96**][Theoren 4.5.5]). Hence, we may talk about *the* regular continued fraction, that represents a/b.

Let a, b be integers with $a \geq b > 0$. We describe how to compute the regular continued fraction, that represents a/b, and its convergents. Set

$$r_0 = a, r_1 = b,$$

and for $k \geq 2$, let r_k be the uniquely determined remainder, that is obtained from division with remainder by

$$r_{k-1} = q_k r_k + r_{k+1}, \; 0 \leq r_{k+1} < r_k.$$

It is $q_k = \lfloor r_{k-1}/r_k \rfloor$ for $1 \leq k \leq n$, where n is chosen, such that r_n is the last non-zero element of the sequence (r_k). The continued fraction $[q_1, \dots, q_n]$ represents a/b. Furthermore, we set $x_0 = 0$, $x_1 = 1$, $y_0 = 1$, $y_1 = 0$, and

$$\begin{array}{rcl} x_{k+1} & = & q_k x_k + x_{k-1}, \\ y_{k+1} & = & q_k y_k + y_{k-1}, \end{array} \quad 1 \leq k \leq n.$$

Then x_{k+1}/y_{k+1} is the k-th convergent of $[q_1, \dots, q_n]$ for $1 \leq k \leq n$, [**Per77**][§2, (12)]. The described algorithm is the extended euclidean algorithm, and in fact, we have $gcd(a, b) = (-1)^n x_n a + (-1)^{n+1} y_n b$, [**Buc99**][Section 1.9].

THEOREM 1.2.1. *Let $a, b \in \mathbb{Z}$ with $b \neq 0$. The regular continued fraction, that represents a/b, and all its convergents can be computed in time $O(max\{\text{size}(a), \text{size}(b)\}^2)$.*

PROOF. (sketch) We use the extended euclidean algorithm to compute the continued fraction and its convergents. For $a \geq b > 0$ the assertion follows from [**Buc99**][Theorem 1.10.1].

Suppose that $b > 0$ and $a < b$. Let $a = q_0 b + r$, $0 \leq r < b$, be the division with remainder. If $r = 0$, then $[q_0]$ is the regular continued fraction, that represents a/b. Otherwise, if $[q_1, \dots, q_n]$ is the regular continued fraction, that represents b/r with $q_1 \geq 1$, obtained by the euclidean algorithm above, then $[q_0, q_1, \dots, q_n]$ is the regular continued fraction, that represents a/b. The estimate on the running time can be obtained from the analysis given in [**Buc99**][Section 1.10], if we note that

$$|q_0| q_1 \cdots q_n \leq |a| + b \leq 2\, max\{|a|, b\}.$$

□

We need some properties of convergents of continued fractions. The first theorem describes, when a rational number is a convergent of a continued fraction.

THEOREM 1.2.2. *Let* $[q_1, \dots, q_n]$ *be a regular continued fraction that represents the rational number* q*, and let* $c, d \in \mathbb{Z}$ *with* $|q - c/d| < 1/(2d^2)$*, then* c/d *is a convergent of* $[q_1, \dots, q_n]$*.*

PROOF. [**Per77**][Satz 2.11]. $\square$

The next theorem states, that the approximation accuracies of convergents increase, and also the denominators of the convergents.

THEOREM 1.2.3. *Let* $[q_0, \dots, q_n]$ *be a regular continued fraction that represents the rational number* q*, and let* c_i *be the* i*-th convergent with* $c_i = a_i/b_i$*,* $a_i, b_i \in \mathbb{Z}$*, and* $gcd(a_i, b_i) = 1$*. Then* $|q - c_i| < |q - c_{i-1}|$ *for* $0 < i \leq n$*, and* $|b_i| > |b_{i-1}|$ *for* $1 < i \leq n$*.*

PROOF. Apply [**Per77**][II.§13 (5)] and [**Bue89**][Theorem 3.13]. $\square$

1.3. Quadratic orders, numbers, and ideals

We introduce the concepts of quadratic number fields which are used in this thesis. We refer to [**BS66**], [**Coh78**], [**Mol96**], and [**Coh95**] for a detailed description.

A *quadratic discriminant* is an integer $\Delta \in \mathbb{Z}$ with $\Delta \equiv 0, 1 \bmod 4$, that is not a square in $\mathbb{Z}$. For positive Δ we denote by $\sqrt{\Delta}$ the square root of Δ in $\mathbb{R}$, which is positive. Then

$$O = \mathbb{Z} + \frac{\Delta + \sqrt{\Delta}}{2}\mathbb{Z}$$

is the *quadratic order* of discriminant Δ. For $\Delta > 0$ the discriminant and the order are called *real quadratic*, because the order is a subset of the real numbers, and for $\Delta < 0$, they are called *imaginary quadratic*.

The *quadratic number field* $K = \mathbb{Q}(\sqrt{\Delta})$ is the field of fractions of O. Any element $\alpha \in K$ can be uniquely written as

$$\alpha = \frac{x + y\sqrt{\Delta}}{z},$$

with $x, y, z \in \mathbb{Z}$, $z > 0$, and $gcd(x, y, z) = 1$. This is the *standard representation* of a quadratic number α.

The following terms are given with regard to O. The *denominator* of α is defined as

$$d(\alpha) = min\{d \in \mathbb{Z}_{>0} | d \cdot \alpha \in O\}.$$

The *conjugate* of α is

$$\sigma(\alpha) = \frac{x - y\sqrt{\Delta}}{z},$$

and the *height* of α is

$$H(\alpha) = max\{|\alpha|, |\sigma\alpha|\}.$$

We call

$$N(\alpha) = \alpha\sigma(\alpha)$$

the *norm* of α.

For $\Delta > 0$ there is a *fundamental unit* η in the unit group of the ring O, i.e.,

$$1 < \eta \in O^*,$$

such that the unit group is generated by -1 and η, i.e.,

$$O^* = \{\pm\eta^k | k \in \mathbb{Z}\}.$$

The positive real number

$$R = \ln \eta$$

is called the *regulator* of O.

In the imaginary case, the unit group is $\{\pm 1\}$ for $\Delta < -4$, $\{\pm 1, \pm i\}$ for $\Delta = -4$, and $\{\pm 1, \pm i, (1 \pm \sqrt{-3})/2\}$ for $\Delta = -3$.

A subset $\{0\} \neq A \subseteq O$ that is an additve subgroup of O and satisfies

$$AO \subseteq O,$$

is called an *integral ideal* of O. A *fractional ideal* of O is a subset $A \subseteq K$ such that dA is an integral ideal of O for some $d \in \mathbb{Z}_{>0}$. We call the minimal d the *denominator* of A and denote it by $d(A)$. For an integral ideal A of O, the residue class group O/A is finite, and its index

$$N(A) = |O/A|$$

is called the *norm* of A. For a fractional O-ideal A we set $N(A) = N(d(A)A)/d(A)$. By $l(A)$ we denote the smallest positive integer in A, i.e.,

$$l(A) = min\{x | x \in A \cap \mathbb{Z}_{>0}\},$$

and $r(A)$ denotes the smallest positive rational number in A, i.e.,

$$r(A) = min\{x | x \in A \cap \mathbb{Q}_{>0}\},$$

For $\alpha \in K$, the set

$$\alpha O = \{\alpha\beta | \beta \in O\},$$

is a fractional O-ideal. It is called the *ideal generated by* α. We have

$$N(\alpha O) = |N(\alpha)|,\ d(\alpha O) = d(\alpha).$$

A fractional O-ideal A, that can be written as

$$A = \alpha O$$

for some $\alpha \in K$ is called *principal* ideal. The set of principal O-ideals is denoted by

$$P_\Delta = \{A | A \text{ principal } O - \text{ideal}\}.$$

The *product* of two fractional O-ideals A and B is the fractional O-ideal

$$AB = \{\sum_{(a,b)\in S} ab : S \subset A \times B \text{ finite}\}.$$

A fractional O-ideal A is *invertible* in O, if there exists another fractional O-ideal B such that

$$A\,B = O.$$

The set of all invertible O-ideals is denoted by

$$I_\Delta = \{A | A \text{ invertible } O - \text{ideal}\}.$$

Together with multiplication I_Δ is an abelian group, in which P_Δ is a subgroup of finite index. The quotient

$$Cl_\Delta = I_\Delta / P_\Delta$$

is called the *class group* of O and its order

$$h = |Cl_\Delta|$$

is the *class number* of O.

We introduce the notion of *equivalence of ideals*. Let I,J be two fractional ideals of O. I and J are said to be equivalent if there exists a number $\alpha \in K$ with $I = \alpha J$. We write

$$I \sim J.$$

For real quadratic orders, we call α a *minimum* of a fractional ideal I of O, if $\alpha > 0$, and if there is no non-zero number $\beta \in I$ such that $|\beta| < |\alpha|$ and $|\sigma\beta| < |\sigma\alpha|$. The set of all minima of I is denoted by

$$Min(I) = \{\alpha \in I | \alpha \text{ minimum of } I\}.$$

We call a real quadratic fractional O-ideal I *reduced*, if 1 is a minimum in I. All reduced ideals in the equivalence class of I are given by the minima of I. More precisely, it is (see [**BTW95**][Corollary 2.22])

$$\{J | J \sim I, J \text{ reduced}\} = \{I/\alpha | \alpha \in Min(I)\}. \tag{1.3.1}$$

1.4. Logarithm functions

Let O be a real quadratic order with field of fractions K. Let R be the regulator and η be the fundamental unit of O. A map

$$l : K^* \to \mathbb{R}$$

is called a *logarithm function* for O, if l is a homomorphic map from the multiplicative group $(K^*, \cdot)$ to the additive group $(\mathbb{R}, +)$, whose kernel contains -1 and which maps the fundamental unit to the regulator, i.e. $l(\eta) = R$.

Because O^* is generated by -1 and η, it follows that $l(O^*) = \mathbb{Z}R$. More precisely, if $\alpha \in O^*$ with $\alpha = s\eta^k$, $s \in \{\pm 1\}$, then $l(\alpha) = kR$.

We give two examples for logarithm functions of O. The first is the map

$$K^* \to \mathbb{R}, \alpha \mapsto \ln |\alpha|.$$

Its kernel is $\{\pm 1\}$, it is a homomorphic map, and the image of the fundamental unit is the regulator, so the map is a logarithm function.

The second example is the Ln function which has been introduced by Hendrik Lenstra, [**Len82**]. It is defined as

$$\begin{aligned} \mathrm{Ln} : \quad & K^* \to \mathbb{R}, \\ & \alpha \mapsto \mathrm{Ln}\, \alpha = (1/2) \ln |\alpha / \sigma(\alpha)|. \end{aligned}$$

The Ln map has the following properties.

LEMMA 1.4.1. *Let $\alpha \in K^*$.*

1. *$kernel(\mathrm{Ln}) = \mathbb{Q}^*$.*
2. *$\mathrm{Ln}\,\alpha = \ln|\alpha| - (1/2)\ln|N\alpha|$.*
3. *$\mathrm{Ln}\,\alpha = sign(x)sign(y)\,\mathrm{Ln}(|x| + |y|\sqrt{\Delta})$ for $\alpha = (x + y\sqrt{\Delta})/z$, $x, y, z \in \mathbb{Z}$.*

Proof. The assertion on the kernel is obvious. The second assertion is true, because $\operatorname{Ln}\alpha = 1/2 \ln|\alpha^2/(\alpha\sigma(\alpha))| = \ln|\alpha| - 1/2\ln|N\alpha|$.

We prove the third assertion. Set $s = sign(x)sign(y)$. We have

$$\operatorname{Ln}\alpha = \frac{1}{2}\ln\left|\frac{sign(x)|x| + sign(y)|y|\sqrt{\Delta}}{sign(x)|x| - sign(y)|y|\sqrt{\Delta}}\right| = \frac{1}{2}\ln\left|\frac{|x| + s|y|\sqrt{\Delta}}{|x| - s|y|\sqrt{\Delta}}\right|.$$

If $s = 1$, the assertion is proven. If $s = -1$, then

$$\operatorname{Ln}\alpha = \frac{1}{2}\ln\left|\frac{|x| - |y|\sqrt{\Delta}}{|x| + |y|\sqrt{\Delta}}\right| = s\frac{1}{2}\ln\left|\frac{|x| + |y|\sqrt{\Delta}}{|x| - |y|\sqrt{\Delta}}\right|.$$

This proves the assertion. □

The Ln map is homomorphic, because ln is. It follows from the second assertion of Lemma 1.4.1, that the image of the fundamental unit is the regulator of O. So the map is a logarithm function.

What is the advantage of the Ln function ? If α is a unit of O, then $\operatorname{Ln}\alpha = \ln|\alpha|$. Thus, if we are interested in approximating the regulator of O, we may use the Ln function instead of ln. Then we can neglect the denominators of the quadratic numbers to save computation time and storage. For example, this is used during the regulator computation with the subexponential method (see [**Jac99**][p.71]).

1.5. $L(1, \chi_\Delta)$ and the analytic class number formula

We introduce the Dirichlet L-function and the analytic class number formula. For a more detailed description we refer to [**MW92**][Section 4].

Let $s \in \mathbb{C}$ and χ be any Dirichlet character. The *(Dirichlet) L-function* is defined as

$$L(s, \chi) = \sum_{n=1}^{\infty} \chi(n)/n^s.$$

An important conjecture on the zeroes of $L(s, \chi)$ is the *extended Riemann hypothesis* (ERH). It states that

$$L(s, \chi) > 0,$$

whenever the real part of s exceeds 1/2. It is still unknown, whether the ERH is true. The assumption of the ERH allows a faster approximation of $L(1, \chi_\Delta)$ (see Chapter 8). Here χ_Δ denotes the Kronecker symbol (see [**Hua82**][12.3]), i.e., for a prime p we have

$$\chi_\Delta(p) = \begin{cases} 0, & \text{if } p|\Delta, \\ 1, & \text{if } p \nmid \Delta, \Delta \text{ square mod } 4p, \\ -1, & \text{if } p \nmid \Delta, \Delta \text{ non-square mod } 4p, \end{cases}$$

and if $0 < n = \prod_{r=1}^{v} p_r$ where p_r are primes, then

$$\chi_\Delta(n) = \prod_{r=1}^{v} \chi_\Delta(p_r).$$

The important property of the L-function for the computation of invariants of quadratic orders O is given by Dirichlet's analytic class number formula, that relates class number, regulator, and the L-function. For $\Delta < 0$ let w be the number of roots of unity in O, i.e. $w = 6$ for $\Delta = -3$, $w = 4$ for $\Delta = -4$, and $w = 2$ for

$D < -4$. If we denote by h the class number of O, then the *analytic class number formula* can be stated as follows

$$L(1,\chi_\Delta) = \begin{cases} 2h\pi/(w\sqrt{|\Delta|}), & \Delta < 0, \\ 2hR/\sqrt{\Delta}, & \Delta > 0. \end{cases} \tag{1.5.1}$$

We can use the class number formula to derive an upper bound on hR as follows. According to [**Hua42**][Lemma 5] we have

$$L(1,\chi_\Delta) < 1 + (\ln\Delta)/2. \tag{1.5.2}$$

It follows from (1.5.1) and (1.5.2) that

$$hR < (1 + (\ln\Delta)/2)\sqrt{\Delta}/2. \tag{1.5.3}$$

CHAPTER 2

Computing with approximations

2.1. Floating point numbers: The `xbigfloat` model

When using approximations to real numbers in number theoretic computations it is necessary to know exactly what the error of the approximation is. Unfortunately, the methods from numerical analysis cannot be used since they only determine the order of magnitude of the errors. In this chapter we present our framework developed in [**BM98**] for dealing with roundoff errors in number theoretic computations.

DEFINITION 2.1.1. Let $r \in \mathbb{R}$, $r \neq 0$. Then we set $b(r) = \lfloor \log |r| \rfloor + 1$. We also set $b(0) = 0$.

Note that for $r \in \mathbb{R}$, $r \neq 0$ we have

$$2^{b(r)-1} \leq |r| < 2^{b(r)}.$$

If r is of the form

$$r = 2^m \sum_{i=1}^{k} b_i 2^{-i}, \quad b_i \in \{0, 1\}, \quad b_1 = 1, m \in \mathbb{Z}, k \in \mathbb{Z}_{\geq 1},$$

then $b(r) = m$. This means that for a non zero integer m the value $b(m)$ is the length of the binary expansion of $|m|$. For $m \in \mathbb{Z}$ we have

$$\mathrm{size}(m) = b(m) + 1.$$

DEFINITION 2.1.2. A *floating point number* is a pair $f = (m, e)$ with $m, e \in \mathbb{Z}$, $m \neq 0$ or $m = 0$ and $e = 0$.

Let $f = (m, e)$ be a floating point number. Then m is called the *mantissa* of f and e is called the *exponent* of f. That floating point number represents the rational number

$$q = m 2^{e-b(m)}.$$

Note that $b(q) = e$. Frequently, we will identify f with the rational number q which is represented by f. We also set

$$\mathrm{size}(f) = \mathrm{size}(m) + \mathrm{size}(e).$$

Floating point numbers are implemented by the data type `xbigfloat`. If `f` is a floating point number then `f.mantissa()` is its mantissa and `f.exponent()` is its exponent.

2.2. Approximations

Next, we introduce relative and absolute approximations.

DEFINITION 2.2.1. Let $r \in \mathbb{R}$ and $k \in \mathbb{Z}$.

1. A *relative k-approximation* to r is a floating point number $f = (m, e)$ with $b(m) \leq k + 3$ and such that there exists an $\varepsilon \in \mathbb{R}$ with $f = r(1 + \varepsilon)$ and $|\varepsilon| < 2^{-k}$.
2. An *absolute k-approximation* to r is a floating point number $f = (m, e)$ such that $|f - r| < 2^{-k}$, and either $e \geq b(m) - k - 1$ or $f = (0, 0)$.

LEMMA 2.2.1. *If $r \in \mathbb{R}$ and $f = (m, e)$ is a floating point number then f is a relative k-approximation to r if and only if $r = 0$ and $(m, e) = (0, 0)$, or $r \neq 0$ and $|f/r - 1| < 2^{-k}$.*

We wish to interpret relative approximations as absolute approximations and vice versa. For this purpose we need a few results.

LEMMA 2.2.2. *Let $r, s \in \mathbb{R} \setminus \{0\}$, $k \in \mathbb{Z}$, and $|r/s - 1| < 2^{-k}$. Then*

1. $|r - s| < 2^{-k+b(s)}$
2. *if* $k \geq 1$ *then* $|b(r) - b(s)| \leq 1$.

PROOF. We have $|r - s| = |r/s - 1||s| < 2^{-k+b(s)}$. Also, $|r/s - 1| < 2^{-k}$ implies $1 - 2^{-k} < |r/s| < 1 + 2^{-k}$. If $k \geq 1$ then $|s|/2 < |r| < 2|s|$. This implies that $|b(r) - b(s)| \leq 1$. □

LEMMA 2.2.3. *Let $r, s \in \mathbb{R} \setminus \{0\}$ and $|r - s| < 2^{-k}$. Then*

1. $|r/s - 1| < 2^{-k-b(s)+1}$ *and*
2. *if* $k \geq -b(s) + 2$ *then* $|b(r) - b(s)| \leq 1$.

PROOF. We have $|r/s - 1| = |r - s|/|s| < 2^{-k-b(s)+1}$. If $k \geq -b(s) + 2$ then $k + b(s) - 1 \geq 1$. Hence Lemma 2.2.2 implies that $|b(r) - b(s)| \leq 1$. □

We estimate the size of approximations. For relative approximations this is easy. If $k \geq 1$ and $f = (m, e)$ is a relative k approximation to $r \in \mathbb{R}$ then we obtain from the definition and from Lemma 2.2.2

$$b(m) \leq k + 3, \quad |e| \leq |b(r)| + 1. \tag{2.2.1}$$

For absolute approximations we obtain the following result.

LEMMA 2.2.4. *Let $r \in \mathbb{R}$, $k \in \mathbb{Z}$. If $f = (m, e) \neq 0$ is an absolute k-approximation to r, then $b(m) \leq \max\{b(r) + k + 2, 2\}$. Furthermore, $|e| \leq |b(r)| + 1$, if $|r| \geq 2^{-(k+1)}$, and $|e| \leq |k| + 2$ otherwise.*

PROOF. Since $|f - r| < 2^{-k}$ it follows that $e = b(f) \leq \max\{b(r), -k\} + 1$. By definition $e \geq b(m) - k - 1$. This implies the assertion for $b(m)$.

Assume that $|r| \geq 2^{-(k+1)}$. Then $k \geq -b(r) + 2$ and applying Lemma 2.2.3 yields $|e| \leq |b(r)| + 1$. Assume that $|r| < 2^{-(k+1)}$. If $e > 0$, then $|e| = e \leq \max\{b(r) + k + 2, 2\} \leq -k + 2$. If $e < 0$, we obtain from the definition of an absolute approximation $|e| = -e \leq k + 1 - b(m) \leq k + 1$. Hence, $|e| \leq |k| + 2$. □

2.3. Operations on floating point numbers

We will now describe a few operations on floating point numbers.

The first one is `truncate`. Let $f = (m, e)$, $l \in \mathbb{Z}$. Then $\texttt{truncate}(f, l)$ returns $(0, 0)$, if $f = 0$ or $l \leq 0$. Otherwise, `truncate` deletes the last $b(m) - l$ bits in m. More precisely, $\texttt{truncate}(f, l)$ returns the floating point number (n, e), where $n = m$ for $l \geq b(m)$, and if $1 \leq l < b(m)$ and

$$m = \sum_{i=1}^{b(m)} m_i 2^{b(m)-i}$$

is the binary expansion of m then

$$n = \sum_{i=1}^{l} m_i 2^{l-i}.$$

Note that

$$|f - \texttt{truncate}(f, l)| < 2^{e-l}. \tag{2.3.1}$$

$\texttt{truncate}(f, l)$ takes time $O(\mathrm{size}(f) + \mathrm{size}(l))$.

We show how to use `truncate` to construct approximations.

Lemma 2.3.1. *Let $r \in \mathbb{R}$, $k \in \mathbb{Z}$, and let $f = (m, e)$ be a floating point number with $|f - r| < 2^{-k-1}$. Then $\texttt{truncate}(f, e + k + 1)$ is an absolute k-approximation to r.*

Proof. Let $g = \texttt{truncate}(f, e + k + 1)$. If $e + k + 1 \leq 0$, then $g = (0, 0)$. Because $e = b(f)$, we have

$$|g - r| \leq |f| + |f - r| < 2^{b(f)} + 2^{-k-1} < 2^{-k},$$

so g is an absolute k-approximation to r. If $e + k + 1 > 0$, we obtain from (2.3.1)

$$|g - r| \leq |g - f| + |f - r| < 2^{-k}.$$

Also, for $g = (n, e)$, we have $b(n) \leq e + k + 1$. So g is an absolute k-approximation to r. □

Lemma 2.3.2. *Let $r \in \mathbb{R}$, $k \in \mathbb{Z}_{\geq 1}$, and let $f = (m, e)$ be a floating point number with $f = r(1 + \varepsilon)$, $\varepsilon \in \mathbb{R}$, $|\varepsilon| < 2^{-k-1}$. Then $\texttt{truncate}(f, k + 3)$ is a relative k-approximation to r.*

Proof. Let $g = (n, e) = \texttt{truncate}(f, k + 3)$. If $r = 0$, then $g = f$ is a relative k-approximation to r. Otherwise, we obtain from (2.3.1) and Lemma 2.2.2

$$\frac{|g - r|}{|r|} \leq \frac{|g - f|}{|r|} + \frac{|f - r|}{|r|} < \frac{|g - f|}{|r|} + 2^{-k-1} \leq 2^{b(f)-b(r)-k-2} + 2^{-k-1} \leq 2^{-k}.$$

Furthermore, $b(n) \leq k + 3$. □

We now describe the basic operations for floating point numbers and estimate their bit complexity.

Let $f = (m, x)$ and $g = (n, y)$ be floating point numbers. Then we set

$$-f = (-m, x).$$

Next we define addition. We set

$$z = min\{x - b(m), y - b(n)\}, \quad p = 2^{x-b(m)-z} m + 2^{y-b(n)-z} n.$$

and

$$f + g = \begin{cases} (p, z + b(p)), & \text{if } p \neq 0, f \neq 0, g \neq 0, \\ (0,0), & \text{if } p = 0, \\ (m, x), & \text{if } g = 0, \\ (n, y), & \text{if } f = 0. \end{cases}$$

We also set

$$f - g = f + (-g).$$

Addition of floating point numbers is, in general, not associative. For example, if $f_1 = (1, -1)$, $f_2 = (-1, -1)$, $f_3 = (1, 0)$ then $(f_1 + f_2) + f_3 = f_3 = (1, 0)$ but $f_1 + (f_2 + f_3) = (1, -1) + (1, -1) = (2, 0)$. Note that the floating point numbers $(1, 0)$ and $(2, 0)$ are different representations of $1/2$. Addition can be made associative if the result is normalized in such a way that the the mantissa is odd. For efficiency reasons we do not do this.

The running time of addition and subtraction is $O(s + |x - y|)$, where $s = max\{\text{size}(f), \text{size}(g)\}$.

We now present an algorithm for adding several floating point numbers. The algorithm receives as input an array f of n floating point numbers and returns the floating point number

$$((((f[1] + f[2]) + f[3]) + f[4]) + \ldots + f[n]).$$

```
xbigfloat sum(array_of_xbigfloat f, int n)
{
  s = 0;
  for (i = 1; i <= n; i++) s = s + f[i];
  return s;
}
```

In the follwing theorem we use the notation $f[i] = f_i = (m_i, x_i)$, $1 \leq i \leq n$ and $s = (p, u)$.

THEOREM 2.3.3. *Computing* $(p, u) = \texttt{sum}(f, n)$ *takes time* $O(n(X + B + \log(n)))$, *where* $X := max\{|x_i| : 1 \leq i \leq n\}$, $B = max\{b(m_i) : 1 \leq i \leq n\}$. *Also,* $b(p) \leq 2X + B + b(n)$ *and* $|u| \leq 3X + 2B + b(n)$.

PROOF. Let $s_0 = (0, 0)$ and $s_i = (p_i, u_i)$ be the value of s after the ith iteration of the for loop. First assume that $s_i, f_i \neq 0$, $1 \leq i \leq n$. Set $y_i = x_i - b(m_i)$, $v_i = u_i - b(p_i)$, $1 \leq i \leq n$, and $v = u - b(p)$. Then $v_1 = y_1$ and

$$v_{i+1} = \min\{v_i, y_{i+1}\}, \quad 1 \leq i < n. \tag{2.3.2}$$

This implies that

$$v = \min\{y_i : 1 \leq i \leq n\}. \tag{2.3.3}$$

If $X = \max\{|x_i| : 1 \leq i \leq n\}$ and $B = \max\{b(m_i) : 1 \leq i \leq n\}$ then

$$|v| \leq X + B. \tag{2.3.4}$$

Also,

$$|p| = |\sum_{i=1}^{n} m_i 2^{y_i - v}| < |\sum_{i=1}^{n} 2^{x_i - v}|.$$

Together with (2.3.4) we obtain

$$b(p) \leq b(n) + 2X + B.$$

Together with (2.3.4) this also implies the bound for $|u|$. It is easy to see that those bounds are also valid for each $b(p_i)$ and $|u_i|$, $1 \leq i \leq n$, even if some of the p_i or m_i are zero. This implies the bound for the running time. $\square$

The product $f \cdot g$ of the floating point numbers f and g is defined as follows.

$$f \cdot g = \begin{cases} (m \cdot n, x + y + b(mn) - b(m) - b(n)), & \text{if } f \neq 0, g \neq 0, \\ (0,0), & \text{otherwise.} \end{cases}$$

Multiplication takes time $O(M(t) + \text{size}(e) + \text{size}(f))$, where $t = max\{\text{size}(m), \text{size}(n)\}$.

Let $k \in \mathbb{Z}_{\geq 1}$ and $g \neq 0$. The function $\texttt{divide}(f, g, k)$ returns a relative k-approximation to the quotient f/g. Applying [**Bre76**][Lemma 2.2] and Lemma 2.3.2 we find that the running time of $\texttt{divide}(f, g, k)$ is $O(M(k) + s)$, where $s = max\{\text{size}(f), \text{size}(g)\}$.

We explain the extraction of square roots. Brent [**Bre76**](see also [**Sch90**]) presents an algorithm which computes a relative k-approximation to $\sqrt{f}$ where $f \geq 1/2$. This algorithm has running time $O(M(k) + \text{size}(f))$. Suppose that the function $\texttt{sqrt\_Brent}(f, k)$ implements that algorithm. Then we define for an arbitrary positive floating point number $f = (m, e)$ the function

$$\texttt{sqrt}(f, k) = \begin{cases} (0,0), & \text{if } f = 0, \\ \texttt{sqrt\_Brent}((m,0), k) \cdot 2^{e/2}, & \text{if } e \text{ is even,} \\ \texttt{sqrt\_Brent}((m,1), k) \cdot 2^{(e-1)/2}, & \text{if } e \text{ is odd,} \end{cases}$$

that computes a relative k-approximation to $\sqrt{f}$ in time $O(M(k) + \text{size}(f))$.

2.4. Approximating the logarithm

We discuss the complexity of computing approximations to the logarithm of a floating point number.

Brent [**Bre76**] (see also [**Sch90**]) presents an algorithm which computes a relative k-approximation to $\ln f$ where k is a positive integer and f is a positive floating point number with $1/2 \leq f \leq 2$. This algorithm has running time $O(M(k) \log k + \text{size}(f))$. Suppose that the function $\texttt{log\_Brent}(f, n)$ implements that algorithm. Then a function $\texttt{log}(f, k)$ which for any floating point number f and positive integer k computes an absolute k-approximation to $\ln f$ works as follows.

```
xbigfloat log(xbigfloat f, int k)
{
  xbigfloat f1, l1, l2, l;
  int e;
  f1.assign(f.mantissa(), 0);
  l1 = log_Brent(f1,k+2);
  e  = f.exponent();
  l2 = log_Brent(2,b(e)+k+2);
  l  = l1 + e * l2;
  return (Truncate(l, l.exponent()+k+1));
}
```

THEOREM 2.4.1. *If $f = (m, e)$ is a floating point number and $k \in \mathbb{Z}_{\geq 1}$ then* **log**(f, k) *returns an absolute k-approximation to $\ln f$ in time $O(M(k + \operatorname{size}(e))log(k + \operatorname{size}(e)) + \operatorname{size}(f))$.*

PROOF. We use the notation of the procedure `log`. Let $f = (m, e) > 0$ be a positive floating point number and $k \in \mathbb{Z}_{\geq 0}$. We have $f1 = (m, 0) = m2^{-b(m)}$. Hence $1/2 \leq f1 < 1$. Therefore, $l1 = \mathtt{log\_Brent}(f1, k + 2)$ is a relative $k + 2$-approximation to $\ln f1$. Since $1/2 \leq f1 < 1$, hence $b(\ln f1) \leq 0$, we obtain $|l1 - \ln f1| < 2^{-k-2}$. Because $l2$ is a relative $b(e) + k + 2$ approximation to $\ln 2$, we have $|e * l2 - e * \ln 2| < 2^{-k-2}$. It follows from Lemma 2.3.1 that l is an absolute k-approximation to $\ln f$.

It follows from (2.2.1) that $e * l2$ can be computed in time $O(M(k + \operatorname{size}(e)))$, which dominates the time for determining $l = l1 + e * l2$. The overall running time can be deduced from Brent's theorem. □

The next goal is to describe how the logarithm of a positive real number can be approximated when r itself is only known approximately. We need the following auxiliary results.

LEMMA 2.4.2. *Let $x \in \mathbb{R}$ with $|x| < 1$ Then*

$$\frac{x}{1+x} \leq \ln(x+1) \leq x.$$

PROOF. We have

$$\ln(1+x) = x - (\frac{x^2}{2} - \frac{x^3}{3}) - (\frac{x^4}{4} - \frac{x^5}{5}) - \ldots \leq x.$$

This proves the upper bound. To prove the lower bound consider the function

$$f(x) = \ln(1+x) - \frac{x}{1+x}.$$

Then

$$f'(x) = \frac{1}{1+x} - \frac{1}{(1+x)^2}.$$

Therefore, $f'(x) \geq 0$ for $x \geq 0$ and $f'(x) \leq 0$ for $x < 0$. Since $f(0) = 0$ this proves the lower bound. □

COROLLARY 2.4.3. *Let $x \in \mathbb{R}$ with $|x| < 1$ Then*

$$\frac{|x|}{1+|x|} \leq |\ln(x+1)| \leq \frac{|x|}{1-|x|}.$$

LEMMA 2.4.4. *Let $l, f, r \in \mathbb{R}$, $f, r > 0$ and $k \in \mathbb{Z}_{\geq 1}$. If $|f/r - 1| < 2^{-k-2}$ and $|l - \ln f| < 2^{-k-1}$ then $|l - \ln r| < 2^{-k}$.*

PROOF. Write $f = r(1 + \varepsilon)$ with $\varepsilon \in \mathbb{R}$. Then $|\varepsilon| < 2^{-k-2}$ and

$$|l - \ln r| \leq |l - \ln f| + |\ln f - \ln r| < 2^{-k-1} + |\ln(1+\varepsilon)|.$$

Applying Corrollary 2.4.3 with $x = \varepsilon$ yields $|\ln(1+\varepsilon)| < 2^{-k-1}$. This proves the assertion. □

LEMMA 2.4.5. *Let $r \in \mathbb{R}_{>0}$ and $f = (m, e)$ be an absolute k-approximation to $\ln r$. Then $|e|, b(m) \leq b(1 + |b(r)|) + |k| + 2$.*

PROOF. We have $|\ln r| = |\ln((r/2^{b(r)})\,2^{b(r)})| \le 1 + |b(r)|$. This implies

$$b(\ln r) \le b(1 + |b(r)|). \tag{2.4.1}$$

Using (2.4.1) and Lemma 2.2.4 proves the assertion for $b(m)$. Suppose that $|\ln r| < 2^{-k-1}$. Then Lemma 2.2.4 implies the assertion for $|e|$. If $|\ln r| \ge 2^{-k-1}$, we obtain $b(\ln r) \ge -k$. Together with (2.4.1) this implies $|b(\ln r)| \le b(1 + |b(r)|) + |k|$. In this case, Lemma 2.2.4 shows that $|e| \le b(1 + |b(r)|) + |k| + 1$. ☐

Next we present the procedure `b_of_ln` that on input of a floating point number $r > 0$ computes $l, u \in \mathbb{Z}$ such that $l \le b(\ln r) \le u$.

```
r.b_of_ln(int & l, int & u)
 {
   if (r >= 2)
    {
      l = b(b(r)-1) - 2;
      u = b(b(r));
    }
   else
    {
      l = b(r-1)-b(1+|r-1|)-1;
      u = b(r-1)-b(1-|r-1|)+1;
    }
 }
```

The correctness of the procedure is an immediate consequence of the following Lemma.

LEMMA 2.4.6. *Let* $r \in \mathbb{R}_{>0}$. *Then*

$$b(b(r) - 1) - 2 \le b(\ln r) \le b(b(r)), \text{ if } 2 \le r,$$
$$b(r - 1) - b(1 + |r - 1|) - 1 \le b(\ln r) \le b(r - 1) - b(1 - |r - 1|) + 1, \text{ if } 0 < r < 2.$$

PROOF. We prove the assertion for $r \ge 2$, i.e., $2^{b(r)-1} \le r < 2^{b(r)}$. We have $|\ln r| = \ln r < \ln 2^{b(r)} < b(r) < 2^{b(b(r))}$ and so $b(\ln r) \le b(b(r))$. And similarly $|\ln r| = \ln r \ge \ln 2^{b(r)-1} > (b(r) - 1)/2 = |(b(r) - 1)/2| \ge 2^{b(b(r)-1)-2}$, so $b(\ln r) \ge b(b(r) - 1) - 2$.

Now assume that $0 < r < 2$. Then we write $r = 1 + x$ with $|x| < 1$. By Lemma 2.4.3 we have $|x|/(1 + |x|) \le |\ln(1 + x)| \le |x|/(1 - |x|)$. This implies that $2^{b(x)-1-b(1+|x|)} < |\ln(1 + x)| < 2^{b(x)-b(1-|x|)+1}$. Substituting $x = r - 1$ proves the assertion. ☐

2.5. Approximating the exponential function

To approximate $\exp f$ for a floating point number $f = (m, e)$ we approximate $\exp(m2^{-b(m)})$ and raise the result to the power 2^e.

Brent [**Bre76**] (see also [**Sch90**]) describes an algorithm that, for a floating point number f with $1/2 \le f \le 1$ and an integer $k \ge 1$, computes a relative k-approximation to $\exp f$ in time $O(M(k)\log(k) + \mathrm{size}(f))$. Assume that the function `exp_Brent`(f, k) implements that algorithm.

Let $k \in \mathbb{Z}_{\ge 1}$ and f be a floating point number. The following function `exp`(f, k) computes a relative k-approximation to $\exp f$:

```
xbigfloat exp(xbigfloat f, int k)
{
  e = f.exponent();
  m = f.mantissa();

  if (m < 0)
   {
     m = -m;
     l = k + 4;
   }
  else
    l = k;

  if (e == 0)
     g = exp_Brent ((m,0), l);
  else
    {
      if (e > 0)
        {
          g = exp_Brent ((m,0), e+l+b(e+1)+3);
          for (i=1; i <= e; i++)
            g = Truncate (g*g, e+l+b(e+1)+3-i);
        }
      else
        {
          e = -e;
          g = exp_Brent ((m,0), l+3);
          for (i=1; i <= e; i++)
            g = Sqrt (g, l+3);
        }
      g = Truncate (g,l+3);
    }

  if (f.mantissa() < 0)
    {
      g = Divide (1, g, k+3);
      g = Truncate (g, k+3);
    }

  return g;
}
```

To be able to analyze exp we need the following results.

LEMMA 2.5.1 (Bernoulli). *Let* $r, \varepsilon \in \mathbb{R}$, $r > 1$, $\varepsilon > -1$, $\varepsilon \neq 0$. *Then* $(1+\varepsilon)^r > 1 + r\varepsilon$.

PROOF. [**Kön84**, p. 67] $\square$

LEMMA 2.5.2. *Let $\varepsilon \in \mathbb{R}_{>0}$, $r \in \mathbb{Z}_{\geq 1}$ with $r\varepsilon < 1$. Then*

$$(1+\varepsilon)^r < 1 + \frac{r\varepsilon}{1-r\varepsilon}.$$

PROOF. $(1+\varepsilon)^r = \sum_{i=0}^{r} \binom{r}{i}\varepsilon^i = 1 + r\varepsilon \sum_{i=0}^{r-1}(r\varepsilon)^i < 1 + r\varepsilon/(1-r\varepsilon)$. $\square$

LEMMA 2.5.3. *Let $\varepsilon \in \mathbb{R}$ with $|\varepsilon| < 1/2$ then*

$$1 - |\varepsilon| \leq \frac{1}{1+\varepsilon} \leq 1 + 2|\varepsilon|.$$

PROOF. If $\varepsilon \geq 0$ then $1 \geq 1 - \varepsilon^2 = (1-|\varepsilon|)(1+\varepsilon)$ implies $1 - |\varepsilon| \leq 1/(1+\varepsilon)$. Also, $1/(1+\varepsilon) \leq 1 \leq 1 + 2\varepsilon$.

If $\varepsilon < 0$ then $1-|\varepsilon| \leq 1 \leq 1/(1-|\varepsilon|) = 1/(1+\varepsilon) = 1+|\varepsilon|/(1-|\varepsilon|) \leq 1+2|\varepsilon|$. $\square$

THEOREM 2.5.4. *If $f = (m, e)$ is a floating point number and $k \in \mathbb{Z}_{\geq 1}$ then* $\exp(f, k)$ *returns a relative k-approximation to* $\exp f$ *in time $O(M(k + |e|)log(k + |e|) + \mathrm{size}(f))$.*

PROOF. First, we show that after the second if-else statement g is a relative l approximation to $\exp|f|$.

Suppose that $e > 0$ and let $g_0 = \texttt{exp\_Brent}((m, 0), e{+}l{+}b(e{+}1){+}3)$. Denote by g_i the value of g after the ith iteration of the first for loop. Write $g_i = g_{i-1}^2(1 + \varepsilon_i)$, with $\varepsilon_i \in \mathbb{R}$, $1 \leq i \leq e$ and $g_i = (n_i, x_i)$, $0 \leq i \leq e$. Then $|\varepsilon_i| < 2^{-l-e-b(e+1)+i-3}$, $0 \leq i \leq e$. So we obtain from Lemma 2.5.2

$$\begin{aligned} g_e &= \exp|f| \prod_{i=0}^{e}(1+\varepsilon_i)^{2^{e-i}} < \exp|f| \prod_{i=0}^{e}(1 + 2^{-l-e-b(e+1)+i-3})^{2^{e-i}} \\ &< \exp|f|(1 + 2^{-l-b(e+1)-2})^{e+1} < \exp|f|(1 + 2^{-l-1}). \end{aligned}$$

In the same way one can use Lemma 2.5.1 to show that $g_e > \exp|f|(1 - 2^{-l-1})$. Hence, $\texttt{truncate}(g_e, l+3)$ is a relative l-approximation to $\exp|f|$ by Lemma 2.3.2. By Lemma 2.2.2 we know that $|x_0| \leq 3$ and therefore, $|x_i| < 2^{i+2}$. Hence, each product takes time $O(M(l + e))$ and g_e can be computed in time $O(M(e + l)[e + \log(e + l)] + \mathrm{size}(f))$.

Now, suppose that $e < 0$ and let $g_0 = \texttt{exp\_Brent}((m, 0), l{+}3)$. Denote by g_i the value of g after the ith iteration of the second for loop. Write $g_i = \sqrt{g_{i-1}}(1 + \varepsilon_i)$, with $\varepsilon_i \in \mathbb{R}$, $1 \leq i \leq e$. Then $|\varepsilon_i| < 2^{-l-3}$, $0 \leq i \leq e$. So we obtain from Lemma 2.5.2

$$g_e = \exp|f| \prod_{i=0}^{|e|}(1+\varepsilon_i)^{2^{-|e|+i}} < \exp|f|(1 + 2^{-l-3})^{\sum_{i=0}^{|e|} 1/2^i} < \exp|f|(1 + 2^{-l-1}).$$

In the same way one can use Lemma 2.5.1 to show that $g_e > (1 - 2^{-l-1})$. Hence, $\texttt{truncate}(g_e, l + 3)$ is a relative l-approximation to $\exp|f|$ by Lemma 2.3.2.

Using Lemma 2.2.2, we can easily derive that for each g_i, $0 \leq i \leq |e|$, we obtain $|b(g_i)| \leq 3$. Then (2.2.1) implies that $\mathrm{size}(g_i) = O(l)$ and therefore, each square root can be computed in time $O(M(l))$. Hence, determining g_e takes time $O(M(l)[|e| + \log(l)] + \mathrm{size}(f))$.

Now, for $f > 0$ the assertions are proven. Suppose that $f < 0$. Then $l = k + 4$ and we obtain for $g = \mathtt{divide}(1, g_e, k + 3)$

$$g = (\exp f)\frac{1+\varepsilon_2}{1+\varepsilon_1}, \quad |\varepsilon_1| < 2^{-k-4}, |\varepsilon_2| < 2^{-k-3}.$$

From Lemma 2.5.3 we obtain

$$\frac{1+\varepsilon_2}{1+\varepsilon_1} \leq (1+\varepsilon_2)(1+2\varepsilon_1) < (1+2^{k-3})^2 < 1+2^{k-1}$$

and

$$\frac{1+\varepsilon_2}{1+\varepsilon_1} \geq (1-|\varepsilon_2|)(1-|\varepsilon_1|) > (1-2^{k-3})^2 > 1-2^{k-2}.$$

□

LEMMA 2.5.5. *Let $r \in \mathbb{R}$. It is* $\exp(x) \geq 1 + x$ *and if* $0 \leq x \leq 1/2$*, then* $\exp(x) \leq 1 + 2x$*.*

PROOF. For $c \in \{1, 2\}$ let $f_c(x) = \exp(x) - 1 - cx$. We have $f_c(0) = 0$ and $f_c'(x) = \exp(x) - c$. Then $f_1'(x) \geq 0$ if and only if $x \geq 0$. Hence $f_1(x) \geq 0$ and so $\exp(x) \geq 1 + x$. Similarly, we have $f_2'(x) < 0$ for $0 \leq x \leq 1/2$. This implies $f_2(x) \leq 0$, hence $\exp(x) \leq 1 + 2x$ for $0 \leq x \leq 1/2$. □

LEMMA 2.5.6. *Let $x \in \mathbb{R}$ with $|x| \leq 1/2$. Then* $\exp(x) = 1 + \varepsilon$ *with* $\varepsilon \in \mathbb{R}$*,* $|\varepsilon| \leq 2|x|$*.*

PROOF. Let $\varepsilon \in \mathbb{R}$ with $\exp(x) = 1 + \varepsilon$. If $-1/2 \leq x < 0$, we have $\varepsilon < 0$ and Lemma 2.5.6 yields $1 + \varepsilon = \exp(x) \geq 1 + x$. This implies $|\varepsilon| = -\varepsilon \leq -x = |x|$. Similarly, if $0 \leq x \leq 1/2$, we have $\varepsilon \geq 0$ and by Lemma 2.5.6 $1+\varepsilon = \exp(x) \leq 1+2x$. Hence, $|\varepsilon| = \varepsilon \leq 2x = 2|x|$. □

2.6. Approximating roots and powers

In this section we present the procedure power that on input of floating point numbers r, n, m with $r > 0$, $m \neq 0$, and an integer $k \geq 1$ computes a relative k approximation to $r^{n/m}$. This is done by computing a relative k approximation to $\exp(n/m \ln r)$.

```
xbigfloat power (xbigfloat r, xbigfloat n, xbigfloat m, int k)
 {
   int l, u, k1, k2;
   xbigfloat x,y,z;

   if (r == 1)
     return 1;

   r.b_of_log(l,u);
   k1 = max( k+b(n)-b(m)+6, -u );
   k2 = k+b(n)-b(m)+6+u;

   x = log(r,k1);
   y = divide(n,m,k2);
   z = exp(x*y,k+3);
```

```
  return truncate(z,k+3);
}
```

LEMMA 2.6.1. *On input of floating point numbers r, n, m, with $r > 0$, $m \neq 0$, and an integer $k \geq 1$, the procedure* `power` *returns a relative k approximation to $r^{n/m}$.*

PROOF. We use the notation of the procedure. For $r = 1$ the function is obviously correct. Let us assume that $r \neq 1$. The function computes an approximation $x \in \mathbb{R}$ to $\ln r$ such that $x - \ln r = \varepsilon_1$ with $|\varepsilon_1| < 2^{-k_1}$. Hence,

$$x = \ln(r)(1 + \varepsilon_1/\ln r),\ |\varepsilon_1| < 2^{-k_1}.$$

Furthermore, it determines $y \in \mathbb{R}$ with

$$y = n/m(1 + \varepsilon_2),\ |\varepsilon_2| < 2^{-k_2}.$$

This implies that $xy - n/m \ln r = n/m(\varepsilon_1 + \varepsilon_1\varepsilon_2 + \varepsilon_2 \ln r)$. We have $|xy - n/m \ln r| < |n/m|(2^{-k-b(n)+b(m)-6} + 2^u 2^{-k-b(n)+b(m)-6-u} + 2^{-k-b(n)+b(m)-6-u}|\ln r|)$. Hence

$$xy - n/m \ln r = \varepsilon_3,\ |\varepsilon_3| < 2^{-k-3}.$$

It follows that we obtain $z = \exp(xy)(1 + \varepsilon_5) = \exp(n/m \ln r)\exp(\varepsilon_3)(1 + \varepsilon_5)$ with $|\varepsilon_5| < 2^{-k-3}$. Because $k \geq 1$, Lemma 2.5.6 yields $\exp(\varepsilon_3) = 1 + \varepsilon_4$ with $|\varepsilon_4| \leq 2|\varepsilon_3| < 2^{-k-2}$. So finally

$$\begin{aligned} z &= \exp(n/m \ln r)(1 + \varepsilon_4)(1 + \varepsilon_5) \\ &= \exp(n/m \ln r)(1 + \varepsilon) \end{aligned}$$

with $|\varepsilon| < 2^{-k-1}$, because $k \geq 1$. By Lemma 2.3.2 the truncated z is a relative k approximation to $\exp(n/m \ln r)$. □

2.7. The IEEE-754 floating point model

In our `xbigfloat` model for floating point numbers described in the previous sections, the mantissa and the exponent of a floating point number are multi-precision integers. We have implemented this model as a C++ class. The C++ language also provides a built-in type for floating point numbers, the data type `double`. In contrast to `xbigfloat`, floating point numbers of type `double` have a fixed length for the size of the exponent and mantissa. But, because this is a built-in data type, programs using `double` are much faster in practice than those using `xbigfloat`. In most of our applications, the speed of the algorithms implemented with `xbigfloat` is good enough in practice and the accuracy of the approximations must be high. But in some parts, the `xbigfloat` efficiency is not sufficient and inputs that are relevant in practice only require so little accuracy such that the approximations can be stored in variables of type `double`.

A very common implementation of the data type `double` that is available on most platforms is based on the IEEE Standard for Binary Floating-Point Arithmetic (cf. [**ANS85**]). We will use that standard when we switch from our model of `xbigfloat` to floating point numbers with fixed lengthes. In the following we will briefly describe the basic facts of the IEEE model and analyze the roundoff error in such a model for those operations that are relevant to our applications. For a detailed description of the model we refer the reader to the standard [**ANS85**].

An IEEE-754 `double` is a string consisting of 64 bits which has the following format:

FIGURE 1. IEEE double format

The value v of such a string is defined as follows:

1. If $e = 2047$ and $f \neq 0$, then v is NaN regardless of s.
2. If $e = 2047$ and $f = 0$, then $v = (-1)^s\infty$.
3. If $0 < e < 2047$, then $v = (-1)^s 2^{e-1023}(1.f)$.
4. If $e = 0$ and $f \neq 0$, then $v = (-1)^s 2^{e-1022}(0.f)$.
5. If $e = 0$ and $f = 0$, then $v = (-1)^s 0$.

Here $\pm\infty$ and NaNs are reserved values used to indicate, e.g., over-/underflow and exceptions. The number of bits in f plus 1 is the precision p, i.e. $p = 53$. Furthermore, the standard defines $E_{max} = 1023$.

The default rounding mode is round to nearest. "In this mode the representable value nearest to the infinitely precise result shall be delivered.... However, an infinitely precise result with magnitude at least $2^{E_{max}}(2 - 2^{-p})$ ($= 2^{1023}(2 - 2^{-53})$) shall round to ∞ with no change in the sign" ([**ANS85**][4.1]). For our purposes we always assume that the mode round to nearest is chosen.

Furthermore, "...., each of the operations shall be performed as if it first produced an intermediate result correct to infinite precision and with unbounded range, and then coerced this intermediate result to fit in the destination's format" ([**ANS85**][5]), according to the rounding mode and the exception handling. An underflow exception is created when, e.g., before rounding, a nonzero result computed as though both the exponent range and the precision were unbounded would lie strictly between $\pm 2^{-1022}$ (see [**ANS85**][7.4]).

In the following we switch between floating point numbers in double format and the value they represent. For a non-zero $x \in \mathbb{R}$ let

$$x = (-1)^s 2^E \sum_{i=0}^{\infty} f_i 2^{-i}, \tag{2.7.1}$$

$f_0 = 1$, $f_i \in \{0, 1\}$ for $i \geq 1$, $s \in \{0, 1\}$, and $E \in \mathbb{Z}$. We define the function

(2.7.2)

$$round\_to\_nearest(x) = \begin{cases} (0,0,0) & \text{if } x = 0. \\ (s, E+1023, (f_1 \ldots f_{52})_2) & \text{if } x \neq 0, f_{53} = 0. \\ (s, E+1023, (f_1 \ldots f_{52})_2 + 1) & \begin{array}{l}\text{if } f_{53} = 1 \text{ and} \\ f_1 = 0 \vee \ldots \vee f_{52} = 0.\end{array} \\ (s, E+1024, (\underbrace{0 \ldots 0}_{52})_2) & \begin{array}{l}\text{if } f_{53} = 1 \text{ and} \\ f_1 = \ldots = f_{52} = 1.\end{array} \end{cases}$$

LEMMA 2.7.1. *Let $x \in \mathbb{R}$ with $2^{-1022} \leq |x| < 2^{1023}(2 - 2^{-53})$. Then $round\_to\_nearest(x)$ is a floating point number in IEEE-754 double format with $round\_to\_nearest(x) = x(1 + \varepsilon)$, $|\varepsilon| < 2^{-53}$.*

PROOF. Let $z = round\_to\_nearest(x)$. If $x = 0$, then z is in double format and $z = x$. For the rest of the proof assume that $x \neq 0$ and let x be represented as in (2.7.1).

We have $2^{-1022} < 2^{E+1}$. Hence, $E > -1023$. Furthermore, we have $2^E \leq |x| < 2^{1024}$. Together we obtain $0 < E + 1023 < 2047$. Also in case of $f_1 = \ldots = f_{53} = 1$, we have $(2 - 2^{-53}) \leq |x|/2^E$. This implies $E < 1023$ and therefore $0 < E + 1024 < 2047$. This shows that z is a floating point number in `double` format.

Now we prove the size of the relative error of z. Set $f_0 = 1$. Then $x = (-1)^s 2^E \sum_{i=0}^{\infty} f_i 2^{-i}$.

If $f_{53} = 0$, then $z = (-1)^s 2^E \sum_{i=0}^{52} f_i 2^{-i}$. This implies $|x - z| \leq 2^E \sum_{i=54}^{\infty} f_i 2^{-i}$. Hence $|x - z| < |x| 2^{-53}$.

If $f_{53} = 1$ and $f_i = 0$ for some $1 \leq i \leq 52$, then $z = (-1)^s 2^E \sum_{i=0}^{52} f_i 2^{-i} + 2^{-52}$. This yields $|x - z| \leq 2^E |2^{-53} + \sum_{i=54}^{\infty} f_i 2^{-i} - 2^{-52}| \leq 2^E 2^{-53}$. Hence, $|x - z| < |x| 2^{-53}$.

In the last case, it is $f_i = 1$ for all $1 \leq i \leq 53$. Then $z = (-1)^s 2^{E+1} = (-1)^s 2^E \sum_{i=0}^{\infty} 2^{-i}$, and again $|x - z| = 2^E | \sum_{i=54}^{\infty} 2^{-i}| < |x| 2^{-53}$. □

THEOREM 2.7.2. *For the operations $+, -, *, /, sqrt, log, exp$ let x be the result of the operation correct to infinite precision and with unbounded range and let z be the coercion of x into double format rounded to nearest and according to the rules for exception handling. Assume that there is no division by zero error, that the argument of the logarithm function is greater than zero, and that the argument of the square root function is positive. If $z \neq \pm\infty$ and there is no underflow exception when x is coerced into z, then $z = x(1 + \varepsilon)$ with $|\varepsilon| < 2^{-53}$.*

PROOF. $z \neq \pm\infty$ implies that $|x| < 2^{1023}(2 - 2^{-53})$. Because there is no underflow exception, we have $|x| \geq 2^{-1022}$. Hence, the resulting `double` z is the representable value nearest to x. Furthermore, Lemma 2.7.1 shows that $w = round\_to\_nearest(x)$ also is a valid `double`. If $x = 0$, then $z = 0$. Otherwise, we have $|x - z| \leq |x - w| < |x| 2^{-53}$ by Lemma 2.7.1 and the fact that z is nearest to x. □

There are two major differences in comparison to the `xbigfloat` model: Unlike the `xbigfloat` operations addition, subtraction, and multiplication there are, in general, rounding errors in each `double` operation. Furthermore, according to Theorem 2.7.2, it is more convenient to describe the errors of the `double` floating point operations in relative notation in contrast to `xbigfloat`, where absolute and relative notation is mixed.

CHAPTER 3

Accuracy constants

In this chapter we describe details of the implementation of the `xbigfloat` model. We present an analysis of the implemented algorithms for approximating the square root and the logarithm of a rational number and for the approximation of the exponential function. The analysis shows how accurate intermediate results must be approximated such that the output is an approximation of a prescribed precision.

3.1. Approximating the square root

Let $x > 0$ be a floating point number. To approximate to $\sqrt{x}$, we use the Newton iteration

$$y_{i+1} = \frac{1}{2} \cdot \left(y_i + \frac{x}{y_i} \right). \tag{3.1.1}$$

LEMMA 3.1.1. *If* $y_i = \sqrt{x}\,(1+\varepsilon_i)$, $i \geq 0$, $\varepsilon_i \neq -1$, *then* $\varepsilon_{i+1} = \frac{\varepsilon_i^2}{2(1+\varepsilon_i)}$.

PROOF.

$$\begin{aligned} y_{i+1} &= \frac{1}{2} \cdot \left(\sqrt{x}\,(1+\varepsilon_i) + \sqrt{x}\,\frac{1}{1+\varepsilon_i} \right) \\ &= \frac{\sqrt{x}}{2} \cdot \left(2 - (1-\varepsilon_i) + \frac{1}{1+\varepsilon_i} \right) \\ &= \frac{\sqrt{x}}{2} \cdot \left(2 + \frac{1-(1-\varepsilon_i)(1+\varepsilon_i)}{1+\varepsilon_i} \right) \\ &= \sqrt{x} \cdot \left(1 + \frac{\varepsilon_i^2}{2(1+\varepsilon_i)} \right). \end{aligned}$$

□

Assume that y_0 is an approximation to $\sqrt{x}$ with

$$y_0 = \sqrt{x}\,(1+E_0),\ |E_0| < 2^{-q},\ q \geq 3, \tag{3.1.2}$$

and set $\overline{y_0} = y_0$.

LEMMA 3.1.2. *For* $i \geq 1$ *let* $\overline{y_i} = 1/2(\overline{y_{i-1}} + x/\overline{y_{i-1}})(1+F_i)$ *with* $|F_i| < 2^{2^i(2-q)-4}$. *Then* $\overline{y_i} = \sqrt{x}\,(1+E_i)$, $|E_i| < 2^{2^i(2-q)-2}$ *for* $i \geq 0$.

PROOF. We use induction on i. For $i = 0$ the assertion follows from (3.1.2). Assume that the assertion holds for $\overline{y_{i-1}}$, $i \geq 1$. Then $\overline{y_{i-1}} = \sqrt{x}(1+E_{i-1})$. Set $\tilde{y}_i = 1/2(\overline{y_{i-1}} + x/\overline{y_{i-1}})$. It follows from Lemma 3.1.1, that

$$\tilde{y}_i = \sqrt{x}(1+\varepsilon_i),\ |\varepsilon_i| < E_{i-1}^2.$$

Hence,

$$\overline{y_i} = \sqrt{x}(1+\varepsilon_i)(1+F_i),$$

so $E_i = \varepsilon_i + F_i + \varepsilon_i F_i$. It is $|E_i| < |E_{i-1}|^2 + |F_i| + |E_{i-1}^2 F_i| < 2^{2(2^{i-1}(2-q)-2)} + 2^{2^i(2-q)-4} + 2^{2(2^{i-1}(2-q)-2)} 2^{2^i(2-q)-4} < 2^{2^i(2-q)-2}$. So the assertion also holds for $\overline{y_i}$. □

We present the function $\mathtt{sqrt}(x,k)$, that on input of a non-negative floating point number x and $k \in \mathbb{Z}_{\geq 1}$ returns a relative k-approximation to $\sqrt{x}$. It uses the function $\mathtt{sqrt\_initialization}(y,q,x)$, that returns the floating point number y and the integer $q \geq 3$, such that $y = \sqrt{x}(1+\varepsilon)$, $|\varepsilon| < 2^{-q}$. This function will be described below.

```
xbigfloat sqrt (xbigfloat x, int k)
 {
   if (x == 0)
      return 0;

   xbigfloat y, w, z;
   sqrt_initialization(y, q, x);

   int t = k+1;
   int n = b(t-2)-b(q-2)+1;
   int i, f;

   for (i=1; i <= n; i++)
    {
       f = 2^i(q-2)-4;

       w = truncate(x,F+4);
       z = truncate(y,F+3);
       y = divide (w,z,F+4);
       y += z;
       y /= 2;
    }

   return truncate(y,k+3);
 }
```

Theorem 3.1.3. *On input of a non-negative floating point number x and an integer $k \geq 1$, the function $\mathtt{sqrt}(x,k)$ returns a relative k-approximation to $\sqrt{x}$.*

Proof. We show that $y = \sqrt{x}(1+\varepsilon)$, $|\varepsilon| < 2^{-k-1}$, before the truncation at the end. Lemma 2.3.2 implies, that the returned value is a relative k-approximation to $\sqrt{x}$.

After the initialization it is $y = \sqrt{x}(1+\varepsilon)$, $|\varepsilon| < 2^{-q}$ with $q \geq 3$. If $k+1 = t < q$, then the initial value of y is accurate enough, so the loop need not to be executed. Otherwise $n \geq 1$ and the loop is executed. Let y_0 denote the value of y before the loop and y_i and f_i denote the value of y and f at the end of the i-th iteration of the loop, respectively. It follows from (2.3.1), that in the i-th iteration of the loop, it is

$$\begin{aligned} w &= x(1+\varepsilon_1),\ |\varepsilon_1| < 2^{-f_i-3}, \\ z &= y_{i-1}(1+\varepsilon_2),\ |\varepsilon_2| < 2^{-f_i-2}. \end{aligned}$$

The division w/z is done with a relative error $|\varepsilon_3| < 2^{-f_i-3}$. Let $1+\varepsilon_4 = (1+\varepsilon_1)(1+\varepsilon_3)/(1+\varepsilon_2)$. Then

$$\begin{aligned} y_i &= 1/2(y_{i-1} + x/y_{i-1})(1 + (y_{i-1}\varepsilon_2 + x/y_{i-1}\varepsilon_4)/(y_{i-1} + x/y_{i-1})), \\ &= 1/2(y_{i-1} + x/y_{i-1})(1+F_i). \end{aligned}$$

Because $x, y_{i-1} \geq 0$, it is $|F_i| \leq max\{|\varepsilon_2|, |\varepsilon_4|\} < 2^{-f_i} = 2^{2^i(q-2)-4}$. It follows from Lemma 3.1.2, that

$$y_n = \sqrt{x}(1+\varepsilon),\ |\varepsilon| < 2^{2^n(2-q)-2}.$$

Because n is chosen, such that $2^n \geq (t-2)/(q-2)$, it follows that $|\varepsilon| < 2^{-t} = 2^{-k-1}$. This proves the assertion. □

To find a start value for the Newton iteration, we truncate x such that it fits into a double in IEEE-754 format and then use the square root function of the IEEE-754 model. We make this more precise.

Let $x = (m, e)$ be the floating point representation of x. Then

$$x = m2^{-b(m)}2^b2^f,$$

where $f = e$, $b = 0$, if e is even, and $f = e-1$, $b = 1$, if e is odd. Let $m = (m_1, \ldots, m_{b(m)})_2$ be the bit representation of m. We set $m_{b(m)+1} = \ldots = m_{53} = 0$, if $b(m) < 53$. Set

$$z = truncate((m, b), 53).$$

Then, according to Section 2.7, the IEEE-754 double representation of z is

$$z = (0, 1022+b, (m_2, \ldots, m_{53})).$$

Furthermore, it follows from (2.3.1), that

$$z2^f = x(1+\varepsilon),\ |\varepsilon| < 2^{-52}. \tag{3.1.3}$$

Now we can apply the `sqrt` function of the IEEE-754 model to approximate $\sqrt{z}$.

```
void sqrt_initialization (xbigfloat & y,
                          int & q,
                          xbigfloat x)
 {
   // Determine b and f
   //
   bigint b, f;

   if (x.exponent().is_odd())
    {
      f = e-1;
      b = 1;
    }
   else
    {
      f = e;
```

```
      b = 0;
    }

   // Truncate to 53 bits.
   //
   bigint m = truncate(x,53).mantissa();

   if (b(m) < 53)
      m = shift_left(m, 53-b(m));
   else
      m = shift_right(m, b(m)-53);

   // create z = m * 2^(-b(m)) * 2^b
   //
   double z = 0;

   while (m > 0)
    {
      if (m.is_odd())
        z += 1;

      z /= 2;
      m >>= 1;
    }

   if (b == 1)
     z *= 2;

   // Approximate the square root.
   //
   y = 2^(f/2) * (xbigfloat) sqrt(z);
   q = 51;
}
```

THEOREM 3.1.4. *On input of a non-negative floating point number x of type* **xbigfloat**, *the function* **sqrt_initialization**(y, q, x) *returns the* **xbigfloat** y *and the integer q, such that $y = \sqrt{x}(1+\varepsilon)$, $|\varepsilon| < 2^{-q}$.*

PROOF. Because the value for f fits into IEEE-754 double format, no roundoff error occurs during the initialization of z. Therefore, we have by (3.1.3) that

$$z2^f = x(1+\varepsilon_1),\ |\varepsilon_1| < 2^{-52}.$$

Set $r = \mathtt{sqrt}(z)$, where `sqrt` is the IEEE-754 square root function. It follows from Theorem 2.7.2, that

$$r = \sqrt{z}(1+\varepsilon_2),\ |\varepsilon_2| < 2^{-53}.$$

Hence $y = r2^{f/2}$ satisfies

$$\begin{aligned} y &= \sqrt{x}(1+\varepsilon_1)^{1/2}(1+\varepsilon_2), \\ &= \sqrt{x}(1+\varepsilon_3). \end{aligned}$$

It is $(1+\varepsilon_1)^{1/2}(1+\varepsilon_2) > (1-2^{-52})^{1/2}(1-2^{-53}) > (1-2^{-52})^2 > 1-2^{-51}$. Similarly, we have $(1+\varepsilon_1)^{1/2}(1+\varepsilon_2) < (1+2^{-52})^{1/2}(1+2^{-53}) < (1+2^{-53})^2 < 1+2^{-51}$. Hence, it is $|\varepsilon_3| < 2^{-51}$. □

3.2. Approximating the logarithm

Let $x \geq 0$ be a floating point number and $k \in \mathbb{Z}$. Our goal is to compute an absolute k-approximation to $\ln x$. If we assume that x is in the range $1 < x < 1 + 2^{-c+1}$, where $c \in \mathbb{Z}_{\geq 1}$, we can use the identity ([**AS64**][(4.1.27)])

$$\ln x = 2 \sum_{i=0}^{\infty} \frac{y^{2i+1}}{2i+1},\ y = \frac{x-1}{x+1}$$

to compute the approximation.

Lemma 3.2.1. *Let $k, c \in \mathbb{Z}_{\geq 1}$ and $1 < x < 1 + 2^{-c+1}$, then for every $n \in \mathbb{N}$ with $n \geq k/2c$, we have*

$$| \ln x - 2 \sum_{i=0}^{n} \frac{y^{2i+1}}{2i+1} | < 2^{-k}, \tag{3.2.1}$$

where $y = (x-1)/(x+1)$.

Proof. see [**Pap93**, Theorem 4.1] □

If x is not in the interval $(1, 2)$, we use the identities

$$\begin{array}{lcll} \ln x & = & 0, & \text{if } x = 1, \\ \ln x & = & -\ln(1/x), & \text{if } 0 < x < 1, \\ \ln x & = & 2 \ln \sqrt{x}, & \text{if } 2 \leq x. \end{array}$$

Hence, according to [**Pap93**, Section 4.4.3], the structure for computing an approximation to the natural logarithm of a floating point number can be formulated as follows:

1. If $x = 1$, return 0.
2. If $0 < x < 1$, invert x.
3. If $2 \leq x$, take square roots until $x < 2$.
4. Use Lemma 3.2.1 to approximate $\ln x$.
5. Multiply the approximation according to steps 2) and 3).

We present the function log(x, k), that on input of a positive floating point number x and $k \in \mathbb{Z}$ returns an absolute k-approximation to $\ln x$.

```
xbigfloat log (xbigfloat v, int r)
 {
   // v == 1
   //
   if (v == 1)
      return 0;

   // 0 < v < 1, invert
   //
   bool inversion;
   xbigfloat w;
   int s;
```

```
if (v < 1)
 {
   // v <= 1-2^(-d), d >= 1.
   int d = 1-b(1-v);
   w = divide (1, v, max{d, r+3});
   s = r+2;
   inversion = true;
 }
else
 {
   w = v;
   s = r+1;
   inversion = false;
 }

// 2 <= w, square root reduction
//
bool square_root;
xbigfloat x;
int t, f;

if (w >= 2)
 {
   x = w;
   n = f;
   while (x >= 2)
    {
      f = f +1;
      x = sqrt(x, max{b(b(w))+s+5,3});
    }
   t = s+f+1;
   square_root = true;
 }
else
 {
   x = w;
   t = s;
   square_root = false;
 }

// We have 1 < x < 2.
//
xbigfloat l;

if (t <= 0)
   l = 0;
else
 {
```

```
int k = t+1;
int c = 1-b(x-1);
int n = ceil( (k+1)/(2c) );

int k_0 = 4+k+b(n+1)-c;
int s_0 = max{ k_0, b(6n+6) };

xbigfloat z_0 = divide(x-1, x+1, s_0 + 2);
z_0 = z_0 * z_0;

// l = 1/(2*n+1) approximation.
//
int k_i = k_0 - c * 2n;
int s = max{ k_i, b(6n+6) };

xbigfloat m = 2*n+1;
l = divide(1, m, s);

// Result in l.
//
xbigfloat h;

for (i=n-1; i >= 0; i--)
 {
   k_i += 2c;
   s = max{ k_i, b(6n+6) };

   // Multiply l by ((x-1)/(x+1))^2 approximation.
   //
   if (s < s_0)
    {
      h = truncate(z_0, s+3);
      l *= h;
    }
   else
      l *= z_0;

   // Add 1/(2i+1) approximation to l.
   //
   m -= 2;
   h = divide (1, m, s);
   l += h;
   l = truncate(l, s+1);
 }

// Multiply l by 2(x-1)/(x+1) approximation.
//
h = divide(x-1, x+1, s_0);
l *= 2*h;
```

```
      l = truncate(l, l.exponent() + t + 1);
    }

   if (square_root)
      l = l * 2^f;

   if (inversion)
      l = -l;

   return truncate(l, l.exponent() + r + 1);
}
```

Before we can prove the correctness of `log`, we need some auxiliary results.

LEMMA 3.2.2. *Let $x \in \mathbb{R}$ with $1 < x < 1 + 2^{-c+1}$, $c \in \mathbb{Z}_{\geq 1}$, $n \in \mathbb{Z}_{\geq 0}$, and $k \in \mathbb{Z}$. Set*

$$s_i = max\{3 + k + b(n+1) - c(2i+1), b(6n+6)\},\ 0 \leq i \leq n. \tag{3.2.2}$$

Furthermore, for $y = (x-1)/(x+1)$, set

$$\begin{aligned} z &= y(1+d), \\ z_i &= y^2(1+d_i),\ 0 \leq i < n, \\ a_i &= 1/(2i+1)(1+e_i),\ 0 \leq i \leq n, \end{aligned}$$

and

$$\begin{aligned} l_n &= a_n, \\ l_i &= (l_{i+1} z_i (1+f_i) + a_i)(1+g_i),\ 0 \leq i < n, \\ l &= 2 z l_0, \end{aligned}$$

where d, d_i, e_i, f_i, $g_i \in \mathbb{R}$ such that

$$|d| < 2^{-s_0},\ |e_i| < 2^{-s_i},\ 0 \leq i \leq n,\ |d_i|, |f_i|, |g_i| < 2^{-s_i},\ 0 \leq i < n.$$

Then $|l - 2\sum_{i=0}^{n} y^{2i+1}/(2i+1)| < 2^{-k}$.

PROOF. For $n - 1 \geq i \geq 0$ we have

$$\begin{aligned} l_i &= z_i \cdot l_{i+1} \cdot (1+f_i)(1+g_i) + a_i(1+g_i) \\ &= z_{n-1} \cdots z_i a_n (1+f_{n-1})(1+g_{n-1}) \cdots (1+f_i)(1+g_i) + \\ &\quad z_{n-2} \cdots z_i a_{n-1} (1+g_{n-1})(1+f_{n-2})(1+g_{n-2}) \cdots (1+f_i)(1+g_i) + \\ &\quad z_i a_{i+1} (1+g_{i+1})(1+f_i)(1+g_i) + \\ &\quad a_i(1+g_i). \end{aligned}$$

Hence,

$$l = 2\sum_{i=0}^{n} (1+E_i) y^{2i+1}/(2i+1), \tag{3.2.3}$$

with

$$1 + E_i = (1+d)(1+e_i)(1+g_i) \prod_{j=0}^{i-1} (1+d_j)(1+f_j)(1+g_j), \tag{3.2.4}$$

where we set $g_n = 0$ and the product to one for $i = 0$.

It follows from (3.2.2), that

$$\textit{Condition}\,1:\ s_0 \geq s_1 \geq \cdots \geq s_n \geq 1. \tag{3.2.5}$$

It follows from (3.2.4) and (3.2.5), that

$$(1-2^{-s_i})^{3i+3} < 1+E_i < (1+2^{-s_i})^{3i+3},\ 0 \leq i \leq n. \tag{3.2.6}$$

Now (3.2.5) and Lemma 2.5.1 imply, that $(1-2^{-s_i})^{3i+3} > 1-(3i+3)2^{-s_i}$ for $0 \leq i \leq n$. It also follows from (3.2.2), that

$$\textit{Condition}\,2:\ s_i \geq b(6i+6),\ 0 \leq i \leq n. \tag{3.2.7}$$

Then (3.2.7) implies $(3i+3)2^{-s_i} < 1/2$, so Lemma 2.5.2 yields $(1+2^{-s_i})^{3i+3} < 1+(3i+3)2^{-s_i+1}$ for $0 \leq i \leq n$. Hence, (3.2.6) implies

$$|E_i| < (3i+3)2^{-s_i+1},\ 0 \leq i \leq n. \tag{3.2.8}$$

It also follows from (3.2.2), that

$$\textit{Condition}\,3:\ s_i \geq 3+k+b(n+1)-c(2i+1),\ 0 \leq i \leq n. \tag{3.2.9}$$

And it follows from $1 < x < 1+2^{-c+1}$, $c \geq 1$, that $y < 2^{-c}$. So (3.2.3), (3.2.8), and (3.2.9) imply, that

$$\begin{aligned}
|l - 2\sum_{i=0}^{n} y^{2i+1}/(2i+1)| &\leq 2\sum_{i=0}^{n} |E_i| y^{2i+1}/(2i+1) \\
&\leq 2\sum_{i=0}^{n} (3i+3)2^{-s_i+1}2^{-c(2i+1)}/(2i+1) \\
&< 2\sum_{i=0}^{n} 2^{-s_i+2-c(2i+1)} \\
&\leq 2\sum_{i=0}^{n} 2^{-3-k-b(n+1)+2} \\
&< 2^{k}.
\end{aligned}$$

□

LEMMA 3.2.3. *Let $x \geq 2$ be a floating point number and $k \in \mathbb{Z}$. Let*

$$\begin{aligned}
z_0 &= x, \\
z_i &= \sqrt{z_{i-1}}(1+\varepsilon_i),\ |\varepsilon_i| < 2^{-max\{b(b(x))+k+5,3\}},\ 1 \leq i \leq n,
\end{aligned}$$

until $z_n < 2$. If l is a floating point number with $|l - \ln z_n| < 2^{-k-n-1}$, then $|2^n l - \ln x| < 2^{-k}$.

PROOF. We estimate the relative error of z_i. For $1 \leq i \leq n$, we have

$$\begin{aligned}
z_i &= z_0^{1/2^i}(1+\varepsilon_1)^{1/2^{i-1}}(1+\varepsilon_2)^{1/2^{i-2}}\cdots(1+\varepsilon_i) \\
&= x^{1/2^i}(1+E_i),
\end{aligned}$$

with $|\varepsilon_i| < 2^{-s}$ for $s = max\{b(b(x))+k+5,3\}$. So

$$(1-2^{-s})^2 < 1+E_i < (1+2^{-s})^2,\ 1 \leq i \leq n. \tag{3.2.10}$$

Because $s \geq 1$, Lemma 2.5.1 implies $(1-2^{-s})^2 > 1-2^{-s+1}$. And because $s \geq 3$, Lemma 2.5.3 implies $(1+2^{-s})^2 < 1+4\cdot 2^{-s}$. So,

$$|E_i| < 2^{-s+2},\ 1 \leq i \leq n. \tag{3.2.11}$$

We derive an upper bound on n. Let $m \in \mathbb{Z}_{\geq 1}$ be minimal with $x^{1/2^m} < 2$. Because $x^{1/2^{m-1}} \geq 2$, we have $2^{2^{m-1}} \leq x$, so $m \leq b(b(x))$. And (3.2.10) implies $2 \leq z_{n-1} = x^{1/2^{n-1}}(1+E_{n-1}) < x^{1/2^{n-1}}(1+2^{-s})^2 \leq x^{1/2^{n-1}}(9/8)^2$, because $s \geq 3$. Hence, $x^{1/2^{n-1}} > 128/81 > \sqrt{2}$, and we obtain $x^{1/2^{n-2}} > 2$, which implies $n-2 \leq m-1$, so

$$n \leq b(b(x))+1. \tag{3.2.12}$$

Because $s \geq 3$, it follows from Corollary 2.4.3 and (3.2.11), that $|\ln(1+E_n)| < |E_n|/(1-|E_n|) < 2|E_n| < 2^{-s+3}$. This implies

$$\begin{aligned}
|2^n l - \ln x| &\leq 2^n|l-\ln z_n| + |2^n \ln z_n - \ln x| \\
&\leq 2^{-k-1} + 2^n|\ln(1+E_n)| \\
&< 2^{-k-1} + 2^n 2^{-s+3} \\
&\leq 2^{-k-1} + 2^{b(b(x))+1}2^{-b(b(x))-k-2} \\
&= 2^{-k}.
\end{aligned}$$

□

Lemma 3.2.4. *Let x be a floating point number with $0 < x < 1$ and let $k \in \mathbb{Z}$. Let $d \in \mathbb{Z}_{\geq 1}$ be minimal, such that $x \leq 1-2^{-d}$.*

If $z = 1/x(1+\varepsilon)$, $|\varepsilon| < 2^{-max\{d,k+2\}}$, then $z > 1$. Furthermore, if l is a floating point number with $|l-\ln z| < 2^{-k-1}$, then $|-l-\ln x| < 2^{-k}$.

Proof. If $\varepsilon > 0$, then $z > 1/x > 1$. If $\varepsilon \leq 0$, then $-\varepsilon = |\varepsilon| < 2^{-d} \leq 1-x$, so $1 < 1/x(1+\varepsilon) = z$. Hence, $z > 1$ in both cases.

Because $|\varepsilon| < 1/2$ it follows from Corollary 2.4.3, that $|\ln(1+\varepsilon)| < |\varepsilon|/(1-|\varepsilon|) < 2|\varepsilon| < 2^{-k-1}$. This implies

$$\begin{aligned}
|-l-\ln x| &= |\ln(1/x)-l| \\
&\leq |\ln(1/x)-\ln z| + |\ln z - l| \\
&< |\ln(1+\varepsilon)| + 2^{-k-1} \\
&< 2^{-k}.
\end{aligned}$$

□

Theorem 3.2.5. *On input of a positive floating point number x and $k \in \mathbb{Z}$, the function **log**(x,k) returns an absolute k-approximation to $\ln x$.*

Proof. We use the notation of the procedure. It consists of 5 parts. An inversion in case of $0 < v < 1$, a square root reduction for $2 \leq w$, a logarithm approximation to $\ln x$ for $1 < x < 2$, an adaption of the logarithm approximation in case of a square root reduction, and a further adaption for a possible inversion.

First, we show that, after the logarithm approximation, l is an absolute t-approximation to $\ln x$. If $t \leq 0$, then $|\ln x| < \ln 2 < 1$, so $l = 0$ is an absolute t-approximation to $\ln x$.

Assume, that $t \geq 1$ and set $k = t + 1$. First, note that $|x - 1| < 2^{b(x-1)}$, so $c = 1 - b(x - 1)$ satisfies $1 < x < 1 + 2^{-c+1}$, $c \geq 1$. Because $n = \lceil (k + 1)/(2c) \rceil$, it follows from Lemma 3.2.1, that

$$| \ln x - 2 \sum_{i=0}^{n} \frac{y^{2i+1}}{2i + 1} | < 2^{-k-1}, \tag{3.2.13}$$

for $y = (x - 1)/(x + 1)$. We set

$$s_i = max\{4 + k + b(n + 1) - c(2i + 1), b(6n + 6)\}, \; 0 \leq i \leq n.$$

Because $s_0 \geq 0$, it follows that

$$z_0 = y^2(1 + d_0), \; |d_0| < 2^{-s_0}.$$

Furthermore, let l_n denote the value of l right before the start of the while-loop. It is

$$l_n = 1/(2n + 1)(1 + e_n), \; |e_n| < 2^{-s_n}.$$

For iteration i, let the first value of h be denoted by z_i, the second value of h be denoted by a_i, and let the value of l at the end of the loop be denoted by l_i, $i = n-1, \ldots, 0$. Furthermore, note that the value of s for iteration i is s_i. We have

$$\begin{aligned} a_i &= 1/(2i + 1)(1 + e_i), \; |e_i| < 2^{-s_i}, \\ z_i &= y^2(1 + d_i), \; |d_i| < 2^{-s_i}, \end{aligned}$$

for $i = n - 1, \ldots, 0$, where the last equation follows from Lemma 2.3.2. And it is

$$l_i = (l_{i+1} z_i(1 + f_i) + a_i)(1 + g_i), \; 0 \leq i \leq n - 1,$$

with $f_i = 0$, because multiplication is done without truncation, and $|g_i| < 2^{-s_i}$ by (2.3.1). Furthermore, let the value of h as the result of the division after the loop be denoted by z. Then

$$z = y(1 + d), \; |d| < 2^{-s_0}.$$

It follows from Lemma 3.2.2, that l, after multiplication with $2z$, satisfies

$$|l - 2 \sum_{i=0}^{n} \frac{y^{2i+1}}{2i + 1}| < 2^{-k-1}. \tag{3.2.14}$$

So (3.2.13) and (3.2.14) yield $|l - \ln x| < 2^{-k} = 2^{-t-1}$. It follows from Lemma 2.3.1, that l is an absolute t-approximation to $\ln x$ after truncation.

Now, we analyze the remaining adaptions. Lemma 3.2.3 yields, that after a possible multiplication by 2^f in case of a square root reduction, we have $|l - \ln w| < 2^{-s}$. And it follows from Lemma 3.2.4, that after a possible negation in case of an inversion, it is $|l - \ln v| < 2^{-r-1}$. So, Lemma 2.3.1 implies, that l is an absolute r-approximation to $\ln v$ after truncation. □

3.3. Approximating the exponential function

Let x be a floating point number and $k \in \mathbb{Z}_{\geq 1}$. We want to compute a relative k-approximation g to $\exp x$. If $0 < x < 2^{-c}$ for some $c \in \mathbb{Z}_{\geq 1}$, then we use the identity ([**AS64**][(4.2.1)])

$$\exp x = \sum_{i=0}^{\infty} x^i / i!.$$

LEMMA 3.3.1. *Let $k, c \in \mathbb{Z}_{\geq 1}$, $x \in \mathbb{R}$ with $0 < x < 2^{-c}$, and $n = \lceil k/c \rceil$. Then*

$$|\exp x - \sum_{i=0}^{n} x^i/i!| < 2^{-k}. \tag{3.3.1}$$

PROOF. It follows from [**Kön84**, III.3.19], that

$$|\exp x - \sum_{i=0}^{n} x^i/i!| < 2\frac{x^{n+1}}{(n+1)!}.$$

And it is $2x^{n+1}/(n+1)! \leq x^{n+1} < 2^{-c(k/c+1)} = 2^{-k-c} < 2^{-k}$. □

If x is not in the interval $(0, 1/2)$, we use the identities

$$\begin{array}{rcll} \exp x & = & 1, & \text{if } x = 0, \\ \exp x & = & 1/\exp(-x), & \text{if } x < 0, \\ \exp x & = & (\exp(x/2^{b(x)}))^{2^{b(x)}}, & \text{if } 1/2 \leq x. \end{array}$$

Hence, according to [**Pap93**, Section 4.4.2], the structure for computing an approximation to the expontial function can be formulated as follows:

1. If $x = 0$, return 1.
2. If $x < 0$, negate x.
3. If $1/2 \leq x$, divide x by $2^{b(x)}$.
4. Use Lemma 3.3.1 to approximate $\exp x$.
5. Power and invert the approximation according to steps 2) and 3).

We present the function exp(v, r), that on input of a floating point number v and $r \in \mathbb{Z}_{\geq 1}$ returns a relative r-approximation to $\exp v$.

```
xbigfloat exp ( xbigfloat v, int r )
 {
    if (v == 0)
       return 1;

    // Negation for v < 0.
    //
    xbigfloat w;
    int s;
    bool negation;

    if (v < 0)
     {
       w = -v;
       s = r+3;
       negation = true;
     }
    else
     {
       w = v;
       s = r+1;
       negation = false;
     }
```

```
// Division for 1/2 <= w.
//
xbigfloat x;
int t;
bool division;
int m;

if (0.5 <= w)
 {
   m = b(w);
   x = w / 2^m;
   t = s+m+b(m+1)+2;
   division = true;
 }
else
 {
   x = w;
   t = r;
   division = false;
 }

// We have 1 < x < 1/2, t >= 1.
//
int k = t+2;
int c = -b(x);
int n = ceil(k/c);

int i;
xbigfloat h, z;

xbigfloat l = 1;
int k_i = k+b(n)+2-(n+1)c;
int s_i;

for (i=n; i >= 2; i--)
 {
    k_i += c;
    s_i = max(k_i, b(2n));

    // Multiply by x/i approximation.
    //
    h = truncate(x, s_i+3);
    z = divide (h, i, s_i+2);
    l *= z;

    // Add 1 and truncate.
    //
    l += 1;
    l = truncate(l, s_i+1);
```

```
    }

    k_i += c;
    s_i = max(k_i, b(2n));

    // Multiply by x approximation and add 1.
    //
    z = truncate(x, s_i+1);
    l *= z;
    l += 1;

    l = truncate(l, t+3);

    // Power by 2^m in case of division.
    //
    if (division)
     {
       k_i = s+m+b(m+1)+2;
       for (i=1; i <= m; i++)
        {
            l = l^2;
            k_i -= 1;
            l = truncate(l, k_i + 1);
        }
     }

    // Invert in case of negation.
    //
    if (negation)
       l = divide (1,l,r+3);

    return truncate(l, r+3);
}
```

Before we can prove the correctness of exp, we need some auxiliary results.

LEMMA 3.3.2. *Let $x \in \mathbb{R}_{<0}$ and $k \in \mathbb{Z}_{\geq 1}$. If $h = \exp(-x)(1+\varepsilon_1)$ with $|\varepsilon_1| < 2^{-k-2}$ and $g = 1/h(1+\varepsilon_2)$ with $|\varepsilon_2| < 2^{-k-2}$, then $g = \exp(x)\,(1+\varepsilon)$ with $|\varepsilon| < 2^{-k}$.*

PROOF. We have $g = \exp(x)\,(1+\varepsilon_2)/(1+\varepsilon_1) = \exp(x)\,(1+\varepsilon)$ with $|\varepsilon| < 2^{-k-2} + 2^{-k-1} + 2^{-2k-3} < 2^{-k}$ by Lemma 2.5.3. □

LEMMA 3.3.3. *Let $1/2 \leq x \in \mathbb{R}$ and $k \in \mathbb{Z}_{\geq 1}$. Set $n = b(x)$. If*

$$\begin{aligned} g_0 &= \exp(x/2^n)\,(1+\varepsilon_0), \\ g_i &= g_{i-1}^2\,(1+\varepsilon_i),\ i = 1, \ldots, n, \end{aligned}$$

with $|\varepsilon_i| < 2^{-k-n-b(n+1)-2+i}$ for $0 \leq i \leq n$, then $g_n = \exp(x)\,(1+\varepsilon)$ with $|\varepsilon| < 2^{-k}$.

PROOF. We have

$$\begin{aligned} g_n &= (\exp(x/2^n))^{2^n} \prod_{i=0}^{n}(1+\varepsilon_i)^{2^{n-i}}, \\ &= \exp(x)\,(1+\varepsilon). \end{aligned}$$

Set $s_i = k + n + b(n+1) + 2 - i$. Applying Lemma 2.5.2 twice yields

$$\begin{aligned} 1+\varepsilon &= \prod_{i=0}^{n}(1+\varepsilon_i)^{2^{n-i}} \\ &< \prod_{i=0}^{n}(1+2^{-s_i})^{2^{n-i}} \\ &< \prod_{i=0}^{n}(1+2^{n-i+1}2^{-s_i}) \\ &= (1+2^{-k-b(n+1)-1})^{n+1} \\ &< 1+2(n+1)2^{-k-b(n+1)-1}) \\ &< 1+2^{-k}. \end{aligned}$$

And the lower bound $1+\varepsilon > 1-2^{-k}$ can be derived in the same way using Lemma 2.5.1. Hence, $|e| < 2^{-k}$. $\square$

LEMMA 3.3.4. *Let $x \in \mathbb{R}$ with $0 < x < 2^{-c}$ for $c \in \mathbb{Z}_{\geq 1}$, $n \in \mathbb{Z}_{\geq 0}$, and $k \in \mathbb{Z}$. Set*

$$s_i = max\{k + b(n) + 2 - ic, b(2n)\},\ 1 \leq i \leq n. \tag{3.3.2}$$

Furthermore, let

$$\begin{aligned} z_i &= x/i(1+d_i),\ 1 \leq i \leq n, \\ l_{n+1} &= 1, \\ l_i &= (l_{i+1}z_i + 1)(1+g_i),\ 2 \leq i \leq n, \\ l_1 &= l_2 z_1 + 1, \end{aligned}$$

where d_i, $g_i \in \mathbb{R}$, such that $|d_i| < 2^{-s_i}$, $1 \leq i \leq n$, and $|g_i| < 2^{-s_i}$, $2 \leq i \leq n$.
Then $|l_1 - \sum_{i=0}^{n} x^i/i!| < 2^{-k}$.

PROOF. For $j = 1, \ldots, n$ we have

$$l_j = \sum_{i=j}^{n}\left(\prod_{k=j}^{i} z_k(1+g_k)\right) + (1+g_j),$$

where we set $g_1 = 0$. It follows that

$$l_1 = \sum_{i=0}^{n} x^i/i!(1+E_i), \tag{3.3.3}$$

with

$$1+E_i = \prod_{j=1}^{i}(1+g_j)(1+d_j),\ 1 \leq i \leq n, \tag{3.3.4}$$

and $E_0 = 1$. It follows from (3.3.2), that

$$\textit{Condition}\,1: \; s_1 \geq s_2 \geq \cdots \geq s_n \geq 1. \tag{3.3.5}$$

Hence, (3.3.4) and (3.3.5) imply

$$(1 - 2^{-s_i})^{2i} < 1 + E_i < (1 + 2^{-s_i})^{2i},\; 1 \leq i \leq n.$$

Because $s_i \geq 1$, it follows from Lemma 2.5.1, that $(1 - 2^{-s_i})^{2i} > 1 - 2i\,2^{-s_i}$ for $1 \leq i \leq n$. It also follows from (3.3.2), that

$$\textit{Condition}\,2: \; 2^{-s_i} > 2i. \tag{3.3.6}$$

Together with Lemma 2.5.2, this implies $(1 + 2^{-s_i})^{2i} < 1 + 4i\,2^{-s_i}$ for $1 \leq i \leq n$. So, it is

$$|E_i| < i\,2^{-s_i+2},\; 1 \leq i \leq n. \tag{3.3.7}$$

It also follows from (3.3.2), that

$$\textit{Condition}\,3: \; s_i \geq k + b(n) + 2 - ic,\; 1 \leq i \leq n. \tag{3.3.8}$$

So (3.3.3), (3.3.7), and (3.3.8) imply

$$\begin{aligned} |l_1 - \sum_{i=0}^{n} x^i/i!| &\leq \sum_{i=1}^{n} x^i/i!|E_i| \\ &< \sum_{i=1}^{n} 2^{-ic} i\,2^{-s_i+2}/i! \\ &\leq \sum_{i=1}^{n} 2^{-k-b(n)} \\ &< 2^{-k}. \end{aligned}$$

□

THEOREM 3.3.5. *Let x be a floating point number and $k \in \mathbb{Z}_{\geq 1}$. The function* **exp**$(x, k)$ *returns a relative k-approximation to* $\exp(x)$.

PROOF. We use the notation of the procedure. It consists of 5 parts. A negation in case of $v < 0$, a division for $1/2 \leq w$, an approximation of $\exp(x)$ for $0 < x < 1/2$, an adaption of that approximation in case of a division, and a further adaption for a possible negation.

First, we show, that, after the approximation of $\exp(x)$, l is a relative t-approximation to $\exp(x)$. Let z_i and l_i denote the value of z and l at the end of the loop iteration for i, respectively, and l_{n+1} the value of l before the start of the loop. It follows from (2.3.1), that in the iteration for i, it is $h = x(1 + \varepsilon_1)$, $|\varepsilon_1| < 2^{-s_i-2}$. Furthermore, $z = h/i(1 + \varepsilon_2)$, $|\varepsilon_2| < 2^{-s_i-2}$. Hence,

$$z_i = x/i(1 + d_i),\; |d_i| < 2^{-s_i},\; 2 \leq i \leq n.$$

It follows from (2.3.1), that $l_i = (l_{i+1}z_i + 1)(1 + g_i)$, $2 \leq i \leq n$, with $|g_i| < 2^{-s_i}$.

Let z_1 denote the value z after the loop, i.e., the truncation of x to $s_1 = k + b(n) + 2 - c$. Then

$$z_1 = x(1 + d_1),\; |d_1| < 2^{-s_1}.$$

Hence, it follows from Lemma 3.3.4, that before the trunction to $t + 3$ bits, it is $|l - \sum_{i=0}^{n} x^i/i!| < 2^{-k} = 2^{-t-2}$. Lemma 3.3.1 implies $|l - \exp(x)| < 2^{-t-1}$. Because

$\exp(x) \geq 1$, it is $l = \exp(x)(1 + \varepsilon)$, $|\varepsilon| < 2^{-t-1}$. And after truncation l is a relative t-approximation to $\exp(x)$ by Lemma 2.3.2.

Now, we analyze the remaining adaptions. Lemma 3.3.3 yields, that after a possible squaring by 2^m in case of a division, we have $|l - \exp w|/|\exp w| < 2^{-s}$. (Denote by l_i the value of l at the end of the i-th iteration of the for-loop. Note that the truncation to $k_i + 1$ yields $l_i = l_{i-1}^2(1 + \varepsilon_i)$ with $|\varepsilon_i| < 2^{-k_i}$ by (2.3.1).)

And it follows from Lemma 3.3.2, that after a possible inversion in case of a negation, it is $|l - \exp v| < 2^{-r-1}$. So, Lemma 2.3.2 implies, that l is a relative r-approximation to $\exp v$ after truncation. $\square$

CHAPTER 4

Computing with quadratic numbers

4.1. Standard representation: `quadratic_number_standard`

Every quadratic number, i.e., element $\alpha \in K$ can be written as

$$\alpha = \frac{x + y\sqrt{\Delta}}{z}, \tag{4.1.1}$$

with $x, y, z \in \mathbb{Z}$, $z > 0$, and $gcd(x, y, z) = 1$. That representation is unique. It is called the *(Δ-)standard representation* of α.

The standard representation of quadratic numbers is implemented by the data type `quadratic_number_standard`. An object a that represents α is the quadruple

$$a = (x, y, z, p),$$

where x, y, z are given by the standard representation of α, and p is a pointer to an object that represents O. We set

$$\mathrm{size}(a) = \mathrm{size}(x) + \mathrm{size}(y) + \mathrm{size}(z) + \mathrm{size}(\Delta) + 1,$$

and

$$\mathrm{size}_s(\alpha) = \mathrm{size}(a).$$

Often we will identify a with the number α represented by a. For a finite set S of objects $a_i = (x_i, y_i, z_i, p_i)$, $1 \leq i \leq n$, where the p_i all point to the same quadratic order object, we set

$$\mathrm{size}(S) = \mathrm{size}(\Delta) + n + \sum_{i=1}^{n} \mathrm{size}(x_i) + \mathrm{size}(y_i) + \mathrm{size}(z_i).$$

Let (x, y, z, p) be the standard representation of α. We have

$$H(\alpha) = \frac{|x| + |y|\sqrt{\Delta}}{z},$$

and

$$d(\alpha) = \begin{cases} z, & \text{if } z \text{ odd },\\ z/2, & \text{if } z \text{ even }. \end{cases}$$

In addition, assume that α is non-zero. First, we estimate b-values. Note that

$$|x + y\sqrt{\Delta}| = \begin{cases} |x| + |y|\sqrt{\Delta}, & \text{if } xy \geq 0,\\ |N(x + y\sqrt{\Delta})|/(|x| + |y|\sqrt{\Delta}), & \text{if } xy < 0. \end{cases} \tag{4.1.2}$$

It follows from (4.1.2), that

$$\begin{aligned} |\alpha| &\leq |x| + |y|\sqrt{\Delta} < 2^{b(x)+b(y)+b(\sqrt{\Delta})+1},\\ |\alpha| &\geq 1/(z(|x| + |y|\sqrt{\Delta})) > 2^{-(b(x)+b(y)+b(\sqrt{\Delta})+b(z)+1)}. \end{aligned}$$

As a consequence, we obtain

$$|b(\alpha)| \leq \text{size}_s(\alpha). \tag{4.1.3}$$

Furthermore, it is

$$2^{b(d(\alpha)\alpha)-b(d(\alpha)\sigma\alpha)-1} < |a/\sigma\alpha| < 2^{b(d(\alpha)\alpha)+b(d(\alpha)\sigma\alpha)+1}.$$

Thus, it follows from (4.1.3) that

$$|b(\alpha/\sigma\alpha)| \leq 2\ \text{size}_s(d(\alpha)\alpha) + 1. \tag{4.1.4}$$

And finally, we have

$$1/z^2 \leq |N\alpha| < 2^{b(\alpha)+b(\sigma\alpha)},$$

which implies, together with (4.1.3) again, that

$$|b(N\alpha)| \leq 2\ \text{size}_s(\alpha). \tag{4.1.5}$$

Furthermore, it is $d(\alpha)H(\alpha) \geq (|x| + |y|\sqrt{\Delta})/2$ and $2d(\alpha) \geq z$. And an easy calculation shows that there exists a constant $c \in \mathbb{N}$, independent of α and O, with

$$\text{size}_s(\alpha) \leq c \log((d(\alpha))^2 H(\alpha)\Delta). \tag{4.1.6}$$

We estimate the absolute values of $\ln|\alpha|$ and $\operatorname{Ln}\alpha$. Note, that $2^{b(\alpha)-1} \leq |\alpha| < 2^{b(\alpha)}$ and so, $(b(\alpha)-1)\ln 2 \leq \ln\alpha < b(\alpha)\ln 2$. It follows that $|\ln\alpha| \leq (1+|b(\alpha)|)\ln 2$, and together with (4.1.3), we obtain

$$|\ln\alpha| < 1 + \text{size}_s(\alpha). \tag{4.1.7}$$

The same calculation together with (4.1.4) yields

$$|\operatorname{Ln}\alpha| < 1 + \text{size}_s(d(\alpha)\alpha). \tag{4.1.8}$$

The absolute value of $\operatorname{Ln}\alpha$ is bounded by the height and norm of α as stated in the following Lemma.

LEMMA 4.1.1. *Let* $\alpha \in K^*$. *Then* $|\operatorname{Ln}\alpha| \leq 1/2\ln(H(\alpha)^2/|N(\alpha)|)$.

PROOF. We have $|\alpha/\sigma\alpha| = |N(\alpha)|/|\sigma\alpha|^2 \geq |N(\alpha)|/H(\alpha)^2$. Similarly, $|\sigma\alpha/\alpha| = |N(\alpha)|/|\alpha|^2 \geq |N(\alpha)|/H(\alpha)^2$. This implies that $max\{\ln|\alpha/\sigma\alpha|, -\ln|\alpha/\sigma\alpha|\} \leq \ln(|N(\alpha)|/H(\alpha)^2)$. Thus $|\operatorname{Ln}\alpha| = 1/2|\ln|\alpha/\sigma\alpha|| \leq 1/2\ln(H(\alpha)^2/|N(\alpha)|)$. □

4.1.1. Operations for the standard model.

We describe the multiplication and inversion of quadratic numbers in Δ-standard representation and estimate their bit complexity.

Let $a = (x, y, z, p)$, $a_i = (x_i, y_i, z_i, p)$, $i = 1, 2$ be quadratic numbers in Δ-standard representation. The procedure a.multiply(a_1, a_2) computes the Δ-standard representation a of a_1a_2.

```
void a.multiply(quadratic_number_standard a_1,
                quadratic_number_standard a_2)
  {
    bigint h = a_1.x * a_2.x + a_1.y * a_2.y * a_1.p.discriminant;

    y = a_1.x * a_2.y + a_1.y * a_2.x;
    x = h;
    z = a_1.z * a_2.z;
```

```
    bigint g = gcd(x,y,z);
    x = x/g; y = y/g; z = z/g;
  }
```

Set $n = max\{\text{size}\, a_1, \text{size}\, a_2\}$. The gcd can be computed in time $O(M(n)\log n)$. And this is the overall running time of the procedure, because the gcd computation dominates the remaining parts. If a quadratic time integer multiplication algorithm is used, then the gcd can be computed in time $O(n^2)$, and again this is the overall running of the procedure in that case. In the following we will also use the notation $a = a_1 a_2$ for $a.\texttt{multiply}(a_1, a_2)$.

Let $a = (x, y, z, p)$ be a non-zero number in Δ-standard representation. The procedure $a.\texttt{invert}()$ transforms a into the standard representation of $1/a$.

```
void a.invert()
  {
    bigint h = x^2 - y^2 * p.discriminant;
    x = z * x;
    y = -z * y;
    z = h;

    bigint g = gcd(x,y,z);
    x = x/g; y = y/g; z = z/g;
  }
```

Computing x, y, and z before the gcd computation takes time $O(M(\text{size}\, a))$, and their size is $O(\text{size}\, a)$. Therefore, the gcd computation takes time $O(M(\text{size}\, a)\log \text{size}\, a)$, and this is the overall running time. In case of a quadratic time integer multiplication algorithm, the gcd can be computed in quadratic time, and the overall running time is $O((\text{size}\, a)^2)$.

4.1.2. Approximating a real quadratic number. We describe the function $a.\texttt{relative\_approximation}(k)$, that computes a relative k-approximation to the real quadratic number $a = (x, y, z, p)$, for $k \geq 1$.

```
xbigfloat a.relative_approximation (int k)
 {
   xbigfloat d, u, v, w;

   if (a == 0)
      w = 0;

   if (y == 0)
      w = divide(x,z,k);
   else {
      bigint Delta = p.discriminant();
      int t = k;

      if (sign(x) != sign(y)) {
         int bd = b(Delta);
         if (bd & 1)
            bd = (bd+1) / 2;
```

```
        else
           bd = bd / 2;

        t += b(y)+bd + max(b(x),b(y)+bd) - b(x^2-y^2 Delta) + 2;
        t = max(t,1);
     }

     d = sqrt (Delta, t+5);
     u = truncate (truncate(y,t+5) * d, t+5);
     v = u + x;
     w = divide (v, z, k+3);
     w = truncate (w, k+3));
   }

  return w;
}
```

Before we prove the correctness of the procedure and analyze its running time, we need some auxiliary results. First, note that the b-value of $\sqrt{\Delta}$ is

$$b(\sqrt{\Delta}) = \begin{cases} b(\Delta)/2, & \text{if } b(\Delta) \text{ even,} \\ (b(\Delta)+1)/2, & \text{if } b(\Delta) \text{ odd.} \end{cases}$$

The crucial part in the approximation of a is the size of $|y\sqrt{\Delta}/(x+y\sqrt{\Delta})|$. Let $x + y\sqrt{\Delta} \neq 0$. By computing b-values, it immediately follows from (4.1.2), that

$$\text{(4.1.9)} \qquad \frac{|y\sqrt{\Delta}|}{|x+y\sqrt{\Delta}|} \leq \begin{cases} 1, & \text{if } xy \geq 0, \\ 2^{b(y\sqrt{\Delta})+max\{b(x),b(y\sqrt{\Delta})\}-b(x^2-y^2\Delta)+2}, & \text{if } xy < 0. \end{cases}$$

PROPOSITION 4.1.2. *For an object $a = (x, y, z, p)$ of type* ***quadratic_number_standard****, and on input of an integer $k \geq 1$, the function a.****relative_approximation****(k) returns a relative k-approximation to $(x + y\sqrt{\Delta})/z$ in time $O(M(k) + \operatorname{size}(a))$, if $xy \geq 0$, and $O(M(k + \operatorname{size}(d(a)a)) + \operatorname{size}(a))$, if $xy < 0$.*

PROOF. We use the notation of the algorithm. First, we prove the correctness of the function. If $a = 0$ or $y = 0$, then w is a relative k-approximation. Suppose that $a, y \neq 0$. By (2.3.1) and Lemma 2.3.2, we have

$$u = y\sqrt{\Delta}(1+\varepsilon_1), |\varepsilon_1| < 2^{-t-2}.$$

This implies

$$v = (x + y\sqrt{\Delta})(1 + \varepsilon_1 y\sqrt{\Delta}/(x + y\sqrt{\Delta})).$$

It follows from (4.1.9) that

$$v = (x + y\sqrt{\Delta})(1+\varepsilon_2), |\varepsilon_2| < 2^{-k-2}.$$

Thus, because $k \geq 1$, we obtain

$$\begin{aligned} w &= (x+y\sqrt{\Delta})/z(1+\varepsilon_2)(1+\varepsilon_3),\ |\varepsilon_3| < 2^{-k-3} \\ &= (x+y\sqrt{\Delta})/z(1+\varepsilon_4),\ |\varepsilon_4| < 2^{-k-1}. \end{aligned}$$

It follows from Lemma 2.3.2, that the truncated w is a relative k-approximation.

Now we analyze the running time. The square root extraction for determining d takes time $O(M(t) + \operatorname{size}\Delta)$ and $|b(d)| = O(\operatorname{size}\Delta)$. u can be found in time

$O(M(t) + \operatorname{size} y + \operatorname{size} \Delta)$ and $|b(u)| = O(\operatorname{size} y + \operatorname{size} \Delta)$. Hence, computing v takes time $O(t + \operatorname{size} x + \operatorname{size} y + \operatorname{size} \Delta)$ and $\operatorname{size} v = O(t + \operatorname{size} x + \operatorname{size} y + \operatorname{size} \Delta)$. Therefore, the division takes time $O(M(t) + \operatorname{size} a)$. This implies an overall running time of $O(M(t) + \operatorname{size} a)$.

If $xy \geq 0$, then $t = k$. Otherwise, $t = O(k + \operatorname{size}(d(a)a))$. This proves the assertions. □

4.1.3. Approximating the logarithm function ln**.** For several applications, for example approximating the regulator, it is necessary to compute logarithms of quadratic numbers. In this section we will describe how to approximate the logarithm function ln. We recall that it is defined as

$$K^* \to \mathbb{R}, \alpha \mapsto \ln |\alpha|.$$

Because we already know how to approximate $a = (x, y, z, p)$, it is very easy to implement a function a.`absolute_ln_approximation`(k), that returns an absolute k-approximation to $\ln |a|$ for $k \geq 0$.

```
xbigfloat a.absolute_ln_approximation(int k)
 {
   xbigfloat w = a.relative_approximation(k+3);
   xbigfloat l = log(|w|, k+2);
   return truncate(l,k+1+b(l));
 }
```

Proposition 4.1.3. *For an object $a = (x, y, z, p)$ of type* ***quadratic_number_standard****, and on input of $k \in \mathbb{Z}_{\geq 0}$, the function a.**absolute_ln_approximation**(k) returns an absolute k-approximation to $\ln |a|$. If $xy > 0$, its bit complexity is $O(M(t) \log t + \operatorname{size}(a))$, and if $xy \leq 0$, it is $O(M(t) \log t + M(\operatorname{size}(d(a)a)) + \operatorname{size}(a))$, where $t = k + \log(\operatorname{size}(a))$.*

Proof. We prove the correctness of the procedure. It follows from Proposition 4.1.2 and Lemma 2.4.4, that $l = \ln |a|(1+\varepsilon)$, $|e| < 2^{-k-1}$, before truncation. (Note, that the fact, that w is a relative $k + 3$-approximation to a, implies that $|w|$ is a relative $k + 3$-approximation to $|a|$, because $k + 3 > 1$.) It follows from Lemma 2.3.1, that the truncated l is an absolute k-approximation.

We estimate the running time. By Proposition 4.1.2, the time for approximating a is $O(M(k)+\operatorname{size}(a))$, if $xy \geq 0$, and $O(M(k+\operatorname{size}(d(a)a))+\operatorname{size}(a))$, if $xy < 0$. Because $k + 3 \geq 1$, it follows from Lemma 2.2.2 and (4.1.3), that

$$|b(w)| \leq |b(a)| + 1 \leq \operatorname{size}(a) + 1.$$

Together with Theorem 2.4.1, we obtain, that l can be computed in time $O(M(k+\log(\operatorname{size}(a))) \log(k+\log(\operatorname{size}(a)))+\operatorname{size}(a))$. This yields the asserted overall running time. □

4.1.4. Approximating the logarithm function Ln**.** We will show how to compute absolute approximations to the logarithm function Ln. We recall that it is defined as

$$\begin{aligned} \operatorname{Ln} :\ & K^* \to \mathbb{R}, \\ & \alpha \mapsto \operatorname{Ln} \alpha = (1/2) \ln |\alpha/\sigma(\alpha)| = \ln |\alpha| - (1/2) \ln |N\alpha|. \end{aligned}$$

We describe the function a.`absolute_Ln_approximation`(k), that on input of a quadratic number $a = (x, y, z, p)$ and $k \in \mathbb{Z}_{\geq 0}$ computes an absolute k-approximation to $\operatorname{Ln}(a)$.

```
xbigfloat a.absolute_Ln_approximation (int k)
 {
   quadratic_number_standard b = (x,y,1,p);
   quadratic_number_standard c = (x,-y,1,p);

   xbigfloat u = b.relative_approximation(k+3);
   xbigfloat v = c.relative_approximation(k+3);

   xbigfloat w = divide(u,v,k+4);
   xbigfloat l = log(|w|, k+1) / 2;

   return truncate(l, k+1+b(l));
 }
```

PROPOSITION 4.1.4. *For an object* $a = (x, y, z, p)$ *of type* ***quadratic_number_standard****, and on input of* $k \in \mathbb{Z}_{\geq 0}$*, the function* $a.$***absolute_Ln_approximation***(k) *computes an absolute* k*-approximation to* $\operatorname{Ln} a$ *in time* $O(M(t)\log t + M(\operatorname{size}(d(a)a)) + \operatorname{size} a)$*, where* $t = k + \log(\operatorname{size}(d(a)a))$.

PROOF. We prove the correctness of the procedure. It follows from Proposition 4.1.2 that

$$u = (x + y\sqrt{\Delta})(1 + \varepsilon_1),\ |\varepsilon_1| < 2^{-k-3},$$
$$v = (x - y\sqrt{\Delta})(1 + \varepsilon_2),\ |\varepsilon_2| < 2^{-k-3}.$$

Because $k \geq 0$, we obtain

$$\begin{aligned} |w| &= |a/\sigma(a)|(1+\varepsilon_1)(1+\varepsilon_3)/(1+\varepsilon_2),\ |\varepsilon_3| < 2^{-k-4},\\ &= |a/\sigma(a)|(1+\varepsilon_4), |\varepsilon_4| < 2^{-k-2}. \end{aligned}$$

Thus, it follows from Lemma 2.4.4, that the first value of l satisfies $l = \ln|a/\sigma(a)|(1+\varepsilon_5)$, $|e_5| < 2^{-k-1}$. The truncated l is an absolute k-approximation by Lemma 2.3.1.

We estimate the running time. By Proposition 4.1.2, the time for computing u and v is $O(M(k + \operatorname{size}(d(a)a)) + \operatorname{size}(a))$. This dominates the time for computing w. Because $k + 2 \geq 1$, it follows from Lemma 2.2.2 and (4.1.4), that

$$|b(w)| \leq |b(b/\sigma(b))| + 1 \leq 2\ \operatorname{size}(b) + 2 \leq 2\ \operatorname{size}(d(a)a) + 2.$$

Together with Theorem 2.4.1, we obtain, that l can be computed in time $O(M(k + \log(\operatorname{size}(d(a)a)))\log(k + \log(\operatorname{size}(d(a)a))) + \operatorname{size}(d(a(a))$. This yields the asserted overall running time. ☐

Neglecting the advantage of not having to deal with denominators, the running time for the Ln approximation as stated in Proposition 4.1.4 is worse than the running time for the approximation of ln stated in Proposition 4.1.3, in the case that the coefficients x,y of $\alpha = (x + y\sqrt{\Delta})/z$ have the same sign. This is because either the coefficients of α or the coefficients of $\sigma(\alpha)$ have different signs. One might think that the asymptotical running time for the approximation of Ln can be improved, if we use the properties (see Lemma 1.4.1)

$$\operatorname{Ln}\alpha = \ln|\alpha| - (1/2)\ln|N\alpha|, \tag{4.1.10}$$

and

$$\operatorname{Ln}\alpha = sign(x)\, sign(y)\ \operatorname{Ln}(|x| + |y|\sqrt{\Delta}). \tag{4.1.11}$$

By (4.1.11) we can always achieve, that the coefficients of the Ln argument have the same sign. Then we use (4.1.10) to approximate the Ln value. Here, because of the same sign, the quadratic number approximation can be accerelated from $M(k+\operatorname{size}(d(\alpha)\alpha))$ to $M(k)$. But in addition, we also have to compute the norm, which requires time $M(\operatorname{size}(d(\alpha)\alpha))$; and the norm approximation cannot be improved to $M(k)$, because the numbers in the sum $x^2 - y^2\Delta$ have different signs. Hence, the asymptotical running time remains the same.

We estimate the size of approximations to Ln.

LEMMA 4.1.5. *There is a constant $c \in \mathbb{N}$, such that for any real quadratic order O with field of fractions K, any $\alpha \in K^*$, $k \in \mathbb{Z}$, and any floating point number (m, e), which is an absolute k-approximation to $\operatorname{Ln}\alpha$, we have*

$$b(m), |e| \le c \log\log((d(\alpha))^2 H(\alpha)\Delta) + |k|,$$

PROOF. Use Lemma 2.4.5 together with (4.1.4) and (4.1.6). □

4.1.5. Estimating Ln. Let

$$\alpha = (x + y\sqrt{\Delta})/z$$

be a real quadratic number with $x, y \neq 0$. When computing relative approximations to $\operatorname{Ln}\alpha$, it is necessary to know an estimate on $|\operatorname{Ln}\alpha|$ beforehand. In this section we derive such an estimate by using the fact that the natural logarithm can be estimated by its argument, if the argument is close to 1, and by the logarithm to base 2, if the argument is large.

To decide whether the argument of the logarithm is close to 1, we introduce the value

$$c(\alpha) = \begin{cases} |x|/(|y|\sqrt{\Delta}), & \text{if } |x| > |y|\sqrt{\Delta}, \\ |y|\sqrt{\Delta}/|x|, & \text{if } |x| < |y|\sqrt{\Delta}. \end{cases}$$

For simplicity, we set $c = c(\alpha)$.

LEMMA 4.1.6. *We have $c > 1$ and $\operatorname{Ln}\alpha = (1/2)sign(x)sign(y)\ln(1+2/(c-1))$.*

PROOF. Because $\alpha \notin \mathbb{Q}$, we have $c > 1$. Lemma 1.4.1 states, that

$$\begin{aligned} \operatorname{Ln}\alpha &= sign(x)sign(y)\operatorname{Ln}(|x| + |y|\sqrt{\Delta}) \\ &= (1/2)sign(x)sign(y)\ln|(|x| + |y|\sqrt{\Delta})/(|x| - |y|\sqrt{\Delta})|. \end{aligned}$$

First assume that $|x| > |y|\sqrt{\Delta}$. Then $c = |x|/(|y|\sqrt{\Delta})$ and $|(|x| + |y|\sqrt{\Delta})/(|x| - |y|\sqrt{\Delta})| = |(c+1)/(c-1)| = (c+1)/(c-1) = 1 + 2/(c-1)$.

Now assume that $|x| < |y|\sqrt{\Delta}$. Then $c = |y|\sqrt{\Delta}/|x|$ and $|(|x| + |y|\sqrt{\Delta})/(|x| - |y|\sqrt{\Delta})| = |(1+c)/(1-c)| = (c+1)/(c-1) = 1 + 2/(c-1)$.

So, in both cases, we have $|(|x| + |y|\sqrt{\Delta})/(|x| - |y|\sqrt{\Delta})| = 1 + 2/(c-1)$, and therefore $\operatorname{Ln}\alpha = (1/2)sign(x)sign(y)\ln(1 + 2/(c-1))$. □

The lemma shows that the argument of the logarithm function becomes closer to 1 the larger c is. We can estimate the logarithm as follows.

COROLLARY 4.1.7. *If $c > 3$, then $1/(c+1) \le |\operatorname{Ln}\alpha| \le 1/(c-1)$.*

PROOF. Apply Lemma 2.4.2 and Lemma 4.1.6. □

COROLLARY 4.1.8. *If* $1 < c \leq 3$*, then* $d/4 < |\operatorname{Ln}\alpha| < (d+2)/2$ *for* $d = -b(c-1)+1$.

PROOF. Note that $c < 3$ implies $d \geq 1$. It follows from the definition of the b-value, that d is the minimal integer with

$$c \geq 1 + 2^{-d}. \tag{4.1.12}$$

It follows from (4.1.12), that $1 + 2/(c-1) \leq 2^{d+2}$. Together with Lemma 4.1.6 this implies

$$|\operatorname{Ln}\alpha| = (1/2)\ln(1+2/(c-1)) \leq (1/2)\ln(2^{d+2}) = (\ln 2)(d+2)/2 < (d+2)/2.$$

Similarly, it follows from (4.1.12), that $c < 1 + 2^{-d+1}$. Hence, $1 + 2/(c-1) \geq 2/(c-1) > 2^d$. Together with Lemma 4.1.6 this implies

$$|\operatorname{Ln}\alpha| = (1/2)\ln(1+2/(c-1)) > (1/2)\ln(2^d) = (\ln 2)d/2 > d/4.$$

□

Let us make the remaining technical details clear. First, we show how to decide whether $c > 3$. Set

$$m = min\{x^2, y^2\Delta\},$$

and let

$$q, r \in \mathbb{Z},$$

such that

$$max\{x^2, y^2\Delta\} = qm + r, \quad 0 \leq r < m.$$

Then we have $c^2 = q + r/m$. We can easily decide whether $c > 3$. We obtain

$$c > 3 \Leftrightarrow q \geq 10 \text{ or } q = 9 \text{ and } r > 0. \tag{4.1.13}$$

Next we describe how we use Corollary 4.1.8 to compute bounds on $|\operatorname{Ln}\alpha|$ in case of $1 < c \leq 3$. Let u be a relative 1-approximation to $c-1$ and set

$$d' = -b(u) + 1.$$

It follows from Lemma 2.2.2 that for $d = -b(c-1)+1$ we have $|d' - d| \leq 1$. By Corollary 4.1.8

$$U = (d'+3)/2$$

satisfies $|\operatorname{Ln}\alpha| < U$. Furthermore, set

$$L = \begin{cases} (d'+1)/4, & \text{if } d' = 0, \\ d'/4, & \text{if } d' = 1, \\ (d'-1)/4, & \text{if } d' > 1. \end{cases}$$

It follows from $d \geq 1$, $|d'-d| \leq 1$, and Corollary 4.1.8, that $0 < L < |\operatorname{Ln}\alpha|$. Hence, we obtain

$$0 < L < |\operatorname{Ln}\alpha| < U, \quad 2(L+1) \geq U. \tag{4.1.14}$$

To derive a relative 1-approximation to $c-1$ we need to know an estimate on

$$d = -b(c-1) + 1$$

beforehand. We derive such an estimate. Let q, r and m be as defined above. It follows from $c^2 = q + r/m$, that $c - 1 = (c^2 - 1)/(c + 1) = (q - 1 + r/m)/(c + 1)$. Because $1 < c \leq 3$, we obtain

$$(q - 1 + r/m)/4 \leq c - 1 < (q - 1 + r/m)/2. \tag{4.1.15}$$

Because $c \leq 3$, we have $q \leq 9$. And if $q \geq 2$, then (4.1.15) implies $1/4 \leq |c-1| < 4$. By taking b-values this yields $-1 \leq b(c - 1) \leq 2$, and we obtain the bounds

$$-1 \leq d \leq 2, \quad \text{if } 2 \leq q \leq 9. \tag{4.1.16}$$

If $q = 1$, then (4.1.15) implies $r/(4m) \leq c - 1 < r/(2m)$. Note that $r > 0$ in this situation, because Δ is not a perfect integral square. By looking at the b-values we obtain $2^{b(r)-b(m)-3} < |c - 1| < 2^{b(r)-b(m)}$, and we can bound the b-value of $c - 1$ by $b(r) - b(m) - 2 \leq b(c - 1) \leq b(r) - b(m)$. This implies

$$-b(r) + b(m) + 1 \leq d \leq -b(r) + b(m) + 3, \quad \text{if } q = 1. \tag{4.1.17}$$

Because $c > 1$, the case $q = 0$ cannot occur.

Next we show how to use Corollary 4.1.7 to derive bounds on $|\operatorname{Ln}\alpha|$ for $c > 3$. Let l and u be relative 4-approximations to $1/(c + 1)$ and $1/(c - 1)$, respectively, i.e.

$$\begin{aligned} l &= (1 + \varepsilon_1)/(c + 1), \quad |\varepsilon_1| < 1/16, \\ u &= (1 + \varepsilon_2)/(c - 1), \quad |\varepsilon_2| < 1/16. \end{aligned}$$

We have

$$(16/15)l > 1/(c + 1) = l/(1 + \varepsilon_1) > (16/17)l > 0.9375\, l,$$

and

$$1.09375\, u > (16/15)u > 1/(c - 1) = u/(1 + \varepsilon_2) > (16/17)u.$$

Set

$$L = 0.9375\, l, \quad U = 1.09375\, u.$$

It follows from Corollary 4.1.7 that

$$0 < L < |\operatorname{Ln}\alpha| < U, \quad L \cdot \omega > U, \tag{4.1.18}$$

where $\omega = 17 \cdot 1.09375/(15 \cdot 0.9375) \cdot (c + 1)/(c - 1) < 3$.

We present the function α.`estimate_Ln`(L, U) that on input of a real quadratic number $\alpha = (x, y, z, p)$ with $x, y \neq 0$ returns L and U with $0 < L < |\operatorname{Ln}\alpha| < U$. If $c \leq 3$, then L and U satisfy (4.1.14). Otherwise, L and U satisfy (4.1.18).

```
void alpha.estimate_Ln(xbigfloat & L, xbigfloat & U)
 {
   bigint m, n, Delta = p.discriminant;

   if (x^2 > y^2 Delta) {
      m = y^2 Delta;
      n = x^2;
    }
   else {
      m = x^2;
      n = y^2 Delta;
    }
```

```
   bigint k, g, q, r;
   div_rem (q,r,n,m);

   bool c_is_gt_3 = (q >= 10) || (q == 9 && r > 0);

   if (c_is_gt_3)
      k = 7;
   else {
      if (q == 1)
         g = -b(r)+b(m)+3;
      else
         g = 2;

      k = g+3;
    }

   xbigfloat a,b,c,D;

   if (x^2 > y^2 Delta) {
      a = truncate(absolute_value(x),k+3);
      b = truncate(absolute_value(y),k+4);
      D = sqrt(Delta, k+4);
      c = divide(a, b*D, k+3);
    }
   else {
      a = truncate(absolute_value(x),k+4);
      b = truncate(absolute_value(y),k+3);
      D = sqrt(Delta, k+3);
      c = divide(b*D, a, k+3);
    }

   if (c_is_gt_3) {
       l = divide(1,c+1,6);
       u = divide(1,c-1,6);
       L = (1/2 + 1/4 + 1/8 + 1/16) * l;
       U = (1 + 1/16 + 1/32) * u;
    }
   else {
       xbigfloat d' = -b(c-1)+1;
       U = (d'+3)/2;
       if (d' == 0)
          L = (d'+1)/4;
       else if (d' == 1)
          L = d'/4;
       else
          L = (d'-1)/4;
    }
}
```

PROPOSITION 4.1.9. *On input of a real quadratic number* $\alpha = (x + y\sqrt{\Delta})/z$ *with* $x, y \neq 0$*, the procedure* α.***estimate_Ln***(L, U) *returns floating point numbers* L *and* U *with* $0 < L < |\operatorname{Ln}\alpha| < U$*. If* $c(\alpha) \leq 3$*, then* $2(L+1) \geq U$*. Otherwise,* $3L \geq U$*.*

PROOF. We use the notation of the procedure. It follows from (4.1.13) that the value of `c_is_gt_3` is correct, i.e., it is true if and only if $c(\alpha) > 3$. Furthermore, it follows from (4.1.17) and (4.1.16) that g has the property

$$c(\alpha) \geq 1 + 2^{-g}.$$

Furthermore, it is $k = 7$, if $c(\alpha) > 3$, and $k = g + 3$, otherwise. We examine the accuray of the approximation c to $c(\alpha)$. Assume that $|x| > |y|\sqrt{\Delta}$. Then

$$\begin{aligned} a &= x(1+\varepsilon_1), |\varepsilon_1| < 2^{-k-3}, \\ b &= y(1+\varepsilon_2), |\varepsilon_2| < 2^{-k-4}, \\ D &= \sqrt{\Delta}(1+\varepsilon_3), |\varepsilon_3| < 2^{-k-4}, \\ c &= a/(bD)(1+\varepsilon_4), |\varepsilon_4| < 2^{-k-3}. \end{aligned}$$

Hence, we have $c = c(\alpha)(1+\varepsilon_1)(1+\varepsilon_4)/((1+\varepsilon_2)(1+\varepsilon_3)) = c(\alpha)(1+\varepsilon_5)$. Lemma 2.5.2 and Lemma 2.5.3 yield

$$c = c(\alpha)(1+\varepsilon_5), \quad |\varepsilon_5| < 2^{-k}. \tag{4.1.19}$$

In case $|x| < |y|\sqrt{\Delta}$ the algorithm enters the else part, and the proof of (4.1.19) is analogous. Furthermore, it follows from (4.1.19) that

$$c - 1 = c(\alpha)(1+\varepsilon_5) - 1 = (c(\alpha) - 1)(1+\varepsilon_6), \tag{4.1.20}$$

where $\varepsilon_6 = c(\alpha)\varepsilon_5/(c(\alpha) - 1)$.

Assume that $c(\alpha) \leq 3$. Then the algorithm enters the else-part of the last if-else statement. In this case we have $|\varepsilon_6| \leq 3 \cdot 2^g |\varepsilon_5| < 1/2$. Hence,

$$c - 1 = (c(\alpha) - 1)(1+\varepsilon_6), \quad |\varepsilon_6| < 1/2,$$

and the assertion follows from (4.1.14).

Suppose that $c(\alpha) > 3$. Then the algorithm enters the if-part. In that case we have $|\varepsilon_6| < 2|\varepsilon_5| < 2^{-6}$, so

$$c - 1 = (c(\alpha) - 1)(1+\varepsilon_6), \quad |\varepsilon_6| < 2^{-6}.$$

Furthermore, we have

$$\begin{aligned} u &= 1/(c-1) \cdot (1+\varepsilon_7)/(1+\varepsilon_6), \quad |\varepsilon_7| < 2^{-6}, \\ &= 1/(c-1) \cdot (1+\varepsilon_8), \qquad |\varepsilon_8| < 2^{-4}. \end{aligned}$$

The same considerations show that

$$l = 1/(c+1) \cdot (1+\varepsilon_9), \qquad |\varepsilon_9| < 2^{-4}.$$

Hence, the assertion follows from (4.1.18). □

Let $\alpha = (x, y, z, p)$ be a real quadratic number. The following procedure α.`sign_of_Ln`() returns $sign(\operatorname{Ln}\alpha)$.

```
int alpha.sign_of_Ln()
 {
   bigint Delta = p.discriminant;
   int sign_a, sign_b;
   quadratic_number_standard beta;

   sign_a = alpha.sign();
   sign_b = alpha.conjugate().sign();

   beta = (x (sign_a-sign_b), y (sign_a+sign_b), 1, p);
   return beta.sign();
 }
```

LEMMA 4.1.10. *On input of a real quadratic number α the procedure α.`sign_of_Ln`() returns $sign(Ln\alpha)$.*

PROOF. We use the notation of the procedure. It is $sign(\operatorname{Ln}\alpha) = sign(|\alpha| - |\sigma\alpha|)$. Let $sign\_a = sign(a)$ and $sign\_b = sign(\sigma\alpha)$. We have $\beta = x(sign\_a - sign\_b) + y\sqrt{\Delta}(sign\_a + sign\_b) = sign\_a \cdot a - sign\_b \cdot b = z(|a| - |b|)$. Hence, $sign(\operatorname{Ln}\alpha) = sign(\beta)$. □

4.2. Power product representation: `quadratic_number_power_product`

The standard representation for quadratic numbers introduced in Section 4.1 is not appropriate for the fundamental unit of O. For example, Lagarias [**Lag80**] shows that there is an infinite set of quadratic orders, such that the binary length of the fundamental unit is exponential in $\log\Delta$. Therefore, if we use the standard representation no polynomial time algorithm for computing the fundamental unit can exist. To circumvent this problem we follow Buchmann, Thiel, and Williams ([**BTW95**]) and introduce a power product representation, where the base elements are in standard representation and the exponents are integers. We explain how to find a power product representation of the fundamental unit whose size is polynomial in $\log\Delta$ in Chapter 7.

Let $\alpha \in K$. A pair of vectors $a = ((\alpha_1, \ldots, \alpha_n), (e_1, \ldots, e_n))$ with integers e_i and α_i of type `quadratic_number_standard` is a ($\Delta-$)*power product representation* of α, if

$$\alpha = \prod_{i=1}^{n} \alpha_i^{e_i}.$$

The power product representation is implemented by the data type `quadratic_number_power_product`. Often we will identify a with the number α represented by it. We set

$$\operatorname{size}(a) = \operatorname{size}\{\alpha_1, \ldots, \alpha_n\} + \sum_{i=1}^{n} \operatorname{size}(e_i).$$

A power product representation $((\alpha_1, \ldots, \alpha_n, \gamma), (2^{n-1}, \ldots, 2^0, 1))$ of $\alpha \in K^*$ is called *reduced* or *compact*, if

$$n \leq 2 + \log\log max\{H(\alpha)d(\alpha), 2\},$$

$$\gamma \in Min(\alpha O), \quad d(\gamma) \leq d(\alpha), \quad H(\gamma) \leq |N(\alpha)|d(\alpha),$$

and

$$0 < d(\alpha_j) \le \Delta^{1/2}, \quad H(d(\alpha_j)\alpha_j) \le 2\Delta^{7/4} \text{ for } 1 \le j \le n.$$

This definition of reduced power products is an extension of the definition of compact representation given in [**BTW95**] from integers to all numbers in K^*.

4.2.1. Operations for power products. We will describe basic operations for power product representations of quadratic numbers and estimate their bit complexity. Let $a = ((a_1, \dots, a_n), (e_1, \dots, e_n))$ be a power product representation of $\alpha \in K$.

There is a function a.`length`() that returns n. Also, there are functions a.`base`(i), a.`exponent`(i) which return a_i, e_i, for $1 \le i \le n$, respectively.

Inversion of a, i.e., $a = ((a_1, \dots, a_n), (-e_1, \dots, -e_n))$, is done by the function a.`invert`(). The function a.`negate`() computes $a = ((-a_1, a_2, \dots, a_n), (e_1, \dots, e_n))$. and a.`square`() determines $a = ((a_1, \dots, a_n), (2\,e_1, \dots, 2\,e_n))$, i.e., a representation for α^2. Their bit complexity is $O(\text{size}(a))$.

Let b be the Δ-standard representation of a number $\beta \in K$. The function c.`multiply`(a, b) computes $c = ((a_1, \dots, a_n, b), (e_1, \dots, e_n, 1))$, i.e., a representation for the product $\alpha\beta$, in time $O(\text{size}(a) + \text{size}(b))$. The function c.`divide`(a, b), that computes $c = ((a_1, \dots, a_n, b), (e_1, \dots, e_n, -1))$, has the same complexity as multiplication.

Furthermore, there is a function a.`invert_base_elements`() that transforms a into $((1/a_1, \dots, 1/a_n), (e_1, \dots, e_n))$. It follows from the results of Section 4.1.1, that the running time of the function is $O(\text{size}\, a + \sum_{i=1}^{n} M(\text{size}\, a_i) \log \text{size}\, a_i)$. If a quadratic time multiplication algorithm is used, then the running time of the function is $O(\text{size}\, a + \sum_{i=1}^{n} (\text{size}\, a_i)^2)$.

And the function a.`multiply_base_element`(i, b) transforms a into $((a_1, \dots, a_{i-1}, a_i \cdot b, a_{i+1}, \dots, a_n), (e_1, \dots, e_n))$ for $1 \le i \le n$. Set $X = max\{\text{size}\, a_i, \text{size}\, b\}$. Then it follows from the results of Section 4.1.1, that the running time of the function is $O(\text{size}\, a + M(X) \log X)$. Again, if a quadratic time multiplication algorithm is used, the running time of the function is $O(\text{size}\, a + X^2)$.

4.2.2. Approximating logarithms. Let $a = ((a_1, \dots, a_n), (e_1, \dots, e_n))$ be a power product representation for $\alpha \in K^*$. We describe the function a.`absolute_ln_approximation`(k), that returns an absolute k-approximation to $\ln|\alpha|$ for $k \in \mathbb{Z}$.

```
xbigfloat a.absolute_ln_approximation (int k)
 {
   int i, n = a.length();
   xbigfloat u, v, l = 0;

   for (i=1; i <= n; i++)
    {
       u = a_i.absolute_ln_approximation(max(k+b(e_i)+b(n)+2, 0));
       v = e_i * u;
       l = l + truncate(v, k+b(n)+2+b(v));
    }
   return truncate(l, k+1+b(l));
```

```
}
```

PROPOSITION 4.2.1. *Let $a = ((a_1, \ldots, a_n), (e_1, \ldots, e_n))$ be a non-zero real quadratic number, and $k \in \mathbb{Z}$. The function a.`absolute_ln_approximation`(k) returns an absolute k-approximation to $\ln|a|$ in time $O(nM(C+B)\log(C+B)+A+\mathrm{size}(a))$, where $B = max_{1\le i\le n}\{b(e_i) + \log \mathrm{size}(a_i)\}$, $A = \sum_{i=1}^{n} M(\mathrm{size}(d(a_i)a_i))$, and $C = max\{max_{1\le i\le n}\{k + b(n) + b(e_i) + 2\}, 0\}$.*

PROOF. We use the notation of the procedure. We prove its correctness. It follows from Proposition 4.1.3 and Lemma 2.3.1, that the truncated v satisfies $|v - e_i \ln|a_i|| < 2^{-k-b(n)-1}$ in the i-th iteration of the loop for $1 \le i \le n$. Thus, the sum l satisfies $|l - \ln|a|| < 2^{-k-1}$, after the loop, but before truncation. Lemma 2.3.1 yields, that the truncated l is an absolute k-approximation to $\ln|a|$.

We examine the running time. Let $c_i = max\{k + b(e_i) + b(n) + 2, 0\}$. It follows from Proposition 4.1.3, that the computation of u in the i-th iteration of the loop takes time $O(M(c_i + \log(\mathrm{size}(a_i)))\log(c_i + \log(\mathrm{size}(a_i))) + M(\mathrm{size}(d(a_i)a_i)) + \mathrm{size}(a_i))$. By Lemma 2.4.5, it is

$$|b(u)|, b(m(u)) \le b(1 + |b(a_i)|) + c_i + 2,$$

and we know from (4.1.3), that $|b(a_i)| \le |\,\mathrm{size}(a_i)|$. Hence,

$$|b(u)|, b(m(u)) = O(\log(\mathrm{size}(a_i)) + c_i),$$

and the time for computing v is $O(M(\log(\mathrm{size}(a_i)) + b(e_i) + c_i))$. Thus, the time for computing all n values of u and v can be estimated by

$$O(nM(C + B)\log(C + B) + A + \mathrm{size}(a)).$$

Obviously, it is $|b(v)|, b(m(v)) = O(\log(\mathrm{size}(a_i)) + b(e_i) + c_i)$, too. Hence, Theorem 2.3.3 implies, that the sum l can be computed in time $O(n(C+B))$. This is dominated by the computation of u and v. □

REMARK 4.2.2. In the situation of Proposition 4.2.1, let $a_i = (x_i, y_i, z_i, p)$ for $1 \le i \le n$. If $x_i y_i > 0$ for all $1 \le i \le n$, then it follows from Proposition 4.1.3, that the term A in the running time of the procedure a.`absolute_ln_approximation`(k) can be omitted.

REMARK 4.2.3. Let $a = ((a_1, \ldots, a_n), (e_1, \ldots, e_n))$ be a non-zero real quadratic number, and $k \in \mathbb{Z}$. A function a.`absolute_Ln_approximation`(k), that returns an absolute k-approximation to $\mathrm{Ln}(a)$ works in the same way as the function for the ln approximation above, except that the call of a_i.`absolute_ln_approximation` is replaced by a_i.`absolute_Ln_approximation`.

It follows from Proposition 4.1.4, that the running time of the procedure a.`absolute_Ln_approximation`(k) is $O(nM(C + B)\log(C + B) + A + \mathrm{size}(a))$, where $B = max_{1\le i\le n}\{\mathrm{size}(e_i) + \log \mathrm{size}(d(a_i)a_i)\}$, $A = \sum_{i=1}^{n} M(\mathrm{size}(d(a_i)a_i))$ and $C = max\{max_{1\le i\le n}\{k + b(n) + b(e_i) + 2\}, 0\}$.

To prove the asserted running time, we may follow the proof of Proposition 4.1.3, if we note that,

$$|b(u)|, b(m(u)) \le b(1 + |b(a_i/\sigma(a_i))|) + c_i + 2,$$

thus, $|b(u)|, b(m(u)) = O(\log(\mathrm{size}(d(a_i)a_i)) + c_i)$ by (4.1.4).

We estimate the time for approximating $\mathrm{Ln}\,\alpha$, when α is in reduced power product representation.

THEOREM 4.2.4. *Let $\alpha \in K^*$ be in reduced power product representation $((\alpha_1, \dots, \alpha_n, \gamma), (2^{n-1}, \dots, 2^0, 1))$, and $k \in \mathbb{Z}_{\geq 0}$. An absolute k-approximation to $\operatorname{Ln} \alpha$ can be determined in time $O(n[M(k + n + \log\log \Delta) \log(k + n + \log\log \Delta) + M(\log \Delta)] + M(k + \log t) \log(k + \log t) + M(t)$, where $t = \log(|N(\alpha)| d^3(\alpha)\Delta)$ and $n = 2 + \log\log max\{H(\alpha)d(\alpha), 2\}$.*

PROOF. The approximation is computed as follows. First, we determine absolute $k + 2$-approximations c and b to $\operatorname{Ln} \gamma$ and $\operatorname{Ln} \prod_{j=1}^{n} \left(\frac{\alpha_j}{d_j}\right)^{2^{n-j}}$, respectively. Then we add c and b, and truncate the sum to obtain an absolute k-approximation.

We estimate the complexity. It follows from Proposition 4.1.4, that c can be determined in time $O(M(k+\log \operatorname{size}(d(\gamma)\gamma)) \log(k+\log \operatorname{size}(d(\gamma)\gamma))+M(\operatorname{size}(d(\gamma)\gamma))+ \operatorname{size} \gamma)$. Because α is reduced, we have

$$d(\gamma) \leq d(\alpha),\ H(\gamma) \leq |N(\alpha)|d(\alpha).$$

Set $t = \operatorname{size} \gamma$. It follows from (4.1.6), that $\operatorname{size}(d(\gamma)\gamma) \leq t \leq c_1 \log(d^2(\gamma)H(\gamma)\Delta) \leq c_1 \log(d^3(\alpha)|N(\alpha)|\Delta)$ for some constant $c_1 \in \mathbb{N}$. So, the running time for approximating $\operatorname{Ln} \gamma$ is $O(M(k + \log t) \log(k + \log t) + M(t)$, where $t = \log(|N(\alpha)|d^3(\alpha)\Delta)$.

Because α is reduced, we have

$$0 < d_j \leq \Delta^{1/2}, H(\alpha_j) \leq 2\Delta^{7/4},$$

for $1 \leq j \leq n \leq 2 + \log\log max\{H(\alpha)d(\alpha), 2\}$. So, we can estimate the size of the numbers as $\operatorname{size}(\alpha_j/d_j) = O(\log \Delta)$. And it follows from Remark 4.2.3, that the time for computing b is $O(nM(k + n + \log\log \Delta) \log(k + n + \log\log \Delta) + nM(\log \Delta))$. $\square$

4.2.3. Computing the norm. Let $a = ((a_1, \dots, a_n), (e_1, \dots, e_n))$ be a reduced power product representation for $\alpha \in K^*$. We describe the function $a.$`norm`(A), that on input of $A = \alpha O$ returns the norm of α.

```
bigrational a.norm (quadratic_ideal A)
 {
   int n = a.length();
   quadratic_number_standard gamma  = a.base(n-1);
   quadratic_number_standard beta_k = a.base(n-2);

   quadratic_ideal B = A/gamma;
   return sign(norm(beta_k)) * norm(gamma) / d(B);
 }
```

PROPOSITION 4.2.5. *Let α be a reduced power product representation with $A = \alpha O$. The function α.norm(A) returns the norm of α.*

PROOF. We know from the definition of reduced power products, that $\alpha = \gamma\beta$, with $\gamma \in Min(\alpha O)$, $\beta = \prod_{j=1}^{k} \beta_j^{2^{k-j}}$, $\beta_j \in K^*$, $1 \leq j \leq k$, $k \in \mathbb{N}$.

The ideal $\beta O = A/\gamma$ is reduced, because $\gamma \in Min(A)$. It follows from Lemma 5.1.3 and Lemma 5.1.1, that $N(\beta O) = 1/d(\beta O) = 1/d(A/\gamma)$. Hence,

$$|N(\beta)| = 1/d(A/\gamma).$$

Furthermore, β_k is the only basis element of the power product β with odd exponent, so its norm determines the sign of the norm of β. Therefore,

$$N(\alpha) = N(\gamma)sign(N(\beta_k))/d(A/\gamma),$$

which proves the assertion. □

4.3. Logarithm representation: `quadratic_number_logarithm`

We introduce a logarithm representation for quadratic numbers $\alpha \in K^*$. A tupel (A, a, k, s, l) is called a (k-)*logarithm representation* of α with regard to Δ, if A is a fractional O-ideal in standard representation, a is a floating point number, $k \in \mathbb{Z}$, $s \in \{\pm 1\}$, and l is a logarithm function for O, such that

$$A = \alpha O, |a - l(\alpha)| < 2^{-k}, sign(\alpha) = s,$$

where $k \geq 3$, if A is reduced, and $k \geq 4$, otherwise. We also require a to be an absolute k-approximation to $l(\alpha)$. If the order is given from the context, we sometimes omit Δ and just say, that the tupel is a (k-)logarithm representation. Those representations are implemented by the data type `quadratic_number_logarithm`. For a tuple $T = (A, a, k, s, l)$, we set

$$\mathrm{size}(T) = \mathrm{size}(A) + \mathrm{size}(a) + \mathrm{size}(k) + 1.$$

LEMMA 4.3.1. *The logarithm representation for non-zero real quadratic numbers is well-defined.*

PROOF. Let $\alpha, \beta \in K^*$ with logarithm representation $T = (A, a, k, s, l)$. We show that $\alpha = \beta$. Let R be the regulator of O. It follows from $\alpha O = A = \beta O$, that $\alpha/\beta \in O^*$. This implies

$$|l(\alpha/\beta)| \leq |a - l(\alpha)| + |a - l(\beta)| < 2^{-2} < R.$$

Hence, $|\alpha| = |\beta|$. Because $sign(\alpha) = s = sign(\beta)$, we obtain $\alpha = \beta$. □

CHAPTER 5

Computing with quadratic ideals

In this chapter we describe the representation of real quadratic ideals. We also introduce the concept of neighbours of minima and ideals, and we explain the cycle of reduced ideals. Furthermore, we describe operations for ideals together with their bit complexity, and we give bounds on the distance of minima and the size of minima. Parts of this chapter are a short introduction with the goal to make the reader familiar with the concepts and to define the notation that is used throughout the rest of the thesis. For a more detailed presentation, we refer to [**BTW95**] and [**Len82**].

5.1. Standard representation: The model `quadratic_ideal`

It follows from [**BTW95**][(2) and Proposition 2.5] that every fractional O-ideal I can uniquely be written as

$$I = q\left(a\mathbb{Z} + \frac{b+\sqrt{\Delta}}{2}\mathbb{Z}\right), \tag{5.1.1}$$

where $q = u/v \in \mathbb{Q}_{>0}$, $gcd(u,v) = 1$, $a, b \in \mathbb{Z}$, $a > 0$, $4a$ divides $b^2 - \Delta$, $\sqrt{\Delta} - 2a < b < \sqrt{\Delta}$, if $a < \sqrt{\Delta}$, and $-a < b \leq a$, if $a \geq \sqrt{\Delta}$. We call this representation the (Δ-)*standard represention* of I.

The fractional ideals of O are implemented by the data type `quadratic_ideal`. An object of type `quadratic_ideal` that represents the ideal I is a quadruple $\mathtt{I} = (q, a, b, p)$, where the numbers q, a, b are given by the standard representation of I, and p is a pointer to the quadratic order O of discriminant Δ. We set

$$\operatorname{size} \mathtt{I} = \operatorname{size} u + \operatorname{size} v + \operatorname{size} a + \operatorname{size} b + \operatorname{size} \Delta + 1,$$

and

$$\operatorname{size} I = \operatorname{size} \mathtt{I}.$$

Lemma 5.1.1. *For a fractional O-ideal I in standard representation we have*
1. *$d(I) = v$.*
2. *$N(I) = q^2 a$.*
3. *$r(I) = qa$.*
4. *$l(I) = ua/gcd(a,v)$.*

Proof. We start with the first assertion. It follows from the standard representation of I that $vI \subseteq O$. So $d(I) \leq v$. On the other hand let $d(I)I = u_2(a_2\mathbb{Z} + (b_2+\sqrt{\Delta})/2\mathbb{Z})$ be the standard representation of $d(I)I$. The uniqueness of the standard representation implies that $d(I) = gcd(u_2, d(I))v$, so $d(I) \geq v$. Hence, $d(I) = v$.

We prove the assertion on the norm. It is $d(I)I = u(a\mathbb{Z}+(b+\sqrt{\Delta})/2\mathbb{Z})$. Because $b \equiv \Delta \mod 2$, the matrix $A = \left(\begin{smallmatrix} ua & 0 \\ 0 & u \end{smallmatrix}\right)$ transforms the $\mathbb{Z}$-basis of O into the $\mathbb{Z}$-basis of $d(I)I$. So, $N(d(I)I) = det(A) = u^2a$. This implies $N(I) = N(d(I)I)/d(I)^2 = u^2a/v^2 = q^2a$.

We prove the third and fourth assertion. It follows from the standard representation of I and the fact that $\sqrt{\Delta} \notin \mathbb{Q}$, that $I \cap \mathbb{Q} = qa\mathbb{Z}$. Hence, $r(I) = qa$, and $l(I) = ua/gcd(ua, v) = ua/gcd(a, v)$. $\square$

Lemma 5.1.2. *For a fractional O-ideal I, it is $r(I) = l(d(I)I)/d(I) \leq N(I)d(I)$.*

Proof. It follows from Lemma 5.1.1, that $r(I) = l(d(I)I)/d(I)$. Furthermore, because $d(I)I$ is integral, we have $N(d(I)I) \in d(I)I$. Hence, $r(I) = l(d(I)I)/d(I) \leq N(d(I)I)/d(I) = N(I)d(I)$. $\square$

We obtain the following criterium for reduced ideals, i.e. ideals with minimum 1.

Lemma 5.1.3. *Let $I = q(\mathbb{Z}a+\mathbb{Z}(b+\sqrt{\Delta})/2)$ be a fractional O-ideal in standard representation. Then I is reduced if and only if $q = 1/a$ and $|b| + \sqrt{\Delta} > 2a$.*

Proof. Suppose that I is reduced. Then $1 \in I$, so $q = 1/a$. If $|b| + \sqrt{\Delta} \leq 2a$, then $|b| + \sqrt{\Delta} < 2a$, because Δ is not a square. Thus, $H((b + \sqrt{\Delta})/(2a)) = (|b|+\sqrt{\Delta})/(2a) < 1$, which contradicts the fact that I is reduced. So, $|b|+\sqrt{\Delta} > 2a$.

Conversely, suppose that $I = \mathbb{Z} + \mathbb{Z}(b + \sqrt{\Delta})/(2a)$ with $|b| + \sqrt{\Delta} > 2a$. Set $H(x, y) = |2ax + yb| + |y|\sqrt{\Delta}$ for integer x, y. We have $1 \in I$, and it is a minimum if and only if $H(x, y) \geq 2a$ for all $x, y \in \mathbb{Z}$, $(x, y) \neq (0, 0)$. If $x = 0$, then $H(x, y) = |y|(|b| + \sqrt{\Delta}) > 2a$, and if $x \neq -yb/(2a)$, then $H(x, y) \geq 2a|x + yb/(2a)| \geq 2a$. If $0 \neq x = -yb/(2a)$, then $H(x, y) \geq |y|\sqrt{\Delta} = 2a|x|\sqrt{\Delta}/|b| > 2a$, because $|b| + \sqrt{\Delta} > 2a$, implies that $|b| < \sqrt{\Delta}$. (Note that a,b satisfy the conditions of the standard representation.) $\square$

By Lemma 5.1.3 the standard representation of a reduced ideal I is

$$I = \frac{1}{a}\left(a\mathbb{Z} + \mathbb{Z}\frac{b+\sqrt{\Delta}}{2}\right). \tag{5.1.2}$$

Lemma 5.1.4. *Let I be a reduced ideal. Then*
1. $d(I) < \sqrt{\Delta}$.
2. $1/\sqrt{\Delta} < N(I) \leq 1$.
3. $r(I) = l(I) = 1$.
4. $\text{size}(I) \leq c \log \Delta$ *for some* $c \in \mathbb{N}$.

Proof. If I is reduced, then we have $q = 1/a$ in the standard representation of I by Lemma 5.1.3. Therefore, the assertions follow from Lemma 5.1.1 and [**BTW95**][Lemma 2.14]. $\square$

5.2. Neighbours of minima and reduced ideals

Let I be a fractional ideal of O and $\alpha \in Min(I)$. We define neighbours of minima. We call the smallest minimum $\beta \in Min(I)$ with $\beta > \alpha$ the *right neighbour* of α. It is denoted by

$$\rho_I(\alpha).$$

The largest minimum $\beta \in Min(I)$ with $\beta < \alpha$ is called the *left neighbour* of α, and it is denoted by

$$\rho_I^{-1}(\alpha).$$

We show that neighbours of minima exist. Let $I = 1/a(a\mathbb{Z} + (b+\sqrt{\Delta})/2\mathbb{Z})$ in standard representation and reduced. As stated in Proposition 2.24 and 2.25 of [**BTW95**], there are explicit formulas for the right and left neighbour of 1 in I:

$$\begin{array}{rcl} \rho_I(1) & = & (b+\sqrt{\Delta})/(2a), \\ \rho_I^{-1}(1) & = & (-\tau(-b,a)+\sqrt{\Delta})/(2a), \end{array} \tag{5.2.1}$$

where $\tau(-b,a)$ is the unique integer τ such that $\tau \equiv -b \bmod 2a$, $-a < \tau \leq a$ if $a > \sqrt{\Delta}$ and $\sqrt{\Delta} - 2a < \tau < \sqrt{\Delta}$ if $a < \sqrt{\Delta}$. Now let I be a fractional O-ideal, not necessarily reduced, and let $\alpha \in Min(I)$. Then 1 is a minimum in I/α. An easy calculation shows that $\alpha\rho_{I/\alpha}^{-1}(1)$ and $\alpha\rho_{I/\alpha}(1)$ are the left and right neighbour of α in I, respectively.

For $i \in \mathbb{Z}$, we also write

$$\rho_I^i(\alpha)$$

to denote the i-th right neighour of α in I, if i is positive, or the $|i|$-th left neighbour, if i is negative, respectively.

We also introduce neighbours of reduced ideals. Let I be a reduced O-ideal, so 1 is a minimum in I. By

$$\rho(I) = I/\rho_I(1) \tag{5.2.2}$$

we denote the *right neighbour* of I. Similarly, we define

$$\rho^{-1}(I) = I/\rho_I^{-1}(1) \tag{5.2.3}$$

as the *left neighbour* of I. And for $i \in \mathbb{Z}$, we also write

$$\rho^i(I)$$

to denote the i-th right neighour of α in I, if i is positive, or the $|i|$-th left neighbour, if i is negative, respectively.

Now, let I be a fractional O-ideal and $\alpha_0 \in Min(I)$. For $i \in \mathbb{Z}$ set

$$\alpha_i = \rho_I^i(\alpha_0),\ I_i = I/\alpha_i.$$

It follows from (1.3.1), that the sequence $(I_i)_{i\in\mathbb{Z}}$ contains all reduced ideals in the equivalence class of I. The following theorem is fundamental.

THEOREM 5.2.1. *The sequence $(I_i)_{i\in\mathbb{Z}}$ is periodic and for every $i \in \mathbb{Z}$ we have*
1. $I_{i+1} = \rho(I_i)$, $I_{i-1} = \rho^{-1}(I_i)$.
2. $\alpha_{i+1} = \alpha_i\rho_{I_i}(1), \alpha_{i-1} = \alpha_i\rho_{I_i}^{-1}(1)$.

PROOF. [**BTW95**] □

Let I be a reduced O-ideal. If we set $\alpha_{-1} = 1$, it follows from Theorem 5.2.1[2.], that

$$1 = \rho_I(1)\rho_{\rho_I}^{-1}(1). \tag{5.2.4}$$

5.3. Properties of minima

In this section we introduce some properties of minima.

LEMMA 5.3.1. *Let I be a fractional O-ideal, α a minimum of I, and $\beta \in K_{>0}$. Then*

1. *$\alpha\beta$ is a minimum in βI.*
2. *$\rho_{\beta I}(\beta\alpha) = \beta\rho_I(\alpha)$.*
3. *$\rho_{\beta I}^{-1}(\beta\alpha) = \beta\rho_I^{-1}(\alpha)$.*

PROOF. We prove the first assertion. Assume that there exists a number $\gamma \in \beta I$ with $|\gamma| < |\beta\alpha|$ and $|\gamma^{(2)}| < |(\beta\alpha)^{(2)}|$. Then $\delta = \gamma/\beta \in I$ and $|\delta| < |\alpha|$, $|\delta^{(2)}| < |\alpha^{(2)}|$. This contradicts the minima property of α.

Now, we prove the second assertion. Let $\gamma = \rho_{\beta I}(\beta\alpha)$. Hence, $\gamma/\beta \in Min(I)$. Assume that there is a minimum $\delta \in Min(I)$ with $\alpha < \delta < \gamma/\beta$. This implies $\alpha\beta < \beta\delta < \gamma$ and $\beta\delta \in Min(\beta I)$, but this contradicts the fact that γ is the right neighbour of $\alpha\beta$ in βI. So γ/β is minimal in $Min(I)$ with $\alpha < \gamma/\beta$ and therefore $\gamma/\beta = \rho_I(\alpha)$.

The assertion for ρ^{-1} can be proven analogously. □

We examine the distance of minima for logarithm functions. Let l be a logarithm function, and $i \in \mathbb{Z}$. We set

$$L_{l,i} = inf\{|l(\alpha) - l(\rho_I^i(\alpha))| \, I \text{ fractional } O-\text{ideal}, \alpha \in Min(I)\},$$

and

$$U_{l,i} = sup\{|l(\alpha) - l(\rho_I^i(\alpha))| \, I \text{ fractional } O-\text{ideal}, \alpha \in Min(I)\}.$$

LEMMA 5.3.2. *It is $L_{\mathrm{Ln},\pm 1} \geq 1/(2\sqrt{\Delta})$, $U_{\mathrm{Ln},\pm 1} \leq 1/2 \ln \Delta$, and $L_{\mathrm{Ln},\pm 2} \geq \ln 2$.*

PROOF. We prove the first and second assertion. Because $\rho_I^{-1}(\alpha)$ also is a minimum of I, it is sufficient to prove that for any $\alpha \in Min(I)$ we have $1/(2\sqrt{\Delta}) < \mathrm{Ln}\,\rho_I(\alpha) - \mathrm{Ln}\,\alpha < 1/2 \ln \Delta$.

Let $\alpha \in Min(I)$ and $J = I/\alpha$. Set $\gamma = \rho_J(1)$. By Theorem 5.2.1 we know that $\mathrm{Ln}\,\rho_I(\alpha) - \mathrm{Ln}\,\alpha = \mathrm{Ln}\,\gamma = 1/2 \ln |\gamma/\sigma\gamma|$. Let $\gamma = (b + \sqrt{\Delta})/(2a)$ and $c = (b^2 - \Delta)/(4a)$. Then $\gamma/\sigma\gamma = (b + \sqrt{\Delta})^2/(4ac)$. Since $|ac| \geq 1$ and $|b| < \sqrt{\Delta}$ by [**BTW95**][Lemma 2.14], we obtain $|\gamma/\sigma\gamma| < \Delta$. So $\mathrm{Ln}\,\gamma < 1/2 \ln \Delta$ and this proves the upper bound.

To prove the lower bound let $x = (\sqrt{\Delta} + b)/(\sqrt{\Delta} - b) - 1$. By [**BTW95**][Proposition 2.15] we have $b \geq 0$. Hence $x \geq 0$. Furthermore, $\ln |\gamma/\sigma\gamma| = \ln |(b+\sqrt{\Delta})/(b-\sqrt{\Delta})| = \ln(x+1) \geq x/(1+x) = 2b/(\sqrt{\Delta}+b) > 1/\sqrt{\Delta}$. So $\mathrm{Ln}\,\gamma = 1/2 \ln |\gamma/\sigma\gamma| > 1/(2\sqrt{\Delta})$, which proves the lower bound.

For third assertion we refer to [**Len82**][(11.2)]. □

LEMMA 5.3.3. *It is $L_{\ln,\pm 1} \geq 0$, $U_{\ln,\pm 1} \leq 1/2 \ln \Delta$, and $L_{\ln,\pm 2} \geq \ln 2$.*

PROOF. [**BTW95**][Lemma 2.27 and 2.28]. □

We estimate the denominator and the height of a minimum and its inverse.

LEMMA 5.3.4. *Let I be a fractional ideal of O and $\alpha \in Min(I)$. Then $d(\alpha) \leq d(I)$ and $d(1/\alpha) < l(I)\sqrt{\Delta}$. Furthermore, if $|\mathrm{Ln}\,\alpha| < c \in \mathbb{R}$, then $H(\alpha) < e^c l(I)\Delta^{1/4}$ and $H(1/\alpha) < e^c/\sqrt{N(I)}$. And if $|\ln \alpha| < c \in \mathbb{R}$, then*

$H(\alpha) < e^c l(I)^2 \Delta^{1/2}$ and $H(1/\alpha) < e^c max\{1, 1/N(I)\}$. Here, e is the Euler constant.

PROOF. We prove the bounds for the denominators. $\alpha \in I$ implies $d(\alpha) \leq d(I)$. Let $J = I/\alpha$. By Lemma 5.3.1 we obtain that 1 is a minimum in J, so J is reduced. Because $l(I)/\alpha \in J$, we have $d(1/\alpha) \leq d(J)l(I) < \sqrt{\Delta} l(I)$ by Lemma 5.1.4.

We estimate the heights. Because $\operatorname{Ln}\alpha = 1/2 \ln|\alpha/\sigma\alpha|$, the condition on $|\operatorname{Ln}\alpha|$ yields $|\alpha/\sigma\alpha|, |\sigma\alpha/\alpha| < e^{2c}$. This implies that $|\alpha|, |\sigma\alpha| < \sqrt{|N(\alpha)|}e^c$ and $|1/\alpha|, |1/\sigma\alpha| < 1/\sqrt{|N(\alpha)|}e^c$, which implies $H(\alpha) < e^c\sqrt{|N(\alpha)|}$ and $H(1/\alpha) < e^c/\sqrt{|N(\alpha)|}$.

The condition $|\ln\alpha| < c$ implies $e^{-c} < |\alpha| < e^c$ and $e^{-c}|N(\alpha)| < |\sigma\alpha| < e^c|N(\alpha)|$. Hence, $|\alpha|, |\sigma\alpha| < e^c max\{1, |N(\alpha)|\}$ and $|1/\alpha|, |1/\sigma\alpha| < e^c max\{1, 1/|N(\alpha)|\}$. Thus, $H(\alpha) < e^c max\{1, |N(\alpha)|\}$ and $H(1/\alpha) < e^c max\{1, 1/|N(\alpha)|\}$.

To prove the assertions, we note that

$$N(I) \leq |N(\alpha)| \leq \sqrt{\Delta} l(I)^2.$$

The lower bound is obvious, because $\alpha \in I$. And the upper bound follows from $l(I)/\alpha \in I/\alpha$ and $N(I/a) \geq 1/\sqrt{\Delta}$ by Lemma 5.1.4; so, $|N(l(I)/\alpha)| \geq 1/\sqrt{\Delta}$. $\square$

5.4. Basic operations for ideals

In this section, let I and J be of type `quadratic_ideal`.

5.4.1. `multiply`. It follows from [**BTW95**][Proposition 2.10] that there is a function H.`multiply`(I, J) which computes the product $H = IJ$ in time $O(M(max\{\operatorname{size} I, \operatorname{size} J\}) \log max\{\operatorname{size} I, \operatorname{size} J\})$. In case of $M(n) = n^2$ multiplication of ideals can be done in time $O((max\{\operatorname{size} I, \operatorname{size} J\})^2)$. For simplicity there also exists a function I.`square`() for computing the square of I.

5.4.2. `rho`. There is a function I.`rho`(α) which transforms the reduced ideal I into the standard representation of $\rho(I)$ and also returns $\alpha = \rho_I(1)$ in standard representation. We have $\rho(I) = I/\alpha$. It follows from (5.2.2) and (5.2.1) that the running time of the function is $O(M(\operatorname{size} I))$.

Let $1/a(\mathbb{Z}a + \mathbb{Z}(b + \sqrt{\Delta})/2)$ be the standard representation of I. We know from (5.2.1) that

$$\alpha = (b + \sqrt{\Delta})/(2a).$$

Thus, $1/\alpha = (b - \sqrt{\Delta})/(2c)$, where $c = (\Delta - b^2)/(4a)$. We know from [**BTW95**][Lemma 2.14] that $a, |b| < \sqrt{\Delta}$, because I is reduced. So, $H(\alpha), H(1/\alpha) < \sqrt{\Delta}$. Furthermore, it follows from Lemma 5.3.4 and Lemma 5.1.1, that $d(\alpha) \leq d(I) = a < \sqrt{\Delta}$ and $d(1/\alpha) \leq d(\rho(I)) < \sqrt{\Delta}$, because $\alpha \in Min(I)$ and $1/\alpha \in Min(\rho(I))$. This shows that α satisfies

$$H(\alpha), H(1/\alpha), d(\alpha), d(1/\alpha) < \sqrt{\Delta}. \tag{5.4.1}$$

It follows from (5.4.1) and (4.1.6) that there is a constant $c \in \mathbb{N}$, indepent of α and Δ, such that

$$\operatorname{size}\alpha \leq c \log \Delta. \tag{5.4.2}$$

5.4.3. `inverse_rho`**.** There is a function I.`inverse_rho`(α) that transforms the reduced ideal I into $\rho^{-1}(I)$ and also returns $\alpha = \rho_I^{-1}(1)$ in standard representation. We have $\rho^{-1}(I) = I/\alpha$. It follows from (5.2.3) and (5.2.1) that the running time of the function is $O(M(\operatorname{size} I))$.

It follows from (5.2.4) that $1/\alpha = \rho_{\rho^{-1}(I)}(1)$. Hence, (5.4.1) implies

$$H(\alpha), H(1/\alpha), d(\alpha), d(1/\alpha) < \sqrt{\Delta}. \tag{5.4.3}$$

And it follows from (5.4.3) and (4.1.6) that there is a constant $c \in \mathbb{N}$, indepent of α and Δ, such that

$$\operatorname{size} \alpha \leq c \log \Delta. \tag{5.4.4}$$

5.4.4. `reduce`**.** In this section, we introduce a function I.`reduce`(α) that transforms the fractional O-ideal I into the reduced ideal $J = I/\alpha$, and returns the reducing minimum α as an object of type `quadratic_number_standard`. Using a result of Schönhage, [**Sch91**], we show that the running time of the procedure is $O(M(\operatorname{size} I) \log(\operatorname{size} I))$, and that $H(\alpha) \leq r(I)$. We have to present some details, to prove the result on the height.

First note, that there is a function I.`is_reduced`(), that returns true, if the ideal is reduced, and false otherwise. Using the criterium given in Lemma 5.1.3, this can be done in time $O(M(\operatorname{size} I))$.

To make use of Schönhage's reduction method, we introduce quadratic forms. They are represented by their corresponding 2×2 matrices,

$$f(x, y) = ax^2 + bxy + cy^2 = (x, y)[a, b, c](x, y)^T,$$

where $[a, b, c] = \left(\begin{smallmatrix} a & b/2 \\ b/2 & c \end{smallmatrix}\right)$, $a, b, c \in \mathbb{Z}$. Here, T stands for taking the transpose. Two forms $[a, b, c]$ and $[A, B, C]$ are called *equivalent*, denoted by

$$[A, B, C] \sim [a, b, c],$$

if and only if there exists a unimodular matrix U with $det\, U = 1$ and

$$[A, B, C] = U[a, b, c]U^T.$$

For $U = \left(\begin{smallmatrix} s & t \\ u & v \end{smallmatrix}\right)$ the coefficients of the equivalent form are given as

$$[A, B, C] = [f(s, t), 2(asu + ctv) + b(sv + tu), f(u, v)]. \tag{5.4.5}$$

An invariant of an equivalence class is the *discriminant* of the form

$$\Delta = b^2 - 4ac.$$

Because we work with real quadratic orders, we are only interested in *indefinite forms*, i.e. forms with discriminant $\Delta > 0$. In the following all forms are assumed to be indefinite.

We explain the connection between forms and the real quadratic order O. Let $f = [a, b, c]$ be a form of discriminant Δ. We set

$$I(f) \quad = \quad \mathbb{Z} + \mathbb{Z}\frac{b + \sqrt{\Delta}}{2|a|}.$$

Then I is a fractional O-ideal. For $U = \left(\begin{smallmatrix} s & t \\ u & v \end{smallmatrix}\right)$, we set

$$\alpha(f, U) = s + t(b + \sqrt{\Delta})/(2a).$$

For the equivalent form $g = U[a,b,c]U^T$, an easy, but somewhat lengthy calculation shows that,

$$I(g) = I(f)/\alpha(f,U). \tag{5.4.6}$$

We sketch the proof. Set $\alpha = (b+\sqrt{\Delta})/(2a)$ and let $g = [A,B,C]$. We use (5.4.5) to show that $(B+\sqrt{\Delta})/(2A) = (v\alpha+u)/(t\alpha+s)$. This implies $I(g) = (\mathbb{Z}(s+t\alpha)+\mathbb{Z}(u+v\alpha))/(s+t\alpha) = I(f)/\alpha(f,U)$, because the unimodular transformation does not change $I(f)$.

Furthermore, if U and V are two transformations, a straight forward calculation shows that

$$\alpha(f,VU) = \alpha(f,U)\cdot\alpha(U\,[a,b,c]\,U^T,V). \tag{5.4.7}$$

So far, we have seen how the equivalence of forms implies the equivalence of the corresponding ideals, and we can read the transforming number from the transformation matrix of the forms. Next, we introduce the notion of reduced forms, and we will see, that reduction of a form implies the reduction of the associated ideal. Then we can apply Schönhage's reduction method for forms to reduce the ideal and to find the transforming number.

We follow [**Buc**] to introduce normal and reduced forms. The form $[a,b,c]$ is *normal*, if $-|a| < b \le |a|$, for $|a| \ge \sqrt{\Delta}$, and $\sqrt{\Delta} - 2|a| < b \le \sqrt{\Delta}$, for $|a| < \sqrt{\Delta}$. If we set

$$s = \begin{cases} sign(a)\lfloor \frac{|a|-b}{2|a|}\rfloor, & \text{for } |a| \ge \sqrt{\Delta}, \\ \lfloor \frac{\lfloor\sqrt{\Delta}\rfloor - b}{2|a|}\rfloor, & \text{for } |a| < \sqrt{\Delta}, \end{cases} \tag{5.4.8}$$

and $U = \left(\begin{smallmatrix} 1 & 0 \\ s & 1 \end{smallmatrix}\right)$, then $[a, b+2sa, as^2+bs+c] = U[a,b,c]U^T$ is normal. Replacing $[a,b,c]$ by this equivalent normal form is called *normalization*. It follows from (5.4.6), that normalization does not change the corresponding ideal of the form.

The form $[a,b,c]$ is *reduced*, if $|\sqrt{\Delta}-2|a|| < b < \sqrt{\Delta}$. If the form is reduced, then $|a| < \sqrt{\Delta}$, so the form is also normal.

LEMMA 5.4.1. *The form $[a,b,c]$ is reduced if and only if it is normal and $b + \sqrt{\Delta} > 2|a|$.*

PROOF. If the form is reduced, then the assertion is obviously. Assume that $[a,b,c]$ is normal with $b+\sqrt{\Delta} > 2|a|$.

If $|a| \ge \sqrt{\Delta}$, then $b+\sqrt{\Delta} > 2|a|$ implies $b > |a|$, which is a contradiction, because the form is normal. So $|a| < \sqrt{\Delta}$, and because the form is normal, it is $\sqrt{\Delta} - 2|a| < b < \sqrt{\Delta}$. Together with $b+\sqrt{\Delta} > 2|a|$, we obtain $|\sqrt{\Delta}-2|a|| < b < \sqrt{\Delta}$. Hence, the form is reduced. □

LEMMA 5.4.2. *If $[a,b,c]$ is normal and not reduced, then $|b| + \sqrt{\Delta} \le 2|a|$.*

PROOF. For $b \ge 0$, the assertion is an immediate consequence of Lemma 5.4.1. For $b < 0$, the assertion immediately follows from the definition of normal forms for both cases $|a| \ge \sqrt{\Delta}$ and $|a| < \sqrt{\Delta}$, respectively. □

The connection between reduced forms and ideals is given by the next Proposition.

PROPOSITION 5.4.3. *Let O be a real quadratic order, and f be a form with discriminant Δ.*

1. *If f is reduced, then $I(f)$ is reduced.*
2. *If f is normal, and $I(f)$ reduced, then f is reduced.*

PROOF. Let $f = [a, b, c]$. Assume that f is reduced. Because f is normal, it follows from Lemma 5.1.3, that $I(f)$ is reduced if and only if $|b| + \sqrt{\Delta} > 2|a|$. But this is surely fulfilled, since f is reduced.

Suppose that f is normal. If $I(f)$ is reduced, then $|b| + \sqrt{\Delta} > 2|a|$. Hence, f must be reduced by Lemma 5.4.2. □

Finally, we introduce the setting of Schönhage's reduction method for forms. Given some treshold $s > 0$, a form $[a, b, c]$ with $a, b/2, c \geq s$ is said to be *minimal above* s, if $a, b/2, c \geq s$ and ($a - b + c < s$ or $b < 2s + min(2a, 2c)$). As described in [**Sch91**], there is a function $(A, B, C, s, t, u, v) = \mathtt{MR}(a, b, c, m)$, that, on input of a form $[a, b, c]$ with $a, b/2, c \geq 2^m$, computes a matrix $M = \left(\begin{smallmatrix} s & t \\ u & v \end{smallmatrix}\right)$ with $det M = 1$ and $M \geq 0$ (all entries ≥ 0), such that $[a, b, c] = M[A, B, C]M^T$, and the form $[A, B, C]$ is minimal above 2^m. If n is the minimal integer such that $a, b, c < 2^{m+n}$, then the running time of the procedure is $O(M(n) \log n)$.

Now, we present the function $I.\mathtt{reduce}(\alpha)$, that on input of a fractional O-ideal I transforms it into the reduced ideal $J = I/\alpha$, and returns α with $H(\alpha) \leq r(I)$, in time $O(M(\operatorname{size} I) \log \operatorname{size} I)$. To simplify the description, we write $[a, b, c] = [A, B, C]$ to indicate that $a = A$, $b = B$, and $c = C$; to program these assignments, further variables must be introduced to implement, e.g., the exchange of a and c. We also use the function `normalization`($[a, b, c]$), that computes the normalization of the form, and `is_reduced`($[a, b, c]$), that returns true, if the form is reduced, and false otherwise.

```
void I.reduce (quadratic_number_standard & alpha)
 {
    quadratic_order O = I.order();

    // I reduced ?
    //
    if (I.is_reduced())
     { alpha.assign_one(O); return; }

    // I/(qa) = Z + Z (b+sqD)/(2a) reduced ?
    //
    alpha = qa;
    q = 1/a;

    if (I.is_reduced())
     { return; }

    // Generate form [a,b,c] with a+c >= 0.
    //
    bigint Delta = O.discriminant();
    bigint c = (b^2-Delta)/(4a);

    if (a+c < 0)
       { [a,b,c] = [-a,b,-c] };
```

```
   // Generate form [a,b,c] with a,b,c >= 1,  U_0, U_1
   //
   if (c < 0)
    {
      alpha *= quadratic_number_standard(b,1,2a,0);
      [a,b,c] = normalization([c,-b,a]);
      q = 1/a;
      return;
    }

   // U_2
   //
   if (b < 0)
      {
        alpha *= quadratic_number_standard(b,1,2a,0);
        [a,b,c] = [c,-b,a];
      }

   // Minimal form over 1/2.
   // [a,b,c] = M [a,b,c] M^T with M = ( s t )
   //                                  ( u v )
   (a,b,c,s,t,u,v) = MR(a,b,c,-1);
   alpha *= quadratic_number_standard(-2as+bu,u,2a,0);

   // Transformation U_3
   //
   if (a^2 >= Delta)
    {
      alpha  *= quadratic_number_standard(2a-b,-1,2a,0);
      [a,b,c] = [a-b+c,b-2c,c];
    }

   // Normalization with U_4
   //
   [a,b,c] = normalization([a,b,c]);

   // Final reduction step,  U_5, U_6
   //
   if (!is_reduced([a,b,c]))
     {
        alpha *= quadratic_number_standard(b,1,2a,0);
        [a,b,c] = normalization([c,-b,a]);
     }

   q = 1/a;
}
```

Before we analyze the correctness and the running time of the procedure, we need some auxiliary results.

LEMMA 5.4.4. *Let $[a, b, c]$ be a form with $a + c \geq 0$ and $a < 0$. Then the normalization of $[a, b, c]$ is reduced.*

PROOF. Let $[a, B, C]$ be the normalization of $[a, b, c]$. It follows from $a + c \geq 0$, that $|a| = -a \leq c = |c|$. Thus, $\Delta = b^2 - 4ac = b^2 + 4|ac| \geq (2|a|)^2$. This implies

$$2|a| \leq \sqrt{\Delta}.$$

Because the form is normal, it is $|\sqrt{\Delta} - 2|a|| = \sqrt{\Delta} - 2|a| < B < \sqrt{\Delta}$. So the form is reduced. □

LEMMA 5.4.5. *If the form $[a, b, c]$ is normal with $|a| < \sqrt{\Delta}$, but not reduced, then the normalization of $[c, -b, a]$ is reduced.*

PROOF. [**Buc**] □

Now, let O be the real quadratic order and

$$I = q(\mathbb{Z}a + \mathbb{Z}(b + \sqrt{\Delta})/2)$$

be the standard representation of I. Assume that I is not reduced. It follows from Lemma 5.1.3, that $I/(qa)$ is reduced if and only if $|b| + \sqrt{\Delta} > 2a$. If this is the case,

$$J = I/(qa) = 1/a(\mathbb{Z}a + \mathbb{Z}(b + \sqrt{\Delta})/2),$$

is the standard representation of $I/(qa)$ and the reducing number is $\alpha = qa$. By Lemma 5.1.1, it is $H(\alpha) = qa = r(I)$. Otherwise, $J = I([a, b, c])$ is not reduced, where $c = (b^2 - \Delta)/(4a)$.

To achieve $a, b, c \geq 1$, before `MR` can be applied, we follow [**Sch91**][section 4]. If $a + c < 0$, then $[a, b, c]$ is changed to $[-a, b, -c]$. This is not an equivalence transformation, but it does not change the ideal, i.e., $I([a, b, c]) = I([-a, b, -c])$. The form $[a, b, c]$ is normal, because a,b are from the standard representation of I, and not reduced. Thus, $[-a, b, -c]$ also is normal and not reduced.

In the following, if a transformation is not applied during the execution of the algorithm, it is implicitely set to the unit matrix $I_2 = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$.

Let $[a, b, c]$ be the form, before the test for $c < 0$. It follows from Lemma 5.4.4, that $a \geq 0$, because the form is normal, but not reduced. It also follows from Lemma 5.4.4, that the normalization of $[c, -b, a]$ is reduced, if $c < 0$. Hence, the function terminates. In this case, the applied transformations are

$$U_1 U_0$$

where $U_0 = S = \left(\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right)$ is the exchange of a and c with negation of b, and $U_1 = \left(\begin{smallmatrix} 1 & 0 \\ s & 1 \end{smallmatrix}\right)$ is the normalization, where s is according to (5.4.8).

Furthermore, if $b < 0$, a transformation

$$U_2$$

with $U_2 = S$ is applied. After that, it is $a, b, c \geq 1$. Note that neither a nor c can be zero, because Δ is not a perfect square, and $a, c > 0$ implies $b \neq 0$, because $\Delta > 0$.

The procedure `MR` applies the matrix

$$M^{-1}$$

and yields the form $[a, b, c]$ minimal above 1/2, i.e., $a, b, c \geq 1$ and $a - b + c \leq 0$, because $\Delta > 0$ implies $b > min(2a, 2c)$ as noted by Schönhage. The next transformation is

$$U_3$$

with $U_3 = \left(\begin{smallmatrix} 1 & -1 \\ 0 & 1 \end{smallmatrix}\right)$, that yields the form $[a - b + c, b - 2c, c]$, if $|a| \geq \sqrt{\Delta}$. Because $a - b + c \leq 0$, it is $(b - 2a)^2 = b^2 + 4a(a - b) \leq \Delta$ and the same is true for c. Hence, $|b - 2a|, |b - 2c| < \sqrt{\Delta}$. Therefore, $2(a - b + c) = 2a - b - b + 2c$ implies $|a - b + c| < \sqrt{\Delta}$.

Let $[a, b, c]$ be the form before the next normalization. The application of

$$U_4$$

with $U_4 = \left(\begin{smallmatrix} 1 & 0 \\ s & 1 \end{smallmatrix}\right)$ normalizes the form, where s is according to (5.4.8). Because $|a| < \sqrt{\Delta}$, it follows from Lemma 5.4.5, that after the application of $U_5 = S$, followed by a normalization with U_6, i.e.,

$$U_6 U_5$$

the form $[a, b, c]$ is finally reduced.

Thus, we have shown, that beginning with a form f, such that $I/(qa) = I(f)$, the procedure computes an equivalent reduced form

$$[A, B, C] = U f U^T,$$

with $U = U_6 U_5 U_4 U_3 M^{-1} U_2 U_1 U_0$ and

$$\alpha = q\, a\, \alpha(f, U).$$

It follows from Proposition 5.4.3, that $1/A(\mathbb{Z}A + \mathbb{Z}(B + \sqrt{\Delta})/2)$ is a reduced ideal, and it is the standard representation of I/α. By [**Sch91**][section 3], the running time of `MR` is $O(M(\operatorname{size} I) \log \operatorname{size} I)$. Furthermore, it follows from [**Sch91**][Lemma 2], that the size of the entries of matrix M found by `MR` is bounded by some constant multiple of $\operatorname{size} I$. Hence, the time for `MR` dominates all other parts of the reduction.

It remains to prove that $H(\alpha) \leq r(I)$. By Lemma 5.1.1, it is $qa = r(I)$. We show that

$$H(\alpha(f, U)) \leq 1.$$

It follows from (5.4.7), that it is sufficient to prove the assertion for every factor of U.

U_1, U_4, U_6 : These transformations are normalizations and do not change the ideal. Hence, the corresponding α is 1.

U_0, U_2, U_5 : If the transformations are non-trivial, they are equal to S and are applied to a form $f = [a, b, c]$, that is normal, but not reduced. Hence, Lemma 5.4.2 yields $H(\alpha(f, S)) \leq 1$.

U_3: If the transformation is non-trivial, it is equal to $\left(\begin{smallmatrix} 1 & -1 \\ 0 & 1 \end{smallmatrix}\right)$ and is applied to a form $f = [a, b, c]$ with $|a| \geq \sqrt{\Delta}$. It is $H(\alpha(f, U_3)) = (|b - 2a| + \sqrt{\Delta})/(2|a|)$. As shown above, we have $|b - 2a| < \sqrt{\Delta}$, so $|b - 2a| + \sqrt{\Delta} < 2\sqrt{\Delta} \leq 2|a|$. Hence, $H(\alpha(f, U_3)) \leq 1$.

M^{-1}: The procedure `MR` is called with a form $f = [a, b, c]$, that satisfies $a, b, c \geq 1$. It remains to prove, that $H(\alpha(f, M^{-1})) \leq 1$. To prove this assertion, we need some more details of `MR`. The matrix M^{-1} is the product of matrizes

$$(L^{-1})^t,\ (H^{-1})^t,$$

where $L^{-1} = \left(\begin{smallmatrix} 1 & 0 \\ -1 & 1 \end{smallmatrix}\right)$ and $H^{-1} = \left(\begin{smallmatrix} 1 & -1 \\ 0 & 1 \end{smallmatrix}\right)$. The application of $(L^{-1})^t$ or $(H^{-1})^t$ to a form $[a, b, c]$ is called a *lower simple step* or a *higher simple step*, respectively (s. [**Sch91**][p. 129]). For $s > 0$, a *simple step above* s on $[a, b, c]$ is the application of a lower simple step, if $a < c$, and the application of a higher simple step, if $a \geq c$. In both cases, $t \geq 0$ is chosen maximal, such that the resulting form $[A, B, C]$ satisfies $A, B/2, C \geq s$.

The procedure `MR` has the following property. If a simple step above $2^{m'}$ is applied to a form $[x, y, z]$, then the form satisfies $x, y/2, z \geq 2^{m'}$. This can be proven, by showing that this is true for any simple step during one iteration, and that $x, y/2, z \geq 2^{m'}$ for every recursive call $\mathtt{MR}(x, y, z, m')$, under the assumption, that at the beginning of the iteration $\mathtt{MR}(a, b, c, m)$, the condition $a, b/2, c \geq 2^m$ holds. Using the fact, that this is true for the initial call of `MR`, completes the proof.

For L^{-1}, applied to a form f, it is $\alpha(f, L^{-1}) = 1$. So $H(\alpha(f, L^{-1})) = 1$. Now, assume that a higher simple step above $s > 0$ is applied to the form $f = [a, b, c]$ that is not minimal above s. Because of the property of `MR`, that $a, b/2, c \geq s$, and the fact, that $[a, b, c]$ is not minimal above s, it follows from the definition of minimality, that

$$a - b + c \geq s.$$

This implies $b < b + s \leq a + c \leq 2a$, because a higher simple step is only applied for $a \geq c$. Furthermore, $a, c > 0$ and $\Delta = b^2 - 4ac$ imply $b > \sqrt{\Delta}$. Together, we obtain $|2a - b| + \sqrt{\Delta} = 2a - b + \sqrt{\Delta} < 2a$. Thus $H(\alpha(f, H^{-1})) = (|2a - b| + \sqrt{\Delta})/(2a) < 1$. Hence, for every factor U of M^{-1}, that is applied to a form f, the corresponding $\alpha(f, U)$ satisfies $H(\alpha(f, U)) \leq 1$. Thus, for M^{-1} that is applied to a form f, it is $H(\alpha(f, M^{-1})) \leq 1$.

Now, we have proven the following

PROPOSITION 5.4.6. *Let O be a real quadratic order and I be a fractional O-ideal in standard representation. The function I.`reduce`(α) transforms I into a reduced ideal I/α in standard representation, and returns the reducing number α as an object of type `quadratic_number_standard`. The running time of the function is $O(M(\operatorname{size} I) \log \operatorname{size} I)$, and $H(\alpha) \leq r(I)$.*

REMARK 5.4.7. If $M(n) = n^2$, then the reduction method presented in [**BTW95**] is faster. Let $I = q(\mathbb{Z}a + \mathbb{Z}(b + \sqrt{\Delta})/2)$ be the standard represenation. If $I/(qa)$ is reduced, we set $\alpha = qa$. Assume that $I/(qa)$ is not reduced. It follows from [**BTW95**][Proposition 2.18], that the number β with $(I/qa)/\beta$ is reduced, satisfies $H(\beta) \leq 1$. Hence, for $\alpha = qa\beta$, we again have $H(\alpha) \leq qa = r(I)$ by Lemma 5.1.1. It has been shown in [**BB97**], that the running of that algorithm is $O((\operatorname{size} I)^2)$, as already mentioned in [**LL90**].

Because the reducing number α is a minimum in I, we finally obtain

$$H(\alpha) \leq r(I), d(\alpha) \leq d(I). \tag{5.4.9}$$

It follows from (5.4.9) and (4.1.6), that there exists a constant $c \in \mathbb{N}$, independent of α, `I`, and Δ, such that

$$\operatorname{size} \alpha \leq c \log((d(I))^2 r(I) \Delta). \tag{5.4.10}$$

5.5. Minimal ideal generator

Let $\omega \in K \setminus \mathbb{Q}$ and $I = \omega O$. Furthermore, let η be the fundamental unit of O, R its regulator, and b be an integer with $|b(R) - b| \leq 1$.

In this section we present an algorithm for computing a logarithm representation of the minimal generator μ of I, i.e., $I = \mu O$ and $0 < \operatorname{Ln} \mu \leq R$. Because $\omega O = \omega \eta O$, the minimal generator exists. For an application of the algoritm see [**Jac99**][Algorithm 6.5, step 6].

Below we present the function `minimal_ideal_generator` that on input of ω, I, η, b, and an integer t computes a t-logarithm representation of μ.

The function proceeds as follows. If $I = O$, then it returns a logarithm representation of $\mu = \eta$. Otherwise it verifies whether there is a generator of I which logarithm is less than 1 in absolute value. If such a generator exists, it is used to determine μ. Otherwise the function computes approximations w and r with

$$|w - \operatorname{Ln} \omega| < 2^{b-4-max\{b-3,4\}}, \quad |r - R| < 2^{2b-b(w)-8-max\{8,b+8\}},$$

and sets

$$z = \lfloor w/r \rfloor.$$

It follows from Lemma 12.1.4 that $|\lfloor (\operatorname{Ln} \omega)/R \rfloor - z| \leq 1$. Then the function determines $-1 \leq i \leq 2$ such that

$$\alpha = \omega / \eta^{z+i}$$

satisfies

$$-R/2 < \operatorname{Ln} \alpha \leq R/2 \quad \text{or} \quad 0 \leq \operatorname{Ln} \alpha < R. \tag{5.5.1}$$

It also determines an approximation l with

$$|l - \operatorname{Ln} \alpha| < 2^{-5}.$$

If l is non-negative, then $\operatorname{Ln} \alpha$ is non-negative. In this case (5.5.1) implies $\mu = \alpha$, and the logarithm representation of μ is $(I, l, 5, 1)$. If l is negative, then $\operatorname{Ln} \alpha$ is negative. It follows from (5.5.1) that $\mu = \alpha\eta$, and for an absolute 5-approximation r to the regulator, a logarithm representation of μ is $(I, l{+}r, 4, 1)$. Then the logarithm representation of μ is refined to a t-logarithm representation.

```
void L.minimal_ideal_generator (

                  quadratic_number_power_product omega,
                  quadratic_ideal I,
                  quadratic_number_power_product eta,
                  long b,
                  long t)
 {
    xbigfloat l, r;
    quadratic_order O = I.order();

    // Check I = O.
    if (I == O)
     {
       r = eta.absolute_Ln_approximation(t);
       L.assign(O,r,t,1);
```

```
   return;
 }

// Check for 0 <= Ln mu < 1.
if (I.is_bounded_equivalent(l,O,1,4))
 {
   L.assign(I,l,4,1);
   L.refine_logarithm_representation(t);
   return;
 }

// Check for -1 < Ln mu -R <= 0.
if (O.is_bounded_equivalent(l,I,1,5))
 {
   r = eta.absolute_Ln_approximation(5);
   L.assign(I,-l+r,4,1);
   L.refine_logarithm_representation(t);
   return;
 }

// Compute absolute k=5 approximation l.
k = 5;
bigint z;

// approximate Ln(omega)
//
xbigfloat w = omega.absolute_Ln_approximation(-b+4+max(b-2+k,4));

// Check for |w| < |r|/2
//
if (|w| - 2^(b-5) < 2^(b-4))
  {
    l = truncate(w,w.exponent()+k+1);
    z = 0;
  }
else
{
  // approximate regulator Ln(eta)
  //
  r = eta.absolute_Ln_approximation(-2b+b(w)+8+max(8,b+3+k));
  r.absolute_value();

  // | floor(Ln(omega)/|Ln(eta)|) - z | <= 1
  //
  z = floor(w/r);

  // Try to find -1 <= i <= 2 such that
  //
  //   -|Ln(eta)|/2 < Ln(omega) - (z+i) |Ln(eta)| <= |Ln(eta)|/2
```

```
   //
   // Set l = w - (z+i) * r.
   //
   long kl = min(-k-1, b-6);
   long kr = 2b-b(w)-8-max(8,b+k+3);
   int  i  = 0;
   l = w - z * r;

   if (l + 2^kl <= -r/2 - 2^(kr-1))
      { i = -1;
        l = l + r; }
   else if (-r/2 + 2^(kr-1) <= l - 2^kl &&
           l + 2^kl <= r/2 - 2^(kr-1))
       { i = 0;
         l = l; }
   else if (r/2 + 2^(kr-1) <= l - 2^kl &&
            l + 2^kl <= 3/2 * r - 2^(kr+1))
      { i = 1;
        l = l - r; }
   else if (3/2 * r + 2^(kr+1) <= l - 2^kl)
      { i = 2;
        l = l - 2*r; }

   // Ln(omega/eta^z) is too close to |Ln(eta)|/2. Find -1 <= i <= 1
   // such that
   //
   //  0 <= Ln(omega) - (z+i) |Ln(eta)| < |Ln(eta)|
   //
   // Set l = w - (z+i) * r.
   //
   else if (r + 2^kr <= l - 2^kl)
     { i = -1;
       l = l - r; }
   else if (l +2^kl <= 0)
     { i = 1;
       l = l + r; }

   z = z + i;
   l = truncate(l, l.exponent()+k+1);
 }

// Decide, whether Ln alpha >= 0.
//
if (l >= 0)
   L.assign(I,l,5,1);
else
 {
   r = eta.absolute_Ln_approximation(5);
   L.assign(I,l+r,4,1);
```

```
    }

   L.refine_logarithm_representation(t);
  }
```

THEOREM 5.5.1. *Let $\omega \in K \setminus \mathbb{Q}$, $I = \omega O$, η be the fundamental unit of O, $b \in \mathbb{Z}$ with $|b - b(R)| \leq 1$, and be $t \in \mathbb{Z}$, where R is the regulator of O.*

On input of ω, I, η, b, t the procedure $L.\mathtt{minimal\_ideal\_generator}(\omega, I, \eta, b, t)$ computes a t-logarithm representation L of a generator μ of I with $0 < \operatorname{Ln} \mu \leq R$.

PROOF. We use the notation of the procedure. If $I = O$ or a generator is found by one of the `is_bounded_equivalent` calls, the output of the procedure is obviously correct. Otherwise, we know, that there is no generator of I which logarithm is less than 1 in absolute value.

We analyze the computation of l. Let $R = |\operatorname{Ln} \eta|$ be the regulator of O and set $W = \operatorname{Ln} \omega \neq 0$, because $\omega \notin \mathbb{Q}$. We will show that the procedure computes $z \in \mathbb{Z}$ and an absolute $k = 5$-approximation l to $\operatorname{Ln} \alpha$, where $\alpha = \omega / \eta^z$, such that $I = \alpha O$ and

$$-R/2 < \operatorname{Ln} \alpha \leq R/2 \text{ or } 0 \leq \operatorname{Ln} \alpha < R. \tag{5.5.2}$$

First, assume that $|w| - 2^{b-5} < 2^{b-4}$ in the first if-statement. Then $|W| < |w| + 2^{b-5} < 2^{b-3} \leq 2^{b(R)-2} \leq |R|/2$. It follows from Lemma 2.3.1 that after truncation w is an absolute k-approximation to W.

In the following let us assume that $|w| - 2^{b-5} \geq 2^{b-4}$. We prove that $|\lfloor W/R \rceil - z| \leq 1$. We have $|W|/2 > (|w| - 2^{b-5})/2 \geq 2^{b-5} > |W - w|$. Hence, $|W - w|/|W| < 1/2$ and it follows from Lemma 2.2.2 that

$$|b(w) - b(W)| \leq 1. \tag{5.5.3}$$

Furthermore, $|W| > |w| - 2^{b-5} \geq 2^{b-4} \geq 2^{b(R)-5} > 2^{-5}|R|$. This implies $b(W) \geq b(R) - 5$ and we obtain

$$b(R) - b(W) - 1 \leq 4. \tag{5.5.4}$$

Let $k_1 = max\{4, b - 2 + k\}$. We have $|w - W| < 2^{b-4-k_1} \leq 2^{b(R)-3-k_1} < |W| 2^{-b(W)+b(R)-2-k_1}$. Hence,

$$|W - w|/|W| < 2^{-(b(W)-b(R)+2+k_1)}. \tag{5.5.5}$$

Similarly, let $k_2 = max\{8, b + 3 + k\}$. We have $|r - R| < 2^{2b-b(w)-8-k_2} < 2^{2b(R)-b(W)-5-k_2} < |R| 2^{b(R)-b(W)-4-k_2}$. Hence,

$$|R - r|/|R| < 2^{-(b(W)-b(R)+4+k_2)}. \tag{5.5.6}$$

Together with (5.5.4), (5.5.5), and (5.5.6), it follows from Lemma 12.1.4, that $|W/R - w/r| < 2^{-4} < 1$. This implies that $|\lfloor W/R \rceil - z| \leq 1$.

In a next step, the procedure computes $l = w - zr$. It then adjusts l and computes $l = w - (z + i)r$ for some $-1 \leq i \leq 2$.

First, we examine how close l approximates $W - (z+i)R$ after the if-statements, but before the truncation. It is $|z| + 2 \leq |W|/|R| + 3 < 2^{b(W)-b(R)+1} + 3 < 2^{b(W)-b(R)+1} + 2^{b(W)-b(R)+6} < 2^{b(W)-b(R)+7}$, because of (5.5.4). So

$$|z| + 2 < 2^{b(w)-b+9}.$$

This implies $|W-(z+i)R-l| \leq |W-w|+(|z|+2)|R-r| < |W-w|+2^{b(W)-b(R)+7}|r-R| \leq 2^{b-4-max\{4,b+2+k\}} + 2^{b(w)-b+9}2^{2b-b(w)-8-max\{8,b+3+k\}}$. Hence,

$$|W-(z+i)R-l| < 2^{min\{-k-1,b-6\}}.$$

It follows from Lemma 2.3.1, that, after truncation, l is an absolute k-approximation to $|W-(z+i)R|$. It remains to prove that $W-(z+i)R$ lies in one of the intervals. First note, that $-R \leq W-zR < 2R$, because of $|\lfloor W/R \rfloor - z| \leq 1$. If we set

$$\begin{array}{ll} i=-1, & \text{if } W-zR \leq -R/2, \\ i=0, & \text{if } -R/2 < W-zR \leq R/2, \\ i=1, & \text{if } R/2 < W-zR \leq 3/2R, \\ i=2, & \text{if } 3/2R < W-zR, \end{array}$$

then $-R/2 < W-(z+i)R \leq R/2$, and similar, if we set

$$\begin{array}{ll} i=-1, & \text{if } W-zR < 0, \\ i=0, & \text{if } 0 \leq W-zR < R, \\ i=1, & \text{if } R \leq W-zR, \end{array}$$

then $0 \leq W-(z+i)R < R$. But because we deal with approximations, we can only decide in which interval $W-zR$ lies in, if it is not too close to the bounds. We make this more precise.

For $k_l = min\{-k-1, b-6\}$, $k_r = 2b-b(w)-8-max\{8, b+3+k\}$, and $l = w-zr$, we know that $|W-zR-l| < 2^{k_l}$ and $|r-R| < 2^{k_r}$. Hence,

(5.5.7)
$$\begin{array}{ll} \text{if } l+2^{k_l} \leq -r/2-2^{k_r-1}, & \text{then } W-zR \leq -R/2, \\ \text{if } -r/2+2^{k_r-1} \leq l-2^{k_l} \text{ and } l+2^{k_l} \leq r/2-2^{k_r-1}, & \text{then } -R/2 < W-zR \leq R/2, \\ \text{if } r/2+2^{k_r-1} \leq l-2^{k_l} \text{ and } l+2^{k_l} \leq 3/2r-2^{k_r+1}, & \text{then } R/2 < W-zR \leq 3/2R, \\ \text{if } 3/2r+2^{k_r+1} \leq l-2^{k_l}, & \text{then } 3/2R < W-zR, \end{array}$$

and similarly,

$$\begin{array}{ll} \text{if } l+2^{k_l} \leq 0, & \text{then } W-zR < 0, \\ \text{if } r+2^{k_r} \leq l-2^{k_l}, & \text{then } R \leq W-zR, \\ \text{if } 2^{k_l} \leq l \text{ and } l+2^{k_l} \leq r-2^{k_r}, & \text{then } 0 \leq W-zR < R. \end{array} \tag{5.5.8}$$

For a discrete subset $M \subset \mathbb{R}$ and $x \in \mathbb{R}$, let $\delta(x, M) = min\{|x-m| | m \in M\}$ be the distance of x from M.

Now, if $\delta(W, R/2+\mathbb{Z}R) \geq 2^{k_r+2}+2^{k_l+1}$, then one of the conditions of (5.5.7) must be fulfilled. We sketch the proof for this assertion. If $W-zR \leq -R/2$, then $W-zR+2^{k_l+1} \leq -R/2-2^{k_r}$, because of the distance. Hence, $l+2^{k_l} < W-zR+2^{k_l+1} \leq -R/2-2^{k_r} < -r/2+2^{k_r-1}-2^{k_r} = -r/2-2^{k_r-1}$, so the first condition of (5.5.7) is fulfilled. The other cases are analogously. Because $W-zR$ must be in one of the 4 intervals on the right side of (5.5.7), one of the conditions on the left side is fulfilled.

Assume, that $\delta(W, R/2+\mathbb{Z}R) < 2^{k_r+2}+2^{k_l+1}$. Now $k_r = 2b-b(w)-8-max\{8, b+3+k\} \leq 2b(R)-b(W)-5-max\{8, b+3+k\} \leq b(R)-8$, because of (5.5.4). Also $k_l \leq b-6$. This implies that $2^{k_r+3}+2^{k_l+2} \leq 2^{b(R)-3}+2^{b(R)-3} = 2^{b(R)-2} \leq R/2$. Hence, we obtain $\delta(W, \mathbb{Z}R) \geq R/2-\delta(W, R/2+\mathbb{Z}R) \geq R/2-2^{k_r+2}-2^{k_l+1} \geq 2^{k_r+3}+2^{k_l+2}-2^{k_r+2}-2^{k_l+1} = 2^{k_r+2}+2^{k_l+1}$ and so, one of the conditions of (5.5.8) must be fulfilled.

This shows that the correct value of i is found and this proves (5.5.2).

It remains to prove that a logarithm representation of the minimal generator μ is determined at the end. Because we know, that there is no generator of I which logarithm is less than 1 in absolute value, and because l is an absolute 5-approximation to $\operatorname{Ln} \alpha$, we have

$$\operatorname{Ln} \alpha \geq 0 \text{ iff } l \geq 0.$$

Hence, if $l \geq 0$, it follows from (5.5.2), that $\mu = \alpha$. Otherwise, if follows from (5.5.2), that $\mu = \alpha\eta$. This proves the theorem. $\square$

CHAPTER 6

Finding minima with prescribed logarithm

Let l be a logarithm function and I be a fractional O-ideal. In this chapter we will present functions which compute both, $\alpha \in Min(I)$ such that $l(\alpha)$ is close to a given number t, and the standard representation of the reduced ideal I/α. The running time of the first function `local_close` is proportional to t. The second function `order_close` can be applied for $I = O$. Its running time is proportional to $\log|t|$. It uses the function `local_close`. The third function `close` can be applied to any fractional O-ideal I. Its running time also is proportional to $\log|t|$. It uses both functions `local_close` and `order_close`.

The methods of this chapter are an extension of the ideas of Buchmann, Thiel, and Williams (see [**BTW95**]) for computing compact representations of quadratic integers. In contrast to [**BTW95**] the procedures described here are not formulated with constant accuracies, only adjusted for computing compact representations, but parameterized accuracies allow a greater flexibility in applications. We also prove more precise complexity statements.

6.1. Minima close to a given distance

DEFINITION 6.1.1. (close) Let O be a real quadratic order and l a logarithm function. Let t be a real number, k a rational integer, and I a fractional O-ideal. The minimum $\alpha \in Min(I)$ is l-close to t with regard to k if $|t - l(\alpha)| < |t - l(\beta)| + 2^{-k}$ for any minimum $\beta \in I$. If the context is clear, we omit l, and just say close.

LEMMA 6.1.1. *Let I be a fractional ideal of O and l a logarithm function. If $\alpha \in Min(I)$ is l-close to t with regard to k then $|t - l(\alpha)| < U_{l,1}/2 + 2^{-k}$.*

PROOF. Let $\beta \in Min(I)$ such that $|l(\beta) - t| \leq |l(\gamma) - t|$ for all $\gamma \in Min(I)$. Then $|l(\beta) - t| \leq U_{l,1}/2$. So $|t - l(\alpha)| < |t - l(\beta)| + 2^{-k} < U_{l,1}/2 + 2^{-k}$. □

LEMMA 6.1.2. *Let I be a fractional O-ideal, $\alpha \in Min(I)$ with left neighbour α_{-1} and right neighbour α_{+1}, respectively. Furthermore, let $|t - l(\alpha)| < L_{l,2}/4$. If $\beta \in Min(I)$ is l-close to t with regard to $3 - b(L_{l,2})$, then $\beta \in \{\alpha_{-1}, \alpha, \alpha_{+1}\}$.*

PROOF. We extend the notation. Let α_{-2} be the left neighbour of α_{-1} and α_{+2} be the right neighbour of α_{+1}, respectively. Set $r = L_{l,2}$ and $k = 3 - b(L_{l,2})$).

It is $|l(\alpha) - l(\alpha_{\pm 2})| > r$. Because $|t - l(\alpha)| < r/4$, we have $|t - l(\alpha_{\pm 2})| > r/2$. But $|t - l(\alpha)| + 2^{-k} < r/4 + r/4 = r/2 < |t - l(\alpha_{\pm 2})|$. So, it follows from the definition, that $\alpha_{\pm 2}$ is not close to t with regard to k. But β is close to t with regard to k, and thus, β must be an element of $\{\alpha_{-1}, \alpha, \alpha_1\}$. □

6.2. Small distance, the procedure `local_close`

We describe the procedure `local_close`. Its input is a fractional O-ideal I, a floating point number t, an integer k, and a logarithm function l. The function returns the standard representation of $\alpha \in Min(I)$ such that α is l-close to t with regard to $k-1$, and transforms I into the standard representation of the reduced ideal I/α. It also returns an approximation a with $|a - l(\alpha)| < 2^{-k}$.

It works as follows. It reduces I to obtain a minimum α of I. If the logarithm approximation of α is smaller than the target t, it uses the ρ operator to find neighboured minima, such that the target t lies between the logarithm approximations of those neighboured minima. If the logarithm approximation of α is larger than t it does the same using the ρ^{-1} operator. At the end it picks up the minimum whose logarithm approximation is closer to t.

```
I.local_close( quadratic_number_standard & alpha,
               xbigfloat & a,
               xbigfloat t,
               long k,
               logarithm_function l)
 {
    quadratic_ideal B, C;
    quadratic_number_standard beta, gamma, mu;
    xbigfloat b, c;

    C = I;

    // Reduce I by division by gamma
    // and approximate l(gamma) by c.
    C.reduce(gamma);
    c = gamma.absolute_l_approximation(k);

    // Choose direction and find b <= t < c
    if (c <= t)
      {
        while (c <= t)
         {
            // Copy C to B
            B = C; beta = gamma ; b = c;

            // Move forward with C
            C.rho(mu);
            gamma *= mu;
            c = gamma.absolute_l_approximation(k);
         }
      }
    else
      {
         B = C; beta = gamma; b = c;
         while (b > t)
          {
            // Copy B to C
            C = B; gamma = beta; c = b;
```

```
          // Move backward with B
          B.inverse_rho(mu);
          beta *= mu;
          b = beta.absolute_l_approximation(k);
        }
      }

    // Choose closest
    if (t - b < c - t)
       { I = B; alpha = beta; a = b; }
    else
       { I = C; alpha = gamma; a = c; }
}
```

Proposition 6.2.1. *Let I be a fractional O-ideal, $k \in \mathbb{Z}$, t be a floating point number, and l be a logarithm function.*

On input of I, t, k, and l the function $I.$`local_close`(α, a, t, k, l) returns $\alpha \in Min(I)$ such that α is l-close to t with regard to $k-1$, and transforms I into I/α. It also returns an absolute k-approximation a to $l(\alpha)$.

Proof. We use the notation of the procedure `local_close`. From the definition of the `reduce`, `rho`, and `inverse_rho` functions we see that I is transformed into I/α, where $\alpha \in Min(I)$, and a is an absolute k-approximation to $l(\alpha)$.

It remains to prove that $|l(\alpha) - t| < |l(\mu) - t| + 2^{-k+1}$ for every minimum μ in I. Let B, β, b and C, γ, c be the variables from `local_close` initialized with the values from the end of the procedure and let $\mu \in Min(I)$.

First, we examine the case that $l(\beta) \le t < l(\gamma)$. Then $|l(\mu) - t| \ge min\{|l(\beta) - t|, |l(\gamma) - t|\} \ge min\{|b-t|, |c-t|\} - 2^{-k} = |a-t| - 2^{-k} \ge |l(\alpha) - t| - 2^{-k+1}$. So α is close to t with regard to $k-1$.

We look at the second case where $l(\beta) < l(\gamma) \le t$. Then $0 \le c - t = c - l(\gamma) + l(\gamma) - t \le c - l(\gamma) < 2^{-k}$. It follows that $|a - t| = min\{|b-t|, |c-t|\} < 2^{-k}$. So $|l(\alpha) - t| < 2^{-k+1} < |t - l(\mu)| + 2^{-k+1}$ for any $\mu \in Min(I)$. We can use the same arguments for the third case where $t < l(\beta) < l(\gamma)$. This proves the Lemma. $\square$

Proposition 6.2.2. *In the situation of Proposition 6.2.1, let the logarithm function l be either $\ln$ or Ln and let $k \ge 1$. Then the running time of the function* `local_close` *is $O(M(\operatorname{size} I)\log(\operatorname{size} I) + D[M(D)logD + M(|k| + \log D)\log(|k| + \log D)] + \operatorname{size} t)$, where $D = |t| + \log(l(I)d(I)\Delta)$.*

Proof. Let γ_0 be the value of γ after the reduction, which can be done in time $O(M(\operatorname{size} I)\log \operatorname{size} I)$. It follows from (5.4.10), that there is a constant $\kappa_1 \in \mathbb{N}$ with

$$\operatorname{size}\gamma_0 \le \kappa_1 \log(l(I)d(I)\Delta).$$

Set $d = \operatorname{size}\gamma_0$. We estimate the number of iterations s of the while loops. It is

$$s \le 2\frac{|t| + |l(\gamma_0)| + 2^{-k}}{|L_{l,\pm 2} - 2^{-k}|}.$$

The condition $k \ge 1$, implies $0.1 < \ln 2 - 2^{-k}$. Because $L_{l,\pm 2} \ge \ln 2$ by Lemma 5.3.2 and Lemma 5.3.3, and $|l(\gamma_0)| < 1 + d$ by (4.1.7) and (4.1.8), we obtain the bound

$$s \le 2(|t| + 2^{-k} + 1 + \operatorname{size}\gamma_0)/(\ln 2 - 2^{-k}).$$

Let γ_i denote the value of γ after the i-th iteration of the first while loop. We estimate its size. If c_i denotes the corresponding logarithm approximation of γ_i, then $|c_i| \le |c_0| + |t| < |t| + |l(\gamma_0)| + 2^{-k}$. This implies

$$|l(\gamma_i)| < |t| + d + 1 + 2^{-k+1}. \tag{6.2.1}$$

We know by (4.1.6) that the size of γ_i is bounded by $\log((d(\gamma_i))^2 H(\gamma_i)\Delta)$, and Lemma 5.3.4 shows that the denominator and the height are bounded by $d(I)$ and $e^{|l(\gamma_i)|} l(I)^2 \Delta^{1/2}$, respectively. Thus, together with (6.2.1), we obtain, that there is a constant $\kappa_2 \in \mathbb{N}$ such that

$$\operatorname{size} \gamma_i \le \kappa_2(|t| + \log(l(I)d(I)\Delta)),\ 0 \le i \le s. \tag{6.2.2}$$

Set $D = |t| + \log(l(I)d(I)\Delta)$. We estimate the running time of the i-th iteration of the first while loop. Computing `rho` takes time $O(M(\log\Delta))$ by Lemma 5.1.4, because C is reduced. Also, $\text{size}(\mu) \le \kappa_3 \log\Delta$ by (5.4.2) for some constant $\kappa_3 \in \mathbb{N}$. By (6.2.2), the time for computing the product is $O(M(D) logD)$. And computing the logarithm takes time $O(M(|k| + \log D)\log(|k| + \log D) + M(D)$ by Proposition 4.1.3 and Proposition 4.1.4.

It follows that all iterations can be done in time $O(s[M(D) logD + M(|k| + \log D)\log(|k| + \log D)])$ and we obtain the same complexity bound for the second while loop. If $s \ge 1$, this dominates the time for computing the logarithm of γ_0. Thus, we obtain the asserted overall bit complexity. ☐

Remark 6.2.3. If $M(n) = n^2$ in the situation of Proposition 6.2.2, then the running time can be simplified to $O(M(\operatorname{size} I) + D[M(D) + M(|k| + \log D)\log(|k| + \log D)] + \operatorname{size} t)$.

It follows from (6.2.2) that the minimum α found by `local_close` satisfies

$$\operatorname{size} \alpha \le \kappa(|t| + \log(l(I)d(I)\Delta)) \tag{6.2.3}$$

for some constant $\kappa \in \mathbb{N}$.

6.3. Minima of the order, the procedure `order_close`

We describe the function `order_close`. Its input is a real quadratic order $I = O$, a floating point number t, an integer k, and a logarithm function l. The function returns the standard representation of $\alpha \in Min(I)$ such that α is l-close to t with regard to $k-1$, and transforms I into the standard representation of the reduced ideal I/α. It also returns an approximation a with $|a - l(\alpha)| < 2^{-k}$.

The function works as follows. First it determines a minimum α_1 of O that is close to

$$t/2^{b(t)}$$

using `local_close`. It also computes $I_1 = O/\alpha_1$. Then the function uses `local_close` to determine a minimum β of I_1^2 whose logarithm is close to

$$t/2^{b(t)-1} - \operatorname{Ln}\alpha_1^2.$$

Then $\alpha_2 = \alpha_1^2\beta$ is a minimum of O whose logarithm is close to $t/2^{b(t)-1}$. After $b(t)$ iterations of this process a minimum close to t will be found.

Instead of only squaring the ideal and the minimum in each iteration, it is also possible to compute 2^m-th powers. This is specified by the input parameter m of the function.

```
I.order_close ( power_product_quadratic_number & alpha,
                xbigfloat & a,
                xbigfloat t,
                long k,
                logarithm_function l,
                long m = 1)
 {
   long n, i, j;
   xbigfloat b;
   alpha = 1;
   a = 0;
   n = ceil( b(t) / m );

   if (n <= 0)
     {
       quadratic_number_standard beta;

       if (k >= 2)
        { I.local_close (beta, b, t, k, l); alpha = beta; a = b; }
     }
   else
     {
       vector<quadratic_number_standard> beta;
       t = t / 2^(m*n);

       for (j=1; j <= n; j++)
        {
          for (i=0; i < m; i++)
           { I = I^2; a = 2*a; t = 2*t; }

          I.local_close (beta[j], b, t-a, k+1, l);
          b = beta[j].absolute_l_approximation(k+1+b(n)+m(n-j+1));
          a += b;
        }

       alpha = ((beta[1],...,beta[n]),(2^(m*(n-1)),...,2^(m*0)));
       a.truncate(b(a)+k+1);
     }
 }
```

Proposition 6.3.1. *Let O be a real quadratic order, t a floating point number, $k \in \mathbb{Z}$, l a logarithm function, and $m \in \mathbb{Z}_{\geq 1}$.*

On input of O, t, k, l, and m the function $O.$`order_close`(α, a, t, k, l, m) returns $\alpha \in Min(O)$ such that α is l-close to t with regard to $k-1$, and transforms O into O/α. It also returns an absolute k-approximation a to $l(\alpha)$.

Proof. We use the notation of the procedure. Assume that $n \leq 0$. If $k \leq 1$, the resulting ideal is O, and we have $\alpha = 1$, and $a = 0$. Hence, the assertion is true, because $|l(\alpha) - t| = |t| < 1 \leq 2^{-k+1}$ and 1 is a minimum in O. Otherwise $k \geq 2$ and the correctness follows from Proposition 6.2.1.

Assume that $n \geq 1$. Then `order_close` enters the else-part. Let $I_0 = O$, $\alpha_0 = 1$, and $a_0 = 0$ be the values of I, α, and a, respectively, immediately before the start of the loop. Furthermore, for $1 \leq j \leq n$, let I_j, β_j, a_j, and b_j be the values of I, $\beta[j]$, a, and b, respectively, at the end of the j-th iteration of the for-loop. Set

$$\alpha_j = \prod_{i=1}^{j} \beta_i^{2^{m(j-i)}}.$$

After the loop we have $\alpha = \alpha_n$.

Using induction on j we find

$$a_j = \sum_{i=1}^{j} 2^{m(n-i)} b_i$$

for $1 \leq j \leq n$. Furthermore, we have $I_j = I_{j-1}^{2^m}/\beta_j = I_0^{2^m}/\alpha_j = O/\alpha_j$. So, α_j is a minimum in O, because I_j is reduced by Proposition 6.2.1.

And we have $|l(\alpha_j) - a_j| \leq \sum_{i=1}^{j} 2^{m(j-i)} |l(\beta_i) - b_i|$ and $|l(\beta_j) - b_j| < 2^{-k-1-b(n)-m(n-j+1)}$ for $1 \leq j \leq n$. So

$$|l(\alpha_j) - a_j| < 2^{-k-m-1}$$

for $0 \leq j \leq n$. Especially, the truncated a_n is an absolute k-approximation to $l(\alpha_n)$.

Set $t_j = t/2^{m(n-j)}$. We prove that the minimum α_j of O is close to t_j with regard to $k-1$ for $1 \leq j \leq n$. Fix j and let $\gamma \in Min(O)$. Set $r = t_j - 2^m a_{j-1}$. By Lemma 5.3.1, we know that $\mu = \gamma/\alpha_{j-1}^{2^m}$ is a minimum in $I_{j-1}^{2^m} = O/\alpha_{j-1}^{2^m}$. So,

$$|l(\beta_j) - r| < |l(\mu) - r| + 2^{-k}.$$

It follows that

$$\begin{aligned}
|l(\alpha_j) - t_j| &= |l(\alpha_{j-1}^{2^m}\beta_j) - t_j| \\
&= |l(\beta_j) - r + l(\alpha_{j-1}^{2^m} - 2^m a_{j-1}| \\
&\leq |l(\beta_j) - r| + 2^m |l(\alpha_{j-1}) - a_{j-1}| \\
&< |l(\mu - r| + 2^{-k} + 2^m |l(\alpha_{j-1}) - a_{j-1}| \\
&= |l(\gamma/\alpha_{j-1}^{2^m}) - t_j + 2^m a_{j-1}| + 2^{-k} + 2^m |l(\alpha_{j-1}) - a_{j-1}| \\
&\leq |l(\gamma) - t_j| + 2^{-k} + 2^{m+1} |l(\alpha_{j-1}) - a_{j-1}| \\
&< |l(\gamma) - t_j| + 2^{-k+1}.
\end{aligned}$$

This shows that α_n is close to $t_n = t$ with regard to $k-1$. This proves the assertions. □

Proposition 6.3.2. *In the situation of Proposition 6.3.1, let the logarithm function l be either* ln *or* Ln, *let $k \geq 1$, and $m = 1$. Then the running time of the function* `order_close` *is* $O(n[\log\Delta[M(\log\Delta)\log\log\Delta + M(|k| + \log\log\Delta)\log(|k| + \log\log\Delta)] + b(m(t)) + M(|k| + n + \log\log\Delta)\log(|k| + n + \log\log\Delta)] + \operatorname{size} t)$, *where* $n = max\{1, b(t)\}$.

Proof. We use the notation of the proof of Proposition 6.3.1. We estimate the time of the j-th iteration of the loop.

Note that the targets for the calls of `local_close` satisfy $|t-a| = 2|t_{j-1} - a_{j-1}| \leq 2|t_{j-1} - l(\alpha_{j-1})| + 2|l(\alpha_{j-1}) - a_{j-1}| \leq U_{l,1} + 2 \cdot 2^{-k} + 2 \cdot 2^{-k-2}$. Hence,

$$|t-a| < U_{l,1} + 2^{-k+1} + 2^{-k-1}. \tag{6.3.1}$$

By Lemma 5.3.2, it is $U_{\ln,1}, U_{\mathrm{Ln},1} \leq (\ln \Delta)/2$. It follows from (6.2.3) and (6.3.1), that

$$\operatorname{size} \beta_j = O(\log \Delta),$$

We estimate the time for the floating point operations. Let a_j denote the value of a and b_j denote the value of b at the end of the j-th iteration. Lemma 2.4.5, (4.1.3) and (4.1.4), and the bound on $\operatorname{size} \beta_j$ imply

$$|e(b_j)|, b(m(b_j)) \leq \kappa_1(\log\log\Delta + |k| + b(n) + (n-j)),$$

for some constant $\kappa_1 \in \mathbb{N}$. Because $a_j = \sum_{i=1}^{j} 2^{j-i} b_i$, it follows from Theorem 2.3.3 that

$$|e(a_j)|, b(m(a_j)) \leq \kappa_2(\log\log\Delta + |k| + n),$$

for some constant $\kappa_2 \in \mathbb{N}$. And if t_j denotes the value of t at the end of the j-th iteration, then

$$t_j = m(t) 2^{j-1}.$$

It follows that the floating point operations, except the approximation of the logarithm, in the j-iteration of the loop take time $O(\log\log\Delta + |k| + n + b(m(t)))$. By Proposition 4.1.3 and Proposition 4.1.4, the time for computing the logarithm approximation b_j is $O(M(|k| + n + \log\log\Delta)\log(|k| + n + \log\log\Delta) + M(\log\Delta))$.

Because I is the square of a reduced ideal, it is

$$\operatorname{size} I = O(\log \Delta),$$

and computing the square takes time $O(M(\log\Delta)\log\log\Delta)$. It follows from Lemma 5.1.4, that

$$l(I) = 1, \; d(I) < \Delta.$$

Proposition 6.2.2 implies, that `local_close` takes time $O(\log\Delta[M(\log\Delta)\log\log\Delta + M(|k| + \log\log\Delta)\log(|k| + \log\log\Delta)] + b(m(t)))$.

Thus, the running time for the loop is $O(n(\log\Delta[M(\log\Delta)\log\log\Delta + M(|k| + \log\log\Delta)\log(|k| + \log\log\Delta)] + b(m(t)) + M(|k| + n + \log\log\Delta)\log(|k| + n + \log\log\Delta)))$.

The time for generating α is $O(n\log\Delta + n^2)$. This is dominated by the time for the loop.

If $n \leq 0$ and $k \geq 2$, then

$$|t| < 1, \tag{6.3.2}$$

so $O(\log\Delta[M(\log\Delta)\log\log\Delta + M(|k| + \log\log\Delta)\log(|k| + \log\log\Delta)] + \operatorname{size} t)$ is the time for `local_close` in that case. This proves the asserted overall running time. $\square$

Remark 6.3.3. If $M(n) = n^2$ in the situation of Proposition 6.3.2, then the running time can be simplified to $O(n[\log\Delta[M(\log\Delta) + M(|k| + \log\log\Delta)\log(|k| + \log\log\Delta)] + b(m(t)) + M(|k| + n + \log\log\Delta)\log(|k| + n + \log\log\Delta)] + \operatorname{size} t)$.

The minimum α of O found by `order_close`, which is l-close to t with regard to $k-1$, is represented as a `quadratic_number_power_product`. Suppose that $m = 1$. Then it is of the form $((\beta_1, \dots, \beta_n), (2^{n-1}, 2^{n-2}, \dots, 2^0))$ and represents the number

$$\alpha = \prod_{j=1}^{n} \beta_j^{2^{n-j}}, \tag{6.3.3}$$

for $n = max\{b(t), 1\}$, with

$$\beta_j \in Min(O/\prod_{i=1}^{j-1} \alpha_i^{2^{j-i}}) \tag{6.3.4}$$

Because β_j is a minimum l-close to a target bounded by (6.3.1) with regard to k or bounded by (6.3.2) with regard to $k-1$, it follows from Lemma 6.1.1, that

$$|l(\beta_j)| < 3/2U_{l,1} + 2^{-k+3}, \tag{6.3.5}$$

for $1 \le j \le n$. Set $I_j = O/\prod_{i=1}^{j} \alpha_i^{2^{j-i}}$ for $1 \le j \le n$ and $I_0 = O$. We know that β_j is a minimum in the square of the reduced ideal I_{j-1}. Especially, $l(I_{j-1}^2) = 1$ and $d(I_{j-1}^2) < \Delta$ by Lemma 5.1.4. Hence, it follows from Lemma 5.3.4, that

$$d(\beta_j) < \Delta,\ d(1/\beta_j) < \sqrt{\Delta}. \tag{6.3.6}$$

Furthermore, it is $1/\Delta < N(I_{j-1}^2)$ by Lemma 5.1.4. Now if the logarithm function is $l = \mathrm{Ln}$, then it follows from Lemma 5.3.4, (6.3.5), and Lemma 5.3.2, that

$$H(\beta_j) < \Delta e^{2^{-k+3}},\ H(1/\beta_j) < \Delta^{5/4} e^{2^{-k+3}}. \tag{6.3.7}$$

Similarly, if the logarithm function is $l = \ln$, Lemma 5.3.4, (6.3.5), and Lemma 5.3.3 imply

$$H(\beta_j) < \Delta^{5/4} e^{2^{-k+3}},\ H(1/\beta_j) < \Delta^{7/4} e^{2^{-k+3}}. \tag{6.3.8}$$

And as derived in the proof of Proposition 6.3.2, it is $\operatorname{size} \beta_j = O(\log \Delta)$.

6.4. Minima of ideals, the procedure `close`

We describe the procedure `close`. Its input is a fractional O-ideal I, a floating point number t, an integer k, and a logarithm function l. The function returns the standard representation of $\alpha \in Min(I)$ such that α is l-close to t with regard to $k-1$, and transforms I into the standard representation of the reduced ideal I/α. It also returns an approximation a with $|a - l(\alpha)| < 2^{-k}$.

It works as follows. First it determines a minimum β in the order of I that is close to t with regard to $k+1$ using `order_close`. Then `local_close` is used to find a minimum γ of I/β that is close to an approximation of $t - l(\beta)$ with regard to k. Then $\alpha = \beta\gamma$ is a minimum of I that is l-close to t with regard to $k-1$.

```
I.close ( power_product_quadratic_number & alpha,
          xbigfloat & a,
          xbigfloat t,
          long k,
          logarithm_function l)
 {
   quadratic_ideal J;
   power_product_quadratic_number beta;
```

```
    quadratic_number_standard gamma;
    xbigfloat b,c;

    J = I.order;
    J.order_close(beta, b, t, k+2, l);

    I = I * J;
    I.local_close (gamma, c, t-b, k+2, l);

    alpha = beta * gamma;
    a = b + c;
    a.truncate(b(a)+k+1);
  }
```

PROPOSITION 6.4.1. *Let I be a fractional O-ideal, $k \in \mathbb{Z}$, t be a floating point number, and l be a logarithm function.*

On input of I, t, k, and l the function $I.\texttt{close}(\alpha, a, t, k, l)$ returns $\alpha \in Min(I)$ such that α is l-close to t with regard to $k-1$, and transforms I into I/α. It also returns an absolute k-approximation a to $l(\alpha)$.

PROOF. We use the notation of the procedure. In addition, let I_0 denote the ideal I at the beginning of the procedure, and let O be its order.

We obtain from Proposition 6.3.1 that $J = O/\beta$ after the call of `order_close`, where the minimum β of O is l-close to t with regard to $k+1$ and $|l(\beta) - b| < 2^{-k-2}$.

Then I is transformed into $I_0 \cdot J = I_0/\beta$ and after the call of `local_close` the resulting ideal is $I = I_0/(\beta\gamma)$, where $\gamma \in Min(I_0/\beta)$. Hence, $\alpha = \beta\gamma$ is a minimum in I_0. Also, Proposition 6.2.1 yields $|l(\alpha) - a| \leq |l(\beta) - b| + |l(\gamma) - c| < 2^{-k-2} + 2^{-k-2} < 2^{-k-1}$ before truncation. Hence, the truncated a is an absolute k-approximation to $l(\alpha)$ by Lemma 2.3.1.

We prove that α is close to t with regard to $k-1$. Let $\mu \in Min(I_0)$. We have $|t - l(\alpha)| = |t - l(\beta) - l(\gamma)| < |t - b - l(\gamma)| + 2^{-k-2}$. Because by Proposition 6.3.1, $\gamma \in Min(I_0/\beta)$ is close $t-b$ with regard to k and μ/β also is a minimum in I_0/β, it follows that $|t - b - l(\gamma)| < |t - b - l(\mu/\beta)| + 2^{-k}$. We know that $|b - l(\beta)| < 2^{-k-2}$. So $|t - l(\alpha)| < |t - l(\mu)| + 2^{-k-2} + 2^{-k} + 2^{-k-2} < |t - l(\mu)| + 2^{-k+1}$, i.e., α is close to t with regard to $k-1$. This completes the proof. $\square$

PROPOSITION 6.4.2. *In the situation of Proposition 6.4.1, let the logarithm function l be either* ln *or* Ln*, and let $k \geq 1$. Then the running time of the function* `close` *is $O(n[\log\Delta[M(\log\Delta)\log\log\Delta + M(|k| + \log\log\Delta)\log(|k| + \log\log\Delta)] + b(m(t)) + M(|k| + n + \log\log\Delta)\log(|k| + n + \log\log\Delta)] + M(\operatorname{size} I)\log(\operatorname{size} I) + D[M(D) logD + M(|k| + \log D)\log(|k| + \log D)] + \operatorname{size} t)$, where $n = max\{1, b(t)\}$ and $D = \log(l(I)d(I)\Delta)$.*

PROOF. We use the notation of the procedure.

We estimate the time for `local_close`. Note that $|t-b| \leq |t - l(\beta)| + |l(\beta) - b|$. It follows from Lemma 6.1.1, that

$$|t - b| < U_{l,1}/2 + 2^{-k-1} + 2^{-k-2}. \tag{6.4.1}$$

Hence, Lemma 5.3.2 and Lemma 5.3.3 imply that for $l = \mathrm{Ln}$ or $l = \ln$, it is

$$|t - b| < (\ln\Delta)/4 + 2^{-k}.$$

Let I_0 denote the ideal I at the beginning of the procedure, and let O be its order. `local_close` is applied to the ideal $I = I_0/\beta$, where O/β is reduced. It follows from Lemma 5.1.4, that

$$
\begin{array}{rcl}
l(I_0/\beta) & \leq & l(I_0), \\
d(I_0/\beta) & \leq & d(I_0)\sqrt{\Delta},
\end{array}
\tag{6.4.2}
$$

and it is $\mathrm{size}(I_0/\beta) \leq \kappa \,\mathrm{size}\, I_0$ for some constant $\kappa \in \mathbb{N}$, because O/β is reduced. Hence, it follows from Proposition 6.2.2, that the time for `local_close` is $O(M(\mathrm{size}\, I_0)\log(\mathrm{size}\, I_0) + D[M(D)logD + M(|k| + \log D)\log(|k| + \log D)] + \mathrm{size}\, t)$, where $D = \log(l(I_0)d(I_0)\Delta)$.

It follows from Proposition 6.3.2, that the time for the call of `order_close` is $O(n(\log\Delta[M(\log\Delta)\log\log\Delta + M(|k| + \log\log\Delta)\log(|k| + \log\log\Delta)] + b(m(t)) + M(|k| + n + \log\log\Delta)\log(|k| + n + \log\log\Delta)) + +\,\mathrm{size}\, t)$, where $n = max\{1, b(t)\}$.

The time for computing $|t-b|$ is $O(|b(t) - b(b)| + \mathrm{size}(t) + \mathrm{size}(b))$. We estimate that time. It follows from (4.1.6), Lemma 5.3.4, and Lemma 6.1.1, that there is a constant $\kappa_1 \in \mathbb{N}$ such that

$$\mathrm{size}_s(\beta) \leq \kappa_1(|t| + \log\Delta).$$

Thus, Lemma 2.4.5 and (4.1.3), (4.1.4) imply the existence of a constant $\kappa_2 \in \mathbb{N}$ with

$$|e(b)|, b(m(b)) \leq \kappa_2(|k| + max\{b(t), 0\} + \log\log\Delta).$$

Hence, the time for computing $|t - b|$ is $O(|k| + max\{b(t), 0\} + \log\log\Delta + \mathrm{size}\, t)$, which is dominated by the time for `order_close`.

This proves the overall running time. □

REMARK 6.4.3. If $M(n) = n^2$ in the situation of Proposition 6.4.2, then the running time can be simplified to $O(n[\log\Delta[M(\log\Delta) + M(|k| + \log\log\Delta)\log(|k| + \log\log\Delta)] + b(m(t)) + M(|k| + n + \log\log\Delta)\log(|k| + n + \log\log\Delta)] + M(\mathrm{size}\, I) + D[M(D) + M(|k| + \log D)\log(|k| + \log D)] + \mathrm{size}\, t)$.

The minimum α found by `close`, which is l-close to t with regard to $k-1$, is represented as a `quadratic_number_power_product`. It follows from (6.3.3), that it is of the form $((\beta_1, \ldots, \beta_n, \gamma), (2^{n-1}, \ldots, 2^0, 1))$, and represents the number

$$\alpha = \gamma \prod_{j=1}^{n} \beta_j^{2^{n-j}}, \tag{6.4.3}$$

where $n = max\{b(t) + 1, 1\}$. Bounds on β_j are given by (6.3.5) - (6.3.8).

Because γ is a minimum in I/β, it follows from Lemma 5.3.4 and (6.4.2), that

$$d(\gamma) < d(I)\sqrt{\Delta},\ d(1/\gamma) < l(I)\sqrt{\Delta}. \tag{6.4.4}$$

Furthermore, Lemma 6.1.1 and (6.4.1) imply

$$|l(\gamma)| < U_{l,1} + 2^{-k+1}. \tag{6.4.5}$$

Now, if the logarithm function is $l = \mathrm{Ln}$, it follows from Lemma 5.3.4 and Lemma 5.3.2, that

$$H(\gamma) < \Delta^{3/4} e^{2^{-k+1}} l(I),\ H(1/\gamma) < \Delta^{3/4} e^{2^{-k+1}} / \sqrt{N(I)}. \tag{6.4.6}$$

Similarly, if the logarithm function is $l = \ln$, it follows from Lemma 5.3.4 and Lemma 5.3.3, that

$$H(\gamma) < \Delta e^{2^{-k+1}} l(I)^2,\ H(1/\gamma) < \Delta e^{2^{-k+1}}/N(I). \tag{6.4.7}$$

CHAPTER 7

Fundamental unit computation

In this chapter we present an algorithm for computing the fundamental unit of a real quadratic order. Its input is an approximation to the regulator.

We begin with an algorithm for converting the logarithm representation of a quadratic number into a reduced power product representation. We apply that algorithm to compute a compact representation of the fundamental unit. In the last section we describe an algorithm that on input of an approximation to the logarithm of a quadratic number returns a more accurate approximation of the logarithm.

7.1. Converting from logarithm to reduced power product representation

Let $\alpha \in K^*$. We present an algorithm for converting a logarithm representation of α into a reduced power product representation.

Let p be of type `quadratic_number_power_product` and $l = (A, a, k, s, \mathrm{Ln})$ be a logarithm representation for $\alpha \in K^*$, i.e., $A = \alpha O$ in standard representation, $|a - \mathrm{Ln}\,\alpha| < 2^{-k}$, $k \in \mathbb{Z}$, and $s \in \{\pm 1\}$ with $sign(\alpha) = s$. Furthermore, $k \geq 3$, if A reduced, and $k \geq 4$, otherwise. We present the function p.`assign_reduced_power_product`(l) of the class `quadratic_number_power_product`, that on input of l, assigns to p a reduced power product representation of α.

```
void p.assign_reduced_power_product ( quadratic_number_logarithm l )
 {
   // Determine the accuracy.
   int r = 3;

   // C = A / gamma reduced
   //
   quadratic_number_standard gamma;
   xbigfloat c;
   quadratic_ideal C = l.A;

   if (C.is_reduced())
    {
      gamma.assign_one(C.order);
      c = 0;
    }
   else
    {
      C.reduce(gamma);
```

```
      c = gamma.absolute_Ln_approximation(r+1);
    }

   xbigfloat a;
   if (l.k <= 4)
      a = l.a;
   else
      a = truncate(l.a, b(l.a)+5);

   // D = O / beta close to Ln(gamma/alpha)
   //
   quadratic_number_power_product beta;
   xbigfloat b;
   xbigfloat m = c - a;
   quadratic_ideal D = l.A.order;
   D.order_close(beta, b, m, r+1);

   // Found C again ?
   if (D != C)
    {
      // If not, move to the left neighbour of D.
      //
      quadratic_number_standard nu;
      quadratic_ideal N = D;
      N.inverse_rho(nu);
      xbigfloat n = nu.get_absolute_Ln_approximation(r+1);

      // If rho^{-2}(C) was found, C is the right neighbour of D.
      // Otherwise, C is the left neighbour of D.
      //
      if (b + n < m - 2^{-r+1})
        D.rho(nu);

      // Multiply last base element of beta by nu.
      beta.multiply_base_element(beta.length, nu);
    }

   // p = gamma / beta;
   beta.invert_base_elements();
   p = beta * gamma;

   // Adjust sign
   if (s == -1)
      p.negate();
}
```

First, we prove that the function determines a power product representation for α.

LEMMA 7.1.1. *Let $\alpha \in K^*$. On input of a logarithm representation l for α with logarithm function* Ln*, the function p.`assign_reduced_power_product`(l) computes a power product representation p of α.*

PROOF. We use the notation of the procedure. Let the logarithm representation of α be $l = (A, a, k, s, \mathrm{Ln})$. As a first step the function reduces A to $C = A/\gamma$ with $\gamma \in K^*$. Hence, for

$$\mu = \gamma/|\alpha|,$$

we have $C = O/\mu$, and μ is a minimum of O. To prove the Lemma, we will show that the procedure computes $\beta = \mu$. Then $|\alpha| = \gamma/\beta$, and p is a power product representation of α after the adjustment of the sign.

We introduce some notation. By $\mu_i = \rho_C^i(\mu)$ for $i = -2, -1, 1, 2$, we denote the two predecessors and successors of μ in the minima set of C. We know from Lemma 5.3.2, that for the choice of $r = 3$, we have

$$2^{-r+2} < |\operatorname{Ln}\mu - \operatorname{Ln}\mu_{\pm 2}|.$$

We examine the approximations to the Ln values that are computed by the procedure. We have $|m - \operatorname{Ln}\mu| = |c - a - \operatorname{Ln}\gamma + \operatorname{Ln}\alpha| \le |c - \operatorname{Ln}\gamma| + |a - \operatorname{Ln}\alpha|$. Because $|a - \operatorname{Ln}\alpha| < 2^{-3}$, if A is reduced and $|a - \operatorname{Ln}\alpha| < 2^{-4}$ otherwise, we obtain

$$|m - \operatorname{Ln}\mu| < 2^{-r}.$$

After initializing D by O, the call of D.`order_close`$(\beta, b, m, r+1)$ yields $D = O/\beta$, where $\beta \in Min(O)$, and

$$|b - \operatorname{Ln}\beta| < 2^{-r-1}.$$

Let N, ν, and n be the values that are determined after the ρ^{-1} step in first `if` statement. We have $N = \rho^{-1}(D) = D/\nu$, and

$$|n - \operatorname{Ln}\nu| < 2^{-r-1}.$$

Now, we prove that the following four assertions hold after the call of `order_close`:

1. $\beta \in \{\mu_{-1}, \mu, \mu_1\}$.
2. $\beta = \mu$ if and only if $C = D$.
3. If $\beta = \mu_1$, then $b + n > m - 2^{-r+1}$.
4. If $\beta = \mu_{-1}$, then $b + n < m - 2^{-r+1}$.

If these assertions are proven, then it immediately follows that at the end of the procedure, it is $\beta = \mu$.

We prove the first assertion. It is $|m - \operatorname{Ln}\mu| < 2^{-3}$. It follows from the property of the function `order_close`, Lemma 6.3.1, that β is close to m with regard to 3. Hence, the assertion follows from Lemma 6.1.2.

We prove the second assertion. Let $C = D$. Then $O/\beta = D = C = O/\mu$. Therefore $\beta = \mu$ or $|\operatorname{Ln}(\beta/\mu)| = R$, the regulator of O. But we know from Proposition 6.3.1, that β is close to m with regard to r. So $|\operatorname{Ln}(\beta/\mu)| = |\operatorname{Ln}\beta - \operatorname{Ln}\mu| \le |m - \operatorname{Ln}\beta| + |m - \operatorname{Ln}\mu| < 2|m - \operatorname{Ln}\mu| + 2^{-r} < 2^{-r+1} + 2^{-r} < 0.4 < R$. Hence, $\beta = \mu$. And $\beta = \mu$ obviously implies $C = D$.

We show the third assertion. Assume that $\beta = \mu_1$. Then $\mu = \beta\nu$ and $|m - b - n| \le |m - \operatorname{Ln}\beta - \operatorname{Ln}\nu| + |b - \operatorname{Ln}\beta| + |n - \operatorname{Ln}\nu| < |m - \operatorname{Ln}\mu| + 2^{-r} < 2^{-r+1}$. Hence, $b + n > m - 2^{-r+1}$.

We prove the fourth assertion. Assume that $\beta = \mu_{-1}$. Then $\mu_{-2} = \beta\nu$. We obtain $b + n < \operatorname{Ln}\beta + 2^{-r-1} + \operatorname{Ln}\nu + 2^{-r-1} = \operatorname{Ln}\mu_{-2} + 2^{-r} \leq \operatorname{Ln}\mu - \ln 2 + 2^{-r} < m + 2^{-r+1} - \ln 2 < m + 2^{-r+1} - 2^{-r+2} = m - 2^{-r+1}$. □

Now, we prove that the power product determined by the procedure is reduced.

LEMMA 7.1.2. *Let $\alpha \in K^*$. On input of a logarithm representation l for α with logarithm function* Ln*, the function p.`assign_reduced_power_product`(l) computes a reduced power product representation p of α.*

PROOF. It has been shown in Lemma 7.1.1, that p is a power product representation for α. It remains to prove that p is reduced. As a first step, the function reduces $A = \alpha O$ by dividing it by γ. We know from (5.4.9) and Lemma 5.1.1 that

$$H(\gamma) \leq |N(\alpha)|d(\alpha),\ d(\gamma) \leq d(\alpha).$$

The call of `order_close` yields $C = O/\beta$. We know from (6.3.3), (6.3.6), and (6.3.7) that

$$\beta = \prod_{j=1}^{n} \beta_j^{2^{n-j}},$$

where $n = max\{b(m), 1\}$, $H(1/\beta_j) < \Delta^{5/4}e^{2^{-1}} < 2\Delta^{5/4}$, and $d(1/\beta_j) < \sqrt{\Delta}$ for $1 \leq j \leq n$.

We estimate n. We have $|m| = |c - a| \leq |c - \operatorname{Ln}\gamma| + |a - \operatorname{Ln}\alpha| + |\operatorname{Ln}(\gamma/\alpha)| < 2^{-3} + |\operatorname{Ln}(\gamma/\alpha)|$. Because γ/α is a minimum of O, the absolute value of the norm of γ/α is at least 1. Also, $|\gamma/\alpha| \leq H(\gamma)/|\alpha| \leq |N(\alpha)|d(\alpha)/|\alpha| \leq H(\alpha)d(\alpha)$. Analogously, we obtain $|\sigma\gamma/\sigma\alpha| \leq H(\alpha)d(\alpha)$. This implies $H(\gamma/\alpha) \leq H(\alpha)d(\alpha)$. Thus, by Lemma 4.1.1, we know that $|\operatorname{Ln}(\gamma/\alpha)| \leq 1/2\ln(H(\gamma/\alpha)^2/N(\gamma/\alpha)|) \leq \ln(H(\alpha)d(\alpha))$. Therefore, $|m| < 1/8 + \ln(H(\alpha)d(\alpha))$. In case of $\ln(H(\alpha)d(\alpha)) < 1/8$, we obtain $b(m) \leq -2$. Otherwise, $b(m) \leq log|m| + 1 < log(2\ln(H(\alpha)d(\alpha))) + 1 = log(2\ln 2) + \log\log(H(\alpha)d(\alpha)) + 1 < 2 + \log\log(H(\alpha)d(\alpha))$. So,

$$n \leq 2 + \log\log max\{H(\alpha)d(\alpha), 2\}.$$

After the call of `order_close` there is at most one further application of the ρ or ρ^{-1} operator. If the procedure does not enter the if-statement, set $\nu = 1$. Otherwise, let ν be the minimum by which β is multiplied. This multiplication is done by replacing β_n by $\beta_n\nu$. In this case, we know from (6.3.4), that β_n is a minimum in $M = O/\prod_{j=1}^{n-1}\beta_j^{2^{n-j}}$. Because $C = M/(\beta_n\nu)$ is reduced, $\beta_n\nu$ is a minimum in M too.

We estimate denominator and height of $1/(\beta_n\nu)$. Because M is the square of a reduced ideal, 1 is in M and so, by Lemma 5.3.4, $d(1/(\beta_n\nu)) < \sqrt{\Delta}$. Furthermore, $|\operatorname{Ln}\nu| = |\operatorname{Ln}\beta - \operatorname{Ln}(\gamma/\alpha)| < |m - \operatorname{Ln}\beta| + |m - \operatorname{Ln}(\gamma/\alpha)|$. Lemma 6.3.1 yields $|m - \operatorname{Ln}\beta| + |m - \operatorname{Ln}(\gamma/\alpha)| < 2|m - \operatorname{Ln}(\gamma/\alpha)| + 2^{-3} < 2^{-2} + 2^{-3}$. Using the estimate on $|\operatorname{Ln}\beta_n|$ given by (6.3.5), we obtain $|\operatorname{Ln}(\beta_n\nu)| \leq |\operatorname{Ln}\beta_n| + |\operatorname{Ln}\nu| < 3/4\ln\Delta + 2^{-2} + 2^{-2} + 2^{-3} < 3/4\ln\Delta + 0.625$. Lemma 5.1.4 yields $1/\Delta < N(M) \leq N(\beta_n\nu)$. Both bounds together with Lemma 5.3.4 show that $H(1/(\beta_n\nu)) < 2\Delta^{5/4}$.

If we replace β_n by $\beta_n\nu$, the resulting power product is of the form

$$p = sign(\alpha)\gamma\prod_{j=1}^{n}(1/\beta_j)^{2^{n-j}}.$$

For $1 \le j \le n$ let d_j be the denominator of $1/\beta_j$. Then we have $0 < d_j \le \sqrt{\Delta}$ and $1/\beta_j = \alpha_j/d_j$ with $\alpha_j \in O$, where the heights are bounded by $H(\alpha_j) \le H(1/\beta_j)d_j < 2\Delta^{7/4}$ for $1 \le j \le n$. Furthermore, $H(\gamma) \le |N(\alpha)|d(\alpha)$, and $d(\gamma) \le d(\alpha)$. So p is a reduced power product representation of α. □

Including the running time for the computation of a reduced power product, we have the final proposition.

Proposition 7.1.3. *Let $\alpha \in K^*$. On input of a logarithm representation $l = (A, a, s, k, Ln)$ for α, the function $p.$`assign_reduced_power_product`(l) computes a reduced power product representation p of α in time $O(N[\log \Delta M(\log \Delta) \log\log \Delta + M(N + \log\log \Delta) \log(N + \log\log \Delta)] + M(\operatorname{size} A) \log(\operatorname{size} A) + k)$, where $N = 2 + \log\log(max\{H(\alpha)d(\alpha), 2\})$.*

Proof. The correctness of the procedure has been proven in Lemma 7.1.2. We estimate the running time.

Reducing A takes time $O(M(\operatorname{size} A) \log(\operatorname{size} A))$, and it follows from (5.4.10), that $\operatorname{size}\gamma \le c_1 \log(r(A)(d(A))^2\Delta)$ for some constant $c_1 \in \mathbb{N}$. It follows from Lemma 5.1.1, that $\operatorname{size}\gamma \le c_2 \operatorname{size} A$ for some constant $c_2 \in \mathbb{N}$. So, Proposition 4.1.4 and (1.1.1) imply, that the logarithm approximation c can be determined in time $O(M(\operatorname{size} A))$, which is dominated by the reduction time.

Lemma 4.1.5 together with (5.4.10) gives the estimate on the size of c, i.e.

$$b(m(c)), |e(c)| = O(\log \operatorname{size} A).$$

Also, Lemma 4.1.5 implies that $b(m(a)), |e(a)| = O(\log\log(d^2(\alpha)H(\alpha)\Delta))$. An easy calculation shows that

$$b(m(a)), |e(a)| = O(\log \operatorname{size} A + \log\log max\{d(\alpha)H(\alpha), 2\}).$$

Hence, m can be found in time $O(\log \operatorname{size} A + \log\log max\{d(\alpha)H(\alpha), 2\})$.

The time for the call of `order_close` is $O(n[\log \Delta M(\log \Delta) \log\log \Delta + b(m(m)) + M(n + \log\log \Delta) \log(n + \log\log \Delta)] + size(m))$, where $n = max\{1, b(m)\}$ by Proposition 6.3.2. We know from the proof of Lemma 7.1.2 that

$$n \le 2 + \log\log max\{H(\alpha)d(\alpha), 2\}.$$

And it follows from the bounds on the sizes of a and c above, that $b(m(m)), \operatorname{size}(m) = O(\log \operatorname{size} A + \log\log max\{d(\alpha)H(\alpha), 2\})$.

We estimate the exponent of b. It is $b(m(b)), |e(b)| = O(\log\log(H(\beta)d^2(\beta)\Delta))$ by Lemma 4.1.5. It follows from (6.3.6) and (6.3.7), that $d(\beta), H(\beta) = O(\Delta^{2^N})$. Hence, $b(m(b)), |e(b)| = O(N + \log\log \Delta)$.

Together with the fact, that the ideals inside the if-part are reduced, this shows that the running time of the rest of the function is dominated by the running times above.

Hence, the overall running time of the procedure is $O(N[\log \Delta M(\log \Delta) \log\log \Delta + M(N + \log\log \Delta) \log(N + \log\log \Delta)] + M(\operatorname{size} A) \log(\operatorname{size} A) + k)$, where $N = 2 + \log\log(max\{H(\alpha)d(\alpha)\}, 2)$. The additional term k appears, because $b(m(l.a)), |e(l.a)| = O(\log\log(H(\alpha)d^2(\alpha)\Delta) + k)$ by Lemma 4.1.5. □

Remark 7.1.4. If $M(n) = n^2$ in the situation of Proposition 7.1.3, the running time of the procedure can be simplified to $O(N[M(\log \Delta) \log \Delta + M(N + \log\log \Delta) \log(N + \log\log \Delta)] + M(\operatorname{size} A) + k)$.

7.2. Computing the fundamental unit from a regulator approximation

Let η be the fundamental unit of O, and $R = \operatorname{Ln} \eta$ be its regulator. Furthermore let r be an approximation with

$$|r - R| < 2^{-k},\ k \geq 3.$$

If we want to compute the fundamental unit using r, we have to specify the representation in which we are interested. For example, the tupel

$$l = (O, r, k, 1, \operatorname{Ln}) \tag{7.2.1}$$

is a logarithm representation of the fundamental unit. Usually computing the fundamental unit means to find a power product representation of η. But this is a straight forward application of the function described in Section 7.1 for converting a logarithm representation (7.2.1) into a power product representation. The following function O.`fundamental_unit`(r, k) returns a reduced power product representation of the fundamental unit of O.

```
quadratic_number_power_product
O.fundamental_unit (

            xbigfloat r,
            long k )
 {
    quadratic_number_logarithm l = (O,r,k,1,Ln);
    quadratic_number_power_product p;
    p.assign_reduced_power_product(l);
    return p;
 }
```

PROPOSITION 7.2.1. *On input of $k \in \mathbb{Z}_{\geq 3}$ and an absolute k-approximation r to the regulator of O, the function O.`fundamental_unit`(r, k) returns a reduced power product representation of the fundamental unit of O and its running time is $O((\log \Delta)^2 M(\log \Delta) \log\log \Delta + k)$.*

PROOF. Because the tupel l is a correct logarithm representation for the fundamental unit, the correctness of the function is an immediate consequence of Proposition 7.1.3. And the Proposition also yields the asserted running time, if we note that $\log\log(H(\eta)) = O(\log \Delta)$, where η denotes the fundamental unit. $\square$

REMARK 7.2.2. If $M(n) = n^2$ in the situation of Proposition 7.2.1, the running time of the procedure can be simplified to $O((\log \Delta)^4 + k)$.

7.3. Refining a logarithm approximation

Let $\alpha \in K^*$. Suppose that we know an absolute k-approximation to $\operatorname{Ln} \alpha$. To determine a more accurate approximation to $\operatorname{Ln} \alpha$ we compute a reduced power product representation of α, and approximate its logarithm to the desired accuracy.

That algorithm is implemented by a function of `quadratic_number_logarithm`. Let $l = (A, a, k, s, Ln)$ be a k-logarithm representation of α, i.e., $A = \alpha O$, a an absolute k-approximation to $\operatorname{Ln} \alpha$ with $k \geq 3$, if A is reduced, and $k \geq 4$, otherwise, and $s = sign(\alpha)$. The following function α.`refine_logarithm_approximation`(r) transforms l into a r-logarithm representation (A, b, r, s, Ln) of α.

```
void l.refine_logarithm_representation (int r)
 {
   quadratic_number_power_product p;

   if (r > k)
    {
      p.assign_reduced_power_product(l);
      a = p.absolute_Ln_approximation(r);
      k = r;
    }
 }
```

PROPOSITION 7.3.1. *Let $\alpha \in K^*$ with k-logarithm representation $l = (A, a, k, s, Ln)$, and $\mathbb{Z} \ni r > k$. The function l.`refine_logarithm_representation`(r) transforms l into a r-logarithm representation of α in time $O(N[\log\Delta M(\log\Delta)\log\log\Delta + M(r+N+\log\log\Delta)\log(r+N+\log\log\Delta)] + M(\operatorname{size} A)\log(\operatorname{size} A) + M(r+\log t)\log(r+\log t)$, where $N = 2+\log\log(max\{H(\alpha)d(\alpha), 2\})$ and $t = \log(|N(\alpha)|d^3(\alpha)\Delta)$.*

PROOF. The correctness follows from Proposition 7.1.3 and Remark 4.2.3.

We estimate the bit complexity of the procedure. By Proposition 7.1.3, the time for finding the reduced power product representation of α is $O(N[\log\Delta M(\log\Delta)\log\log\Delta + M(N+\log\log\Delta)\log(N+\log\log\Delta)] + M(\operatorname{size} A)\log(\operatorname{size} A) + k)$, where $N = 2+\log\log(max\{H(\alpha)d(\alpha), 2\})$.

By Theorem 4.2.4, the approximation of $\operatorname{Ln}\alpha$, for α in reduced power product representation, is $O(N[M(r+N+\log\log\Delta)\log(r+N+\log\log\Delta) + M(\log\Delta)] + M(r+\log t)\log(r+\log t) + M(t)$, where $t = \log(|N(\alpha)|d^3(\alpha)\Delta)$.

Hence, the overall running time is $O(N[\log\Delta M(\log\Delta)\log\log\Delta + M(r+N+\log\log\Delta)\log(r+N+\log\log\Delta)] + M(\operatorname{size} A)\log(\operatorname{size} A) + M(r+\log t)\log(r+\log t) + M(t))$. This proves the assertion, if we note that $t \leq c \operatorname{size} A$ for some constant $c \in \mathbb{N}$. $\square$

REMARK 7.3.2. If $M(n) = n^2$ in the situation of Proposition 7.3.1, then the running time can be simplified to $O(N[\log\Delta M(\log\Delta) + M(r+N+\log\log\Delta)\log(r+N+\log\log\Delta)] + M(\operatorname{size} A) + M(r+\log t)\log(r+\log t)$.

CHAPTER 8

Approximation of $L(1, \chi_\Delta)$

Throughout this chapter let Δ be a quadratic discriminant and O the corresponding quadratic order. In this chapter, we show, how the results of Eric Bach [**Bac95**] and of Chapter 2 can be used to compute an approximation to $L(1, \chi_\Delta)$.

8.1. The procedure L1chi

For positive real numbers x set

$$B(x, \chi_\Delta) = \prod_{p<x} \left(1 - \frac{\chi_\Delta(p)}{p}\right)^{-1}, \tag{8.1.1}$$

where the product is taken over all prime numbers less than x.

Let n be an integer, $n \geq 2$, and the weights a_i be defined by

$$a_i(n) = \frac{(n+i)\ln(n+i)}{\sum_{j=0}^{n-1}(n+j)\ln(n+j)}, \quad 0 \leq i \leq n-1. \tag{8.1.2}$$

Assuming the ERH, Eric Bach has proven the following theorem. In the theorem we use

$$\ell(n, \Delta) \quad = \quad \sum_{i=0}^{n-1} a_i(n) \ln B(n+i, \chi_\Delta), \quad n \in \mathbb{Z}_{\geq 1}. \tag{8.1.3}$$

Also, fix any triplet n_0, A, B from Table 1 and set

$$C(n) = \frac{A \ln|\Delta| + B}{\sqrt{n}\ln n}.$$

THEOREM 8.1.1. *(ERH) Let n_0, A, B be any triplet from Table 1. Then for $n \geq n_0$ we have*

$$|\ln L(1, \chi_\Delta) - \ell(n, \Delta)| \leq C(n).$$

PROOF. [**Bac95**] □

The following procedure O.L1chi(k) computes a relative k-approximation to $L(1, \chi_\Delta)$. It makes use of the function O.number_of_terms(F), which, given a floating point number F with $0 < F < 1$, returns an integer $n \geq 2$, such that $C(n) \leq F$. Furthermore, it uses the function O.Ell(n, k), which determines an absolute k-approximation to $\ell(n, \Delta)$. Those functions are presented below.

```
xbigfloat O.L1chi(int k)
{
  n = O.number_of_terms(2^(-k-2));
  l = O.Ell(n, k+3);
```

n_0	A	B
5	16.397	47.183
10	12.170	38.831
50	8.628	29.587
100	7.962	27.145
500	7.106	22.845
1000	6.897	21.528
5000	6.593	19.321
10000	6.510	18.606
50000	6.378	17.397
100000	6.338	17.031
500000	6.269	16.409
1000000	6.246	16.217

TABLE 1

```
  L = Truncate(exp(l, k+3), k+3);
  return L;
}
```

PROPOSITION 8.1.2. *(ERH) If $k \geq 1$, then $O.$`L1chi`(k) returns a relative k-approximation to $L(1, \chi_\Delta)$.*

PROOF. We use the notation from the procedure `L1chi`. It follows from Theorem 8.1.1 that

$$|\ln(L(1, \chi_\Delta)) - l| < 2^{-k-2} + 2^{-k-3}. \tag{8.1.4}$$

Set

$$x = \frac{\exp l}{L(1, \chi_\Delta)} - 1.$$

Using (8.1.4) it is easy to verify that $|x| < 1$. Hence, Lemma 2.4.3, $k \geq 1$, and (8.1.4) imply

$$|x| < \frac{|\ln(1+x)|}{1 - |\ln(1+x)|} < 2^{-k-1}.$$

Therefore, we can write

$$\exp l = L(1, \chi_\Delta)(1 + \varepsilon_1), \quad |\varepsilon_1| < 2^{-k-1}. \tag{8.1.5}$$

Next, we can write

$$L = \exp l(1 + \varepsilon_2)(1 + \varepsilon_3), \quad |\varepsilon_2| < 2^{-k-3}, |\varepsilon_3| < 2^{-k-2}.$$

Together with (8.1.5) we obtain

$$\begin{aligned} L &= L(1, \chi_\Delta)(1 + \varepsilon_1)(1 + \varepsilon_2)(1 + \varepsilon_3) \\ &= L(1, \chi_\Delta)(1 + \varepsilon_4), \quad |\varepsilon_4| < 2^{-k}. \end{aligned}$$

$\square$

The bottleneck of the approximation of $L(1, \chi_\Delta)$ is the rapid growth of the number of terms. In the procedure `L1chi`, if a user asks for a relative k-approximation L to $L(1, \chi_\Delta)$, i.e.,

$$L = L(1, \chi_\Delta)(1 + \epsilon),\ |\epsilon| < 2^{-k},$$

the numbers of terms, n, is determined such that

$$C(n) \leq 2^{-k-2}.$$

In the following, we will present a method that allows to choose n such that

$$C(n) \leq \rho 2^{-k}/(1 + 2^{-k}),$$

where ρ can be any number with $0 < \rho < 1$. This changes the upper bound on $C(n)$ by a factor of approximately $\rho/4$; the larger k, the better this factor is approached. It follows from the definition of $C(n)$, that the number of necessary terms changes by a factor of approximately $(\rho/4)^2$. By choosing ρ close to 1, the number of terms necessary for computing a relative k-approximation can be reduced by a factor of approximately 16. This reduction of the number of terms is paid by an increase in the accuracy for the approximation of $l(n, \Delta)$ and the approximation of $\exp(l)$. For a constant ρ, this accuracy is approximately doubled. We describe this more precise in the following.

The number of terms can be further decreased, if we change the specification of the error bound from 2^{-k} to floating point numbers B. So, in the following, we will present a function $O.$`L1chi`(B), that on input of a floating point number $0 < B \leq 1/2$, returns an approximation L to $L(1, \chi_\Delta)$ such that

$$L = L(1, \chi_\Delta)(1 + \epsilon),\ |\epsilon| < B.$$

To obtain an approximation to $L(1, \chi_\Delta)$ with a relative error less than B in absolute value with the function described above, the error bound would have to be $2^{-k} = 2^{b(B)}$. Using B instead, saves up to a factor of approximately 2 in the error bound, depending on the distance of B from $2^{b(B)}$. This results in a reduction of the number of terms by a further factor of up to approximately 4.

```
xbigfloat O.L1chi(xbigfloat B)
 {
    xbigfloat rho = 63 / 64;
    xbigfloat E = rho * B;

    // Determine F < E / (1+E) - 2^(-k) with same order
    // of magnitude as E / (1+E).
    //
    int k = -b(B)+8;
    xbigfloat h = divide(E,1+E, k + b(B));
    xbigfloat F = h - 2^(-k+1);

    // Guarantee C(n)+2^(-k) < ln 2.
    //
    if (F+2^(-k) > 0.5)
      F = 0.5 - 2^(-k);

    // Determine n with C(n) < F.
    //
```

```
    int n = O.number_of_terms(F);

    // Determine L.
    //
    xbigfloat l = O.Ell(n,k);
    xbigfloat L = exp(l, -b( (1-rho)/2 * B)+1);
    return L;
}
```

PROPOSITION 8.1.3. *(ERH) On input of a floating point number $0 < B \leq 1/2$, the function O.**L1chi**(B) returns an approximation L, such that $L = L(1, \chi_\Delta)(1+\epsilon)$ with $|\epsilon| < B$. The approximation $L = (m, e)$ satisfies $b(m) = O(|b(B)|)$.*

PROOF. We use the notation of the procedure and follow the proof of Proposition 8.1.2. It follows from Theorem 8.1.1 that

$$|\ln(L(1, \chi_\Delta)) - l| < C(n) + 2^{-k} < \ln 2. \tag{8.1.6}$$

Set

$$x = \frac{\exp l}{L(1, \chi_\Delta)} - 1.$$

Using (8.1.6), it is easy to verify that $|x| < 1$. Hence, Lemma 2.4.3 and (8.1.6) imply

$$|x| < \frac{|\ln(1+x)|}{1 - |\ln(1+x)|} < (C(n) + 2^{-k})/(1 - C(n) - 2^{-k}).$$

Therefore, we can write

$$\exp l = L(1, \chi_\Delta)(1 + \varepsilon_1), \quad |\varepsilon_1| < (C(n) + 2^{-k})/(1 - C(n) - 2^{-k}). \tag{8.1.7}$$

It is $C(n) \leq F = (E/1+E)(1+\varepsilon) - 2^{-k+1}$, $|\varepsilon| < 2^{-k-b(E/(1+E))}$. This implies that $F < E/(1+E) - 2^{-k}$ and we obtain $C(n) + 2^{-k} < E/(1+E)$. Because $E > -1$ and $C(n) + 2^{-k} < 1$, this implies

$$(C(n) + 2^{-k})/(1 - C(n) - 2^{-k}) < E = \rho B. \tag{8.1.8}$$

Next, we can write

$$L = \exp l(1 + \varepsilon_2), \quad |\varepsilon_2| < (1 - \rho)/2B.$$

Together with (8.1.7) and (8.1.8), we obtain

$$\begin{aligned} L &= L(1, \chi_\Delta)(1 + \varepsilon_1)(1 + \varepsilon_2) \\ &= L(1, \chi_\Delta)(1 + \varepsilon_3), \quad |\varepsilon_3| < B. \end{aligned}$$

The assertion on the number of bits of the mantissa of L follows from the fact, that L is a relative $-b((1-\rho)/2B) + 1 = O(|b(B)|)$ approximation. □

REMARK 8.1.4. (Size of the result) Suppose that $B = 2^{-k}$ for some integer $k \geq 1$. We compare the size of the result $L = (m, e)$ of the procedure O.`L1chi`(B) to that of the procedure O.`L1chi`(k). Because L is a relative $-b((1-\rho)/2B) + 1$ -approximation, it is $b(m) \leq k + |b(1-\rho)| + 5$. Thus, for $\rho = 63/64$, we obtain

$$b(m) \leq k + 12,$$

compared with $b(m) \leq k+3$ for a relative k-approximation. Furthermore, $B \leq 1/2$ implies that $|b(L) - b(L(1,\chi_\Delta))| \leq 1$. So, $e = b(L)$ satisfies

$$|e - b(L(1,\chi_\Delta))| \leq 1,$$

as in the case of $O.$`L1chi`(k). Hence, the only difference between the two versions of the $L(1,\chi_\Delta)$ computation is the increase of the mantissa by $b(1-\rho)$ bits.

REMARK 8.1.5. (Choice of ρ and k) We explain why the choice of the weight ρ and the accuracy k is appropriate. Without rounding errors, we would require that

$$C(n) \leq B/(1+B). \tag{8.1.9}$$

Instead we require for $\kappa \geq 1$ that

$$C(\kappa n) \leq \rho B/(1+\rho B) - 2^{-k}. \tag{8.1.10}$$

Because $C(\kappa n) < 1/\sqrt{\kappa}C(n)$, it follows from (8.1.9), that (8.1.10) is fulfilled, if

$$1/\sqrt{\kappa} \leq (1+B)(1/(1/\rho + B) - 2^{-k}/B). \tag{8.1.11}$$

It is $(1+B)(1/(1/\rho+B) - 2^{-k}/B) = 1 - (1-\rho)/(1+\rho B) - (1+B)/(2^k B) > \rho - (1+B)/(2^k B) \geq \rho - 3 \cdot 2^{-k-b(B)-1}$. So, (8.1.11) implies, that (8.1.10) is satisfied, if

$$1/\sqrt{\kappa} \leq \rho - 3 \cdot 2^{-k-b(B)-1}. \tag{8.1.12}$$

We find it acceptable, if we have to compute at most 1/10 terms more, because of rounding errors. Thus, we accept

$$\kappa = 11/10.$$

Hence, we choose $\rho = 63/64$ and $k = -b(B) + 8$ to satisfy (8.1.12).

8.2. The procedure `Ell`

Next, we explain the approximate computation of $\ell(n,\Delta)$. To simplify this computation set

$$\omega_p(n) = \begin{cases} 1, & \text{if } p < n \\ \sum_{j=p-n+1}^{n-1} a_j(n), & \text{if } n \leq p < 2n-1. \end{cases} \tag{8.2.1}$$

As stated in [**JLW95**] we have

$$\ell(n,\Delta) = \sum_{p<2n-1} \omega_p(n) \cdot \ln\left(\frac{p}{p - \chi_\Delta(p)}\right).$$

We use a function `Init_Primes`(P,m), $m \in \mathbb{Z}_{\geq 0}$, which initializes the array P with all prime numbers less than m in increasing order and returns the number of primes less than m. For technical reasons it sets $p[0] = 0$. Furthermore, we use a function `chi`(Δ, p) that computes $\chi_\Delta(p)$.

For $n \in \mathbb{Z}_{\geq 2}$ let

$$bl(n) = \begin{cases} 0, & \text{if } n = 2, \\ -b(n-2)+1, & \text{if } n \geq 3. \end{cases} \tag{8.2.2}$$

Note that for $n \geq m \geq 2$ we have $bl(n) \leq bl(m) \leq 0$.

```
xbigfloat O.Ell(int n, int k)
{
  // Initialize prime table
  int N = Init_Primes(primes, 2n-1 );

  // Compute denominator of w_p(n)
  xbigfloat w_den = 0;
  int b = b(log(n,1)-0.5), j;
  for(j = 0; j < n; j++)
      w_den += (n+j)*log(n+j,max{0,max{0,k+b(N)+bl(n)+3}+5-b});

  // ell(n,Delta) : part for n <= p < 2*n-1
  xbigfloat l = 0, w_num = 0, w, x; int q = n, i = N, c;
  for(p = primes[N]; p >= n; p = primes[i])
    {
      for(j = q-1; j >= p-n+1; j--)
        w_num += (n+j)*log(n+j, max{0,max{0,k+b(N)+bl(n)+3}+5-b});
      q = p-n+1;
      i--;
      w = divide(w_num,w_den, max{0,k+b(N)+bl(p)+3}+4);
      w = truncate(w, max{0,k+b(N)+bl(p)+3}+3);

      c = chi(Delta,p);
      if (c != 0)
       {
         x = divide(p,p-c, k+b(N)+5);
         l += w*log(x, k+b(N)+4);
       }
    }

  // ell(n,Delta) : part for 2 <=  p < n
  for(p = primes[i]; p >= 2; p = primes[i])
    {
      i--;
      c = chi(Delta,p);
      if (c != 0)
       {
         x = divide(p,p - chi(Delta,p), k+b(N)+3);
         l += log(x, k+b(N)+2);
       }
    }
  l = truncate(l, l.exponent()+k+1);
  return l;
}
```

We will now prove that `Ell` is correct. For this purpose we need the following result.

LEMMA 8.2.1. *Let p be a prime. Then*

$$\frac{1}{p+2} \le \left|\ln\left(\frac{p}{p-\chi_\Delta(p)}\right)\right| \le \left\{ \begin{array}{ll} \ln 2, & \text{if } p=2 \\ \frac{1}{p-2}, & \text{if } p \ge 3 \end{array} \right\} \le 2^{bl(p)}.$$

PROOF. We have

$$|\ln(p/(p-\chi_\Delta(p)))| = \begin{cases} \ln(1+1/p), & \text{if } \chi_\Delta(p) = -1 \\ \ln(1+1/(p-1)), & \text{if } \chi_\Delta(p) = 1. \end{cases}$$

Hence,

$$|\ln(1+1/p)| \le |\ln(p/(p-\chi_\Delta(p)))| \le \ln(1+1/(p-1)).$$

For $p=2$ the assertion is easy to verify. For $p \ge 3$, we apply Corollary 2.4.3 with $x = 1/p$ and $x = 1/(p-1)$. □

LEMMA 8.2.2. *Let $k \ge 0$. The procedure $O.\mathtt{Ell}(n,k)$ returns an absolute k-approximation to $\ell(n,\Delta)$.*

PROOF. First, we analyze how accurately w approximates the weight $\omega_p(n)$ for each prime p. Use b and N from procedure `Ell` and let $l_{n+j} = \mathtt{log}(n+j, max\{0, max\{k+b(N)+bl(n)+3\}+5-b\})$.

Fix a prime p, $n \le p < 2n-1$ and let $t_p = max\{0, k+b(N)+bl(p)+3\}$. Then the relative errors in the computation of w_{num} and w_{den} can be estimated as follows.

$$\frac{\left|\sum_{j=p-n+1}^{n-1}(n+j)(l_{n+j}-\ln(n+j))\right|}{\left|\sum_{j=p-n+1}^{n-1}(n+j)\ln(n+j)\right|} \quad < \quad \frac{2^{-t_p-5+b}\, 2n\,(2n-p-1)}{\ln(n)\, n\,(2n-p-1)} \le 2^{-t_p-3},$$

$$\frac{\left|\sum_{j=0}^{n-1}(n+j)\,(l_{n+j}-\ln(n+j))\right|}{\left|\sum_{j=0}^{n-1}(n+j)\,\ln(n+j)\right|} \quad < \quad \frac{2^{-t_p-5+b}\, 2n^2}{\ln(n)\, n^2} \le 2^{-t_p-3}.$$

Let $w = \mathtt{divide}(w_{num}, w_{den}, max\{0, k+b(N)+bl(p)+3\}+4)$ and $w_p = \mathtt{truncate}(w, \max\{0, k+b(N)+bl(p)+3\}+3)$. Then

$$w = \frac{w_{num}}{w_{den}}(1+\varepsilon_0) = w_p(n)\frac{(1+\varepsilon_0)(1+\varepsilon_1)}{1+\varepsilon_2}, \quad |\varepsilon_0| < 2^{-t_p-4}, |\varepsilon_1|, |\varepsilon_2| < 2^{-t_p-3}.$$

This implies that

$$w_p = \omega_p(n)(1+\varepsilon_3), \quad |\varepsilon_3| < 2^{-t_p}.$$

Now, we determine how accurately l approximates $\ell(n,\Delta)$. Let $\xi_p = p/(p-\chi_\Delta(p))$.

Suppose that $\xi_p \ne 1$. For $n \le p < 2n-1$ let $x_p = \mathtt{divide}(p, p-\chi_\Delta(p), k+b(N)+5)$, $l_p = \mathtt{log}(x_p, k+b(N)+4)$. Then

$$l_p = \ln(\xi_p)\,(1+\varepsilon_4), \quad |\varepsilon_4| < 2^{-k-b(N)-3}/|\ln(\xi_p)|.$$

Lemma 8.2.1 yields $2^{-bl(p)} \le 1/|\ln(\xi_p)|$. Hence,

$$\begin{aligned} w_p l_p &= \omega_p(n)\ln(\xi_p)(1+\varepsilon_3)(1+\varepsilon_4) \\ &= \omega_p(n)\ln(\xi_p)(1+\varepsilon_5), \quad |\varepsilon_5| < 2^{-k-1-b(N)}/|\ln(\xi_p)|. \end{aligned}$$

Because $|\omega_p(n)| \le 1$, we obtain

$$|\omega_p(n)\,\ln(\xi_p) - w_p l_p| < 2^{-k-1-b(N)}. \tag{8.2.3}$$

For $2 \le p < n$ let $x_p = \texttt{divide}(p, p-\chi_\Delta(p), k+b(N)+3)$, $l_p = \texttt{log}(x_p, k+b(N)+2)$. Then

$$|\ln(\xi_p) - l_p| < 2^{-k-1-b(N)}. \tag{8.2.4}$$

Hence, with the value of l before `truncate` is applied, we have by (8.2.3) and (8.2.4)

$$\begin{aligned}
|\ell(n,\Delta) - l| &\le \sum_{2\le p<n} |\ln(\xi_p) - l_p| + \sum_{n\le p<2n-1} |\omega_p(n)\ln(\xi_p) - w_p l_p| \\
&< \sum_{2\le p<2n-1} 2^{-k-1-b(N)} \\
&< 2^{-k-1}.
\end{aligned}$$

and the truncated l is an absolute k-approximation to $\ell(n,\Delta)$ by Lemma 2.3.1. ☐

LEMMA 8.2.3. *Let $k \ge 0$. The bit complexity of the procedure $O.$`Ell`(n,k) is $O(n\,[M(k+\operatorname{size} n)\,\log(k+\operatorname{size} n) + M(\operatorname{size} n + \operatorname{size}\Delta)\,\log(\operatorname{size} n + \operatorname{size}\Delta)])$ and $|e(l)|, b(m(l)) = O(k+\operatorname{size} n)$.*

PROOF. We use the notation from procedure `Ell`. For a floating point number f we denote by $m(f)$ the mantissa of f and by $e(f)$ its exponent.

According to [**BS96**][p 298] all primes less than $2n-1$ can be determined in time $O(n\operatorname{size}(n))$.

First, we examine the computation of all w . It is easy to verify that $b(N) + bl(n) \le 3$. By Theorem 2.4.1 all logarithms $l_{n+j} = \texttt{log}(n+j, max 0, max 0, k+b(N)+bl(n)+3+5-b)$ can be computed in time $O(n[M(k+\operatorname{size}(\operatorname{size}(n)))\log(k+\operatorname{size}(\operatorname{size}(n)))+\operatorname{size}(n)])$ and by Lemma 2.2.4 we obtain for their sizes $b(m(l_{n+j})) = O(k+\operatorname{size} n)$, $|e(l_{n+j})| = O(\operatorname{size} n)$. This implies that computing all products $(n+j)l_{n+j}$ takes time $O(nM(k+\operatorname{size}(n)))$, which, by Theorem 2.3.3, dominates the time for computing their sum w_{den}. Also by Theorem 2.3.3 we have $\operatorname{size}(w_{den}) = O(k+\operatorname{size} n)$. Clearly, the whole sum w_{num} can be computed in the same time and also $\operatorname{size}(w_{num}) = O(k+\operatorname{size} n)$. Therefore, all w can be computed from w_{den} and w_{num} in time $O(n[M(k)+\operatorname{size}(n)])$. Hence, the overall running time for computing all w is $O(n[M(k+\operatorname{size}(\operatorname{size}(n)))\log(k+\operatorname{size}(\operatorname{size}(n)))+M(k+\operatorname{size}(n))])$.

We estimate the size of each w. Because $1/(4n) \le |\omega_p(n)| \le 1$, we obtain $|b(\omega_p(n))| < b(n)+2$. By (2.2.1) we have for each w that

$$b(m(w)) = O(k), \quad |e(w)| = O(\operatorname{size} n). \tag{8.2.5}$$

Next, we examine the time for computing all x and the approximations to their logarithms.

All relative approximations x can be computed in time $O(n\,[M(k+\operatorname{size} n) + M(\operatorname{size} n + \operatorname{size}\Delta)\,\log(\operatorname{size} n + \operatorname{size}\Delta)])$, where the second term in the sum is due to the computations of the Kronecker symbols $\chi_\Delta(p)$ (see [**SGV94**]).

For each x, let l_x be the absolute approximation to the logarithm of x. Because $1/2 \le p/(p-\chi_\Delta(p)) \le 2$, we obtain $|b(p/(p-\chi_\Delta(p)))| \le 2$. (2.2.1) yields $b(m(x)) = O(k+\operatorname{size} n)$ and $|e(x)| = O(1)$. Hence, by Theorem 2.4.1, we know that computing all l_x from x takes time $O(nM(k+\operatorname{size} n)\log(k+\operatorname{size} n))$.

Lemma 2.4.5 yields that for each x we have $b(m(l_x)), |e(l_x)| = O(k + \operatorname{size} n)$. Together with (8.2.5) this implies that all products $w \cdot l_x$ can be computed in time $O(n\,M(k + \operatorname{size} n))$ and Theorem 2.3.3 yields that summing up all terms to l takes time $O(n(k+\operatorname{size} n))$. Theorem 2.3.3 also implies that $|e(l)|, b(m(l)) = O(k+\operatorname{size} n)$.

Hence, the overall running time is given by computing all x and their logarithms l_x which proves the asserted bit complexity. □

8.3. The number of terms

Next, we describe the procedure O.`number_of_terms`(F). On input of a floating point number $0 < F < 1$, it returns $n \geq 2$, such that $C(n) \leq F$. For $n \geq 2$ it uses the functions `A`(n) which returns a and `B`(n) which returns b from the triplet (n, a, b) according to Table 1. For $n = 0$ the functions return the corresponding maximal value of the table.

```
int O.number_of_terms (xbigfloat F)
{
     // Initial n with C(n) < F.
     //
     xbigfloat d = log(absolute_value(O.discriminant), 6);
     xbigfloat C = (A(0) * d + B(0)) + 0.5;
     C = divide(C,F,2);
     C *= 1.5;
     int n = ceil(C)^2;
     int l = 2;

     int m;
     xbigfloat S,L;

     // Find better smaller n with C(n) < F.
     //
     do
     {
        m = l + floor((n-l)/2);

        S = sqrt(m, 3+ b(F)+ ceil(b(m)/2 + b(b(m))));
        L = log (m, 4+ b(F)+ ceil(b(m)/2));
        C = A(m) * d + B(m);

        if (F*S*L - 0.5 >= C + 0.5)
           n = m;
        else
           l = m+1;
     }
     while (l < n);
}
```

Proposition 8.3.1. *Let $0 < F < 1$. Then the prodedure O.`number_of_terms`(F) determines an integer $n \geq 2$, such that $C(n) \leq F$ in time $O((|b(F)| + \log\log|\Delta|)[M(\operatorname{size} F + |b(F)| + \log\log|\Delta|) + M(|b(F)| + \log\log|\Delta|)\log(|b(F)| + \log\log|\Delta|)] + \operatorname{size}\Delta)$ and $n = O(4^{|b(F)|}\log^2|\Delta|)$.*

PROOF. For every $m \geq 0$, we have

$$|A(m)d + B(m) - (A(m)\ln|\Delta| + B(m))| < 1/2.$$

For $m \geq 2$ set $k = b(F) + b(\sqrt{m} + b(\ln m)$. In each iteration of the loop, we have $S = \sqrt{m}(1+\varepsilon_1)$, $|\varepsilon_1| < 2^{-3-k)}$ and $L = \ln m(1+\varepsilon_2)$, $|\varepsilon_2| < 2^{-3-k)}$, for $m \geq 2$. Hence, $SLF = F\sqrt{m}\ln m(1+\varepsilon_3)$, $|\varepsilon_3| < 2^{-1-k}$ and thus

$$|SLF - F\sqrt{m}\ln m| < 1/2.$$

Hence, before the loop starts and at the end of each loop iteration, we have

$$F\sqrt{n}\ln n \geq A(n)\ln|\Delta| + B(n),$$

which implies $C(n) \leq F$ at the end of the procedure.

Let n_0 be the value of n, before the loop starts. To prove the termination, we note that the distance $|l - n|$ strictly decreases, and the loop is terminated after $O(\log n_0)$ iterations.

d can be found in time $O(M(b(b(\Delta)))\log(b(b(\Delta))) + \operatorname{size}(\Delta))$. The costs for computing the first value of C are dominated by the costs for the computations inside the loop. Computing S, L, and C inside the loop takes time $O(M(|b(F)| + \log\log|\Delta|)\log(|b(F)| + \log\log|\Delta|))$ and the costs for computing the product SLF are $O(M(\operatorname{size} F + |b(F)| + \log\log|\Delta|))$. Because $n_0 = O(4^{|b(F)|}\log^2|\Delta|)$, we obtain the asserted overall running time. Because the return value n is less or equal to n_0, we also obtain the asserted bound on the size of n. $\square$

8.4. Bit complexity

We analyze the running time of the procedures `L1chi`.

THEOREM 8.4.1. *(ERH) Let $k \in \mathbb{Z}_{\geq 1}$. O.`L1chi`(k) computes a relative k-approximation L to $L(1, \chi_\Delta)$ in time $O(4^k M(k + \log|\Delta|)\log(k + \log|\Delta|)\log^2|\Delta|)$.*

PROOF. We estimate the bit complexity of algorithm O.`L1chi`(k). Use l and L from procedure.

Lemma 8.2.3 implies that l can be computed in time $O(n\,[M(k+\operatorname{size} n)\log(k+\operatorname{size} n) + M(\operatorname{size} n + \operatorname{size}\Delta)\log(\operatorname{size} n + \operatorname{size}\Delta)])$ and $|e(l)|, \operatorname{size}(l) = O(k + \operatorname{size} n)$. Theorem 2.5.4 yields that L can be computed from l in time $O(M(k+\operatorname{size} n)\log(k+\operatorname{size} n))$.

By Proposition 8.3.1 we have $n = O(4^k\log^2|\Delta|)$, which implies $\operatorname{size}(n) = O(k + \log\log|\Delta|)$. Hence, the overall running time is $O(4^k M(k + \log|\Delta|)\log(k + \log|\Delta|)\log^2|\Delta|)$. $\square$

THEOREM 8.4.2. *(ERH) Let $0 < B \leq 1/2$. The procedure O.`L1chi`(B) computes an approximation $L = (m, e)$ to $L(1, \chi_\Delta)$, such that $L = L(1, \chi_\Delta)(1+\varepsilon)$, $|\varepsilon| < B$, and $b(m) = O(|b(B)|)$ in time $O(\operatorname{size} B + 4^{|b(B)|}M(|b(B)| + \log|\Delta|)\log(|b(B)| + \log|\Delta|)\log^2|\Delta|)$.*

PROOF. We use the notation of the procedure. For $\rho = 63/64$ and $E = \rho B$, we have $B/2 < E/(1+E) < B$. This implies that $b(B) - 1 \leq b(E/(1+E)) \leq b(B)$. It follows together with Lemma 2.2.2, that

$$b(B) - 2 \leq b(h) \leq b(B) + 1. \tag{8.4.1}$$

Furthermore, we have

$$k = -b(B) + 8. \tag{8.4.2}$$

The first value of F is $F = h - 2^{-k+1}$. Obviously $F < 2^{b(h)}$. A lower bound follows from $F \geq 2^{b(h)-1} - 2^{b(B)-7} \geq 2^{b(B)-3} - 2^{b(B)-7} > 2^{b(B)-4}$. Hence,

$$b(B) - 3 \leq b(F) \leq b(B) + 1. \tag{8.4.3}$$

If $F + 2^{-k} \geq 1/2$, then F is replaced by $F = 1/2 - 2^{-k}$. We show that (8.4.3) is also fulfilled in that case. It follows from (8.4.2) and $B \leq 1/2$, that $-k \leq b(B) - 8 \leq -8$ and so

$$1/2 - 2^{-k} = 2^{-2}(2 - 2^{-k+2}) \geq 1/4. \tag{8.4.4}$$

Suppose that $F + 2^{-k} \geq 1/2$. Then (8.4.4) implies that $b(F) \geq -1$ and thus, by (8.4.3), $-2 \leq b(B) \leq 0$. It also follows from (8.4.4), that the new value $F = 1/2 - 2^{-k}$ satisfies $1/4 \leq F < 1/2$ and so $b(F) = -1$. This shows that $b(B) - 1 \leq b(F) \leq b(B) + 1$, so again (8.4.3) is valid.

It follows from (8.4.2), that F can be computed in time $O(\operatorname{size} B + |b(B)|)$. Proposition 8.3.1 and (8.4.3) imply, that the number of terms n can be determined in time $O((|b(B)| + \log\log|\Delta|) M(|b(B)| + \log\log|\Delta|) \log(|b(B)| + \log\log|\Delta|) + \operatorname{size}\Delta)$ and $n = O(4^{|b(B)|}\log^2|\Delta|)$. Note that $b(m(F)) = O(|b(B)|)$.

It follows from Lemma 8.2.3, that l can be computed in time $O(n\,[M(k + \operatorname{size} n)\,\log(k + \operatorname{size} n) + M(\operatorname{size} n + \operatorname{size}\Delta)\,\log(\operatorname{size} n + \operatorname{size}\Delta)])$ and $|e(l)|, b(m(l)) = O(k + \operatorname{size} n)$.

Because the time for computing L from l is dominated by the running times above, and because $\operatorname{size} n = O(|b(B)| + \log\log|\Delta|)$, the overall running time is $O(\operatorname{size} B + 4^{|b(B)|} M(|b(B)| + \log|\Delta|) \log(|b(B)| + \log|\Delta|) \log^2|\Delta|)$.

$\square$

8.5. Using the built-in type `double`

In Section 8.2 we described the approximation of $l(n, \Delta)$ using the `xbigfloat` model. To accelerate the computations in practice, it is necessary to use machine types, i.e. `double`, whenever possible (see Appendix A.3 for timings). We assume that the data type `double` is implemented according to the IEEE-754 floating point model (Section 2.7), which is available for most of the machine platforms today. Because that model differs from our `xbigfloat` model, we have to examine the procedure $O.$`Ell`(n, k) again.

The general idea is as follows. From Theorem 2.7.2, we know that each operation in the IEEE-754 model yields a relative error which is less than 53 in absolute value. We estimate all relative accuracies, that are required during the approximation of $l(n, \Delta)$. If the accuracy is less than 53, we may use `double` to do the operation. If a first part of the computation is done with `double`s, but then the required accuracy becomes too large, we convert the `double` value into an `xbigfloat`. Note that these conversions are carried out without roundoff error. The same applies here for the conversion of integers to `double`.

```
xbigfloat O.Ell(int n, int k)
{
  int P = 53;

  // Initialize prime table
```

```
int N = Init_Primes(primes, 2n-1);

// Decide which parts can be approximated as double.
int use_double_for_w;
int use_double_for_l;
int b = b(log(n,1)-0.5);

if (max{k+b(N)+bl(n)+3,0}+b(n)+8 <= P)
   use_double_for_w = 2;
else if (max{k+b(N)+bl(n)+3,0} +b(floor(7b(n)/10)) +7-b <= P)
   use_double_for_w = 1;
else
   use_double_for_w = 0;

if (use_double_for_w == 2 &&
    5+b(primes[N]+2) <= P &&
    k+b(N)+11 <= P &&
    k+2b(N)+4 <= P)
   use_double_for_l = 1;
else
   use_double_for_l = 0;

// Compute denominator w_den of w_p(n)
xbigfloat w_den_x = 0;
double w_den_d = 0;
int j;

if (use_double_for_w == 2)
   for(j = 0; j < n; j++)
      w_den_d += (double)(n+j) * log((double)(n+j));

else if (use_double_for_w == 1)
      for(j = 0; j < n; j++)
         w_den_x += (xbigfloat)((double)(n+j)*log((double)(n+j)));
else
      for(j = 0; j < n; j++)
         w_den_x += (n+j)*log(n+j,max{0,max{0,k+b(N)+bl(n)+3}+5-b});

// ell(n,Delta) : part for n <= p < 2*n-1
int q = n, i = N, c;
xbigfloat l_x = 0, w_num_x = 0, w_x, x_x;
double l_d = 0, w_num_d = 0, w_d, x_d;
int kbN11;

if (k+b(N)+11 <= P)
   kbN11 = 1;
else
   kbN11 = 0;
```

```
for(p = primes[N]; p >= n; p = primes[i])
  {
    if (use_double_for_w == 2)
       for(j = q-1; j >= p-n+1; j--)
           w_num_d += (double)(n+j) * log((double)(n+j));

    else if (use_double_for_w == 1)
       for(j = q-1; j >= p-n+1; j--)
           w_num_x += (xbigfloat)((double)(n+j)*log((double)(n+j)));

    else
       for(j = q-1; j >= p-n+1; j--)
          w_num_x += (n+j)*log(n+j,max{0,max{0,k+b(N)+bl(n)+3}+5-b});

    q = p-n+1;
    i--;

    if (use_double_for_w == 2)
       w_d = w_num_d / w_den_d;
    else
     {
       w_x = divide(w_num_x,w_den_x, max{0,k+b(N)+bl(p)+3}+4);
       w_x = truncate(w_x, max{0,k+b(N)+bl(p)+3}+3);
     }

    if (k+b(N)+1+bl(p) >= 0)
     {
       c = chi(Delta,p);
       if (c != 0)
        {
          if (b(p+2)+5 <= P && kbN11)
           {
             x_d = (double)p / (double)(p-c);
             h_d = log(x_d);

             if (use_double_for_l)
                l_d += (w_d * h_d);
             else if (use_double_for_w == 2)
                l_x += (xbigfloat)(w_d*h_d);
             else
                l_x += w_x *(xbigfloat)(h_d);
           }
          else
           {
             x_x = divide(p,p-c, k+b(N)+5);
             h_x = log(x_x, k+b(N)+4);

             if (use_double_for_w == 2)
                 l_x += (xbigfloat)(w_d)*h_x;
```

```
              else
                  l_x += w_x * h_x;
            }
         } // end if c
      }
   } // end for p

 // ell(n,Delta) : part for 2 <=  p < n
 for(p = primes[i]; p >= 2; p = primes[i])
   {
     i--;

     if (k+b(N)+1+bl(p) >= 0)
      {
        c = chi(Delta,p);
        if (c != 0)
         {
           if (b(p+2)+5 <= P && kbN11)
            {
              x_d = (double)p / (double)(p-c);
              h_d = log(x_d);

              if (use_double_for_l)
                 l_d += h_d;
              else
                 l_x += (xbigfloat)(h_d);
            }
           else
            {
               x_x = divide(p,p-c, k+b(N)+3);
               h_x = log(x_x, k+b(N)+2);
               l_x += h_x;
            }
         } // end if c
      }
   } // end for p

 if (use_double_for_l)
  l_x = l_d;

 l_x = truncate(l_x, l_x.exponent()+k+1);
 return l_x;
}
```

We give some auxiliary results, before we prove the correctness of the algorithm.

LEMMA 8.5.1. *Let* $a_i \in \mathbb{R}_{>0}$, $0 \le i < n$, $n \ge 2$, *and* $k \in \mathbb{Z}_{\ge 0}$. *If*

$$\begin{aligned} z_i &= a_i\,(1+g_i),\ |g_i| < 2^{-k-1}, 0 \le i < n, \\ u_0 &= z_0, \\ u_i &= (u_{i-1}+z_i)\,(1+h_i),\ |h_i| < 2^{-k-b(n-1)-3}, 1 \le i < n, \end{aligned}$$

then $u_{n-1} = (\sum_{i=0}^{n-1} a_i)(1+\varepsilon)$ *with* $|\varepsilon| < 2^{-k}$.

PROOF. We have

$$\begin{aligned} u_{n-1} &= \textstyle\sum_{j=0}^{n-1} a_j(1+g_j)\prod_{i=j}^{n-1}(1+h_i) \\ &= \textstyle\sum_{j=0}^{n-1} a_j(1+\varepsilon_j), \end{aligned} \tag{8.5.1}$$

where we set $h_0 = 0$. It is

$$(1-2^{-k-b(n-1)-3})^{n-1}(1-2^{-k-1}) < 1+\varepsilon_j < (1+2^{-k-b(n-1)-3})^{n-1}(1+2^{-k-1}). \tag{8.5.2}$$

Because $k + b(n-1) + 3 \ge 1$, it follows from Lemma 2.5.1, that $(1 - 2^{-k-b(n-1)-3})^{n-1} \ge (1-(n-1)2^{-k-b(n-1)-3})$. And because $k+3 \ge 1$, it follows from Lemma 2.5.2, that $(1+2^{-k-b(n-1)-3})^{n-1} < 1+2(n-1)2^{-k-b(n-1)-3}$. Hence, it follows from (8.5.2) and $k \ge 0$, that

$$|\varepsilon_j| < 2^{-k},\ 0 \le j < n. \tag{8.5.3}$$

Set $a = \sum_{i=0}^{n-1} a_i$. Now, (8.5.1) and (8.5.3) imply $|u_{n-1} - a|/|a| \le (\sum_{j=0}^{n-1} a_j|\varepsilon_j|)/|a| < 2^{-k}$, where we use the fact, that $a_j > 0$ for $0 \le j < n$. □

LEMMA 8.5.2. *Let* $a_i \in \mathbb{R}$ *with* $|a_i| \le 2^{k_i}$, $k_i \in \mathbb{Z}$, $0 \le i < n$, $n \ge 2$, *such that* $k_0 \le k_1 \le \ldots \le k_{n-1}$, *and let* $k \in \mathbb{Z}$.

If $k + b(n) + k_{n-1} < 0$, *set* $u_{n-1} = 0$. *Otherwise, let* $m \in \mathbb{Z}$ *be minimal with* $0 \le m < n$, *such that* $k + b(n) + k_m \ge 0$. *Let*

$$z_i = a_i\,(1+g_i),\ |g_i| < 2^{-k-b(n)-2-k_i},\ m \le i < n,$$

and set

$$\begin{aligned} u_m &= z_m, \\ u_i &= (u_{i-1}+z_i)\,(1+h_i),\ |h_i| < 2^{-k-2b(n)-3-k_i}, \end{aligned}$$

for $m+1 \le i < n$. *Then* $|u_{n-1} - \sum_{i=0}^{n-1} a_i| < 2^{-k}$.

PROOF. We have

$$\begin{aligned} u_{n-1} &= \textstyle\sum_{i=m}^{n-1} a_i(1+g_i)\prod_{j=i}^{n-1}(1+h_j) \\ &= \textstyle\sum_{i=m}^{n-1} a_i(1+\varepsilon_i), \end{aligned} \tag{8.5.4}$$

where we set $h_m = 0$. Set $s_i = k + 2b(n) + 3 + k_i$ and $t_i = k + b(n) + 2 + k_i$ for $m \le i < n$. Because $s_m \le s_{m+1} \le \ldots \le s_{n-1}$, we have

$$(1-2^{-s_i})^{n-i}(1-2^{-t_i}) < 1+\varepsilon_i < (1+2^{-s_i})^{n-i}(1+2^{-t_i}). \tag{8.5.5}$$

Because $s_i \ge 1$ for $i \ge m$, it follows from Lemma 2.5.1, that $(1-2^{-s_i})^{n-i} \ge 1-(n-i)2^{-s_i}$. Furthermore, $s_i \ge b(n)+1$ implies $(n-i)2^{-s_i} < 1/2$, so it is $(1+2^{-s_i})^{n-i} < 1+2(n-i)2^{-s_i}$ by Lemma 2.5.2. Hence, we obtain

$$(1-2^{-k-b(n)-2-k_i})^2 < 1+\varepsilon_i < (1+2^{-k-b(n)-2-k_i})^2$$

for $m \le i < n$. Because $k + b(n) + k_i + 2 \ge 0$, this yields

$$|\varepsilon_i| < 2^{-k-b(n)-k_i},\ m \le i < n. \tag{8.5.6}$$

It follows from (8.5.4) and (8.5.6), that $|u_{n-1} - \sum_{i=0}^{n-1} a_i| \le \sum_{i=0}^{m-1} |a_i| + \sum_{i=m}^{n-1} |a_i||\varepsilon_i| < \sum_{i=0}^{n-1} 2^{-k-b(n)} < 2^{-k}$. □

Lemma 8.5.3. *Let $a \in \mathbb{R}_{>0}$ with $|\ln a| \ge 1/m$, $m \in \mathbb{R}_{>0}$ and $k \in \mathbb{Z}_{\ge -2}$. If $x = a(1+\varepsilon_1)$ with $|\varepsilon_1| < 2^{-max\{k+b(m)+3\},1}$, and $l = \ln x(1+\varepsilon_2)$ with $|\varepsilon_2| < 2^{-k-2}$, then $l = \ln a(1+\varepsilon)$ with $|\varepsilon| < 2^{-k}$.*

Proof. We have

$$\begin{aligned} l &= \ln(a(1+\varepsilon_1))(1+\varepsilon_2) \\ &= (\ln a + \ln(1+\varepsilon_1))(1+\varepsilon_2) \\ &= \ln a(1 + \frac{\ln(1+\varepsilon_1)}{\ln a})(1+\varepsilon_2) \\ &= \ln a(1+\varepsilon), \end{aligned}$$

with

$$\begin{aligned} |e| &\le \frac{|\ln(1+\varepsilon_1)|}{|\ln a|} + |\varepsilon_2| + |\varepsilon_2|\frac{|\ln(1+\varepsilon_1)|}{|\ln a|} \\ &\le m|\ln(1+\varepsilon_1)| + |\varepsilon_2| + |\varepsilon_2|m|\ln(1+\varepsilon_1)| \\ &< m2|\varepsilon_1| + |\varepsilon_2| + |\varepsilon_2|m2|\varepsilon_1| \\ &< 2^{-k-2} + 2^{-k-2} + 2^{-2k-4} \\ &< 2^{-k}, \end{aligned}$$

because $k \ge -2$. Here, we use, that Corollary 2.4.3 implies $|\ln(1+\varepsilon_1)| < 2|\varepsilon_1|$, because $|\varepsilon_1| < 1/2$. □

Theorem 8.5.4. *Let $k \ge 0$. The procedure $O.\mathtt{Ell}(n,k)$ returns an absolute k-approximation to $\ell(n,\Delta)$.*

Proof. We rename the variables of the procedure. Let $w_{den,d}$, $w_{num,d}$, l_d, and $w_{den,x}$, $w_{num,x}$, l_x denote the value of *w_den_d*, *w_num_d*, *l_d*, and *w_den_x*, *w_num_x*, *l_x*, respectively. For a prime p, let $x_{p,d}$, $l_{p,d}$, $w_{p,d}$, and $x_{p,x}$, $l_{p,x}$, $w_{p,x}$ denote the value of *x_d*, *h_d*, *w_d*, and *x_x*, *h_x*, *w_x* in the iteration for p, respectively.

Fix a prime p, $n \le p < 2n-1$.

First, we analyze how accurately $w_{p,d}$ and $w_{p,x}$ approximate the weight $\omega_p(n)$. Use b, N, and P from the procedure `Ell`.

Assume that *use_double_for_w* $= 2$, i.e.

$$max\{k + b(N) + bl(n) + 3, 0\} + b(n) + 8 \le P. \tag{8.5.7}$$

Set $t_{p,d} = max\{k + b(N) + bl(p) + 3, 0\} + 2$. It follows from (8.5.7) and Theorem 2.7.2, that the relative error in the approximation of $(n+j)\ln(n+j)$ is less than $2^{-t_{p,d}-4}$ and that for each addition, the relative error is less than $2^{-t_{p,d}-6-b(n)}$. Hence, Lemma 8.5.1 implies

$$w_{den,d} = \left(\sum_{j=0}^{n-1}(n+j)\ln(n+j)\right)(1+\varepsilon_1),\ |\varepsilon_1| < 2^{-t_{p,d}-3}.$$

The same considerations yield

$$w_{num,d} = \left(\sum_{j=p-n+1}^{n-1} (n+j)\ln(n+j) \right) (1+\varepsilon_2),\ |\varepsilon_2| < 2^{-t_{p,d}-3}.$$

It follows from (8.5.7) and Theorem 2.7.2, that

$$\begin{aligned} w_{p,d} &= w_{num,d}/w_{den,d}(1+\varepsilon_3),\ |\varepsilon_3| < 2^{-t_{p,d}-4}, \\ &= \omega_p(n)(1+\varepsilon_2)(1+\varepsilon_3)/(1+\varepsilon_1). \end{aligned}$$

By applying Lemma 2.5.3 and using that $t_{p,d} \geq 0$, we obtain

$$w_{p,d} = \omega_p(n)(1+\varepsilon_4),\ |\varepsilon_4| < 2^{-t_{p,d}}. \tag{8.5.8}$$

Now, assume that $use\_double\_for\_w = 1$. Because $b(\ln(2n)) \leq b(\lfloor 7b(n)/10 \rfloor + 2)$, this implies

$$max\{k + b(N) + bl(n) + 3, 0\} + b(\ln(2n)) + 7 - b \leq P. \tag{8.5.9}$$

Set $t_{p,x} = max\{0, k + b(N) + bl(p) + 3\}$. It follows from (8.5.9) and Theorem 2.7.2, that the relative error in the approximation of $(n+j)\ln(n+j)$ is less than $2^{-t_{p,x}-b(\ln(2n))-5+b}$. Hence, the absolute error is less than $2^{-t_{p,x}-5+b}2n$, and we obtain

$$\frac{\left| w_{den,x} - \sum_{j=0}^{n-1} (n+j)\ln(n+j) \right|}{\left| \sum_{j=0}^{n-1} (n+j)\ \ln(n+j) \right|} < \frac{2^{-t_{p,x}-5+b}\,2n^2}{\ln(n)\,n^2} \leq 2^{-t_{p,x}-3}.$$

The same considerations hold for the numerator, so

$$\frac{\left| w_{num,x} - \sum_{j=p-n+1}^{n-1} (n+j)\ln(n+j) \right|}{\left| \sum_{j=p-n+1}^{n-1} (n+j)\ln(n+j) \right|} < \frac{2^{-t_{p,x}-5+b}\,2n\,(2n-p-1)}{\ln(n)\,n\,(2n-p-1)} \leq 2^{-t_{p,x}-3}.$$

As in the proof of Lemma 8.2.2, this implies

$$w_{p,x} = \omega_p(n)(1+\varepsilon_8), \quad |\varepsilon_8| < 2^{-t_{p,x}}. \tag{8.5.10}$$

If $use\_double\_for\_w = 0$, the algorithm remains the same as in Lemma 8.2.2. So, (8.5.10) is also true in that case.

Let $\xi_p = p/(p - \chi_\Delta(p))$. We examine how accurate $\ln \xi_p$ is approximated.

Assume that

$$5 + b(p+2) \leq P,\ k + b(N) + 11 \leq P. \tag{8.5.11}$$

It follows from (8.5.11) and Theorem 2.7.2, that

$$\begin{aligned} x_{p,d} &= \xi_p(1+\varepsilon_9),\ |\varepsilon_9| < 2^{-max\{0,k+b(N)+bl(p)+3\}-5-b(p+2)}, \\ l_{p,d} &= \ln x_{p,d}(1+\varepsilon_{10}),\ |\varepsilon_{10}| < 2^{-max\{0,k+b(N)+bl(p)+3\}-4}, \end{aligned}$$

because $bl(p) + b(p+2) \leq 3$. We know from Lemma 8.2.1, that $|\ln \xi_p| \geq 1/(p+2)$. So Lemma 8.5.3 implies

$$l_{p,d} = \ln \xi_p(1+\varepsilon_{11}),\ |\varepsilon_{11}| < 2^{-t_{p,d}}. \tag{8.5.12}$$

It follows from (8.5.12) and Lemma 8.2.1, that we also have

$$l_{p,d} = \ln \xi_p(1+\varepsilon_{11}),\ |\varepsilon_{11}| < 2^{-k-b(N)-5}/|\ln \xi_p|. \tag{8.5.13}$$

If (8.5.11) is not valid, the algorithm remains the same as in Lemma 8.2.2. So,

$$l_{p,x} = \ln \xi_p(1 + \varepsilon_{12}),\ |\varepsilon_{12}| < 2^{-k-b(N)-3}/|\ln \xi_p|. \tag{8.5.14}$$

Assume, that *use_double_for_w* $= 2$. It follows from (8.5.8), and (8.5.12), that

$$l_{p,d} w_{p,d} \quad = \quad \omega_p(n) \ln \xi_p(1 + \varepsilon_4)(1 + \varepsilon_{11})(1 + \varepsilon_{13}),$$

with $|\varepsilon_4|, |\varepsilon_{11}| < 2^{-max\{0,k+b(N)+bl(p)+3\}-2}$, and $|\varepsilon_{13}| < 2^{-max\{0,k+b(N)+bl(p)+3\}-3}$, because of (8.5.7). Hence,

$$l_{p,d} w_{p,d} = \omega_p(n) \ln \xi_p(1 + \varepsilon_{14}),\ |\varepsilon_{14}| < 2^{-k-b(N)-bl(p)-3}. \tag{8.5.15}$$

It follows from Lemma 8.2.1, (8.5.15) and $|\omega_p(n)| \le 1$, that

$$|l_{p,d} w_{p,d} - \omega_p(n) \ln \xi_p| < 2^{-k-b(N)-3}. \tag{8.5.16}$$

Together with (8.5.10) and (8.5.14), it follows from the proof of Lemma 8.2.2, that

$$|w_{p,x} l_{p,x} - \omega_p(n) \ln \xi_p| < 2^{-k-b(N)-1}. \tag{8.5.17}$$

Now, fix a prime p with $2 \le p < n$.

As in (8.5.12), we have

$$l_{p,d} = \ln \xi_p(1 + \varepsilon_{15}),\ |\varepsilon_{15}| < 2^{-k-b(N)-bl(p)-3}. \tag{8.5.18}$$

By using Lemma 8.2.1 again, this implies

$$|l_{p,d} - \ln \xi_p| < 2^{-k-b(N)-3}. \tag{8.5.19}$$

And if $l_{p,x}$ is computed, the algorithm remains the same as in Lemma 8.2.2. So,

$$|l_{p,x} - \ln \xi_p| < 2^{-k-b(N)-1}. \tag{8.5.20}$$

Now, we prove that the sum is approximated correctly.

Assume, that *use_double_for_l* $= 1$. Then all approximations of $\omega_p(n)$ and $\ln \xi_p$ are computed as `doubles` in $w_{p,d}$ and $l_{p,d}$, respectively. Furthermore, we have

$$k + 2b(N) + 4 \le P. \tag{8.5.21}$$

It follows from (8.5.21) and Theorem 2.7.2, that the relative error in each addition step for l_d is less than $2^{-k-2b(N)-4}$. So (8.5.15), (8.5.18), and Lemma 8.5.2 imply

$$|\ell(n, \Delta) - l_d| < 2^{-k-1}. \tag{8.5.22}$$

Assume, that *use_double_for_l* $= 0$. For a prime $2 \le p < 2n - 1$, set $z_p = 0$, if $k + b(N) + 1 + bl(p) < 0$. Otherwise, set

$$z_p = \begin{cases} (\texttt{xbigfloat}) w_{p,d} l_{p,d}, & \text{for } n \le p < 2n-1, \textit{use\_double\_for\_w} = 2, \\ w_{p,x} (\texttt{xbigfloat})(l_{p,d}), & \text{for } n \le p < 2n-1, \textit{use\_double\_for\_w} \ne 2, \\ (\texttt{xbigfloat})(l_{p,d}), & \text{for } 2 \le p < n, \end{cases}$$

for the case that $l_{p,d}$ is computed. It then follows from (8.5.16), (8.5.13), (8.5.10), (8.5.18), and the proof of Lemma 8.2.2, that

$$\left.\begin{array}{rr} n \le p < 2n-1, & |z_p - \omega_p(n) \ln \xi_p| \\ 2 \le p < 2, & |z_p - \ln \xi_p| \end{array}\right\} < 2^{-k-b(N)-1}. \tag{8.5.23}$$

Furthermore, set

$$z_p = \begin{cases} (\texttt{xbigfloat})(w_{p,d})l_{p,x}, & \text{for } n \leq p < 2n-1, use\_double\_for\_w = 2, \\ w_{p,x}l_{p,x}, & \text{for } n \leq p < 2n-1, use\_double\_for\_w \neq 2, \\ l_{p,x}, & \text{for } 2 \leq p < n, \end{cases}$$

for the case that $l_{p,x}$ is computed. Then (8.5.23) follows from (8.5.8), (8.5.17), (8.5.20), and the proof of Lemma 8.2.2.

Hence, with the value of l_x before `truncate` is applied, we have by (8.5.23)

$$|\ell(n,\Delta) - l_x| \leq \sum_{2 \leq p < 2n-1} |\ln(\xi_p) - z_p| < \sum_{2 \leq p < 2n-1} 2^{-k-1-b(N)} < 2^{-k-1}. \tag{8.5.24}$$

It follows from (8.5.22), (8.5.24), and Lemma 2.3.1, that the truncated l_x is an absolute k-approximation to $\ell(n,\Delta)$. □

8.6. Approximating hR

Let Δ be a real quadratic discriminant and O the corresponding real quadratic order. The analytic class number formula (1.5.1) states that

$$hR = L(1, \chi_\Delta)\sqrt{\Delta}/2,$$

where h is the class number and R is the regulator of O. In the previous sections we described how to approximate $L(1, \chi_\Delta)$. This enables us to approximate the product hR using the analytic class number formula. The computation of an approximation to hR is necessary for example in the $\Delta^{1/5}$ algorithm for approximating the regulator described in Chapter 11.

The following function determines an approximation $H = hR(1+\varepsilon)$ with $|\varepsilon| < B$, where $0 < B \leq 1/2$ is a floating point number.

```
xbigfloat O.relative_hR_approximation(xbigfloat B)
 {
    xbigfloat eta = 511/512;
    xbigfloat d = sqrt(O.discriminant, -b( (1-eta)/2 B ) + 1);
    xbigfloat L = O.relative_L1chi_approximation( eta B );
    return d*L/2;
 }
```

THEOREM 8.6.1. *(ERH) On input of a floating point number* $0 < B \leq 1/2$*, the function* $O.\texttt{relative\_hR\_approximation}(B)$ *returns an approximation* $H = (m, e)$ *to hR with* $H = hR(1+\varepsilon)$, $|\varepsilon| < B$ *and* $b(m) = O(|b(B)|)$*, in time* $O(\operatorname{size} B + 4^{|b(B)|} M(|b(B)| + \log \Delta) \log(|b(B)| + \log \Delta) \log^2 \Delta)$.

PROOF. Because $\eta B < 1$, it immediately follows that $H = hR(1+\varepsilon)$ with $|\varepsilon| < B$. It follows from Theorem 8.4.2, that $b(m(L)) = O(|b(B)|)$. And because d is a relative $O(|b(B)|)$-approximation, it is $b(m) = O(|b(B)|)$.

The running time is dominated by the approximation of $L(1, \chi_\Delta)$ and so, follows from Theorem 8.4.2. □

REMARK 8.6.2. (Choice of η) We explain the choice of the weight η. It follows from the analysis of the $L(1, \chi_\Delta)$ computation, formula (8.1.10), that with the optimal weight $\eta = 1$, the number of terms, n, would be determined such that

$$C(n) \leq \rho B/(1+\rho B) - 2^{-k}, \tag{8.6.1}$$

where $\rho = 63/64$ and $k = -b(B) + 8$. Instead we compute the number of terms, such that for $1/2 \le \eta < 1$, it is

$$C(\kappa n) \le \eta\rho B/(1+\eta\rho B) - 2^{-k}, \tag{8.6.2}$$

where $\kappa \ge 1$. Note that

$$0 \le 2^{-k} 1/(1 + 1/(\rho B)) \le 2^{-9}.$$

This implies

$$\begin{aligned}
\eta\rho B/(1+\eta\rho B) - 2^{-k} &\ge \eta\rho B/(1+\rho B) - 2^{-k} \\
&= (\rho B/(1+\rho B) - 2^{-k}) \frac{\eta\rho B/(1+\eta\rho B) - 2^{-k}}{\rho B/(1+\eta\rho B) - 2^{-k}} \\
&\ge (\rho B/(1+\rho B) - 2^{-k})(\rho - 2^{-k} 1/(1+1/(\rho B))) \\
&\ge (\rho B/(1+\rho B) - 2^{-k})(\rho - 2^{-9}).
\end{aligned}$$

Because $C(\kappa n) \le 1/\sqrt{\kappa} C(n)$, it follows from (8.6.1), that (8.6.2) is satisfied, if

$$1/\sqrt{\kappa} \le \rho - 2^{-9}. \tag{8.6.3}$$

We accept $\kappa = 101/100$, i.e. an increase of $1/100$ in the number of terms, and thus, we may choose $\rho = 511/512$ to satisfy (8.6.3). Note that we have accepted an increase of $1/10$ in the number of terms due to the weight used in the function O.`relative_L1chi_approximation`. By accepting an additional increase of $1/100$ now, we obtain an overall increase of less than $12/100$.

We describe how to compute an absolute approximation to hR. By computing an initial approximation to hR using the analytic class number formula, we derive F with $hR < B/F$. Then we use the function above to compute an approximation H with

$$|hR - H|/(hR) < F.$$

Then the absolute error is $|hR - H| < B$. We present the function O.`absolute_hR_approximation`(B), that implements that method for a bound $B > 0$ on the absolute error.

```
xbigfloat O.absolute_hR_approximation(xbigfloat B)
 {
   xbigfloat L = O.relative_L1chi_approximation(2^(-2));
   xbigfloat d = sqrt(O.discriminant, 12) / 2;
   xbigfloat E = divide(B, d*L, 12);
   xbigfloat F = E * (1-2^(-12))^2 * (1-2^(-2));
   xbigfloat H = O.relative_hR_approximation(min(F, 1/2));
   return H;
 }
```

THEOREM 8.6.3. *(ERH) On input of a floating point number $B > 0$, the function O.`absolute_hR_approximation`(B) returns a floating point number $H = (m, e)$ with $|H - hR| < B$, and $b(m) = O(G)$, in time $O(4^G M(G + \log\Delta)\log(G + \log\Delta)\log^2\Delta)$, where $G = |b(B) - b(\Delta)/2| + b(\log\log\Delta)$.*

Proof. The function computes the following approximations:

$$\begin{aligned} L &= L(1,\chi_\Delta)(1+\varepsilon_1),\ |\varepsilon_1| < 2^{-2}, \\ d &= \sqrt{\Delta}/2(1+\varepsilon_2),\ |\varepsilon_2| < 2^{-12}, \\ E &= B/(dL)(1+\varepsilon_3), |\varepsilon_3| < 2^{-12}, \\ F &= E(1-2^{-12})^2(1-2^{-2}). \end{aligned}$$

It follows that

$$\begin{aligned} hRF &= hRE(1-2^{-12})^2(1-2^{-2}) \\ &= \frac{hRB}{hR}\frac{(1-2^{-12})^2(1-2^{-2})(1+\varepsilon_3)}{(1+\varepsilon_1)(1+\varepsilon_2)} \\ &< B. \end{aligned}$$

H satisfies $|H - hR|/|hR| < F$. Thus, $|H - hR| < B$.

We estimate the running time of the procedure. It follows from the analytic class number formula, that there is some function $c_1 \in \mathbb{Z}$, such that $b(F) = b(B) - b(\Delta)/2 - b(L(1,\chi_\Delta)) + c_1$. We know from [**JLW95**][(5.1)], that there are some constants $c_2, c_3 \in \mathbb{N}$, such that

$$(1 + o(1))(c_1 \ln\ln\Delta)^{-1} < L(1,\chi_\Delta) < (1 + o(1))c_2 \ln\ln\Delta.$$

Together with Theorem 8.6.1, it follows that the running time for the second relative approximation of hR is $O(4^G \log^2\Delta\, M(G + \log\Delta)\log(G + \log\Delta))$. And this is the overall running time. Also, we have $b(m) = O(G)$ by Theorem 8.6.1. $\square$

CHAPTER 9

The bounded equivalence problem

Let A and B be reduced fractional O-ideals. In this chapter we examine algorithms for computing a quadratic number α with $A = \alpha B$, if such a number exists whose logarithm is below a given bound. An algorithm for solving this problem has been presented by Biehl and Buchmann [**BB94**]. We extend their work and describe an efficient implementation of that algorithm.

9.1. Bounded equivalence

Throughout this section let A and B be reduced fractional O ideals.

We call a number $\alpha \in K^*$ a *generator of A relative to B,* if

$$A = \alpha B.$$

If α is a generator and η is the fundamental unit of O, then the set of all generators of A relative to B is

$$\{\pm \alpha \eta^k | k \in \mathbb{Z}\}. \tag{9.1.1}$$

The generator α is called the *minimal generator of A relative to B*, if it is the smallest generator with $\alpha > 1$. The minimal generator satisfies

$$\operatorname{Ln} \alpha = min\{\operatorname{Ln} \beta | 1 < \beta \in K^*, A = \beta B\}.$$

and as a consequence from (9.1.1), it also satisfies

$$0 < \operatorname{Ln} \alpha \leq R, \tag{9.1.2}$$

where R is the regulator of O.

The *bounded equivalence problem* can be stated as follows: Given the reduced ideals A and B and $u \in \mathbb{R}_{\geq 1}$, if there is a generator α of A relative to B with $0 < \operatorname{Ln} \alpha < u$, compute the minimal generator. The algorithm for solving this problem is based on the following Theorem which is a slightly modification of [**BB94**][Proposition 6.12]. Because the details are important, we also give a proof of the Theorem, which is almost the same as in [**BB94**].

THEOREM 9.1.1. *Let O be a real quadratic order, A, B be reduced O ideals, and $u \in \mathbb{R}_{\geq 1}$. Let $\alpha > 1$ be a generator of A relative to B and set $q = max\{0, \lceil (\operatorname{Ln} \alpha - ((\ln \Delta)/4 + 1))/\sqrt{u} - 1\rceil\}$.*

For $q = 0$, set $\beta = 1$. For $q \geq 1$, let β be any minimum in A with $|\operatorname{Ln} \beta - q\sqrt{u}| < (\ln \Delta)/4 + 1$. Then for $\gamma = \alpha/\beta$ the inverse $1/\gamma$ is a minimum of B with $0 < \operatorname{Ln} \gamma < \sqrt{u} + (\ln \Delta)/2 + 2$. Also, if $\operatorname{Ln} \alpha < u$, then $q < \sqrt{u}$.

PROOF. Let $\alpha > 1$ be a generator of A relative to B and let q be as above. Suppose that $q = 0$. Then $\gamma = \alpha$ and $0 < \operatorname{Ln} \gamma = \operatorname{Ln} \alpha < \sqrt{u} + (\ln \Delta)/4 + 1$.

Now assume that $q > 0$. Then

$$(q - 1)\sqrt{u} < \operatorname{Ln} \alpha - ((\ln \Delta)/4 + 1) - \sqrt{u} \leq q\sqrt{u}$$

and therefore

$$(\ln \Delta)/4 + 1 < \operatorname{Ln} \alpha - q\sqrt{u} \leq \sqrt{u} + (\ln \Delta)/4 + 1. \tag{9.1.3}$$

Also

$$|\operatorname{Ln} \beta - q\sqrt{u}| < (\ln \Delta)/4 + 1. \tag{9.1.4}$$

Since $\gamma = \alpha/\beta$, the inverse $1/\gamma$ is a minimum in B, and it follows from (9.1.3) and (9.1.4), that $0 < \operatorname{Ln} \gamma < \sqrt{u} + (\ln \Delta)/2 + 2$. It also follows from (9.1.3), that $q < \sqrt{u}$, if $\operatorname{Ln} \alpha < u$. □

We present the function A.`is_bounded_equivalent_engine` (a, B, u, k) for reduced ideals A and B. If it returns false, there is no generator α of A relative to B with $0 < \operatorname{Ln} \alpha < u$ for $u \in \mathbb{R}_{\geq 1}$. If it returns true, it determines an absolute k approximation a to $\operatorname{Ln} \alpha$, where α is the minimal generator of A relative to B.

The procedure is a member function of the class `quadratic_ideal`. It makes use of another member function A.`is_bounded_equivalent_babysteps` (a, H, B, u, k) that will be presented later. If that function returns false, there is no generator α of A relative to B with $0 < \operatorname{Ln} \alpha < \sqrt{u} + (\ln \Delta)/2 + 2$. If it returns true, it determines an absolute k approximation a to $\operatorname{Ln} \alpha$, where α is the minimal generator of A relative to B. If it returns false, it also computes a hash table H of pairs (C, c), where C is a quadratic ideal and c a floating point number. In that case, H contains a pair $(\gamma B, c)$, where $|c - \operatorname{Ln} \gamma| < 2^{-k}$, for each minimum $1/\gamma$ of B with $0 < \operatorname{Ln} \gamma < \sqrt{u} + (\ln \Delta)/2 + 2$ and every pair in the table satisfies $0 < \operatorname{Ln} \gamma < \sqrt{u} + (\ln \Delta)/2 + 3$. The class `quadratic_ideal_with_logarithm` implements such pairs. Hashing is done using the component a of a quadratic ideal $q(\mathbb{Z}a + \mathbb{Z}(b + \sqrt{\Delta})/2)$.

After having computed the babysteps, the procedure `is_bounded_equivalent_engine` determines minima β of A close to $q\sqrt{u}$ for $q = 1, 2, \ldots$ until A/β is found in the hash table. Then the approximations b to $\operatorname{Ln} \beta$ and c to $\operatorname{Ln} \gamma$ are used to determine $a = b + c$, i.e. $\alpha = \beta\gamma$.

```
bool A.is_bounded_equivalent_engine (

      xbigfloat & a,
      quadratic_ideal B,
      xbigfloat u,
      int k)
 {
    // Babysteps
    //
    // Test for B gamma = A with 0 < Ln gamma < sqrt{u} + (ln Delta)/2 + 2.
    //
    hash_table<quadratic_ideal_with_logarithm> H;

    if (A.is_bounded_equivalent_babysteps(a,H,B,u,k+2))
      {
        a.truncate(a.exponent()+k+1);
        return true;
      }

    // Giantsteps
```

```
//
// absolute b(sqrt(u))+3 - approx. v to sqrt(u)
//
xbigfloat v = sqrt(u, 3+2 * ceil(b(u)/2) );

// Start value for q is ceil(h / sqrt{u}),
// where |h - (3 + \ln\D)/4| < 1/4.
//
xbigfloat h;
h = log(A.order.discriminant, 0);
h += 3;
h /= 4;

bigint q = 1;
while (q^2 u < h^2) ++q;

// start loop
//
xbigfloat b, c;
quadratic_number_power_product beta;
quadratic_ideal_with_logarithm *P;
quadratic_ideal C;

xbigfloat s     = (q-1) v + 1/8;
bool      found = false;

while (q^2 < u && !found )
  {
    // abs. 3 approximation s to  q * sqrt(u) + 1/8
    //
    s += v;

    // Find minimum of A close to s with regard to 2.
    //
    C = A;
    C.close(beta,b,s,max(k+2,3));

    // Search for C in H
    //
    P = H.search(C);

    if (P != NULL)
     {
        found = true;
        c = P->get_logarithm();

        // alpha = beta * gamma
        //
        a = b+c;
```

```
            a.truncate(a.exponent()+k+1);
          }

         ++q;
      }

    return found;
 }
```

To be able to prove the correctness of the procedure we need an auxiliary result.

LEMMA 9.1.2. *Let O be a real quadratic order, A, B be reduced fractional O ideals, and $u \in \mathbb{R}_{\geq 1}$. Let α be the minimal generator of A relative to B, $\operatorname{Ln}\alpha \geq \sqrt{u} + (\ln \Delta)/2 + 2$, and let $q = max\{0, \lceil (\operatorname{Ln}\alpha - ((\ln\Delta)/4 + 1))/\sqrt{u} - 1\rceil\}$. If for any $q' \in \mathbb{Z}$, with $1 \leq q' \leq q$, $\beta' \geq 1$ is a minimum of A with $A/\beta' = B\gamma'$, $0 < \operatorname{Ln}\gamma' < \operatorname{Ln}\alpha$, and $|\operatorname{Ln}\beta' - q'\sqrt{u}| < (\ln\Delta)/4 + 1$, then $\beta'\gamma' = \alpha$.*

PROOF. Let β' be a minimum of A with $A/\beta' = B\gamma'$, $0 \leq \operatorname{Ln}\gamma' < \operatorname{Ln}\alpha < R$ and $|\operatorname{Ln}\beta' - q'\sqrt{u}| < (\ln\Delta)/4 + 1$. Because $q' \leq q$, we obtain the upper bound

$$\operatorname{Ln}\beta' < q\sqrt{u} + (\ln\Delta)/4 + 1. \tag{9.1.5}$$

First, assume that $\operatorname{Ln}\beta' > q\sqrt{u} - (\ln\Delta)/4 - 1$. Then it follows from (9.1.5) that

$$|\operatorname{Ln}\beta' - q\sqrt{u}| < (\ln\Delta)/4 + 1. \tag{9.1.6}$$

Because $q \geq 1$, it follows from (9.1.6) and Theorem 9.1.1, that for

$$\gamma = \alpha/\beta',$$

we have $0 \leq \operatorname{Ln}\gamma < \sqrt{u} + (\ln\Delta)/2 + 2 \leq \operatorname{Ln}\alpha < R$ and therefore

$$|\operatorname{Ln}(\beta'\gamma') - \operatorname{Ln}\alpha| \quad = \quad |\operatorname{Ln}\gamma' - \operatorname{Ln}\gamma| < R.$$

It follows from (9.1.1), that $\alpha = \beta'\gamma'$.

Now, assume that $\operatorname{Ln}\beta' \leq q\sqrt{u} - (\ln\Delta)/4 - 1$. Choose $s \in \mathbb{Q}$ with $q\sqrt{u} \leq s < q\sqrt{u} + 1/4$ and $\beta \in Min(A)$ close to s with regard to 2, i.e.,

$$|\operatorname{Ln}\beta - q\sqrt{u}| < (\ln\Delta)/4 + 1. \tag{9.1.7}$$

Because $q \geq 1$, it follows from (9.1.7) and Theorem 9.1.1, that for

$$\gamma = \alpha/\beta,$$

we have $\operatorname{Ln}\gamma \geq 0$. Hence (9.1.2) implies

$$\operatorname{Ln}\beta \leq \operatorname{Ln}\beta + \operatorname{Ln}\gamma = \operatorname{Ln}\alpha < R.$$

As a consequence, we obtain

$$0 \leq \operatorname{Ln}\beta' \leq q\sqrt{u} - (\ln\Delta)/4 - 1 < \operatorname{Ln}\beta < R.$$

Because $0 < \operatorname{Ln}\gamma' < \operatorname{Ln}\alpha$ and $0 \leq \operatorname{Ln}\beta' < R$, we have

$$0 < \operatorname{Ln}(\beta'\gamma') < \operatorname{Ln}\alpha + R,$$

i.e., $\beta'\gamma' = \alpha$ by (9.1.1). $\square$

In each iteration of the loop, the procedure determines a new value for s. We derive bounds on s. The procedure computes v with $|v-\sqrt{u}|/\sqrt{u} < 2^{-3-2b(\sqrt{u})}$, so $|v-\sqrt{u}| < 2^{-3-b(\sqrt{u})}$. Let s and q be with the values that they have at the beginning of each loop iteration after s has been increased by v. It is $q < \sqrt{u}$, $s = qv + 1/8$, and $|s - q\sqrt{u} - 1/8| = |q||v - \sqrt{u}| < \sqrt{u}2^{-3-b(\sqrt{u})} < 2^{-3}$. This implies

$$q\sqrt{u} \le s < q\sqrt{u} + 1/4, \tag{9.1.8}$$

before the computation of β in the while loop.

Now, in the while loop, β is chosen to be close to s with regard to 2. So

$$|s - \operatorname{Ln}\beta| < (\ln\Delta)/4 + 1/4, \tag{9.1.9}$$

and (9.1.8) and (9.1.9) yield

$$|q\sqrt{u} - \operatorname{Ln}\beta| < (\ln\Delta)/4 + 1/2. \tag{9.1.10}$$

Because we choose the start value of q as $\lceil l/\sqrt{u}\rceil$, where $|l - (3+\ln\Delta)/4| < 1/4$, we get $q\sqrt{u} > l > (\ln\Delta)/4 + 1/2$, so

$$\operatorname{Ln}\beta > 0. \tag{9.1.11}$$

Why may we omit the smaller values of q ? If $1 \le q \le \lceil l/\sqrt{u}\rceil - 1$, then $q\sqrt{u} \le l < (\ln\Delta)/4 + 1$. In this case, $\beta = 1$ is a minimum in A that satisfies $|q\sqrt{u} - \operatorname{Ln}\beta| < (\ln\Delta)/4 + 1$, which is sufficient (compare to Theorem 9.1.1). But we can ignore $\beta = 1$, because a match for it would have been found during the babysteps already.

Further note, that the approximation b of $\operatorname{Ln}\beta$ satisfies

$$|b - \operatorname{Ln}\beta| < 2^{-k-2}, \tag{9.1.12}$$

in each iteration of the loop. We will now prove the correctness of the procedure.

Theorem 9.1.3. *Let O be a real quadratic order, A, B be reduced O ideals, $u \in \mathbb{R}_{\ge 1}$, and $k \in \mathbb{Z}$. If the function A.`is_bounded_equivalent_engine` (a, B, u, k) returns false, then there is no generator α of A relative to B with $0 < \operatorname{Ln}\alpha < u$. If it returns true, then it also returns an absolute k-approximation a to $\operatorname{Ln}\alpha$, where α is the minimal generator of A relative to B with $0 < \operatorname{Ln}\alpha < u + \sqrt{u} + 3/4\ln\Delta + 4$.*

Proof. First, we show, that in case, a generator α of A relative to B is found, a is an absolute k approximation to $\operatorname{Ln}\alpha$. If a generator is found during the babysteps, then the correctness of the approximation follows from Theorem 9.1.4 and 2.3.1. Otherwise, it follows from Theorem 9.1.4, that the hash table H contains a pair $(\gamma B, c)$ with $|c - \operatorname{Ln}\gamma| < 2^{-k-2}$, for each minimum $1/\gamma$ of B with $0 \le \operatorname{Ln}\gamma < \sqrt{u} + (\ln\Delta)/2 + 2$. Hence, if for a minimum β of A a match is found in the while loop, i.e., $A/\beta = \gamma B$ for some $(\gamma B, c)$ in H, $\alpha = \beta\gamma$ is the generator that is found. By (9.1.12) we have $|b - \operatorname{Ln}\beta| < 2^{-k-2}$ and so $|b + c - \operatorname{Ln}\alpha| < 2^{-k-1}$. So $a = b + c$ is an absolute k approximation to $\operatorname{Ln}\alpha$ after truncation by Lemma 2.3.1.

Now we can prove the assertion of the Lemma in case that the function returns false. Let α be the minimal generator of A relative to B. Assume that $0 \le \operatorname{Ln}\alpha < u$. Set

$$q(\alpha) = max\{0, \lceil(\operatorname{Ln}\alpha - ((\ln\Delta)/4 + 1))/\sqrt{u} - 1\rceil\}.$$

If $q(\alpha) = 0$, it follows from Theorem 9.1.1, that $1/\alpha$ is a minimum in B such that $A = B\alpha$ and $0 < \operatorname{Ln}\alpha < \sqrt{u} + (\ln\Delta)/2 + 2$. Hence, α is determined during the computation of the babysteps and the function returns true. If the start value of q would be larger than $q(\alpha)$, then by the conclusions above, $\beta = 1$ would be a suitable minimum in A for $q = q(\alpha)$. But in that case, a match would have been found during the babysteps.

Suppose that $q(\alpha) > 0$. Assume that we have $q = q(\alpha)$ at the beginning of the loop. It follows from (9.1.10), that the minimum determined by `close` satisfies $|q\sqrt{u} - \operatorname{Ln}\beta| < (\ln\Delta)/4 + 1$. It follows from Theorem 9.1.1, that the inverse of $\gamma = \alpha/\beta$ is a minimum in B with $0 < \operatorname{Ln}\gamma < \sqrt{u}+(\ln\Delta)/2+2$. It is $C = A/\beta = B\gamma$, so C is found in the hash table. So we have shown, that the function will find α, if $q = q(\alpha)$. By Theorem 9.1.1 we have $q(\alpha) < u$, so the function will find a match, at latest for $q = q(\alpha)$, and hence, will return true.

This implies, that the return value false indicates that there is no minimal generator α of A relative to B with $0 < \operatorname{Ln}\alpha < u$, and so, no generator at all with that property. Now we prove the assertion for the case of the return value being true.

If the function returns true, then a generator has been found and it follows from our conclusions at the beginning, that the logarithm representation of it is correct and stores an absolute k approximation to the Ln value. If that generator has been found during the babysteps, its minimality follows from Theorem 9.1.4. Otherwise, the generator is determined in the while loop. By (9.1.10), the minimum β computed by `close` satisfies

$$|q\sqrt{u} - \operatorname{Ln}\beta| < (\ln\Delta)/4 + 1, \tag{9.1.13}$$

and by (9.1.11), we have $\beta > 1$. Because no match was found during the babysteps, we also get $0 < \operatorname{Ln}\gamma < \operatorname{Ln}\alpha$ and $\operatorname{Ln}\alpha \geq \sqrt{u} + (\ln\Delta)/2 + 2$. So, Lemma 9.1.2 implies, that $\beta\gamma$ is the minimal generator of A relative to B.

By Theorem 9.1.4, every pair $(B\gamma, c)$ in H satisfies

$$0 < \operatorname{Ln}\gamma < \sqrt{u} + (\ln\Delta)/2 + 3.$$

Together with (9.1.11) and (9.1.13), we obtain that

$$0 < \operatorname{Ln}\alpha = \operatorname{Ln}\beta + \operatorname{Ln}\gamma < u + \sqrt{u} + 3/4\ln\Delta + 4.$$

□

We describe the function A.`is_bounded_equivalent_babysteps` (a, H, B, u, k) as specified above.

```
bool A.is_bounded_equivalent_babysteps (

      xbigfloat & a,
      hash_table<quadratic_ideal_with_logarithm> & H,
      quadratic_ideal B,
      xbigfloat u,
      int k)
 {
   // Absolute 2-approx. d to (ln Delta)/2.
   //
   xbigfloat d;
   log(d, A.order.discriminant, 1);
   d /= 2;

   // absolute 1 - approximation v to sqrt(u)
   //
```

```
xbigfloat v = sqrt(u, 1 + ceil(b(u)/2));

// Upper bound n on the number of babysteps.
//
bigint n = ceil( 2.9375 * ( v + d + 11/4 ) + 1 );

// Babysteps
//
quadratic_number_standard delta;
quadratic_ideal_with_logarithm p;
xbigfloat c, h;

quadratic_ideal C = B;
bigint i = 1;
d += 5/2;

H.initialize(n);
H.set_key_function(quadratic_ideal_with_logarithm_key);

do
  {
    // Neighbour of C
    //
    C.inverse_rho(delta);

    // Absolute max(2,k) approx. c to Ln of gamma = 1 / sum(i) delta[i].
    //
    c -= delta.get_absolute_Ln_approximation(b(n)+max(2,k+1));

    // Add, if Ln gamma could be less than sqrt{u} + (ln Delta)/2 + 2
    //
    h = (c - d)^2;

    if (h < u || c < d)
     {
       p.assign(C,c);
       H.hash(p);

       // Found A ?
       //
       if (C == A)
        {
          // alpha = gamma.
          //
          a = c;
          a.truncate(a.exponent()+k+1);
          return true;
        }
     }
```

```
        else
           return false;

        ++i;
     }
    while (i <= n);

    return false;
}
```

THEOREM 9.1.4. *Let O be a real quadratic order, A, B be reduced O ideals, $u \in \mathbb{R}_{\geq 1}$, and $k \in \mathbb{Z}$. A, B, u, and k are the input of the function $A.$***is_bounded_equivalent_babysteps***(a, H, B, u, k).*

If the function returns true, then it also returns an absolute k-approximation a to $\operatorname{Ln}\alpha$, where α is the minimal generator of A relative to B.

If it returns false, then there is no generator α of A relative to B with $0 < \operatorname{Ln}\alpha < \sqrt{u} + (\ln\Delta)/2 + 2$. In this case it also returns a hash table H of pairs (C, c), where C is a quadratic ideal and c a floating point number. More precisely, $C = \gamma B$, where $1/\gamma$ is a minimum of B and $|c - \operatorname{Ln}\gamma| < 2^{-k}$. The table contains a pair $(\gamma B, c)$ for each minimum $1/\gamma$ of B with $0 < \operatorname{Ln}\gamma < \sqrt{u} + (\ln\Delta)/2 + 2$, and every pair in the table satisfies $0 < \operatorname{Ln}\gamma < \sqrt{u} + (\ln\Delta)/2 + 3$.

PROOF. First the procedure determines approximations d and v, such that

$$|d - (\ln\Delta/2)| < 2^{-2},\ |v - \sqrt{u}| < 2^{-1}.$$

Let δ_i denote the number determined by the call of $C.$`inverse_rho` in the i-th iteration of the loop. Set

$$\gamma_i = 1/\sum_{j=1}^{i}\delta_j.$$

Then $1/\gamma_i$ is a minimum of B, the i-th neighbour right from 1 in B, computed in the i-th iteration of the loop. Let C_i be the value of C in the i-th iteration, then $C_i = B\gamma_i$.

We prove that n is an upper bound on the sufficient number of applications of the `inverse_rho` operator. We have $n \geq 2.9375(v + 2^{-1} + d + 2^{-2} + 2) + 1 \geq \lceil 2/\ln 2(\sqrt{u} + (\ln\Delta)/2 + 2)\rceil$. It follows from Lemma 5.3.2, that

$$\operatorname{Ln}\gamma_n > n(\ln 2)/2 \geq \sqrt{u} + (\ln\Delta)/2 + 2. \tag{9.1.14}$$

Let c_i be the value of c in the i-th iteration of the loop. It follows from (9.1.14) that

$$|c_i - \operatorname{Ln}\gamma_i| < 2^{-max\{2,k+1\}}.$$

If the function returns true in the i-th iteration, then $C_i = A$, so γ_i is the minimal generator of A relative to B. Hence, $a = c$ is an absolute k approximation to $\operatorname{Ln}\gamma_i$ after truncation by Lemma 2.3.1. If the function returns false, because the loop condition is violated, it follows from (9.1.14), that the hash table H contains a pair $(\gamma B, c)$, where $|c - \operatorname{Ln}\gamma| < 2^{-k}$, for each minimum $1/\gamma$ of B with $0 < \operatorname{Ln}\gamma < \sqrt{u} + (\ln\Delta)/2 + 2$.

If the function returns false in loop iteration i, because of the return statement inside the loop, we have $(c_i - d)^2 \geq u$ and $c_i \geq d$, hence $c_i - d \geq \sqrt{u}$. Ignoring the substitution $d+=5/2$, we obtain for the computed minimum

$$\operatorname{Ln}\gamma_i > c_i - 2^{-2} \geq \sqrt{u} + d + 5/2 - 2^{-2} > \sqrt{u} + (\ln\Delta)/2 + 2.$$

So, also in this case, H contains a pair for each $1/\gamma$ of B with $0 < \operatorname{Ln}\gamma < \sqrt{u} + (\ln\Delta)/2 + 2$. This proves the correctness of the return values.

If in the i-th iteration the pair $(\gamma_i B, c_i)$ is inserted into the hash table, the condition $(c_i - d)^2 < \sqrt{u}$ or $c_i < d$ must be fulfilled. Ignoring the substitution $d+=5/2$, this implies $c_i - 2^{-2} < \sqrt{u} + d + 2^{-2} + 2$. Hence

$$\operatorname{Ln}\gamma_i < c_i + 2^{-2} < \sqrt{u} + (\ln\Delta)/2 + 3.$$

□

9.2. Accelerating the giant steps

The time consuming part for the computation of the giantsteps in the function A.is_bounded_equivalent_engine(a, B, u, k) as presented above, is the repeated call of the close function, which always starts from the beginning to find a minimum in the given distance. We can easily accelerate the computation of those giantsteps by determining a minimum δ of O in distance approximately $\sqrt{u}$ and the corresponding ideal $D = O/\delta$ once. To find minima of A in distance approximately $\sqrt{u}, 2\sqrt{u}, 3\sqrt{u}, \ldots$, we repeat the process of multiplying by D followed by a correction step. This faster method is realized by the following variant of the procedure.

```
bool A.is_bounded_equivalent_engine (

     xbigfloat & a,
     quadratic_ideal B,
     xbigfloat u,
     int k)
{
   // Babysteps
   //
   // Test for B gamma = A with 0 < Ln gamma < sqrt{u} + (ln Delta)/2 + 2.
   //
   hash_table<quadratic_ideal_with_logarithm> H;

   if (A.is_bounded_equivalent_babysteps(a,H,B,u,k+2))
     {
       a.truncate(a.exponent()+k+1);
       return true;
     }

   // absolute 3 + b(sqrt(u)) - approx. v to sqrt(u)
   //
   xbigfloat v = sqrt(u, 3 + 2 * ceil(b(u)/2) );

   // Accuracy for approximations.
   //
   long l = max(2,k+2)+ceil(b(u)/2)+2;
```

```
// Now we search for minima beta in A such that
// |s - \Ln\b| < (ln D)/4 + 1/4 with q sqrt(u) <= s < q sqrt(u)+1/4,
// and A/beta = B*gamma for some B*gamma in H.
//

// Start value for q is ceil(h / sqrt(u)),
// where |h - (3 + ln D)/4| < 1/4.
//
xbigfloat h;
h = log(A.order.discriminant, 0);
h += 3;
h /= 4;

bigint q = 1;
while (q^2 u < h^2) ++q;

// Find minimum beta of A close to (q-1) v + 1/8 with regard to l-1.
//
quadratic_ideal C;
quadratic_number_power_product beta;
xbigfloat b;
xbigfloat s = (q-1) v + 1/8;

C = A;
C.close(beta,b,s,l);

// Find minimum delta of O close to v with regard to l-1.
//
quadratic_ideal D;
quadratic_number_power_product delta;
xbigfloat d;

D = A.order;
D.order_close(delta,d,v,l);

// For q >= 1, find minimum beta_q of A close to q v + 1/8
// by finding a minimum mu_q of A / (beta_(q-1) delta) close
// to qv+1/8 - b_(q-1) - d and setting beta_q = beta_(q-1) delta mu_q.
//
quadratic_ideal_with_logarithm *P;
xbigfloat h, m;
quadratic_number_standard mu;

bool found = false;

while (q^2 < u && !found)
  {
    // absolute 3 approximation s to q * sqrt(u) + 1/8.
    //
```

```
        s += v;

        // Find minimum beta of A close to s with regard to l-1.
        //
        C *= D;
        b += d;
        C.local_close(mu,m,s-b,l+1);
        b += m;

        // Search for C in H.
        //
        P = H.search(C);

        if (P != NULL)
         {
            // alpha = beta * gamma
            //
            a = b+P->get_logarithm();
            a.truncate(a.exponent()+k+1);
            found = true;
         }

        ++q;
      }

    return found;
}
```

THEOREM 9.2.1. *Let O be a real quadratic order, A, B be reduced O- ideals, $u \in \mathbb{R}_{\geq 1}$, and $k \in \mathbb{Z}$. If the function A.`is_bounded_equivalent_engine` (a, B, u, k) returns false, then there is no generator α of A relative to B with $0 < \operatorname{Ln}\alpha < u$. If it returns true, then it also returns an absolute k-approximation a to $\operatorname{Ln}\alpha$, where α is the minimal generator of A relative to B.*

PROOF. The implementation of A.`is_bounded_equivalent_engine`(a, B, u, k) differs from the first variant of the function only in the computation of the giantsteps, i.e. the minima β and the approximations b of their Ln values, are computed in different ways.

The computations of s and of the start value for q remain the same. So (9.1.8) and (9.1.11) are still valid. Once we have shown, that also (9.1.9) and (9.1.12) are fulfilled, we may apply the proof of Lemma 9.1.4 to prove the assertions. I.e., we must prove that

$$|s - \operatorname{Ln}\beta| < (\ln \Delta)/4 + 1/4, \tag{9.2.1}$$

and

$$|b - \operatorname{Ln}\beta| < 2^{-k-2} \tag{9.2.2}$$

at the end of each loop iteration.

We introduce some notation. Let $q_0 = \lceil l/\sqrt{u} \rceil - 1$. Set $s_i = iv + 1/8$ for $i \geq q_0$. By β_{q_0},b_{q_0} we denote the value of β, b determined by the first call of `close`, respectively. By β_i, b_i, μ_i, m_i, we denote the value of β, b, μ, m for $q = i$, immediately before the search in the hash table is done.

First note, that $q < \sqrt{u}$ at the beginning of each iteration of the loop. Furthermore, $l = max\{2, k+2\} + \lceil b(u)/2 \rceil + 2 \geq max\{2, k+2\} + b(2\sqrt{u}) + 1)$. Hence, at the beginning of each loop iteration, we have

$$(2q+1)2^{-l} < 2^{-max\{2,k+2\}}. \tag{9.2.3}$$

Fix $q_0 < q < \sqrt{u}$. Then $|b_q - \operatorname{Ln}\beta_q| = |b_{q_0} + \sum_{i=q_0+1}^{q}(d + m_i) - \operatorname{Ln}\beta_0 - \sum_{i=q_0+1}^{q}(\operatorname{Ln}\delta + \operatorname{Ln}\mu_i)| \leq |b_0 - \operatorname{Ln}\beta_0| + q|d - \operatorname{Ln}\delta| + \sum_{i=q_0+1}^{q}|m_i - \operatorname{Ln}\mu_i| < (2q+1)2^{-l} < 2^{-k-2}$ by (9.2.3). This proves (9.2.2).

Similarly, we have $|s_q - \operatorname{Ln}\beta_q| = |s_q - (b_{q-1} + d) + (b_{q-1} + d) - \operatorname{Ln}\beta_{q-1} - \operatorname{Ln}\delta - \operatorname{Ln}\mu_q| \leq |s_q - (b_{q-1} + d) - \operatorname{Ln}\mu_q| + |b_{q-1} + d - \operatorname{Ln}\beta_{q-1} - \operatorname{Ln}\delta| \leq (\ln\Delta)/4 + 2^{-l} + (2(q-1)+2)2^{-l} = (\ln\Delta)/4 + (2q+1)2^{-l} < (\ln\Delta)/4 + 1/4$ by (9.2.3). This proves (9.2.1). □

9.3. Constant accuracy and refinement

The approximation of logarithms is a time consuming part, when solving the bounded equivalence problem $A = \alpha B$. Hence, we use the following strategy. We call A.`is_bounded_equivalent_engine`$(a, B, u, 3)$, presented in the previous section, to obtain an absolute 3-approximation to the logarithm. Then we build a logarithm representation for α and refine the logarithm approximation using the function α.`refine_logarithm_approximation`(k) presented in Section 7.3.

```
bool A.is_bounded_equivalent (

          xbigfloat & a,
          quadratic_ideal B,
          xbigfloat u,
          long k )
 {
    bool found = A.is_bounded_equivalent_engine(a,B,u,3);

    if (found)
     {
       if (k < 3)
         a = truncate(a, k+a.exponent()+1);

       else if (k > 3)
         {
           quadratic_number_logarithm alpha (A/B,a,3,1);
           alpha.refine_logarithm_approximation(k);
           a = alpha.logarithm_approximation();
         }
     }

    return found;
 }
```

THEOREM 9.3.1. *Let O be a real quadratic order, A, B be reduced, invertible O-ideals, $k \in \mathbb{Z}$, $u \in \mathbb{R}_{\geq 1}$, and $k \in \mathbb{Z}$. A, B, u, and k are the input of the function A.***is_bounded_equivalent** (a, B, u, k).*

If the function returns false, then there is no generator β of A relative to B with $0 < \operatorname{Ln} \beta < u$. If it returns true, then it also returns an absolute k-approximation a to $\operatorname{Ln} \alpha$, where α is the minimal generator of A relative to B.

There are constants $c_0, c_1 \in \mathbb{N}$ such that for constant accuracy k the running time of the function is $O(\sqrt{u}(\log u)^{c_0}(\log \Delta)^{c_1})$.

PROOF. The correctness follows from Theorem 9.1.3. We only sketch the proof of the running time. There are constants $c_2, c_3 \in \mathbb{N}$ such that the time for each operation in one loop iteration of the babysteps and in one loop iteration of the giantsteps in the function `is_bounded_equivalent` takes time $O((\log u)^{c_2}(\log \Delta)^{c_3})$. Therefore, the time for the loops and so the time for the whole function is $O(\sqrt{u}(\log u)^{c_2}(\log \Delta)^{c_3})$.

Because A and B are reduced, the time for computing A/B is polynomial in $\log \Delta$. Furthermore, α is a minimum in A, so $\log \log(H(\alpha)d(\alpha)) \leq \log(u) + \log \log \Delta$ by Lemma 5.1.4 and Lemma 5.3.4. It follows from Proposition 7.3.1 that there are constants $c_4, c_5 \in \mathbb{N}$ such that the time for refining the approximation is $O((\log u)^{c_4}(\log \Delta)^{c_5})$.

Therefore, there are constants $c_0, c_1 \in \mathbb{N}$ such that the overall running time is $O(\sqrt{u}(\log u)^{c_0}(\log \Delta)^{c_1})$. □

9.4. Bounded regulator

Using the function for solving the bounded equivalence problem, we can easily approximate the regulator of O, if it is below a given bound. We describe the function O.`regulator_if_lt`(r, u, k). If it returns false, then the regulator R of O satisfies $R \geq u$. If it returns true, it also returns an absolute k-approximation r to the regulator.

```
bool O.regulator_if_lt (

          xbigfloat & r,
          xbigfloat u,
          int k)
 {
    return O.is_bounded_equivalent(r,O,u,k);
 }
```

THEOREM 9.4.1. *Let O be a real quadratic order, $u \in \mathbb{R}_{\geq 1}$, and $k \in \mathbb{Z}$. If $R < u$, then the function O.***regulator_if_lt**(r, O, u, k) *returns true. If the function returns true, then it also returns an absolute k-approximation r to R.*

There are constants $c_0, c_1 \in \mathbb{N}$ such that for constant accuracy k the running time of the function is $O(\sqrt{u}(\log u)^{c_0}(\log \Delta)^{c_1})$.

PROOF. Apply Theorem 9.3.1. □

CHAPTER 10

Regulator approximation from regulator multiple

Throughout this chapter let R be the regulator of O. In this chapter we assume that we know an approximation a to mR for $m \in \mathbb{N}$ with $|a - mR| < 2^{-3}$. We describe algorithms for computing an absolute k-approximation to R.

If $m = 1$, this is just a refinement of the approximation. For the case, that $m > 1$, we present a function in Section 10.2 that for a number $p \in \mathbb{N}$ returns false, if p does not divide m. If p divides m, it returns true and also an approximation to mR/p. If we know an upper bound on m, we can execute this test for all primes less than the bound to obtain an approximation to R. This is done in Section 10.3.

10.1. Refining a regulator approximation

We describe the function O.`regulator_refinement`(a, k), that on input of a floating point number r with $|r - R| < 2^{-3}$, and an integer k, returns an absolute k-approximation to the regulator R of O. It makes use of the function O.`fundamental_unit`(r, k) presented in Section 7.2, that computes a reduced power product representation of the fundamental unit of O.

```
xbigfloat O.regulator_refinement(xbigfloat r, int k)
 {
    quadratic_number_power_product eta = O.fundamental_unit(r,3);
    return eta.absolute_Ln_approximation(k);
 }
```

Proposition 10.1.1. *On input of an absolute 3-approximation r to the regulator R of O, and $k \in \mathbb{Z}_{\geq 3}$, the function O.`regulator_refinement`(r, k) returns an absolute k-approximation to R in time* $O(\log \Delta[\log \Delta M(\log \Delta) \log\log \Delta + M(k + \log \Delta) \log(k + \log \Delta)])$.

Proof. We note that this is a special case of Proposition 7.3.1. □

Remark 10.1.2. If $M(n) = n^2$ in the situation of Proposition 10.1.1, then it follows from Remark 7.3.2, that the running time can be simplified to $O((\log \Delta)^4 + \log \Delta (k + \log \Delta)^2 \log(k + \log \Delta))$.

10.2. Divisor of regulator multiple

Let $m \in \mathbb{N}$. We describe the procedure `is_divisor`. Its input consists of a floating point number a with $|a - mR| < 2^{-3}$, an integer $p \geq 2$, and an integer k. If the procedure returns false, then $p \nmid m$. If the procedure returns true, then b is an absolute k-approximation to qR for some $q \in \mathbb{Z}$; if, in this case, $R/p \geq (1+3\ln \Delta)/4$, then $q = m/p$, i.e., p is a divisor of m.

```
bool O.is_divisor (
```

```
                xbigfloat & b,
                xbigfloat a,
                bigint p,
                int k)
{
   // absolute 3-approximation to mR/p
   //
   xbigfloat t = divide (a,p,5+b(a)-b(p));

   // search for order
   //
   quadratic_number_power_product beta;
   quadratic_ideal I = 0;
   I.order_close(beta,b,t,max(k+2,4));

   quadratic_number_standard mu;
   bool new_unit;

   I.inverse_rho(mu);
   if (I == 0)
     {
       b += mu.absolute_Ln_approximation(k+2);
       new_unit = true;
     }
   else
     {
       I.rho();
       if (I == 0)
         { new_unit = true; }
       else
         {
           I.rho(mu);
           if (I == 0)
              {
                 b += mu.absolute_Ln_approximation(k+2);
                 new_unit = true;
              }
           else
              new_unit = false;
         }
     }

   if (new_unit)
     b.truncate(b.exponent+k+1);

   return new_unit;
}
```

To prove the correctness of the procedure, we need two auxiliary results.

LEMMA 10.2.1. *Let $m, p \in \mathbb{N}$, and $I = O/\alpha$ with $\alpha \in Min(O)$. If $p \not| m$ and $|mR/p - \operatorname{Ln}\alpha| < R/p$, then $I \neq O$.*

PROOF. Let $q, r \in \mathbb{Z}$ with $m = qp + r$ and $-p/2 < r \le p/2$. Then $|m/p - q| = |r|/p \ge 1/p$. Because q is a nearest integer approximation to m/p, we obtain

$$|(m/p)R - zR| \ge R/p, \ \forall z \in \mathbb{Z}. \tag{10.2.1}$$

If $I = O$, then $\operatorname{Ln}\alpha = zR$ for some $z \in \mathbb{Z}$. This implies $|mR/p - zR| = |mR/p - \operatorname{Ln}\alpha| < R/p$. But this is a contradiction to (10.2.1). Hence, $I \neq O$. □

LEMMA 10.2.2. *Let $m, p \in \mathbb{N}$, and $t \in \mathbb{R}$ with $|mR/p - t| < 2^{-3}$. Furthermore, let $I = O/\alpha$ with $\alpha \in Min(O)$ and α close to t with regard to 3.*

1. *If $p|m$, then $\rho^i(I) = O$ for some $-1 \le i \le 1$. If, in addition, $R > \ln\Delta$, then there is only one integer i with $-1 \le i \le 1$ and $\rho^i(I) = O$; furthermore, $\operatorname{Ln}\rho_I^i(\alpha) = mR/p$.*
2. *If $p \not| m$ and $R/p \ge (1 + 3\ln\Delta)/4$, then $\rho^i(I) \neq O$ for every $-1 \le i \le 1$.*

PROOF. We prove the first assertion. Suppose that $p|m$. Let β be the m/p-th power of the fundamental unit of O. Then β is a minimum of O with $\operatorname{Ln}\beta = mR/p$. Lemma 6.1.2 implies that there exists an integer i with $-1 \le i \le 1$ and $\rho^i(I) = O/\beta = O$. If $R > \ln\Delta$, it follows from Lemma 5.3.2, that $\rho^{i\pm j}(I) \neq O$ for $j = 1, 2$. Hence, there is only one integer i with $-1 \le i \le 1$ and $\rho^i(I) = O$. It is $\rho_I^i(\alpha) = \beta$, so $\operatorname{Ln}\rho_I^i(\alpha) = mR/p$.

We prove the second assertion. Suppose that $p \not| m$ and $R/p \ge (1 + 3\ln\Delta)/4$. Let $\alpha_i = \rho_I^i(\alpha)$ for $-1 \le i \le 1$. We have $|\operatorname{Ln}\alpha - \operatorname{Ln}\alpha_i| < 1/2\ln\Delta$ for $-1 \le i \le 1$ by Lemma 5.3.2. This implies, that $|(m/p)R - \operatorname{Ln}\alpha_i| < |(m/p)R - \operatorname{Ln}\alpha| + 1/2\ln\Delta \le |t - \operatorname{Ln}\alpha| + |(m/p)R - t| + 1/2\ln\Delta < (\ln\Delta)/4 + 2^{-3} + 2^{-3} + 1/2\ln\Delta = (1 + 3\ln\Delta)/4$ for $-1 \le i \le 1$. Hence,

$$|(m/p)R - \operatorname{Ln}\alpha_i| < R/p, \text{ for } -1 \le i \le 1.$$

It follows from Lemma 10.2.1, that $\rho^i(I) = O/\alpha_i \neq O$ for every $-1 \le i \le 1$. □

Now, we can prove the correctness of `is_divisor`.

PROPOSITION 10.2.3. *Let $m \in \mathbb{N}$, $a \in \mathbb{R}$ with $|a - mR| < 2^{-3}$, $p \in \mathbb{Z}_{\ge 2}$, and $k \in \mathbb{Z}$. If O.**is_divisor**(b, a, p, k) returns false, then p does not divide m. If it returns true, then b is an absolute k-approximation to qR for some $q \in \mathbb{Z}$, and if, in addition, $R/p \ge (1 + 3\ln\Delta)/4$, then $q = m/p$, i.e., p divides m.*

PROOF. First, we prove that $|t - (m/p)R| < 2^{-3}$. We have $|a - mR| < 2^{-3}$, $t = (1 + \epsilon)a/p$ with $|\epsilon| < 2^{-5-b(a)+b(p)}$, and $a \neq 0$. So, $|t - (m/p)R| \le |t - a/p| + |a/p - (m/p)R| < 2^{-5-b(a)+b(p)}|a|/p + 2^{-3}/p < 2^{-4} + 2^{-4} = 2^{-3}$.

By calling `order_close` the procedure computes $I = O/\beta$, $\beta \in Min(O)$, with β close to t with regard to 3. If the procedure returns false, then $\rho^i(I) \neq O$ for $-1 \le i \le 1$. It follows from Lemma 10.2.2, that p does not divide m.

If the procedure returns true, then it determines an absolute k approximation b to $\operatorname{Ln}\beta$, where β is a minimum in O such that $O/\beta = O = \rho^i(I)$, $-1 \le i \le 1$, i minimal with that property. Hence, $\operatorname{Ln}\beta = qR$ for some $q \in \mathbb{Z}$. If $R/p \ge (1 + 3\ln\Delta)/4$, then the second part of Lemma 10.2.2 implies, that $p|m$. Because $p \ge 2$, we also have $R > \ln\Delta$, and it follows from the first part of Lemma 10.2.2, that $\operatorname{Ln}\beta = (m/p)R$. □

10.3. Approximating the regulator from a regulator multiple

Let a be a floating point number with $|a - mR| < 2^{-3}$ for $m \in \mathbb{N}$. On input of a, an upper bound $M \geq m$, and an integer k, the following function O.`regulator_from_multiple`(a, M, v, k) determines an absolute k-approximation to R. If v is true, then the function verifies that $R/M \geq (1 + 3\ln\Delta)/4$. If v is false, the function assumes that this condition is valid and does not verify it. The function is a member function of the class `quadratic_order`. It uses the procedure `InitPrimes`(P, B), that initializes the prime list P with all primes less than B in increasing order from $P[0]$ to $P[P.length - 1]$.

```
xbigfloat O.regulator_from_multiple (

                xbigfloat a,
                xbigfloat M,
                bool verify_bound,
                int k)
 {
    xbigfloat r;

    if (verify_bound)
     {
       // Verify R >= M (1+3 ln Delta)/4
       //
       xbigfloat u = M/4 * (3 * (log(O.discriminant(),0)+1) + 1);

       if (O.regulator_if_lt(r,u,k))
          return r;
     }

    // Initialize prime array
    //
    prime_list P;
    P.init(M+1);

    // Find factors of m
    //
    bool r_changed = false;
    r = a;

    for (i=P.length - 1; i >= 0; i--)
     {
        while (O.is_divisor(a,r,P[i],max(k+1,3)))
         {
           r = a;
           r_changed = true;
         }
     }

    // Truncate or refine, if m = 1.
```

```
  //
  if (r_changed)
     r.truncate(r.exponent+k+1);
  else
     r = O.regulator_refinement(a,k);

  return r;
}
```

PROPOSITION 10.3.1. *Let a be a floating point number with $|a - mR| < 2^{-3}$ for $m \in \mathbb{N}$. On input of a, an upper bound $M \geq m$, a boolean v, and an integer k, the function O.*`regulator_from_multiple` (a, M, v, k) *returns an absolute k-approximation to R.*

PROOF. We use the notation of the procedure. If v is true, it determines $u \geq M(3\ln\Delta{+}1)/4$. If $R < u$, the assertion follows from Theorem 9.4.1. Otherwise, we have

$$R/q \geq (1 + 3\ln\Delta)/4, \tag{10.3.1}$$

for every prime q of the prime list. If v is false, it is guaranteed by the caller of the function, that this condition holds.

Let $m = \prod_p p^{e_p}$, $e_p \in \mathbb{Z}_{\geq 0}$ be the prime factorization of m. Because of (10.3.1), Lemma 10.2.3 implies that the call of `is_divisor` returns true, if and only if, q divides m.

If $m = 1$, then r is unchanged and we have $|r - R| < 2^{-3}$ after the loop. So the assertion follows from Lemma 10.1.1.

If $m > 1$, then r is changed in the loop. Let q be a prime that is less or equal to the largest prime in the factorization of m. Using induction on q, it follows from Lemma 10.2.3 and (10.3.1), that at the end of the loop iteration for prime q, it is

$$|r - nR| < 2^{-max(k+1,3)},\ n = m/\prod_{p\geq q} p^{e_p}.$$

When the loop is finished, we have $|r - R| < 2^{-k-1}$, and after truncation, r is an absolute k-approximation to the regulator R. $\square$

REMARK 10.3.2. An upper bound on m can be derived from a, if a lower bound B on the regulator is known. Let $|a - mR| < 2^{-3}$ and $B \leq R$. Furthermore, let $k \in \mathbb{Z}_{\geq 1}$ and

$$x = (1 + \epsilon)(a + 2^{-3})/B,\ |\epsilon| < 2^{-k}.$$

Then we obtain the upper bound

$$m < x/(1 - 2^{-k}).$$

CHAPTER 11

Deterministic regulator computation

In this chapter we present deterministic algorithms for approximating the regulator of a real quadratic order. Each algorithm computes an absolute 3-approximation to the regulator. The method of Section 10.1 can be used to compute an absolute k-approximation for $k > 3$, once an absolute 3-approximation is known.

11.1. R method

If the regulator of a quadratic order O is small, we can approximate the regulator by walking through the principal cycle step by step. This is known as the continued fraction method ([**Gau86**]).

We present the function O.`regulator_R`() that computes an absolute 3-approximation to the regulator using this method. The function uses the `rho` operator to walk through the principal cycle and collects all minima into a power product. At the end it computes an absolute 3-approximation to the logarithm of the power product. This is an approximation to the regulator.

```
xbigfloat O.regulator_R ()
 {
   quadratic_number_power_product alpha;
   quadratic_number_standard beta;
   quadratic_ideal A;

   A.assign(O);
   alpha.assign_one(O);

   do
   {
     A.rho(beta);
     alpha *= beta;
   }
   while (A != O);

   xbigfloat r = alpha.absolute_Ln_approximation(3);
   return r;
 }
```

Theorem 11.1.1. *The function O.`regulator_R`() returns an absolute 3-approximation to the regulator R of O. There is a constant $c \in \mathbb{N}$ such that the running time of the function is $O(R(\log \Delta)^c)$.*

PROOF. The function is correct. To prove the running time we note that the length k of the principal cycle satisfies

$$2R/\ln\Delta \leq k \leq 2R/\ln 2 \tag{11.1.1}$$

by [**BTW95**][Lemma 2.33]. It follows from Lemma 5.1.4 and Section 5.4.2 that the operations of the loop can be done in time polynomial in $\log\Delta$. Therefore, (11.1.1) implies that there is some constant $c_1 \in \mathbb{N}$ such that the time for the loop is $O(R(\log\Delta)^{c_1})$. Furthermore, it follows from (11.1.1) and Remark 4.2.3 that the time for the computation of r is $O(R(\log\Delta)^{c_2})$ for some constant $c_2 \in \mathbb{N}$. This proves the running time. $\square$

11.2. $R^{1/2}$ method

We present the function O.`regulator_sqrt_R`() that returns an absolute 3-approximation to the regulator R of O by an iterated Babystep-Giantstep method. This variant of Shanks' Babystep-Giantstep algorithm was suggested by Biehl and Buchmann (cf. [**BB94**]).

```
xbigfloat O.regulator_sqrt_R ()
 {
    xbigfloat u = 50;
    xbigfloat r;

    while (!O.regulator_if_lt(r,u,3))
     {
        u = 2*u;
     }
   return r;
 }
```

THEOREM 11.2.1. *The function O.`regulator_sqrt_R`() returns an absolute 3-approximation to the regulator R. There is a constant $c \in \mathbb{N}$ such that the running time of the function is $O(R^{1/2}(\log\Delta)^c)$.*

PROOF. Apply (1.5.3) and Theorem 9.4.1. $\square$

11.3. $\Delta^{1/4}$ method

The function O.`regulator_delta_1_over_4`() implements the method of Shanks [**Sha73**]. It computes an upper bound on the regulator and then uses the Babystep-Giantstep algorithm to compute the regulator.

```
xbigfloat O.regulator_delta_1_over_4 ()
 {
    xbigfloat d = sqrt(O.discriminant, 1);
    xbigfloat l = 1 + (log(O.discriminant, -1))/2;
    xbigfloat u = d*(l+1);

    xbigfloat r;
    O.regulator_if_lt(r,u,3);
    return r;
 }
```

THEOREM 11.3.1. *The function O.`regulator_delta_1_over_4`() returns an absolute 3-approximation to the regulator of O. There is a constant $c \in \mathbb{N}$ such that the running time of the function is $O(\Delta^{1/4}(\log \Delta)^c)$.*

PROOF. We use the notation of the procedure. We have $d = \sqrt{\Delta}(1+\varepsilon)$ with $|\varepsilon| < 1/2$. This implies

$$\sqrt{\Delta}/2 < d.$$

The approximation l satisfies $|l - (1 + (\ln \Delta)/2)| < 1$. So

$$1 + (\ln \Delta)/2 < l + 1.$$

It follows from (1.5.3) that $R < (1 + (\ln \Delta)/2)\sqrt{\Delta}/2$. This implies

$$R < d(l+1) = u.$$

It follows from Theorem 9.4.1 that O.`regulator_if_lt`$(u, 3)$ returns an absolute 3-approximation to the regulator. This proves the correctness of the function.

We prove the assertion on the running time. We have $u = O(\Delta^{1/2})$. Given u, the time for finding the regulator is $O(\sqrt{u}(\log \Delta)^{c_1})$ for some constant $c_1 \in \mathbb{N}$ by Theorem 9.4.1. Because u can be computed in polynomial time in $\log \Delta$, the overall running time is $O(\Delta^{1/4}(\log \Delta)^c)$ for some constant $c \in \mathbb{N}$. □

11.4. $\Delta^{1/5}$ method

In this section we describe the $\Delta^{1/5}$ algorithm for approximating the regulator. It is a variant of the Babystep-Giantstep algorithm, where the search interval is narrowed by use of the analytic class number formula (1.5.1). It was first described by Lenstra and Schoof (cf. [**Len82**], [**Sch82**], [**MW92**]).

The estimate of its running time $O(\Delta^{1/5}(\log \Delta)^c)$ for some constant $c \in \mathbb{N}$ assumes the ERH, but once a regulator approximation is found, it is unconditionally correct. Recently, Srinivasan [**Sri98**] presented a probabilistic algorithm with the same expected running time without assuming the ERH.

```
xbigfloat O.regulator_delta_1_over_5 ()
 {
   bigint Delta = O.discriminant;
   xbigfloat r;

   // |d - Delta^(2/5)| < 1
   //
   xbigfloat d = power(Delta,2,5, ceil(2 b(Delta)/5));
   xbigfloat B = d + log(Delta,0)/4 + 1.3175;

   // Test for regulator < B
   //
   if (O.regulator_if_lt(r,3,B))
    {
       return r;
    }

   // regulator >= B > Delta^(2/5), proceed.
   //
```

```
xbigfloat S = O.absolute_hR_approximation(d);

// Ideal I close to S with regard to 3.
//
quadratic_number_power_product alpha;
xbigfloat a;
quadratic_ideal I = O;
I.order_close(alpha,a,S,4);

xbigfloat b;

if (I == O)
   r = a;

// Search for approximation b to Ln beta
// with I = beta O, 0 <= Ln beta < B
//
else if (I.is_bounded_equivalent(b,O,B,4))
     r = a + b;

// Search for approximation b to Ln beta
// with O = beta I, 0 <= Ln beta < B
//
else if (O.is_bounded_equivalent(b,I,B,4))
    r = a - b;

// Not found. ERH is wrong.
//
else
 { r = 0; return r; }


// Test for r approximates 0 or r < 0. Then ERH wrong.
//
if (r == 0 || b(r) <= -3)
 { r = 0; return r; }

if (r < 0)
 { r = 0; return r; }

// Now |r-mR| < 2^(-3). If ERH correct, then m = h or m = h+1.
// Determine approximation to R by dividing out
// prime divisors of m. If ERH correct, then m approx. Delta^(1/10),
// because R >= Delta^(2/5).

// Determine M with m = mR/R < (r+1/8)/B < M.
//
divide(d, r+1/8, B, b(r+1/8)-b(B)+1);
M = d+1;
```

```
    // Verify R/M >= (1+3 ln Delta)/4.
    //
    d = (1 + 3 (log(Delta,0)+1)) / 4;

    if (B < M*d)
     if (O.regulator_if_lt(r,3,M*d))
      {
        return r;
      }

    // Approximate regulator by dividing out factors of m.
    //
    r = O.regulator_from_multiple(r, M, false, 3);
    return r;
  }
```

THEOREM 11.4.1. *If the function O.`regulator_delta_1_over_5`() returns* 0*, then the ERH is wrong. If it returns $r \neq 0$, then r is a positive, absolute* 3*-approximation to the regulator of O. Assuming the ERH, there is a constant $c \in \mathbb{N}$ such that the running time of the procedure is $O(\Delta^{1/5}(\log\Delta)^c)$.*

PROOF. We use the notation of the procedure. In a first step the function determines $d = \Delta^{2/5}(1+\varepsilon_1)$ with $|\varepsilon_1| < 2^{-\lceil 2b(\Delta)/5\rceil - 1}$. Hence

$$|d - \Delta^{2/5}| = \Delta^{2/5}|\varepsilon_1| < 1.$$

Set $B = d + \mathit{log}(\Delta, 0)/4 + 1.3175$. If the regulator is less than B, an absolute 3-approximation is determined by `regulator_if_lt`. We know from Section 12.1.2, that $R \geq 2^{-2}$, so r is positive. It follows from Theorem 9.4.1, that due to the choice of B, the running time for `regulator_if_lt` is $O(\Delta^{1/5}(\log\Delta)^{c_1})$ for some constant $c_1 \in \mathbb{N}$.

Otherwise, we have

$$R \geq B > \Delta^{2/5}. \tag{11.4.1}$$

Under the assumption of the ERH, the function O.`absolute_hR_approximation`(B) returns an approximation S with

$$|S - hR| < d.$$

Due to the choice of d, it follows from Theorem 8.6.3, that the running time for determining S is $O(\Delta^{1/5}(\log\Delta)^{c_2})$ for some constant $c_2 \in \mathbb{N}$.

Using `order_close`, the procedure determines a minimum α of O and $I = O/\alpha$, such that α is close to S with regard to 3. This implies that

$$\begin{aligned} |hR - \operatorname{Ln}\alpha| &\leq |hR - S| + |S - \operatorname{Ln}\alpha| \\ &< d + (\ln\Delta)/4 + 2^{-3} \\ &< d + 1 + (\ln\Delta - 1)/4 + 1/4 + 1/8 \\ &< B, \\ &\leq R, \end{aligned} \tag{11.4.2}$$

assuming the ERH. Also,

$$|a - \operatorname{Ln}\alpha| < 2^{-4}.$$

If $I = O$ or one of the `is_bounded_equivalent` returns true, then

$$|r - mR| < 2^{-3},$$

for some $m \in \mathbb{Z}$. Because $R \geq 2^{-2}$, we have $mR = 0$ iff $|r| < 2^{-3}$ iff $r = 0$ or $b(r) \leq -3$. This also implies, that $r < 0$ iff $mR < 0$ iff $m < 0$, in the case of $mR \neq 0$.

Assume that $m \geq 1$, i.e., $r > 0$ and $b(r) \geq -2$. Then the procedure determines $M \in \mathbb{Z}$ with

$$m = mR/R < (r + 2^{-3})/B < M.$$

If the regulator is less than $M(1 + 3 \ln \Delta)/4$, then an absolute 3-approximation is found by `regulator_if_lt`. Otherwise, we know that $R/M \geq (1 + 3 \ln \Delta)/4$. Hence, it follows from Theorem 10.3.1, that `regulator_from_multiple` determines an absolute 3-approximation to the regulator. In both cases, the same arguments as above show, that a positive, absolute 3-approximation to the regulator is returned.

Now, we prove, assuming the ERH, that we have $m \in \{h, h+1\}$ and we prove the running time of the procedure under this assumption.

Let η be the fundamental unit of O. If $I = O$, then it follows from (11.4.2), that $\alpha = \eta$, so $m = h$. For the following, assume that $I \neq O$. If $\operatorname{Ln} \alpha < hR$, then it follows from (11.4.2), that η^h/α is the minimal relative generator of I with respect to O, which is found by the call of I.`is_bounded_equivalent`$(b, O, B, 4)$, because $|\operatorname{Ln}(\eta^h/\alpha)| < B$. Hence, $m = h$. If $\operatorname{Ln} \alpha \geq hR$, it follows from (11.4.2), that η^{h+1}/α is the minimal relative generator of I with respect to O. If it is found by I.`is_bounded_equivalent`$(b, O, B, 4)$, then $m = h + 1$. Assume that it is not found. Then the minimal relative generator of O with respect to I, which is α/η^h, is found by O.`is_bounded_equivalent`$(b, I, B, 4)$, because $|\operatorname{Ln}(\alpha/\eta^h)| < B$. In this case, we have $m = h$ again. Hence, we have $m \in \{h, h+1\}$. Due to the choice of B, it follows from Theorem 9.3.1, that the running time for each test of bounded equivalence is $O(\Delta^{1/5}(\log \Delta)^{c_3})$ for some constant $c_3 \in \mathbb{N}$.

To prove the overall running time, it is sufficient to note, that $M = O(\Delta^{1/10}(\log \Delta)^{c_4})$ for some constant $c_4 \in \mathbb{N}$ assuming the ERH, and this is the running time of `regulator_from_multiple`. All other parts of the algorithm require time $O(\Delta^{1/5}(\log \Delta)^{c_5})$ for some constant $c_5 \in \mathbb{N}$. So, assuming the ERH, the overall running time is $O(\Delta^{1/5}(\log \Delta)^{c})$ for some constant $c \in \mathbb{N}$. $\square$

Some practical improvements have been implemented. The first improvement is the approximation of hR. As described, the function `absolute_hR_approximation` uses the error bound of Bach (see Chapter 8) to choose the number of terms in the sum for the approximation of $L(1, \chi_\Delta)$. This is necessary to guarantee a theoretical bound on the error of the approximation, but in practice, much less terms are necessary to achieve the same quality of approximations. And we can make use of that fact in the situation of the $\Delta^{1/5}$ algorithm, because, for the correctness of the algorithm, it is sufficient to find an approximation of an integer multiple of the regulator. The theoretical accuracy is only needed to prove the complexity result. Hence, based on experiments, we use much less terms for the approximation of $L(1, \chi_\Delta)$ to establish an approximation S of hR. Then we proceed as described. If one of the `is_bounded_equivalent` calls is successfull, we have found an integer multiple of the regulator. Otherwise, we approximate S again with the theoretical required number of terms, and search again. Note that in all our tests, the small number of terms was always sufficient, and this modification of the algorithm does not change its correctness.

The second improvement is the combination of the calls of `regulator_if_lt` and `is_bounded_equivalent` (see Chapter 9). Here, we use the same giant steps for both O.`regulator_if_lt` and O.`is_bounded_equivalent`. And we reuse the hash table for the babysteps for I.`is_bounded_equivalent`.

CHAPTER 12

Regulator computation in subexponential time

An algorithm for approximating the regulator whose running time is subexponential in $\log \Delta$ was suggested by Buchmann [**Buc90**]. In the version of Abel [**Abe94**] the running time of the algorithm is $L_\Delta[1/2, 5\sqrt{3}/6]$ assuming the ERH. The correctness of the algorithm also relies on the assumption of the ERH.

Our implementation of the subexponential method is joint work with Michael Jacobson [**Jac99**]. In this chapter we describe our main contributions. For a general introduction to the subexponential method we refer to [**Buc90**] and [**Coh95**].

12.1. Finding a generating unit

Let $u_1, \ldots, u_c \in O^*$ be units. The set generated by those units is defined as

$$< u_1, \ldots, u_c >= \{\pm \prod_{j=1}^{c} u_j^{e_j}, e_j \in \mathbb{Z}\}.$$

It is a subgroup of the unit group. In this section, we explain how to find a unit $z \in O^*$, such that

$$< z >=< u_1, \ldots, u_c > .$$

We sketch the algorithm. First we remove those units which are rational. Assume that c units remain. We proceed as follows: for $1 \leq i \leq \lfloor c/2 \rfloor$ we determine a unit u_i' with $< u_i' >=< u_{2i-1}, u_{2i} >$ by applying Proposition 12.1.1 below. If c is odd, we set $u'_{\lceil c/2 \rceil} = u_c$. Then the units $u_1', \ldots, u'_{\lceil c/2 \rceil}$ generate $< u_1, \ldots, u_c >$, and we have reduced the problem of finding a generating unit for c units to the problem of finding a generating unit for $\lceil c/2 \rceil$ units. We iterate this process until only one unit z remains. Then $< z >=< u_0, \ldots, u_{c-1} >$.

Proposition 12.1.1. *Let u_1 and u_2 be units of O with $\operatorname{Ln} u_j = N_j R$, $N_j \in \mathbb{Z}$ for $j = 1, 2$, where R is the regulator of O. There exist $x, y \in \mathbb{Z}$, such that*

$$x \operatorname{Ln} u_1 + y \operatorname{Ln} u_2 = gcd(N_1, N_2)R. \qquad (12.1.1)$$

And for all integers x, y which satisfy (12.1.1), we have $< u_1^x u_2^y >=< u_1, u_2 >$.

Proof. Set $M = gcd(N_1, N_2)$. Because $N_1\mathbb{Z} + N_2\mathbb{Z} = M\mathbb{Z}$, there exist integers x, y with $xN_1 + yN_2 = M$. Multiplying both sides with R yields (12.1.1).

Let x, y be integers satisfying (12.1.1). Set $z = u_1^x u_2^y$. We have $z \in< u_1, u_2 >$, so $< z >\subset< u_1, u_2 >$.

Furthermore, (12.1.1) implies $\operatorname{Ln}(z^{N_1/M}/u_1) = 0$. So, $z^{N_1/M}/u_1 \in kernel(\operatorname{Ln}) \cap O^* = \mathbb{Q} \cap O^* = \{\pm 1\}$. This shows that $u_1 \in< z >$ and the same arguments yield $u_2 \in< z >$. Hence, $< u_1, u_2 >\subset< z >$. ∎

We present an algorithm for finding integers x and y satisfying (12.1.1) in Section 12.1.1. That algorithm needs a lower bound on the regulator. We show in Section 12.1.2 how to determine such a bound. In Section 12.1.3 we describe how to decide whether a unit is rational or not. The complete algorithm for finding a generating unit is described in Section 12.1.4.

12.1.1. A real-gcd algorithm. We introduce the term real-gcd. Throughout this section let $R \in \mathbb{R} \setminus \{0\}$, and let $R_j = N_j R$ with $N_j \in \mathbb{Z} \setminus \{0\}$ for $j = 1, 2$. The real-gcd of R_1 and R_2 is defined as

$$rgcd(R_1, R_2) = gcd(N_1, N_2)R.$$

A representation of the real-gcd of R_1 and R_2 is a pair x, y of integers with

$$xR_1 + yR_2 = rgcd(R_1, R_2).$$

In this section, we present an algorithm for computing a representation of a real-gcd.

There exist two ideas for approximating real-gcds. In [**Coh95**] Cohen suggests to apply the euclidean integer gcd algorithm to approximate a real-gcd, but no details concerning the required accuracies are given. Approximating a real-gcd by application of the LLL algorithm has been independently described by Buchmann and Kessler (cf. [**BK92**]) and Ge (cf. [**Ge93**]). Its application is not limited to the one dimensional case, but can be applied to $\mathbb{R}^n$ with $n > 1$. Because of its generality, the running time of the LLL method is worse than that of our new approach. We present a new, fast method, that uses continued fraction expansions.

We explain the idea for finding a representation x, y of the real-gcd of R_1 and R_2. Set

$$M_1 = R_1/rgcd(R_1, R_2) \text{ and } M_2 = R_2/rgcd(R_1, R_2).$$

Assume that we know M_1 and M_2. We have $M_1, M_2 \in \mathbb{Z}$ and $gcd(M_1, M_2) = 1$. We use the extended euclidean algorithm to determine integers x and y with

$$1 = xM_1 + yM_2.$$

If we multiply both sides by $rgcd(R_1, R_2)$, we see that the pair x, y is a representation of the real-gcd of R_1 and R_2.

The main idea for finding M_1 and M_2 is as follows: We assume that we know an upper bound S on $|M_2|$. Then we approximate the real numbers R_1 and R_2 by rational numbers r_1 and r_2, respectively, such that

$$|R_1/R_2 - r_1/r_2| < 1/(2S^2). \tag{12.1.2}$$

Then the fraction $M_1/M_2 = R_1/R_2$ is a convergent in the continued fraction expansion of r_1/r_2 and the absolute value of the denominator of all following convergents is greater than S (see Lemma 12.1.3). This enables us to identify the convergent M_1/M_2 in the continued fraction expansion of r_1/r_2. (S, r_1, and r_2 satisfying (12.1.2) can be determined in several ways. We address this problem again in Remark 12.1.6 at the end of this section.)

We present a function that implements the described idea. On input of S, r_1, and r_2 satisfying (12.1.2), the function `rgcd_cfrac`(x, y, r_1, r_2, S) below returns a representation x, y of $rgcd(R_1, R_2)$. It makes use of a function `cfrac_convergent`(p, q, a, b, S), which, for $a, b, S \in \mathbb{Z}$, returns the numerator and denominator of the last convergent (p, q), $gcd(p, q) = 1$, in the continued fraction expansion of a/b with $|q| \leq S$. It also uses the function `xgcd`(x, y, a, b) that returns integers x, y with $gcd(a, b) = x\,a + y\,b$.

```
void rgcd_cfrac ( bigint & x,
                  bigint & y,
                  xbigfloat r_1,
                  xbigfloat r_2,
                  bigint S )
 {
   // Find the convergent.
   e = r_1.exponent() - r_2.exponent();
   if (e >= 0)
    cfrac_convergent(M_1,M_2, r_1.mantissa()*2^e,r_2.mantissa(),S);
   else
    cfrac_convergent(M_1,M_2, r_1.mantissa(),r_2t.mantissa()*2^(-e),S);

   // Determe the representation.
   xgcd (x,y,M1,M2);
 }
```

We prove some auxiliary results.

Lemma 12.1.2. *Let $M_1, M_2, A, B \in \mathbb{Z}$, $S \in \mathbb{R}_{>0}$. If $0 < |M_1/M_2 - A/B| < 1/(S\,|M_2|)$, then $|B| > S$.*

Proof. Let $\delta_1/\delta_2 = |M_1/M_2 - A/B|$, $\delta_1, \delta_2 \in \mathbb{Z}_{\geq 0}$ with $\gcd(\delta_1, \delta_2) = 1$. This implies $\delta_2 > S\,|M_2|$. We have

$$\left|\frac{A}{B}\right| = \left|\frac{M_1}{M_2} \pm \frac{\delta_1}{\delta_2}\right| = \left|\frac{M_1 \cdot \delta_2 \pm M_2 \cdot \delta_1}{M_2 \cdot \delta_2}\right| = \frac{\left|\frac{M_1 \cdot \delta_2 \pm M_2 \cdot \delta_1}{d}\right|}{\left|\frac{M_2 \cdot \delta_2}{d}\right|},$$

where $d = gcd\,(M_1 \cdot \delta_2 \pm M_2 \cdot \delta_1, M_2 \cdot \delta_2)$.

We show that $d \leq M_2^2$. Write $d = d_1 \cdot d_2$ with $d_1 | M_2$ and $\gcd(d_2, M_2/d_1) = 1$. Hence d_2 is a divisor of δ_2 and because $d_2 \,|\, d \,|\, (M_1 \cdot \delta_2 \pm M_2 \cdot \delta_1)$, we have $d_2 \,|\, (M_2 \cdot \delta_1)$. Because $d_2 \,|\, \delta_2$ and $gcd(\delta_1, \delta_2) = 1$, we obtain $gcd(d_2, \delta_1) = 1$, hence $d_2 \,|\, M_2$. This implies $d = d_1 \cdot d_2 \leq M_2^2$. As a consequence we obtain

$$|B| \geq \frac{|M_2| \cdot \delta_2}{d} \geq \frac{|M_2| \cdot \delta_2}{|M_2|^2} = \frac{\delta_2}{|M_2|} > S.$$

□

Lemma 12.1.3. *Let M_1, M_2 be rational integers with upper bound $S \geq |M_2|$ and $m_1, m_2 \in \mathbb{Q}$ with $|m_1/m_2 - M_1/M_2| < 1/(2S^2)$. Then M_1/M_2 is the last convergent in the continued fraction expansion of m_1/m_2, whose denominator is less or equal to S.*

Proof. Because $|m_1/m_2 - M_1/M_2| < 1/(2S^2) \leq 1/(2M_2^2)$, Theorem 1.2.2 yields, that M_1/M_2 is a convergent in the continued fraction expansion of m_1/m_2.

Let M_1/M_2 be the k-th convergent A_k/B_k, $A_k, B_k \in \mathbb{Z}$. Then we know from Theorem 1.2.3, that the next convergent A_{k+1}/B_{k+1}, if it exists, is closer to m_1/m_2. Hence

$$\left|\frac{M_1}{M_2} - \frac{A_{k+1}}{B_{k+1}}\right| < \left|\frac{M_1}{M_2} - \frac{m_1}{m_2}\right| + \left|\frac{A_{k+1}}{B_{k+1}} - \frac{m_1}{m_2}\right| < \frac{1}{S^2} < \frac{1}{S|M_2|}.$$

We apply Lemma 12.1.2 and obtain $|B_{k+1}| > S$. Because by Theorem 1.2.3, the denominators of the convergents increase monotonely, the absolute values of the denominators of all following convergents are also larger than S. □

LEMMA 12.1.4. *Let $0 \neq R_1, R_2 \in \mathbb{R}$, and $k \in \mathbb{Z}$. Let r_i be an approximation to R_i with $|r_i/R_i - 1| < 2^{-max\{b(R_1)-b(R_2)+4+k,1\}}$ for $i = 1, 2$, then $|r_1/r_2 - R_1/R_2| < 2^{-k}$.*

PROOF. We have

$$\begin{aligned}\left|\frac{R_1}{R_2} - \frac{r_1}{r_2}\right| &= \left|\frac{R_1}{R_2} - \frac{r_1}{R_2} + \frac{r_1}{R_2} - \frac{r_1}{r_2}\right| \\ &= \left|\frac{R_1 - r_1}{R_2} + \frac{r_1(r_2 - R_2)}{r_2 R_2}\right| \\ &\leq \left|\frac{R_1 - r_1}{R_1}\right| \cdot \left|\frac{R_1}{R_2}\right| + \left|\frac{r_2 - R_2}{R_2}\right| \cdot \left|\frac{r_1}{r_2}\right|.\end{aligned}$$

Lemma 2.2.2 implies $b(r_1) \leq b(R_1) + 1$ and $b(r_2) \geq b(R_2) - 1$, because $|r_i/R_i - 1| < 1/2$ for $i = 1, 2$. Hence,

$$\left|\frac{R_1}{R_2} - \frac{r_1}{r_2}\right| < \left|\frac{R_1 - r_1}{R_1}\right| \cdot 2^{b(R_1)-b(R_2)+1} + \left|\frac{r_2 - R_2}{R_2}\right| \cdot 2^{b(R_1)-b(R_2)+3} < 2^{-k}.$$

□

THEOREM 12.1.5. *Let $R \in \mathbb{R}$ with $|R| \geq 2^m$, $m \in \mathbb{Z}$. Also, let $M_1, M_2, M \in \mathbb{Z}$ with $R_i = M_i M R \neq 0$, $i = 1, 2$, and $gcd(M_1, M_2) = 1$. Furthermore, let $S = 2^{b(R_2)-m+c}$ with $c \in \mathbb{N}_0$, and $k \geq b(R_1) + b(R_2) - 2m + 2c + 4$.*

On input of relative k-approximations r_i to R_i, $i = 1, 2$, `rgcd_cfrac`(x, y, r_1, r_2, S) returns $x, y \in \mathbb{Z}$, such that $xR_1 + yR_2 = rgcd(R_1, R_2)$ with $|x| \leq |M_2|$ and $|y| \leq |M_1|$ in time $O(k^2)$.

PROOF. We have $|M_2| 2^m \leq |M_2 M R| < 2^{b(R_2)}$, so $S = 2^{b(R_2)-m+c}$ satisfies

$$S \geq |M_2|.$$

Because $b(R_i) \geq m$, it is

$$k \geq b(R_1) + b(R_2) - 2m + 2c + 4 \geq max\{b(R_1) - b(R_2) + 2 \log_2 S + 4, 1\}.$$

It follows from Lemma 12.1.4, that $|r_1/r_2 - R_1/R_2| < 2^{-2 \log_2 S} = 1/(2S^2)$. Because $r_1/r_2 = M_1/M_2$ with $gcd(M_1, M_2) = 1$, it follows from Lemma 12.1.3, that M_1, M_2 is computed by `cfrac_convergent`. The call of `xgcd` returns $x, y \in \mathbb{Z}$ with $xM_1 + yM_2 = 1$, and $|x| \leq |M_2|$, $|y| \leq |M_1|$. So, $xR_1 + yR_2 = MR$.

We analyze the bit complexity. Let $e(r_i)$ be the exponent of r_i for $i = 1, 2$. Because $k \geq 1$, Lemma 2.2.2 yields $|e(r_i) - b(R_i)| \leq 1$. Hence, $|e(r_1) - e(r_2)| \leq |b(R_1) - b(R_2)| + 2 \leq k$. Using the result of Theorem 1.2.1, we find that the continued fraction expansion can be computed in time $O((k + |e(r_1) - e(r_2)|)^2) = O(k^2)$.

For $i = 1, 2$, we have $b(M_i) \leq b(R_i) - m$. We know from Section 1.1 that computing the extended gcd of M_1 and M_2 takes time $O((max\{b(R_1) - m, b(R_2) - m\})^2) = O(k^2)$. □

REMARK 12.1.6. To satisfy the conditions of Theorem 12.1.5, we can compute relative 1-approximations r_i to R_i, $i = 1, 2$. It follows from Lemma 2.2.2, that $|b(r_i) - b(R_i)| \leq 1$. Hence,

$$S = 2^{b(r_2)-m+1},\ k = b(r_1) + b(r_2) - 2m + 10,$$

are suitable parameters. Once relative k-approximations r_i to R_i, $i = 1, 2$, are computed, it follows from Theorem 12.1.5, that a representation of the real gcd can be found in time $O((b(R_1) + b(R_2) - 2m)^2)$.

12.1.2. Lower bounds on the regulator. To be able to apply the real-gcd algorithm to determine a generating unit, we need to know a lower bound $2^m < R$ to the regulator R. We describe two possibilities for finding such a lower bound. The first variant only needs the discriminant Δ of the order and uses the following formula from [**JLW95**],

$$R \geq \ln((\sqrt{\Delta - 4} + \sqrt{\Delta})/2). \tag{12.1.3}$$

The function O.`lower_regulator_bound`() uses (12.1.3) to determine m.

```
bigint O.lower_regulator_bound()
 {
   bigint Delta = O.discriminant();

   if (Delta == 5)
     return -2;

   if (Delta == 8)
     return -1;

   if (Delta <= 29)
     return 0;

   // |\sqrt{D-4}| < \sqrt{D} < 2^{b(D)/2 + 1}
   //
   xbigfloat d;
   sqrt(d, Delta-4, b(Delta)/2 + 2);
   d -= 0.5;

   // l >= 2^{b(l)-1}
   //
   xbigfloat l;
   log(l, d, 1);
   l -= 0.5;

   return (b(l)-1);
 }
```

THEOREM 12.1.7. *Let O be a real quadratic order. The function O.`lower_regulator_bound`() returns an integer m, such that $2^m < R$, where R is the regulator of O.*

PROOF. It follows, for example from [**Coh95**][appendix B.2], that m is correct for $\Delta \leq 29$. Assume that $\Delta > 29$. (12.1.3) can be simplified to

$$R > \ln(\sqrt{D - 4}).$$

We have $d = \sqrt{\Delta - 4}(1 + \varepsilon_1) - 1/2$, $|\varepsilon_1| < 2^{-b(\sqrt{\Delta})-1}$. So $1 < d < \sqrt{\Delta - 4}$. Furthermore, $l < \ln d < R$. So $2^{b(l)-1} \leq l$ justifies the choice of $m = b(l) - 1$. □

In [**Lag80**] Lagarias presents an infinite set of non square free quadratic discriminants Δ, such that the regulators of the corresponding quadratic orders are larger than $c\sqrt{\Delta}/\log\Delta$ for some constant c. For those orders (12.1.3) is a bad lower regulator bound.

In the situation of the subexponential algorithm, we know a multiple $h_1 = nh$ of the class number h. We can use the analytic class number formula (1.5.1) to derive a lower bound on the regulator. This is done in the following function O.`lower_regulator_bound`(h_1), that returns m with $2^m < R < n2^{m+4}$.

```
bigint O.lower_regulator_bound( bigint h1 )
 {
   bigint    N = O.number_of_terms (0.25);
   xbigfloat l = O.Ell(N, 8);
   xbigfloat L = exp(l, 9);

   xbigfloat d = sqrt(Delta,4) * L;
   xbigfloat r = divide (d, 2*h1, 4);

   return(b(r)-2);
 }
```

THEOREM 12.1.8. *Let O be a real quadratic order. On input of a multiple $h_1 = nh$, $n \in \mathbb{N}$ of the class number h of O, the function O.`lower_regulator_bound`(h_1) returns an integer m with $2^m < R < n2^{m+4}$, where R is the regulator of O.*

PROOF. We use the notation of the procedure. First we analyze how accurate L approximates $L(1,\chi_\Delta)$. It follows from Theorem 8.1.1 that

$$|\ln(L(1,\chi_\Delta)) - l| < 2^{-2} + 2^{-8}. \tag{12.1.4}$$

Set

$$x = \frac{\exp l}{L(1,\chi_\Delta)} - 1.$$

Using (12.1.4) it is easy to verify that $|x| < 1$. Hence, Lemma 2.4.3 and (12.1.4) imply

$$|x| < \frac{|\ln(1+x)|}{1 - |\ln(1+x)|} < \frac{2^{-2}+2^{-8}}{1-2^{-2}-2^{-8}} = 65/191.$$

Therefore, we can write

$$\exp l = L(1,\chi_\Delta)(1+\varepsilon_1), \quad |\varepsilon_1| < 65/191. \tag{12.1.5}$$

Next, we can write

$$L = \exp l(1+\varepsilon_2), \quad |\varepsilon_2| < 2^{-9}.$$

Together with (12.1.5) we obtain

$$\begin{aligned} L &= L(1,\chi_\Delta)(1+\varepsilon_1)(1+\varepsilon_2) \\ &= L(1,\chi_\Delta)(1+\varepsilon_3), \quad |\varepsilon_3| < 1/2. \end{aligned}$$

Furthermore, $d = L(1,\chi_\Delta)\sqrt{\Delta}(1+\varepsilon_3)(1+\varepsilon_4)$, $|\varepsilon_4| < 2^{-4}$. Hence,

$$\begin{aligned} r &= L(1,\chi_\Delta)\sqrt{\Delta}/(2h_1)(1+\varepsilon_3)(1+\varepsilon_4)(1+\varepsilon_5), \quad |\varepsilon_5| < 2^{-4}, \\ &= R/n(1+\varepsilon_6), \quad |\varepsilon_6| < 1/2+1/4, \end{aligned}$$

using the analytic class number formula $R = L(1,\chi_\Delta)\sqrt{\Delta}/(2h)$. Therefore the regulator is bounded by

$$n2^{b(r)-2} < R < n2^{b(r)+2},$$

and for $m = b(r) - 2$ it is $2^m < R < n2^{m+4}$. $\square$

We explain the choice of the parameters. L is computed that way, because the same value is needed for verifying the class number and regulator; because the approximation of $L(1,\chi_\Delta)$ is the most expensive part in those computations, it is computed only once and reused in O.`verify_hR`(h,R) (see Section 12.2). This choice of L implies that with an exact computation of $\sqrt{\Delta}$ and an exact division by h_1, we already would obtain $n2^{b(r)-2} < R < n2^{b(r)+2}$, if we express the bounds in terms of n and powers of two. The square root and the quotient are approximated with a minimal accuracy, such that these inequations remain valid.

12.1.3. Testing for rational number. In this section we present an algorithm that decides whether a unit $u \in O^*$ is rational. On input of an integer m with $2^m \le R$, where R is the regulator of the order, the following function u.`is_rational`(m) returns true, if the unit u is rational and false otherwise. It is a member function of `quadratic_number_power_product`.

```
bool u.is_rational(bigint m)
 {
  // absolute -m+1 approx. to Ln u
  //
  xbigfloat l = u.absolute_Ln_approximation(-m+1);

  // u rational iff |l| < 2^{m-1}
  //
  if (l.is_zero())
    return true;
  else if (b(l) <= m-1)
    return true;
  else
    return false;
 }
```

THEOREM 12.1.9. *Let O be a real quadratic order and $u \in O^*$. On input of $m \in \mathbb{Z}$ with $2^m \le R$, where R is the regulator of O, the function u.`is_rational`(m) returns true if and only if u is rational.*

PROOF. Because u is a unit, we have $\operatorname{Ln} u \in \mathbb{Z}R$. Let $|l - \operatorname{Ln} u| < 2^{m-1}$. Then u is rational if and only if $\operatorname{Ln} u = 0$, and this is true if and only if $|l| < 2^{m-1}$. $\square$

12.1.4. The `generating_unit` algorithm and its bit complexity. Let $u_0, \dots, u_{c-1} \in O^*$. In this section we present an algorithm that computes a unit z such that $< z > = < u_0, \dots, u_{c-1} >$.

We assume that the units are given as

$$u_j = \prod_{i=0}^{r-1} \alpha_i^{f_{i,j}},\ 0 \le j < c,$$

with the $r \times c$ integer matrix $(f_{i,j})$, where $\alpha_i = \prod_{k=0}^{n_j} \gamma_{i,k}$, $0 \le i < r$, with $\gamma_{i,k} \in K$ in standard representation. We represent the units by the vector of vectors $\gamma = ((\gamma_{0,0}, \dots, \gamma_{0,n_0}), \dots, (\gamma_{r-1,0}, \dots, \gamma_{r-1,n_{r-1}}))$ and the matrix $f = (f_{i,j})$.

We present the function O.`generating_unit`(e, γ, f), that on input of γ and f returns the integer vector $e = (e_0, \dots, e_{r-1})$, such that

$$z = \prod_{i=0}^{r-1} \alpha_i^{e_i},$$

generates $< u_0, \dots, u_{c-1} >$.

We describe the principle of the function. First we remove those units which are rational as described in Section 12.1.3. Assume that c units remain. We proceed as follows: we compute $x_i, y_i \in \mathbb{Z}$ such that for $u_i' = u_{2i}^{x_i} u_{2i+1}^{y_i}$ we have

$$< u_i' >=< u_{2i}, u_{2i+1} >$$

for $0 \le i < \lfloor c/2 \rfloor$. This is done using the real-gcd algorithm of Section 12.1.1 (see Proposition 12.1.1). If c is odd, set $u'_{\lceil c/2 \rceil -1} = u_{c-1}$. Then the units $u'_0, \dots, u'_{\lceil c/2 \rceil}$ generate $< u_0, \dots, u_{c-1} >$, and we have reduced the problem of finding a generating unit for c units to the problem of finding a generating unit for $\lceil c/2 \rceil$ units. We iterate this process until only one unit z remains. Then $< z >=< u_0, \dots, u_{c-1} >$. The structure of that computation is a binary tree, where the leaves are the units $u_0, \dots, u_{c-1}$ and the root is z.

All units are represented as power products, where the base elements are $\alpha_0, \dots, \alpha_{r-1}$ and the exponents are given by the columns of the matrix f. The units are manipulated by changing the exponent matrix f. The exponent vector of the unit $u_{2i}^{x_i} u_{2i+1}^{y_i}$ is computed by multiplying the $2i$-th column of the matrix f by x_i and the $2i+1$-th column by y_i, respectively, and adding the columns. The resulting vector is stored in column $2i$, so replaces the exponent vector of u_{2i}. At the end of the process the exponent vector of z is the first column of f.

```
void O.generating_unit (

            vector<bigint> & e,
            vector< vector<quadratic_number_standard> > gamma,
            matrix<bigint> f )
 {
   // Initialize.
   bigint m = O.lower_regulator_bound();
   bigint r = f.no_of_rows();
   bigint c = f.no_of_columns();
   bigint i,j,k,n,t;

   // F[i] = max_{0 <= j < c} {|f_i,j|}, 0 <= i < r
   vector< bigint > F;
   for (i=0; i < r; i++)
     F[i] = f.max_abs_value_in_row(i);

   // |l_i - Ln alpha_i| < 2^{-(-m+1+b(r)+b(F[i]))}
   vector< xbigfloat > l;
   for (i=0; i < r; i++)
```

```
 {
   n = gamma[i].length();
   t = -m+1+b(r)+b(F[i])+b(n);
   l[i] = 0;
   for (k=0; k < n; k++)
     l[i] += gamma[i][k].absolute_Ln_approximation(t);
 }

// B = max_{0 <= j < c} {b_j}, |b_j - b(Ln u_j)| <= 1.
// Remove all columns that correspond to rational numbers.
B = m-1;
i = 0;
for (j=0; j < c; j++)
 {
   // |L - Ln u_j| < 2^{m-1}
   L = 0;
   for (k=0; k < r; k++)
     L += f[k][j] * l[k];

   // Ln u_j rational iff |L| < 2^{m-1}
   if (L != 0 && b(L) > m-1)
    {
      f.swap_columns(i,j);
      i++;
      B = max(B, b(L));
    }
 }
B += 1;

c = i;
if (c == 0)
 {
   for (i=0; i < r; i++)
     e[i] = 0;
   return;
 }
else if (c == 1)
 {
   for (i=0; i < r; i++)
     e[i] = f[i][0];
   return;
 }

f.set_no_of_columns_to(c);

// |l_i - Ln alpha_i| < 2^{-(B-2m+12+b(r)+b(F_i)+(B-m+1)b(c))}
for (i=0; i < r; i++)
 {
   n = gamma[i].length();
```

```
      t = B-2m+12+b(r)+b(F[i])+(B-m+1) b(c)+b(n);
      l[i] = 0;
      for (k=0; k < n; k++)
        l[i] += gamma[i][k].absolute_Ln_approximation(t);
    }

   // Real gcd computations with binary tree.
   bigint step_size = 2;
   bigint neighbour = 1;
   bigint x, y;
   xbigfloat u,v;

   for (k=0; k < b(c); k++)
    {
      for (j=0; j+neighbour < c; j += step_size)
       {
         v = 0;
         for (i=0; i < r; i++)
            v += f[i][j] * l[i];
         v = truncate(v, 2B-2m+13);

         w = 0;
         for (i=0; i < r; i++)
            w += f[i][j+neighbour] * l[i];
         w = truncate(w, 2B-2m+13);

         S = 2^(b(w)+1-m);

         rgcd_cfrac(x,y, v,w, S);

         for (i=0; i < r; i++)
             f[i][j] = f[i][j] * x + f[i][j+neighbour] * y;
       }

      step_size *= 2;
      neighbour *= 2;
    }

  for (i=0; i < r; i++)
    e[i] = f[i][0];
 }
```

Theorem 12.1.10. *Let O be a real quadratic order and v be some constant $v \in \mathbb{R}_{>0}$. Let the units $u_0, \dots, u_{c-1}$ be given as*

$$u_j = \prod_{i=0}^{r-1} \alpha_i^{f_{i,j}}, \; 0 \le j < c,$$

with the $r \times c$ integer matrix $(f_{i,j})$, where $\alpha_i = \prod_{k=0}^{n_i} \gamma_{i,k}$, $0 \le i < r$, with $\gamma_{i,k}$ in standard representation.

On input of $\gamma = ((\gamma_{0,0}, \dots, \gamma_{0,n_0}), \dots, (\gamma_{r-1,0}, \dots, \gamma_{r-1,n_{r-1}}))$ *and* $f = (f_{i,j})$, *the function* $O.\mathtt{generating\_unit}(e, \gamma, f)$ *returns* $e = (e_0, \dots, e_{r-1})$, *such that*

$$z = \prod_{i=0}^{r-1} \alpha_i^{e_i},$$

generates $< u_1, \dots, u_{c-1} >$. *Furthermore, if* $size(\gamma_{i,k}) = O(\log \Delta)$ *and* r, c, $size(f_{i,j})$, $n_i = L_\Delta[1/2, v]$, *then the running time is* $L_\Delta[1/2, 3v]$ *and* $size(e_i) = L_\Delta[1/2, v]$.

Proof. First, we prove the correctness. We use the notation of the procedure. Let $F_i = max_{0 \le j < c}\{|f_{i,j}|\}$. As a first step, the function computes approximations l_i to $\operatorname{Ln} \alpha_i$ such that

$$|l_i - \operatorname{Ln} \alpha_i| < 2^{-1+m-b(r)-b(F_i)},\ 0 \le i < r. \tag{12.1.6}$$

In the following loop, let L_j denote the value of L in the j-th iteration. We have $L_j = \sum_{i=0}^{r-1} f_{i,j} l_i$ and it follows from (12.1.6), that

$$|\operatorname{Ln} u_j - L_j| < 2^{m-1},\ 0 \le j < c.$$

Because 2^m is a lower bound on the regulator, we know that u_j is a rational number, i.e., $\operatorname{Ln} u_j = 0$, if and only if, $|L_j| < 2^{m-1}$. If this is the case, the column j is removed from f. Otherwise, we have $|\operatorname{Ln} u_j| \ge 2^m$, and so $|L_j / \operatorname{Ln} u_j - 1| < 1/2$. Lemma 2.2.2 implies $|b(L_j) - b(\operatorname{Ln} u_j)| \le 1$. Hence, after incrementation, the value B satisfies

$$max_{0 \le j < c}\{b(\operatorname{Ln} u_j)\} \le B \le max_{0 \le j < c}\{b(\operatorname{Ln} u_j)\} + 2, \tag{12.1.7}$$

in the case, that there exists at least one non-rational unit.

The second computation of approximations l_i to $\operatorname{Ln} \alpha_i$ yields

$$|l_i - \operatorname{Ln} \alpha_i| < 2^{-(B-2m+12+b(r)+b(F_i)+(B-m+1)b(c))},\ 0 \le i < r. \tag{12.1.8}$$

We analyze the real-gcd computations. Let $f_{i,j}^{(-1)}$ denote the entries of the matrix f, before the start of the first outer loop iteration for k, and let $f_{i,j}^{(k)}$ denote the entries of f at the end of the k-th outer loop iteration, $0 \le k < b(c)$. We set

$$u_j^{(k)} = \prod_{i=0}^{r-1} \alpha_i^{f_{i,j}^{(k)}},$$

for $-1 \le k < b(c)$, $0 \le j < c$. By induction on k, we prove that

$$|f_{i,j}^{(k)}| < (2^{B-m+1})^{k+1} F_i,\ 0 \le i < r,\ 0 \le j < c, \tag{12.1.9}$$

and

$$< u_j^{(k)} > = < u_j, \dots, u_{min\{j+2^{k+1}-1, c-1\}} >, \tag{12.1.10}$$

for $0 \le j < c$, $j \equiv 0 \mod 2^{k+1}$, $-1 \le k < b(c)$.

The assertions are true for $k = -1$. Let $k \ge 0$ and assume that the assertions are true for all $-1 \le k' < k$. Fix an index $0 \le j < c$ with $j \equiv 0 \mod 2^{k+1}$.

If $j + 2^k \ge c$, then $f_{i,j}^{(k)} = f_{i,j}^{(k-1)}$ and $u_j^{(k)} = u_j^{(k-1)}$. Furthermore, $min\{j + 2^{k+1} - 1, c-1\} = c - 1 = min\{j + 2^k - 1, c - 1\}$, so the assertions follow from the induction hypothesis.

Suppose, that $j + 2^k < c$. It follows from (12.1.10), that $|\operatorname{Ln} u_j^{(k-1)}| \leq |\operatorname{Ln} u_j|$ and $|\operatorname{Ln} u_{j+2^k}^{(k-1)}| \leq |\operatorname{Ln} u_{j+2^k}|$. Together with (12.1.7), this implies

$$b(\operatorname{Ln} u_j^{(k-1)}), b(\operatorname{Ln} u_{j+2^k}^{(k-1)}) \leq B.$$

It follows from (12.1.8) and the induction hypothesis (12.1.9), that in iteration k of the outer loop and j of the inner loop, we have

$$\begin{aligned} |v/\operatorname{Ln} u_j^{(k-1)} - 1| &< 2^{-b(\operatorname{Ln} u_j^{(k-1)})-B+2m-10}, \\ |w/\operatorname{Ln} u_{j+2^k}^{(k-1)} - 1| &< 2^{-b(\operatorname{Ln} u_{j+2^k}^{(k-1)})-B+2m-10}, \end{aligned} \tag{12.1.11}$$

Here, we apply Lemma 2.3.2 to obtain the relative approximations by truncation.

So we may apply Theorem 12.1.5 and Remark 12.1.6, and obtain $rgcd(\operatorname{Ln} u_j^{(k-1)}, \operatorname{Ln} u_{j+2^k}^{(k-1)}) = x \operatorname{Ln} u_j^{(k-1)} + y \operatorname{Ln} u_{j+2^k}^{(k-1)}$ with $x, y \in \mathbb{Z}$ and

$$|x| \leq |\operatorname{Ln} u_{j+2^k}^{(k-1)}|/2^m,\ |y| \leq |\operatorname{Ln} u_j^{(k-1)}|/2^m. \tag{12.1.12}$$

This yields $|x|, |y| < 2^{B-m}$, which implies $|f_{i,j}^{(k)}| \leq |x||f_{i,j}^{(k-1)}| + |y||f_{i,j+2^k}^{(k-1)}| < 2^{B-m}|f_{i,j}^{(k-1)}| + 2^{B-m}|f_{i,j+2^k}^{(k-1)}| \leq (2^{B-m+1})^{k+1} F_i$ by induction hypothesis.

We have $u_j^{(k)} = (u_j^{(k-1)})^x (u_{j+2^k}^{(k-1)})^y$, and it follows from Proposition 12.1.1 and the induction hypothesis that

$$\begin{aligned} < u_j^{(k)} > &= < u_j^{(k-1)}, u_{j+2^k}^{(k-1)} > \\ &= < u_j, \ldots, u_{min\{j+2^k-1, c-1\}}, u_{j+2^k}, \ldots, u_{min\{j+2^k+2^k-1, c-1\}} > \\ &= < u_j, \ldots, u_{j+2^k-1}, u_{j+2^k}, \ldots, u_{min\{j+2^{k+1}-1, c-1\}} > . \end{aligned}$$

Hence, the assertions of the induction are proven. It follows from (12.1.10), that

$$< z > = < u_0^{(b(c)-1)} > = < u_0, \ldots, u_{c-1} > .$$

This proves the correctness of the procedure.

We analyze the bit complexity. Set $L[v] = L_\Delta[1/2, v]$. Suppose, that $size(\gamma_{i,k}) = O(\log \Delta)$, and $r, c, size(f_{i,j}), n_i = L[v]$.

We have $|\operatorname{Ln} u_j| \leq \sum_{i=0}^{r-1} |f_{i,j}||\operatorname{Ln} \alpha_i| \leq L[v] 2^{L[v]}(L[v] \log \Delta)$ by (4.1.8) for $0 \leq j < c$. Hence, (12.1.7) implies

$$B = L[v]. \tag{12.1.13}$$

The approximation of $\operatorname{Ln} \alpha_i$ by l_i is dominated by the second loop, because of higher accuracies. We analyze this part. The required accuracy satisfies $t = L[v]$. It follows from Proposition 4.1.4, that each $\operatorname{Ln} \gamma_{i,k}$ can be approximated in time $O(M(L[v]) \log L[v]) = L[v]$. Hence, approximating all $L[v]^2$ logarithms requires time $O(L_\Delta[1/2, 3v])$. Let $c_{i,k}$ denote the approximation of $\operatorname{Ln} \gamma_{i,k}$. It follows from Lemma 2.4.5, that

$$|b(c_{i,k})|, |b(m(c_{i,k}))| \leq b(1 + |b(\gamma_{i,k}/\sigma(\gamma_{i,k}))| + L[v] + 2,$$

thus $|b(c_{i,k})|, |b(m(c_{i,k}))| = L[v]$ by (4.1.4). Hence, it follows from Theorem 2.3.3, that the computation of one sum l_j requires time $L[2v]$. It also follows from Theorem 2.3.3, that

$$|b(l_i)|, |b(m(l_i))| = L[v],\ 0 \leq i < r. \tag{12.1.14}$$

So, computing all sums l_i to $\operatorname{Ln} \alpha_i$ takes time $L[3v]$.

We analyze the binary tree rgcd-computations. The inner loop for j is executed $L[v]$ times. We estimate the costs for one loop iteration, and we begin with the computations of v and w. It follows from (12.1.9), (12.1.13), and $m \geq -2$, that

$$size(f_{i,j}^{(k)}) = L[v],\ 0 \leq i < r,\ 0 \leq j < c,\ -1 \leq k < b(c). \tag{12.1.15}$$

Together with (12.1.14), this implies that each product can be computed in time $O(M(L[v])) = L[v]$, and the size of the mantissa and the exponent of the product are bounded by $L[v]$ too. Applying Theorem 2.3.3 again, shows that one v and one w can be computed in time $L[2v]$. Because v and w are relative $2B - 2m + 10$-approximations, it follows from Theorem 12.1.5, that the computation of x and y takes time $O((2B - 2m + 10)^2)$, i.e. time $L[2v]$. Also updating f takes time $2\,L[v]M(L[v]) = L[2v]$, so one iteration of the loop takes time $L[2v]$. Hence, the overall time for all outer and inner loop iterations is $L[3v]$.

Thus, the overall running time of the procedure is $L[3v]$. Furthermore, (12.1.9) implies

$$|e_i| = |f_{i,0}^{(b(c)-1)}| < (2^{B+3})^{b(c)} F_i,\ 0 \leq i < r, \tag{12.1.16}$$

for the exponent vector e of z. So, $size(e_i) = L[v]$. This completes the proof. □

12.1.5. Practical improvements. We describe changes in the `generating_unit` algorithm that allow a more efficient implementation. We briefly recall the situation. We have units $u_0, \ldots, u_{c-1}$ with

$$u_j = \prod_{i=0}^{r-1} \alpha_i^{f_{i,j}},\ 0 \leq j < c,$$

with the $r \times c$ integer matrix $(f_{i,j})$, where $\alpha_i = \prod_{k=0}^{n_i} \gamma_{i,k}$, $0 \leq i < r$, with $\gamma_{i,k} \in K$ in standard representation. We use the function O.`generating_unit` of Section 12.1.4 to compute a generating unit.

The first change in the implementation of the algorithm is the fact, that the numbers α_i are not stored as power products. Instead the power products are small enough (see [**Jac99**]), such that they can be evaluated. So, the α_i's are stored as `quadratic_number_standard`, and the units u_j are of type `quadratic_number_power_product`.

The most expensive part of the `generating_unit` algorithm is the approximation of the logarithms of the base elements α_i. Therefore, we minimize the number of those computations, and we try to use smaller accuracies than required by theory to accelerate the computation of the approximations. This is achieved in several steps.

First note that the $\operatorname{Ln}\alpha_i$ are approximated twice in `generating_unit`. The purpose of the first approximation is to find numbers L_j with

$$|\operatorname{Ln} u_j - L_j| < 2^{m-1}, \tag{12.1.17}$$

where $2^m \leq R$ is a lower bound on the regulator. Those approximations are used to decide whether or not u_j is rational, and to find a bound B with

$$max_{0 \leq j < c}\{b(\operatorname{Ln} u_j)\} \leq B.$$

(Note that the upper bound derived on B in the previous section is only necessary to prove the complexity result.) After removing the rationals, B is used to determine the accuracies for the second approximation of $\operatorname{Ln}\alpha_i$.

We reverse this process. First, we determine a bound B, and then compute approximations that are sufficient for finding L_j satisfying (12.1.17), and which are assumed to be accurate enough for the rest of the computation too. To determine B, we begin with computing upper bounds

$$|\operatorname{Ln}\alpha_i| \le U_i,\ 0 \le i < r,$$

using the method for estimating Ln derived in Section 4.1.5. Note that this is much faster than approximating Ln by standard methods. Then we derive the upper bound B as

$$B = b(max_{0\le j<c}\{\sum_{i=0}^{r-1} |f_{i,j}|U_i\}).$$

The second change concerns the accuracies which are used to approximate the $\operatorname{Ln}\alpha_i$. In practice, we compute approximations l_i with

$$|l_i - \operatorname{Ln}\alpha_i| < 2^{-(B-2m+12+b(r)+b(F_i)+3)}, \tag{12.1.18}$$

for $0 \le i < r$. To explain that choice, we need the following Lemma. We use the notation of the proof of Theorem 12.1.10.

Lemma 12.1.11. *For each layer of the binary tree, the sum of the b-values of the unit and the exponent increases by a constant term only. More precisely, we have*

$$b(\operatorname{Ln} u_j^{(k)}) + b(f_{i,j}^{(k)}) < B + b(F_i) + 3(k+1),$$

for $-1 \le k < b(c)$, $0 \le i < r$, and $0 \le j < c$.

Proof. We use induction on k. For $k = -1$ the assertion follows from the choice of B and F_i. Let $k \ge 0$ and assume that the assertion is true for all $-1 \le k' < k$. Fix an index $0 \le j < c$ with $j \equiv 0 \mod 2^{k+1}$. If $j + 2^k \ge c$, then $f_{i,j}^{(k)} = f_{i,j}^{(k-1)}$ and $u_j^{(k)} = u_j^{(k-1)}$. So the assertion follows from the induction hypothesis.

Suppose, that $j + 2^k < c$. Let $\operatorname{Ln} u_j^{(k-1)} = M_1 R$ and $\operatorname{Ln} u_{j+2^k}^{(k-1)} = M_2 R$ with $M_1, M_2 \in \mathbb{Z}$. Theorem 12.1.5 yields

$$|x| \le |M_2| \le |\operatorname{Ln} u_{j+2^k}^{(k-1)}|/2^m,\ |y| \le |M_1| \le |\operatorname{Ln} u_j^{(k-1)}|/2^m.$$

Assume w.l.o.g. that $|M_1| \ge |M_2|$; otherwise use $\operatorname{Ln} u_{j+2^k}^{(k-1)}$ and M_2 in the following. Then $|\operatorname{Ln} u_j^{(k)} f_{i,j}^{(k)})| \le |\operatorname{Ln} u_j^{(k-1)}/M_1|(|f_{i,j}^{(k-1)}| + f_{i,j+2^k}^{(k-1)}|)|M_1| < 2^{B+b(F_i)+3k+1}$ by induction hypothesis. This implies

$$2^{b(\operatorname{Ln} u_j^{(k)})+b(f_{i,j}^{(k)})} \le 4|\operatorname{Ln} u_j^{(k)} f_{i,j}^{(k)})| < 2^{B+b(F_i)+3k+3},$$

so $b(\operatorname{Ln} u_j^{(k)}) + b(f_{i,j}^{(k)}) < B + b(F_i) + 3(k+1)$. This proves the assertion. □

(12.1.18) is based on the following assumption: there is a small constant c_0 such that

$$\operatorname{Ln} u_j^{(0)} \le c_0 \cdot R, \tag{12.1.19}$$

for $j \equiv 0 \mod 2$. This assumption is justified by practical experiments. Therefore, the logarithms of the units at the second layer of the binary tree are close to the regulator, and their b-values are almost the same, especially $b(\operatorname{Ln} u_j^{(0)}) \approx b(\operatorname{Ln} u_{j+2}^{(0)})$.

If we assume that they are equal, Lemma 12.1.11 yields $b(\operatorname{Ln} u_{j+2}^{(0)}) + b(f_{i,j}^{(0)}) < B + b(F_i) + 3$. So, to obtain the approximation v with

$$|v - \operatorname{Ln} u_j^{(0)}| < 2^{-b(\operatorname{Ln} u_{j+2}^{(0)})+2m-10},$$

as required by Theorem 12.1.5 for the next rgcd-computation, it suffices to approximate $\operatorname{Ln} \alpha_j$ as in (12.1.18). The same arguments apply for the approximation of $\operatorname{Ln} u_{j+2}^{(0)}$. And because of (12.1.19), we expect $x = 0$, $y = 1$ or vice versa for each subsequent rgcd-computation. Then the exponent matrix $f_{i,j}^{(k)}$ remains constant for $0 \le k < b(c)$, and the approximations (12.1.18) are also sufficient for $1 \le k < b(c)$.

Note, that (12.1.19) is not true for $\operatorname{Ln} u_{c-1}^{(0)}$, if c is odd, because in that case, $\operatorname{Ln} u_{c-1}^{(0)} = \operatorname{Ln} u_{c-1}^{(-1)}$. To achieve (12.1.19), we compute the rgcd of $\operatorname{Ln} u_{c-2}^{(-1)}$ and $\operatorname{Ln} u_{c-1}^{(-1)}$ too, and set $\operatorname{Ln} u_{c-1}^{(0)}$ to the result, if c is odd.

By using the heuristic arguments above to reduce the accuracies and not using the theoretically required accuracies, we cannot be sure, that the procedure really finds the generating unit. We circumvent this problem by computing the theoretically required accuracy before each rgcd-computation, and refine the approximations to $\operatorname{Ln} \alpha_i$, if necessary. The theoretically required accuracy is determined as described in Remark 12.1.6. Here, the relative 1-approximations are computed via absolute $-m+1$-approximations. If our assumption is fulfilled, no refinements are necessary to derive the theoretically accuracies. Experiments show, that this is the case in practice (see Tables 3,4 in the appendix).

The third change concerns the number of units that are used. Because of (12.1.19), only a few units are necessary to determine the regulator, instead of $c = L_\Delta[1/2, v]$ as theoretically required. In practice $c = 25$ is sufficient.

Finally, we used the lower bound 2^m to the regulator given by formula (12.1.3) in `generating_unit` to formulate the function independently from the subexponential algorithm. But before a generating unit must be determined during the subexponential algorithm, an integer multiple h_1 of the class number is known. So instead of only using (12.1.3), we also use the function O.`lower_regulator_bound`(h_1) of Section 12.1.2, that makes use of the analytic class number formula, to find a larger lower bound on the regulator. This reduces the required accuracies for the logarithms.

12.2. Verifying class number and regulator

Let Δ be a quadratic discriminant, O be the corresponding quadratic order of discriminant Δ, h' be an integer multiple of the class number h, and R' be an integer multiple of the regulator R, if $\Delta > 0$. In this section, we show how the analytic class number formula (1.5.1) can be used to decide whether h' is equal to h in case of $\Delta < 0$, and whether $h'R'$ equals hR in case of $\Delta > 0$.

For $\Delta < 0$ let w be the number of roots of unity in O, i.e. $w = 6$, if $\Delta = -3$, $w = 4$, if $\Delta = -4$, and $w = 2$, if $\Delta < -4$. We set

$$\kappa = \frac{2\pi}{w\sqrt{|\Delta|}}.$$

For $\Delta > 0$ we set

$$\kappa = \frac{2R}{\sqrt{\Delta}}.$$

The analytic class number formula (1.5.1) can be stated as follows

$$h\kappa = L(1, \chi_\Delta). \tag{12.2.1}$$

Set

$$\kappa' = \begin{cases} \kappa, & \Delta < 0, \\ 2R'/\sqrt{\Delta}, & \Delta > 0. \end{cases}$$

To decide whether h' equals h in case of $\Delta < 0$ or $h'R'$ equals hR in case of $\Delta > 0$, it is sufficient to decide whether

$$h'\kappa' \stackrel{?}{=} h\kappa.$$

We need some further notation. Let $x, y \in \mathbb{R}_{>0}$. We define the distance of x and y as

$$d(x, y) = \left| ln\left(\frac{x}{y}\right)\right|. \tag{12.2.2}$$

To decide whether $h'\kappa' = h\kappa$ we first compute n with $C(n) \leq 1/4$. Then we approximate

$$z = \sqrt{2}\exp(l(n, \Delta))$$

by H and $h'\kappa'$ by G such that the distance (12.2.2) of H from z and G from $h'\kappa'$ is less than $(\ln\sqrt{2} - 1/4)/2$. Then $G < H$ if and only if $h'\kappa' = h\kappa$, because z has the property, that $h\kappa < z < 2h\kappa$, and the distance (12.2.2) of z to $h\kappa$ and $2h\kappa$, respectively, is at least $\ln\sqrt{2} - 1/4$.

Let O be an imaginary quadratic order. We present the procedure O.`verify_h`(h'), which on input of an integer multiple h' of the class number h returns true if $h' = h$ and false otherwise. It uses the function O.`number_of_terms` (F) that returns an integer n such that $C(n) \leq F$. It also uses the function O.`Ell`(n, k) that returns an absolute k-approximation to $l(n, \Delta)$ for $k \geq 0$ (see Chapter 8). Furthermore, it uses the function `Pi`(k) that for $k \geq 1$ returns a relative k-approximation to π.

```
bool O.verify_h(bigint h')
 {
   int n, w;
   xbigfloat l, L, z, H, d, G;
   bigint Delta = O.discriminant();

   // Approximate sqrt(2) * exp(l(n,Delta))
   //
   n = O.number_of_terms(0.25);
   l = O.Ell(n, 8);
   L = exp(l, 9);
   z = sqrt(2, 10);
   H = truncate(z*L, 9);

   // w = number of roots of unity
   //
   if (Delta == -3)
      w = 6;
   else if (Delta == -4)
```

```
      w = 4;
   else
      w = 2;

   // Approximate h' * kappa
   //
   d = sqrt(-Delta, 11);
   G = divide(truncate(h', 12), (w/2) * d, 11);
   G = truncate(G * Pi(7), 9);

   // Check  h' * kappa < sqrt(2) * exp(l(n,Delta))
   //
   if (G < H)
    return true;
   else
    return false;
 }
```

Let O be a real quadratic order. We present the procedure O.`verify_hR`(h', r), which on input of an integer multiple h' of the class number h, and a relative 7-approximation r to an integer multiple R' of the regulator R returns true if $h' = h$ and $R' = R$ and false otherwise. The procedure uses the same functions `number_of_terms`(F) and `Ell`(n, k) as above.

```
bool O.verify_hR(bigint h', xbigfloat r)
 {
   int n;
   xbigfloat l, L, z, H, d, G;
   bigint Delta = O.discriminant();

   // Approximate sqrt(2) * exp(l(n,Delta))
   //
   n = O.number_of_terms(0.25);
   l = O.Ell(n, 8);
   L = exp(l, 9);
   z = sqrt(2, 10);
   H = truncate(z*L, 9);

   // Approximate h' * kappa'
   //
   d = sqrt(Delta, 11);
   G = divide(truncate(h', 12), d/2, 11);
   G = truncate(G * r, 9);

   // Check h' * kappa' < sqrt(2) * exp(l(n,Delta))
   //
   if (G < H)
    return true;
   else
    return false;
```

```
}
```

To prove the correctness of the functions we need some auxiliary results.

We can easily derive some simple properties of the distance function.

LEMMA 12.2.1. *Let* $x, y, \delta \in \mathbb{R}_{>0}$.
1. *If* $d(x,y) < \delta$, *then* $e^{-\delta} y < x < e^{\delta} y$.
2. *If* $sign(x-y)\, d(x,y) \geq \delta$, *then* $e^{\delta/2} y \leq e^{-\delta/2} x$.

PROOF. The first property is an immediate consequence of the definition as well as the second one if we note that $sign(x-y)\, d(x,y) = ln(x/y) \geq \delta$. □

The next lemma relates relative approximations and distances.

LEMMA 12.2.2. *Let* $x, y, \varepsilon \in \mathbb{R}$, $y \neq 0$. *If* $|x/y - 1| < \varepsilon < 1$, *then* $d(x,y) < max\{ln(1+\varepsilon), ln(1/(1-\varepsilon))\}$.

PROOF. Assume, that $x/y - 1 \geq 0$. Then $x/y < 1+\varepsilon$ and $d(x,y) = ln(x/y) \leq ln(1+\varepsilon)$. Now suppose that $x/y - 1 < 0$. In this case we have $y/x < 1/(1-\varepsilon)$ and $d(x,y) = ln(y/x) < ln(1/(1-\varepsilon))$. □

LEMMA 12.2.3. *Let* $z \in \mathbb{R}$ *and* $\delta \in \mathbb{R}_{>0}$, *such that*

$$\begin{aligned} sign(z - h\kappa) d(h\kappa, z) &\geq \delta, \\ sign(2h\kappa - z) d(2h\kappa, z) &\geq \delta. \end{aligned}$$

Let h' *and* κ' *be integer multiples of* h *and* κ, *respectively, and let* $H, G \in \mathbb{R}$ *with*

$$\begin{aligned} d(z, H) &< \delta/2, \\ d(h'\kappa', G) &< \delta/2. \end{aligned}$$

Then $h'\kappa' = h\kappa$ *if and only if* $G < H$.

PROOF. Assume, that $h' = h$ and $\kappa' = \kappa$. Lemma 12.2.1 yields $G < h\kappa e^{\delta/2} \leq ze^{-\delta/2} < H$. Now suppose, that $h'\kappa' = \nu h\kappa$, $\nu \geq 2$. Then Lemma 12.2.1 yields $H < ze^{\delta/2} \leq 2h\kappa e^{-\delta/2} \leq \nu h\kappa e^{-\delta/2} < G$. □

THEOREM 12.2.4. *(ERH) Let* h' *be an integer multiple of the class number* h *and* r *be a relative* 7*-approximation to an integer multiple* R' *of the regulator* R. *For imaginary quadratic orders,* $O.$**`verify_h`**(h') *returns true if and only if* $h' = h$, *and for real quadratic orders,* $O.$**`verify_hR`**(h', r) *returns true if and only if* $h' = h$ *and* $R' = R$.

PROOF. We use the notation of the procedures.

First we show that H is a relative 6-approximation to $z = \sqrt{2}\exp(l(n,\Delta))$. We have

$$|l(n,\Delta) - l| < 2^{-8}. \tag{12.2.3}$$

Set

$$x = \frac{\exp(l(n,\Delta))}{\exp l} - 1.$$

Using (12.2.3) it is easy to verify that $|x| < 1$. Hence, Lemma 2.4.3 and (12.2.3) imply

$$|x| < \frac{|\log(1+x)|}{1 - |\log(1+x)|} < 2^{-7}.$$

Therefore, we can write

$$\exp l = \exp(l(n,\Delta))(1+\varepsilon_1), \quad |\varepsilon_1| < 2^{-7}. \tag{12.2.4}$$

Next, we can write

$$H = \exp l(1+\varepsilon_2)\sqrt{2}(1+\varepsilon_3)(1+\varepsilon_4), \quad |\varepsilon_2| < 2^{-9}, |\varepsilon_3| < 2^{-10}, |\varepsilon_4| < 2^{-8}.$$

Together with (12.2.4) we obtain

$$\begin{aligned} H &= \sqrt{2}\exp(l(n,\Delta))(1+\varepsilon_1)(1+\varepsilon_2)(1+\varepsilon_3)(1+\varepsilon_4) \\ &= z(1+\varepsilon_5), \quad |\varepsilon_5| < 2^{-6}. \end{aligned}$$

Let $\kappa' = 2\pi/(w\sqrt{|\Delta|})$, if Δ is negative and $\kappa' = 2R'/\sqrt{\delta}$, if Δ is positive. We show that G is a relative 6-approximation to $h'\kappa'$.

Let $G = \mathit{divide}(\mathit{truncate}(h', 12), (w/2) * d, 11)$ in the procedure `verify_h`. We have

$$\begin{aligned} G &= h'w/(2\sqrt{|\Delta|})(1+\varepsilon_1)(1+\varepsilon_3)/(1+\varepsilon_2), \quad |\varepsilon_1| < 2^{-12}, |\varepsilon_2| < 2^{-11}, |\varepsilon_3| < 2^{-11} \\ &= h'w/(2\sqrt{|\Delta|})(1+\varepsilon_4), \quad |\varepsilon_4| < 2^{-9}. \end{aligned}$$

Hence, for $G = \mathit{truncate}(G * \mathit{Pi}(7), 9)$ we obtain

$$\begin{aligned} G &= h'\kappa'(1+\varepsilon_4)(1+\varepsilon_5)(1+\varepsilon_6), \quad |\varepsilon_5| < 2^{-7}, |\varepsilon_6| < 2^{-8} \\ &= h'\kappa'(1+\varepsilon_7), \quad |\varepsilon_7| < 2^{-6}. \end{aligned}$$

If we replace $w/2$ by $1/2$ and $\mathit{Pi}(7)$ by r, the same analysis shows that G in procedure `verify_hR` also is a relative 6-approximation to $h'\kappa'$.

Now we apply Lemma 12.2.3 to prove the assertions. Because n has been chosen such that $C(n) \le 1/4$, Lemma 8.1.1 and (12.2.1) yield, for $\delta = \ln\sqrt{2} - C(n)$, that

$$\begin{aligned} \mathit{sign}(z - h\kappa)d(h\kappa, z) &\geq \delta > 0, \\ \mathit{sign}(2h\kappa - z)d(2h\kappa, z) &\geq \delta > 0. \end{aligned} \tag{12.2.5}$$

Because H is a relative 6-approximation to z and G a relative 6-approximation to $h'\kappa'$, Lemma 12.2.2 yields

$$d(z, H), d(h'\kappa', G) < \delta/2.$$

Lemma 12.2.3 implies that $h'\kappa' = h\kappa$ if and only if $G < H$. Because h' is an integer multiple of h and R' is an integer multiple of R, we have $h'\kappa' = h\kappa$ if and only if $h' = h$ for $\delta < 0$ and $h'\kappa' = h\kappa$ if and only if $h' = h$ and $R' = R$ for $\delta > 0$. □

We justify the condition $C(n) \le 1/4$ and the choice of $z = \sqrt{2}\exp(l(n,\Delta))$. Suppose that $z = \mu\exp(l(n,\Delta))$ for some n. It follows from (12.2.5) that z must be chosen such that there is a $\delta > 0$ with

$$\begin{aligned} \mathit{sign}(z - h\kappa)d(h\kappa, z) \geq \delta, \\ \mathit{sign}(2h\kappa - z)d(2h\kappa, z) \geq \delta. \end{aligned}$$

Lemma 8.1.1 and (12.2.1) imply $\mathit{sign}(z - h\kappa)d(h\kappa, z) \geq \ln\mu - C(n)$ and $\mathit{sign}(2h\kappa - z)d(2h\kappa, z) \geq \ln(2/\mu) - C(n)$. Hence, the optimal choice is $\mu = \sqrt{2}$. And $k = 2$ is minimal such that $\delta = \ln\sqrt{2} - 2^{-k} > 0$.

APPENDIX A

Timings and statistical data

We present timings and statistical data for algorithms described in this thesis. The running times have been measured on a 296 MHz SUN UltraSparc-II with 512 MB RAM.

A.1. Regulators with subexponential algorithm

We present timings and statistical data for the approximation of the regulator using the subexponential algorithm (see Chapter 12). All given digits of the regulators are correct (assuming the ERH). For sake of completeness, we give the class number and the structure of the class group too. As mentioned before, these results have been computed using an implementation which is joint work with Michael Jacobson [**Jac99**].

We have chosen two types of discriminants: $\Delta = 4(10^x + 3)$ and $\Delta = 10^x + 1$, x odd, with $10 \le x \le 66$.

Tables 1 and 2 contain the regulator, the class number and the structure of the class group. The class group is presented as $[m_1 m_2 \dots m_s]$, where the m_i are the elementary divisors. Tables 3 and 4 contain some statistics acquired during the computation. We have units $u_0, \dots, u_{c-1}$ with

$$u_j = \prod_{i=0}^{r-1} \alpha_i^{f_{i,j}},\ 0 \le j < c,$$

with the $r \times c$ integer matrix $(f_{i,j})$, where α_i are in standard representation. In the second column we give c/rat, where rat is the number of rational units. The third column contains r, the number of base elements. In the fourth column we give b_{max}/b_{avg}, where b_{max} is the number of bits of the maximal entry of the matrix $(f_{i,j})$, and b_{avg} the average over all non-zero entries. Furthermore, "est. acc." is the estimated absolute accuracy, that is used to determine the approximations l_i, i.e.,

$$|l_i - \operatorname{Ln} \alpha_i| < 2^{-(B-2m+12+b(r)+b(F_i)+3)},\ 0 \le i < r,$$

We give the largest estimated accuracy, and the average on the estimated accuracies. Column "req. acc." is the average of the difference of the estimated accuracy and the required accuracy, where the required accuracy has been computed according to Remark 12.1.6. Note that in every case, the required accuracy has been smaller than the estimated accuracy, so no recomputation was necessary. Finally, we give m and $b(R)$, where $2^m \le R$ is the lower bound on the regulator, that is determined at the beginning of the computation. Note that in all computations, the logarithm of the generating unit has been the regulator.

In Tables 5 and 6 we present the running times for those computations. "B" is the time for computing the upper bound B with

$$max_{0\leq j<c}\{b(\operatorname{Ln} u_j)\} \leq B,$$

and "l_i" is the time for determining the approximations l_i to $\operatorname{Ln}\alpha_i$. The time for removing the rationals from the units is given by "rat", and "L_j" is the time for approximating all units $\operatorname{Ln} u_j^{(k)}$, i.e., computing the linear combinations using the precomputed l_i's in depth k of the binary tree. Similarly, "cfrac" is the time for computing all continued fraction expansions, and "$f_{i,j}$" denotes the time for updating the exponents of the units after the real-gcd computations. Finally, "total" gives the overall run-time for finding the generating unit.

TABLE 1. Regulators for $\Delta = 4(10^x + 3)$.

x	R_Δ	h_Δ	Cl_Δ
10	53775.001969	2	[2]
11	84547.762021	5	[5]
12	84349.856943	24	[2 12]
13	203526.135160	16	[2 2 4]
14	740796.623628	16	[2 2 4]
15	24831357.959768	2	[2]
16	61016404.402980	2	[2]
17	396110178.76241	1	[1]
18	340282870.605283	5	[5]
19	33252126.42757	96	[2 2 2 12]
20	63383850.349644	224	[2 112]
21	4819697885.230607	8	[8]
22	495890769.267202	288	[2 2 6 12]
23	18709457902.995567	20	[2 10]
24	26084523859.129802	72	[2 2 18]
25	138284636780.527333	20	[2 10]
26	913413183322.746472	12	[2 6]
27	23713480365005.243777	2	[2]
28	377845390774.801539	336	[2 2 2 42]
29	92301190804382.194976	4	[2 2]
30	850448782136195.175169	2	[2]
31	24073389576854.979875	128	[2 4 4 4]
32	3537151221926935.535254	4	[2 2]
33	2814581184076163.735278	16	[2 2 2 2]
34	1826965195839367.118136	80	[2 40]
35	29243345042806926.348237	16	[2 8]
36	17877925357485391.079171	96	[2 2 24]
37	547768083567046937.092650	6	[6]
38	2623477252608642340.458110	4	[2 2]
39	40619345327358443695.129211	1	[1]

TABLE 1. Regulators for $\Delta = 4(10^x + 3)$ (continued)

x	R_Δ	h_Δ	Cl_Δ
40	10977261769104950698.597808	16	[2 2 4]
41	12189301684306228042.524785	32	[2 2 2 4]
42	125190442865962867802.162379	14	[14]
43	1405727345454365334397.757348	2	[2]
44	1807647714616989888619.710008	8	[2 2 2]
45	1268407016092463169810.216240	32	[2 2 2 4]
46	12725144883960588441298.862295	8	[2 2 2]
47	112516397150405250345958.892665	4	[2 2]
48	201547014864565003875247.753440	8	[2 2 2]
49	3464876984662566497614.078542	912	[2 2 2 114]
50	114547733277803595367769.458944	128	[2 2 2 2 8]
51	2688812353888755165626430.768210	16	[2 2 4]
52	14162272664880841826416262.59950	8	[2 4]
53	110628022019933014627321368.511250	4	[2 2]
54	35280071359556694157008697.747451	46	[46]
55	480781665843330888065733567.2971	8	[2 2 2]
56	13143612028502251426594740254.181230	1	[1]
57	9749997757037759186937113877.484878	4	[2 2]
58	14224257145496711597498162215.038301	8	[2 4]
59	13334479066623179554517304114.324658	30	[30]
60	3390554654214272811414345771.769040	552	[2 2 138]
61	5856792352107065590103502346.476626	384	[2 2 2 2 2 2 6]
62	44696865354307259924308428275.357485	32	[2 2 2 4]
63	4097672962674921402852586990915.812140	12	[2 6]
64	19541453353581366002926616746131.630931	8	[2 2 2]
65	47951423585568877326276779255512.465709	8	[2 2 2]
66	337049807681967700992857054753927.354132	6	[6]

TABLE 2. Regulators for $\Delta = 10^x + 1$,x odd.

x	R_Δ	h_Δ	Cl_Δ
11	62150.604837	4	[2 2]
13	1440291.67360	2	[2]
15	318685.981804	128	[2 2 2 4 4]
17	7457176.673332	36	[2 18]
19	728838566.710806	5	[5]
21	263529007.982452	96	[2 2 2 2 6]
23	3024714392.46445	72	[2 2 18]
25	547182052889.12278	8	[2 2 2]
27	89108763078.319304	384	[2 2 2 2 24]
29	89825403350606.317519	2	[2]
31	3307823296451706.336219	1	[1]
33	35573559512657.892991	768	[2 2 2 2 2 24]
35	16042489907871303.354251	16	[2 2 4]
37	66849033021006595.31538	48	[2 24]
39	2748262910616652.191476	9600	[2 2 2 2 2 2 150]
41	62646429291439957047.49834	4	[4]
43	94115191015848522 1162.591874	4	[2 2]
45	14124843760510730052.445071	2048	[2 2 2 2 2 2 2 2 24]
47	6702955058699141 7938495.88259	4	[2 2]
49	7153013659868030 09132682.148670	4	[2 2]
51	1602664131508976 22724723.22783	192	[2 2 2 2 2 6]
53	216281543395858433665504572.216885	1	[1]
55	26225871174283899217117744.506568	128	[2 2 2 2 2 2 2]
57	1829829004282273688531209157.995901	16	[2 2 2 2]
59	21538283082663054236956352546.970548	12	[2 6]
61	615740315660188904798394501743.618884	6	[6]
63	96211204670294287200610 0399.957940	30720	[2 2 2 2 2 2 2 4 60]

TABLE 3. Subexponential regulator algorithm. Statistics for $\Delta = 4(10^x + 3)$.

x	c/rat	r	$f_{i,j}$	$est.acc.$	$req.acc.$	$m/b(R)$
10	10/9	49	12/4	376/372	368	13/16
11	40/39	92	19/3	37/27	21	13/17
12	40/5	95	18/12	37/27	10	14/17
13	41/40	108	20/3	38/29	23	15/18
14	43/32	113	21/7	37/28	13	17/20
15	40/8	123	26/18	37/28	9	22/25
16	43/42	120	27/3	37/24	23	23/26
17	41/15	131	30/18	40/31	13	26/29
18	40/39	138	28/19	34/27	21	26/29
19	51/4	169	32/26	51/42	12	22/25
20	54/6	239	45/38	76/65	12	23/26
21	52/4	175	113/106	198/182	13	30/33
22	52/2	198	46/40	72/64	17	26/29
23	52/4	199	121/113	208/190	12	32/35
24	25/3	215	134/128	237/224	14	32/35
25	25/6	225	137/127	236/226	17	35/38
26	25/2	245	112/107	182/177	10	37/40
27	25/2	258	204/196	359/348	15	42/45
28	25/1	270	187/177	337/327	13	36/39
29	25/1	276	133/127	213/205	17	44/47
30	25/2	305	214/207	369/351	15	47/50
31	25/0	324	224/209	402/385	30	42/45
32	25/4	323	228/220	394/376	16	49/52
33	25/0	332	220/212	377/359	12	49/52
34	25/3	407	173/163	285/273	14	48/51
35	25/2	367	229/218	390/379	14	52/55
36	25/1	385	211/200	360/349	17	51/54
37	25/2	430	312/302	548/531	15	56/59
38	25/1	429	324/315	569/552	14	59/62
39	25/3	441	227/217	366/351	18	63/66
40	25/1	482	338/325	590/570	15	61/64
41	25/2	481	337/322	589/567	18	61/64
42	25/0	527	349/334	606/588	15	64/67
43	25/1	558	389/378	679/659	15	68/71
44	25/2	576	256/245	414/401	17	68/71
45	25/3	552	259/245	421/406	18	68/71
46	25/1	567	379/367	655/637	16	71/74
47	25/4	566	291/278	470/453	15	74/77
48	25/2	619	326/312	539/523	13	75/78
49	25/2	633	335/325	571/558	13	69/72
50	25/1	807	569/554	1028/1005	15	74/77
51	25/0	962	473/458	826/809	15	79/82
52	25/1	1012	527/510	931/914	17	81/84
53	25/1	1108	465/449	804/789	18	84/87
54	25/2	1220	588/573	1053/1035	15	82/85
55	25/2	1321	670/651	1210/1191	17	86/89
56	25/1	1359	695/679	1249/1232	18	91/94
57	25/6	1524	702/686	1264/1246	18	90/93
58	25/0	1615	1019/1007	1898/1876	21	91/94
59	25/2	1725	803/785	1468/1450	16	90/94

TABLE 3. Subexponential regulator algorithm. Statistics for $\Delta = 4(10^x + 3)$ (continued)

x	c/rat	r	$f_{i,j}$	$est.acc.$	$req.acc.$	$m/b(R)$
60	25/3	1782	850/829	1563/1541	16	89/92
61	25/3	2019	1009/992	1880/1864	21	90/93
62	25/2	2242	1036/1016	1924/1901	17	96/99
63	25/1	2440	1114/1093	2078/2055	19	98/102
64	25/0	2633	1215/1196	2271/2252	19	101/104
65	25/2	2834	1348/1329	2531/2512	16	103/106
66	25/2	3157	1351/1329	2534/2513	17	106/109

TABLE 4. Subexponential regulator algorithm. Statistics for $\Delta = 10^x + 1$,x odd.

x	c/rat	r	$f_{i,j}$	$est.acc.$	$req.acc.$	$m/b(R)$
11	40/39	85	18/2	35/26	19	12/16
13	40/39	101	22/2	34/24	20	18/21
15	41/40	130	18/4	36/28	23	16/19
17	42/2	143	30/25	52/43	10	20/23
19	51/2	155	35/28	47/37	13	27/30
21	52/2	181	51/46	85/77	11	25/28
23	51/1	190	58/53	90/82	13	29/32
25	25/3	238	78/72	114/106	12	37/39
27	25/2	267	153/145	271/258	13	34/37
29	25/1	268	155/147	260/250	19	43/47
31	25/2	360	146/138	228/218	11	49/52
33	25/2	330	215/205	380/367	15	43/46
35	25/0	349	220/209	372/358	12	51/54
37	25/1	399	178/167	285/269	12	53/56
39	25/2	492	249/236	437/422	15	49/52
41	25/5	471	231/218	374/358	20	63/66
43	25/4	480	293/276	488/470	14	67/70
45	25/1	571	288/271	493/472	17	61/64
47	25/1	608	284/270	460/444	15	73/76
49	25/3	610	327/315	540/527	22	77/80
51	25/4	1013	441/424	770/751	18	75/78
53	25/2	1111	525/512	920/907	16	85/88
55	25/1	1327	580/561	1038/1017	15	81/85
57	25/3	1466	716/701	1296/1278	16	88/91
59	25/0	1724	797/780	1451/1435	16	92/95
61	25/1	2035	852/838	1555/1541	20	96/99
63	25/1	2437	1114/1095	2102/2084	19	86/90

TABLE 5. Subexponential regulator algorithm. Running times for $\Delta = 4(10^x + 3)$.

x	B	l_i	rat	L_j	$cfrac$	$f_{i,j}$	$total$
10	0.02 s	0.28 s	0 s	0.02 s	0 s	0 s	0.32 s
11	0.02 s	0.04 s	0.02 s	0.02 s	0 s	0 s	0.1 s
12	0.03 s	0.03 s	0.03 s	0.1 s	0 s	0 s	0.19 s
13	0.02 s	0.07 s	0.03 s	0.1 s	0 s	0 s	0.22 s
14	0.03 s	0.04 s	0.04 s	0.06 s	0 s	0 s	0.17 s
15	0.04 s	0.03 s	0.04 s	0.15 s	0.01 s	0 s	0.27 s
16	0.05 s	0.06 s	0.02 s	0.15 s	0.01 s	0 s	0.29 s
17	0.04 s	0.07 s	0.05 s	0.17 s	0.01 s	0 s	0.34 s
18	0.05 s	0.06 s	0.07 s	0.17 s	0.01 s	0 s	0.36 s
19	0.08 s	0.1 s	0.08 s	0.35 s	0.01 s	0.02 s	0.64 s
20	0.11 s	0.11 s	0.1 s	0.35 s	0.01 s	0.02 s	0.7 s
21	0.09 s	0.32 s	0.12 s	0.47 s	0.03 s	0.01 s	1.04 s
22	0.08 s	0.13 s	0.11 s	0.49 s	0.01 s	0.03 s	0.85 s
23	0.09 s	0.38 s	0.14 s	0.5 s	0.04 s	0.04 s	1.19 s
24	0.06 s	0.63 s	0.07 s	0.3 s	0.01 s	0.01 s	1.08 s
25	0.06 s	0.49 s	0.07 s	0.27 s	0.01 s	0.02 s	0.92 s
26	0.06 s	0.34 s	0.09 s	0.35 s	0 s	0.03 s	0.87 s
27	0.07 s	1.15 s	0.1 s	0.41 s	0 s	0.06 s	1.79 s
28	0.09 s	0.93 s	0.09 s	0.41 s	0.02 s	0.02 s	1.56 s
29	0.07 s	0.52 s	0.1 s	0.38 s	0 s	0.04 s	1.11 s
30	0.08 s	1.41 s	0.11 s	0.47 s	0.01 s	0.07 s	2.15 s
31	0.1 s	1.96 s	0.16 s	0.67 s	0.04 s	0.08 s	3.01 s
32	0.09 s	1.69 s	0.12 s	0.51 s	0.01 s	0.06 s	2.48 s
33	0.11 s	2 s	0.15 s	0.68 s	0.02 s	0.09 s	3.05 s
34	0.1 s	1.41 s	0.12 s	0.52 s	0.02 s	0.07 s	2.24 s
35	0.11 s	2.5 s	0.18 s	0.72 s	0.02 s	0.08 s	3.61 s
36	0.1 s	1.98 s	0.16 s	0.71 s	0 s	0.07 s	3.02 s
37	0.13 s	3.76 s	0.2 s	0.87 s	0.03 s	0.12 s	5.11 s
38	0.13 s	4.69 s	0.24 s	0.99 s	0.01 s	0.12 s	6.18 s
39	0.13 s	2.19 s	0.17 s	0.72 s	0.03 s	0.06 s	3.3 s
40	0.14 s	5.77 s	0.27 s	1.11 s	0.09 s	0.14 s	7.52 s
41	0.15 s	6.08 s	0.28 s	1.15 s	0.06 s	0.17 s	7.89 s
42	0.15 s	5.65 s	0.28 s	1.2 s	0.02 s	0.19 s	7.49 s
43	0.17 s	7.06 s	0.31 s	1.26 s	0.06 s	0.19 s	9.05 s
44	0.16 s	3.93 s	0.24 s	1.09 s	0.03 s	0.12 s	5.57 s
45	0.16 s	4.85 s	0.28 s	1.22 s	0 s	0.18 s	6.69 s
46	0.18 s	6.73 s	0.34 s	1.48 s	0.01 s	0.2 s	8.94 s
47	0.18 s	4.51 s	0.26 s	1.23 s	0.01 s	0.15 s	6.34 s
48	0.19 s	6.13 s	0.34 s	1.48 s	0.06 s	0.18 s	8.38 s
49	0.19 s	7.55 s	0.38 s	1.67 s	0.02 s	0.22 s	10.03 s
50	0.29 s	30.29 s	0.84 s	3.68 s	0.08 s	0.61 s	35.79 s
51	0.33s	22.5s	0.82s	4.03s	0.09s	0.5s	28.27s
52	0.55s	29.79s	1.15s	4.43s	0.05s	0.62s	36.59s
53	0.4s	22.97s	0.85s	3.81s	0.07s	0.62s	28.72s
54	0.45s	45.07s	1.2s	5.83s	0.09s	1.02s	53.66s
55	0.47s	1.07m	1.5s	7.38s	0.11s	1.19s	1.24m
56	0.53s	1.24m	1.78s	8.31s	0.1s	1.43s	1.45m
57	0.48s	1.39m	1.48s	7.71s	0.1s	1.21s	1.58m
58	0.73s	3.71m	3.53s	17.76s	0.26s	3.15s	4.13m
59	0.71s	2.16m	2.52s	12.6s	0.12s	2.13s	2.47m

TABLE 5. Subexponential regulator algorithm. Running times for $\Delta = 4(10^x + 3)$ (continued).

x	B	l_i	rat	L_j	$cfrac$	$f_{i,j}$	$total$
60	$0.7s$	$2.63m$	$2.75s$	$13.82s$	$0.16s$	$2.15s$	$2.95m$
61	$0.89s$	$4.76m$	$4.13s$	$21.19s$	$0.21s$	$3.39s$	$5.26m$
62	$0.93s$	$5.28m$	$4.71s$	$24.29s$	$0.17s$	$4.32s$	$5.85m$
63	$1.09s$	$7.04m$	$5.76s$	$28.16s$	$0.24s$	$5.33s$	$7.71m$
64	$1.22s$	$9.44m$	$6.97s$	$39.02s$	$0.32s$	$6.45s$	$10.34m$
65	$1.42s$	$13.10m$	$8.12s$	$44.8s$	$0.37s$	$7.62s$	$14.14m$
66	$1.45s$	$14.33m$	$8.95s$	$49.25s$	$0.37s$	$8.48s$	$15.47m$

TABLE 6. Subexponential regulator algorithm. Running times for $\Delta = 10^x + 1$,x odd.

x	B	l_i	rat	L_j	$cfrac$	$f_{i,j}$	$total$
11	0.02 s	0.02 s	0.02 s	0 s	0 s	0 s	0.06 s
13	0.03 s	0.03 s	0.03 s	0 s	0 s	0 s	0.09 s
15	0.04 s	0.07 s	0.04 s	0 s	0 s	0 s	0.15 s
17	0.04 s	0.07 s	0.06 s	0.27 s	0.01 s	0 s	0.45 s
19	0.06 s	0.06 s	0.08 s	0.3 s	0.02 s	0.02 s	0.54 s
21	0.08 s	0.1 s	0.1 s	0.5 s	0 s	0.01 s	0.79 s
23	0.08 s	0.18 s	0.12 s	0.49 s	0.02 s	0.03 s	0.92 s
25	0.06 s	0.19 s	0.06 s	0.28 s	0.01 s	0.01 s	0.61 s
27	0.08 s	0.66 s	0.09 s	0.4 s	0.01 s	0.03 s	1.27 s
29	0.08 s	0.81 s	0.1 s	0.42 s	0.02 s	0.03 s	1.46 s
31	0.11 s	0.79 s	0.1 s	0.47 s	0.02 s	0.01 s	1.5 s
33	0.08 s	1.35 s	0.12 s	0.51 s	0.01 s	0.05 s	2.12 s
35	0.11 s	1.74 s	0.16 s	0.7 s	0 s	0.08 s	2.79 s
37	0.11 s	1.79 s	0.16 s	0.69 s	0.01 s	0.08 s	2.84 s
39	0.16 s	3.15 s	0.21 s	0.91 s	0.04 s	0.09 s	4.56 s
41	0.14 s	2.43 s	0.18 s	0.74 s	0.03 s	0.06 s	3.58 s
43	0.14 s	3.54 s	0.21 s	0.99 s	0.02 s	0.1 s	5 s
45	0.18 s	5.01 s	0.29 s	1.2 s	0.04 s	0.18 s	6.9 s
47	0.18 s	4.57 s	0.3 s	1.22 s	0.03 s	0.14 s	6.44 s
49	0.18 s	6.07 s	0.33 s	1.42 s	0.04 s	0.18 s	8.22 s
51	$0.32s$	$18.91s$	$0.63s$	$3.21s$	$0.07s$	$0.4s$	$23.54s$
53	$0.45s$	$31.97s$	$0.97s$	$4.57s$	$0.06s$	$0.7s$	$38.72s$
55	$0.46s$	$45.94s$	$1.34s$	$6.2s$	$0.11s$	$1.12s$	$55.17s$
57	$0.52s$	$1.43m$	$1.78s$	$9.06s$	$0.1s$	$1.52s$	$1.65m$
59	$0.68s$	$2.14m$	$2.7s$	$13.99s$	$0.13s$	$2.22s$	$2.47m$
61	$0.79s$	$2.94m$	$3.29s$	$16.27s$	$0.17s$	$2.63s$	$3.32m$
63	$1.11s$	$7.31m$	$5.79s$	$28.28s$	$0.26s$	$5.25s$	$7.98m$

A.2. Comparison of regulator algorithms and strategy

We present running times for computing an absolute 3-approximation to the regulator. We examine the deterministic algorithms described in Chapter 11 and the subexponential method of Chapter 12.

The timings are listed in Table 7. Each row of the table shows the running times in seconds for the computation of an absolute 3-approximation to the regulator for 1000 quadratic orders whose discriminants Δ have been chosen from the interval $10^x \leq \Delta < 10^{x+1}$ at random.

As a consequence of Table 7 we obtain the following strategy for approximating the regulator.

for discriminant Δ	use algorithm
$0 < \Delta < 10^4$	R
$10^4 \leq \Delta < 10^8$	$\Delta^{1/4}$
$10^8 \leq \Delta < 10^{12}$	$\Delta^{1/5}$
$10^{12} \leq \Delta$	subexponential

The table also shows that the $R^{1/2}$-algorithm is not relevant in practice.

Table 7. Absolute 3-approximation to regulator for 1000 discriminants between 10^x and 10^{x+1}.

x	R	$R^{1/2}$	$\Delta^{1/4}$	$\Delta^{1/5}$	*subexp*
3	8 s	12 s	10 s	33 s	–
4	24 s	31 s	18 s	44 s	–
5	62 s	50 s	34 s	62 s	–
6	202 s	137 s	62 s	85 s	–
7	706 s	279 s	121 s	132 s	–
8	–	445 s	205 s	175 s	–
9	–	804 s	394 s	264 s	–
10	–	1402 s	749 s	399 s	–
11	–	–	1476 s	640 s	819 s
12	–	–	–	988 s	872 s
13	–	–	–	1581 s	913 s

A.3. Approximation of $L(1, \chi_\Delta)$

We present running times for the approximation of $l(n, \Delta)$ to show that the acceleration obtained by the use of the IEEE-754 double precision floating point numbers (see Section 8.5) is significant.

Tables 8 and 9 show timings for the computation of an absolute 8-approximation to $l(n, \Delta)$, where n has been chosen such that $C(n) \leq 1/4$ and the discriminants are of the form $\Delta = 4(10^x + 3)$ and $\Delta = 10^x + 1$, x odd. Such approximations are computed in the subexponential algorithm (see the call of $O.\texttt{Ell}(n, 8)$ in the functions of Sections 12.1.2 and 12.2).

The tables show that the function from Section 8.5 that uses `double`s for the approximation of $l(n, \Delta)$ is approximately 100 times faster than the function from Section 8.2 which exclusively uses `xbigfloat`s.

TABLE 8. Absolute 8-approximation to $l(n, \Delta)$ for $\Delta = 4(10^x + 3)$.

x	n	`Ell` Section 8.2	`Ell` Section 8.5
10	7744	7 s	0.05 s
15	12544	11 s	0.07 s
20	20736	18 s	0.13 s
25	25600	22 s	0.16 s
30	36864	33 s	0.22 s
35	43264	36 s	0.26 s
40	57600	44 s	0.35 s
45	65536	52 s	0.39 s
50	82944	63 s	0.50 s
55	100715	83 s	0.60 s
60	102400	81 s	0.63 s
65	123904	62 s	0.76 s

TABLE 9. Absolute 8-approximation to $l(n, \Delta)$ for $\Delta = 10^x + 1$.

x	n	`Ell` Section 8.2	`Ell` Section 8.5
11	7744	7 s	0.04 s
15	12091	10 s	0.08 s
21	20736	18 s	0.13 s
25	25600	22 s	0.15 s
31	36864	31 s	0.21 s
35	43264	36 s	0.26 s
41	57600	49 s	0.35 s
45	65536	55 s	0.40 s
51	82944	66 s	0.51 s
55	96160	74 s	0.58 s
61	102400	81 s	0.62 s
65	123904	95 s	0.75 s

A.4. Fundamental units in compact representation

The algorithm of Section 7.2 computes a compact representation of the fundamental unit when an approximation to the regulator is given. We present running times of that algorithm and we list fundamental units in compact representation for selected discriminants. As in Appendix A.1 we choose the discriminants $\Delta = 4(10^x + 3)$ and $\Delta = 10^x + 1$, x odd.

Table 10. Fundamental unit computation with algorithm of Section 7.2

x	$\Delta = 4(10^x + 3)$
10	0.02 s
11	0.02 s
12	0.02 s
13	0.03 s
14	0.04 s
15	0.04 s
16	0.04 s
17	0.05 s
18	0.05 s
19	0.05 s
20	0.05 s
21	0.07 s
22	0.06 s
23	0.07 s
24	0.07 s
25	0.08 s
26	0.08 s
27	0.10 s
28	0.08 s
29	0.11 s
30	0.12 s
31	0.11 s
32	0.12 s
33	0.13 s
34	0.12 s
35	0.13 s
36	0.14 s
37	0.15 s
38	0.16 s
39	0.17 s

x	$\Delta = 10^x + 1, x$ odd
11	0.02 s
13	0.03 s
15	0.03 s
17	0.04 s
19	0.05 s
21	0.06 s
23	0.06 s
25	0.08 s
27	0.08 s
29	0.11 s
31	0.12 s
33	0.11 s
35	0.14 s
37	0.15 s
39	0.12 s
41	0.19 s
43	0.21 s
45	0.17 s
47	0.24 s
49	0.25 s
51	0.24 s
53	0.28 s
55	0.27 s
57	0.29 s
59	0.30 s
61	0.34 s
63	0.31 s

TABLE 10. Fundamental unit computation with algorithm of Section 7.2 (continued)

x	$\Delta = 4(10^x + 3)$
40	0.17 s
41	0.17 s
42	0.17 s
43	0.20 s
44	0.18 s
45	0.21 s
46	0.20 s
47	0.23 s
48	0.23 s
49	0.22 s
50	0.23 s
51	0.26 s
52	0.25 s
53	0.28 s
54	0.25 s
55	0.29 s
56	0.30 s
57	0.30 s
58	0.31 s
59	0.34 s
60	0.33 s
61	0.35 s
62	0.36 s
63	0.38 s
64	0.38 s
65	0.38 s
66	0.38 s

In the following we list fundamental units in compact representation for quadratic orders of discriminant Δ.

$\underline{\Delta = 4(10^{10}+3)}$

$$\left(\frac{200000+\sqrt{\Delta}}{6}\right)^{8192} \cdot \left(\frac{-200000+\sqrt{\Delta}}{2}\right)^{4096} \cdot \left(\frac{199984+\sqrt{\Delta}}{355542}\right)^{2048} \cdot \left(\frac{-137129767+686\sqrt{\Delta}}{5487}\right)^{1024}$$
$$\cdot \left(\frac{-4599760+73\sqrt{\Delta}}{39366}\right)^{512} \cdot \left(\frac{1479356+11\sqrt{\Delta}}{3422}\right)^{256} \cdot \left(\frac{-1647776+9\sqrt{\Delta}}{89638}\right)^{128} \cdot \left(\frac{5869959+65\sqrt{\Delta}}{66979}\right)^{64}$$
$$\cdot \left(\frac{32718888+197\sqrt{\Delta}}{53702}\right)^{32} \cdot \left(\frac{-34753586+201\sqrt{\Delta}}{283108}\right)^{16} \cdot \left(\frac{-2011884+49\sqrt{\Delta}}{4591}\right)^{8} \cdot \left(\frac{3706824+23\sqrt{\Delta}}{176006}\right)^{4}$$
$$\cdot \left(\frac{688254+\sqrt{\Delta}}{28}\right)^{2} \cdot (14)$$

$\underline{\Delta = 4(10^{15}+3)}$

$$\left(\frac{63245552+\sqrt{\Delta}}{76107654}\right)^{4194304} \cdot \left(\frac{-11225566653482+177493\sqrt{\Delta}}{1774924}\right)^{2097152} \cdot \left(\frac{-16751096486+376\sqrt{\Delta}}{40193671}\right)^{1048576}$$
$$\cdot \left(\frac{371311897294+1843\sqrt{\Delta}}{38465964}\right)^{524288} \cdot \left(\frac{18655837186+764\sqrt{\Delta}}{5370919}\right)^{262144} \cdot \left(\frac{-4611397255+118\sqrt{\Delta}}{1193583}\right)^{131072}$$
$$\cdot \left(\frac{-61058719+6\sqrt{\Delta}}{886151}\right)^{65536} \cdot \left(\frac{143371715218+2677\sqrt{\Delta}}{63750452}\right)^{32768} \cdot \left(\frac{129470445038+192\sqrt{\Delta}}{16353001}\right)^{16384}$$
$$\cdot \left(\frac{51982162192+1577\sqrt{\Delta}}{13547142}\right)^{8192} \cdot \left(\frac{-139221268+5\sqrt{\Delta}}{129146}\right)^{4096} \cdot \left(\frac{40511500381+1069\sqrt{\Delta}}{32517011}\right)^{2048}$$
$$\cdot \left(\frac{251338789463+5616\sqrt{\Delta}}{59569943}\right)^{1024} \cdot \left(\frac{1611647550422-24961\sqrt{\Delta}}{14823084}\right)^{512} \cdot \left(\frac{10172446742-83\sqrt{\Delta}}{4146442}\right)^{256}$$
$$\cdot \left(\frac{132120628840-1429\sqrt{\Delta}}{60022533}\right)^{128} \cdot \left(\frac{553599535330+4237\sqrt{\Delta}}{32567724}\right)^{64} \cdot \left(\frac{74657708182+209\sqrt{\Delta}}{20361158}\right)^{32}$$
$$\cdot \left(\frac{155349051228-1277\sqrt{\Delta}}{42478049}\right)^{16} \cdot \left(\frac{-24557050464+1609\sqrt{\Delta}}{2702438}\right)^{8} \cdot \left(\frac{-3789291223+153\sqrt{\Delta}}{43420739}\right)^{4}$$
$$\cdot \left(\frac{-340697685734+9231\sqrt{\Delta}}{59609428}\right)^{2} \cdot (29804714)$$

$\underline{\Delta = 4(10^{20}+3)}$

$$\left(\frac{20000000000+\sqrt{\Delta}}{6}\right)^{4194304} \cdot \left(\frac{20000000003-1\sqrt{\Delta}}{13333333333}\right)^{2097152} \cdot \left(\frac{444444444437777777782-2222222221\sqrt{\Delta}}{4444444444}\right)^{1048576}$$
$$\cdot \left(\frac{823045270864197532+41152263\sqrt{\Delta}}{3621399169}\right)^{524288} \cdot \left(\frac{-667242453442734007+33362123\sqrt{\Delta}}{667242459}\right)^{262144}$$
$$\cdot \left(\frac{3341686731023-23\sqrt{\Delta}}{73820503}\right)^{131072} \cdot \left(\frac{33126095069770-613\sqrt{\Delta}}{9654671356}\right)^{65536} \cdot \left(\frac{-183927244686190+16661\sqrt{\Delta}}{3313125978}\right)^{32768}$$
$$\cdot \left(\frac{183141446355784+1163\sqrt{\Delta}}{3006317917}\right)^{16384} \cdot \left(\frac{943684689912166-35885\sqrt{\Delta}}{20770617852}\right)^{8192} \cdot \left(\frac{491109990742094-22094\sqrt{\Delta}}{425861029}\right)^{4096}$$
$$\cdot \left(\frac{159561999106329+7693\sqrt{\Delta}}{9854187533}\right)^{2048} \cdot \left(\frac{6259381493844+33401\sqrt{\Delta}}{2297572142}\right)^{1024} \cdot \left(\frac{-343473693791502+18929\sqrt{\Delta}}{9603870884}\right)^{512}$$
$$\cdot \left(\frac{497034435234062+17482\sqrt{\Delta}}{5412088629}\right)^{256} \cdot \left(\frac{477144491584568+20555\sqrt{\Delta}}{3004212046}\right)^{128} \cdot \left(\frac{305698146732844+11497\sqrt{\Delta}}{999141874}\right)^{64}$$
$$\cdot \left(\frac{-196249792784950+10763\sqrt{\Delta}}{15672659556}\right)^{32} \cdot \left(\frac{426908678627894+36181\sqrt{\Delta}}{5559121394}\right)^{16} \cdot \left(\frac{233743648864178+5450\sqrt{\Delta}}{1383488469}\right)^{8}$$
$$\cdot \left(\frac{41977057841267-872\sqrt{\Delta}}{761697521}\right)^{4} \cdot \left(\frac{-759332000438+57\sqrt{\Delta}}{623092}\right)^{2} \cdot (311546)$$

$\underline{\Delta = 4(10^{25}+3)}$

$$\left(\frac{6324555320336+\sqrt{\Delta}}{4798212423558}\right)^{17179869184} \cdot \left(\frac{-5519839867278860616736778+872763315001\sqrt{\Delta}}{5236579890004}\right)^{8589934592}$$
$$\cdot \left(\frac{3536205908428513482-28048\sqrt{\Delta}}{1819471915169}\right)^{4294967296} \cdot \left(\frac{10519462842208659546-1128343\sqrt{\Delta}}{9021775791724}\right)^{2147483648}$$
$$\cdot \left(\frac{3591341122476462948+984629\sqrt{\Delta}}{1271962857727}\right)^{1073741824} \cdot \left(\frac{621877627447717 37+238310\sqrt{\Delta}}{1401701976639}\right)^{536870912}$$
$$\cdot \left(\frac{-1025385025524600193+298939\sqrt{\Delta}}{1284205577643}\right)^{268435456} \cdot \left(\frac{2972024350703366 84-42337\sqrt{\Delta}}{136150810022}\right)^{134217728}$$
$$\cdot \left(\frac{4589404580700819220+464293\sqrt{\Delta}}{1841108634034}\right)^{67108864} \cdot \left(\frac{7244682302184207404+942365\sqrt{\Delta}}{10008817891122}\right)^{33554432}$$
$$\cdot \left(\frac{-3219385305191880407+1013498\sqrt{\Delta}}{1226743032719}\right)^{16777216} \cdot \left(\frac{5485721406656335218+146699\sqrt{\Delta}}{9712387945996}\right)^{8388608}$$
$$\cdot \left(\frac{6181536057359867452+848181\sqrt{\Delta}}{400080555793}\right)^{4194304} \cdot \left(\frac{12885924719848 15999-172943\sqrt{\Delta}}{2899452724437}\right)^{2097152}$$
$$\cdot \left(\frac{13146424100617841008+2460461\sqrt{\Delta}}{4123212841126}\right)^{1048576} \cdot \left(\frac{7571921285936139444+197395\sqrt{\Delta}}{6561470577922}\right)^{524288}$$
$$\cdot \left(\frac{-2948817356016579 32+554255\sqrt{\Delta}}{566790625278}\right)^{262144} \cdot \left(\frac{5403936792180196 78+76153\sqrt{\Delta}}{1121628339284}\right)^{131072}$$
$$\cdot \left(\frac{-4110790052454042 26+121189\sqrt{\Delta}}{147842545226}\right)^{65536} \cdot \left(\frac{822819375037796 00+37569\sqrt{\Delta}}{2273227002107}\right)^{32768}$$
$$\cdot \left(\frac{8454473621979660518-268983\sqrt{\Delta}}{6636017456972}\right)^{16384} \cdot \left(\frac{8700450524029298514-901189\sqrt{\Delta}}{3925103161814}\right)^{8192}$$
$$\cdot \left(\frac{11650952062197127648+1633139\sqrt{\Delta}}{1886157605037}\right)^{4096} \cdot \left(\frac{8389010217404123723-1170472\sqrt{\Delta}}{4378048645009}\right)^{2048}$$
$$\cdot \left(\frac{5777201756090992 24+1530061\sqrt{\Delta}}{2434084549398}\right)^{1024} \cdot \left(\frac{616482605615890 39-382\sqrt{\Delta}}{23057133697}\right)^{512}$$
$$\cdot \left(\frac{2647991287305093 08-11921\sqrt{\Delta}}{2244459824102}\right)^{256} \cdot \left(\frac{-12904718315339091224+2967737\sqrt{\Delta}}{8194676691814}\right)^{128}$$
$$\cdot \left(\frac{37843118122608644348+5068711\sqrt{\Delta}}{12045031222674}\right)^{64} \cdot \left(\frac{38869432137330916 3232-61356025\sqrt{\Delta}}{6903690784698}\right)^{32}$$
$$\cdot \left(\frac{5056873308656719010+956473\sqrt{\Delta}}{462502621124}\right)^{16} \cdot \left(\frac{-883626685600458 46+27699\sqrt{\Delta}}{427872743234}\right)^{8}$$
$$\cdot \left(\frac{-3791140246682779 86+117526\sqrt{\Delta}}{2232783257611}\right)^{4} \cdot \left(\frac{-146365607825376 42+10235\sqrt{\Delta}}{398768708}\right)^{2} \cdot (199384354)$$

$\underline{\Delta = 4(10^{30}+3)}$

$$\left(\frac{2000000000000000+\sqrt{\Delta}}{6}\right)^{35184372088832} \cdot \left(\frac{2000000000000003-1\sqrt{\Delta}}{1333333333333333}\right)^{17592186044416}$$
$$\cdot \left(\frac{2000000000000024+7\sqrt{\Delta}}{54}\right)^{8796093022208} \cdot \left(\frac{-1999999999998860+\sqrt{\Delta}}{3127572016460014}\right)^{4398046511104}$$
$$\cdot \left(\frac{-21659800000000002280+108301\sqrt{\Delta}}{354294}\right)^{2199023255552} \cdot \left(\frac{7999999972221895+4\sqrt{\Delta}}{127466923134367}\right)^{1099511627776}$$
$$\cdot \left(\frac{752438425736355935382 02+27226679\sqrt{\Delta}}{1024436854643884}\right)^{549755813888} \cdot \left(\frac{1455118072119162978 5002-4067295\sqrt{\Delta}}{554814088475686}\right)^{274877906944}$$
$$\cdot \left(\frac{8462671826672308959500+3891451\sqrt{\Delta}}{35875831719153}\right)^{137438953472} \cdot \left(\frac{-5178379830657428623369+2652713\sqrt{\Delta}}{1034848053817563}\right)^{68719476736}$$
$$\cdot \left(\frac{1512189895399947369 1558+3975497\sqrt{\Delta}}{231746989928036}\right)^{34359738368}$$
$$\cdot \left(\frac{9542178259235589422362+1604357\sqrt{\Delta}}{668299564578766}\right)^{17179869184} \cdot \left(\frac{3243581905332172199 9792-11634365\sqrt{\Delta}}{1143351476132469}\right)^{8589934592}$$
$$\cdot \left(\frac{1626183596694896706 5507-5038171\sqrt{\Delta}}{373871066312911}\right)^{4294967296} \cdot \left(\frac{5967678350812889679 5594+17636351\sqrt{\Delta}}{920955963196108}\right)^{2147483648}$$
$$\cdot \left(\frac{-6878414666161759501408+6184027\sqrt{\Delta}}{498284218119699}\right)^{1073741824} \cdot \left(\frac{1068424443241616709 7074+1295297\sqrt{\Delta}}{100945248510244}\right)^{536870912}$$

$$
\begin{aligned}
&\cdot\left(\frac{1192391383236709824 50+147386\sqrt{\Delta}}{9509056589401}\right)^{268435456}\cdot\left(\frac{5262935053633355335 69096+25342475\sqrt{\Delta}}{1523750229707002}\right)^{134217728}\\
&\cdot\left(\frac{-12313436100997373 6176982+64010505\sqrt{\Delta}}{1057197292299988}\right)^{67108864}\cdot\left(\frac{-6201329388320176191930+3873503\sqrt{\Delta}}{77159414587978}\right)^{33554432}\\
&\cdot\left(\frac{8142942874026603469794+4293416\sqrt{\Delta}}{1247345472729079}\right)^{16777216}\cdot\left(\frac{67286952367669184863684+55565167\sqrt{\Delta}}{2513839046377566}\right)^{8388608}\\
&\cdot\left(\frac{61535581760736628730062-27595807\sqrt{\Delta}}{703087431620756}\right)^{4194304}\cdot\left(\frac{13544705116852852267954-42460\sqrt{\Delta}}{164938047808161}\right)^{2097152}\\
&\cdot\left(\frac{4332357355196325035524+1353571\sqrt{\Delta}}{630815345311606}\right)^{1048576}\cdot\left(\frac{-15884299200932173682309+9262480\sqrt{\Delta}}{101484782198599}\right)^{524288}\\
&\cdot\left(\frac{10001763709278897037778+2664953\sqrt{\Delta}}{3477340494188988}\right)^{262144}\cdot\left(\frac{18022654892391091048430+8528754\sqrt{\Delta}}{100800592001677}\right)^{131072}\\
&\cdot\left(\frac{-37502868478745907260891+23642021\sqrt{\Delta}}{1007647402272939}\right)^{65536}\cdot\left(\frac{85061113885432771072786-28118045\sqrt{\Delta}}{2005654252493388}\right)^{32768}\\
&\cdot\left(\frac{8456949956107930441904-2241425\sqrt{\Delta}}{51134522237781}\right)^{16384}\cdot\left(\frac{8655726400435761327614+4559675\sqrt{\Delta}}{1575863547860532}\right)^{8192}\\
&\cdot\left(\frac{8328829566643441847 84+1622053\sqrt{\Delta}}{47502990065401}\right)^{4096}\cdot\left(\frac{-8128034302837913384191+4840231\sqrt{\Delta}}{1361301206338339}\right)^{2048}\\
&\cdot\left(\frac{14217735704967187392970+3516559\sqrt{\Delta}}{41194723414284}\right)^{1024}\cdot\left(\frac{-4091930162753555921 74+463891\sqrt{\Delta}}{1634268415359514}\right)^{512}\\
&\cdot\left(\frac{99771966643502340486106-43544054\sqrt{\Delta}}{887403498454389}\right)^{256}\cdot\left(\frac{47319191384480541470053+27808346\sqrt{\Delta}}{1084605534441423}\right)^{128}\\
&\cdot\left(\frac{5063069415430602919597+8576002\sqrt{\Delta}}{228292764909991}\right)^{64}\cdot\left(\frac{21364328091343344313268-5563537\sqrt{\Delta}}{3191079645713058}\right)^{32}\\
&\cdot\left(\frac{-50638823753089434860 99+11543864\sqrt{\Delta}}{199312898905911}\right)^{16}\cdot\left(\frac{5141956211794109041 45+843733\sqrt{\Delta}}{195073879687249}\right)^{8}\\
&\cdot\left(\frac{37238530816467275482940-11185957\sqrt{\Delta}}{1293789917417634}\right)^{4}\cdot\left(\frac{807598476457790+\sqrt{\Delta}}{4}\right)^{2}\cdot(2)
\end{aligned}
$$

$\underline{\Delta = 4(10^{35}+3)}$

$$
\begin{aligned}
&\left(\frac{632455532033675866+\sqrt{\Delta}}{252842256024575028}\right)^{2251799813685248}\\
&\cdot\left(\frac{110645556728141947733650300369682-174945986119148\sqrt{\Delta}}{1749459861191481}\right)^{1125899906842624}\\
&\cdot\left(\frac{25237371743075841285388825+3828254\sqrt{\Delta}}{165665629311802353}\right)^{562949953421312}\\
&\cdot\left(\frac{188761431345834272925711418-117383017\sqrt{\Delta}}{548720319561560492}\right)^{281474976710656}\\
&\cdot\left(\frac{351210000538714860463758 42+138446545\sqrt{\Delta}}{85468244695640146}\right)^{140737488355328}\\
&\cdot\left(\frac{22348258451064840956886454+1944270\sqrt{\Delta}}{68164927189165501}\right)^{70368744177664}\\
&\cdot\left(\frac{-111721122439145381767231 44+88419373\sqrt{\Delta}}{323082503903770406}\right)^{35184372088832}\\
&\cdot\left(\frac{144382556085892594543692 19+149229914\sqrt{\Delta}}{333365457829449999}\right)^{17592186044416}\\
&\cdot\left(\frac{716210702189726356412930 78+317786771\sqrt{\Delta}}{158665493865881604}\right)^{8796093022208}\\
&\cdot\left(\frac{175003756808183772536555 62+19602680\sqrt{\Delta}}{24239714904367961}\right)^{4398046511104}\\
&\cdot\left(\frac{27945402524474987656026004-1224317\sqrt{\Delta}}{664052116357366194}\right)^{2199023255552}\\
&\cdot\left(\frac{86575281455849923434156490+189802607\sqrt{\Delta}}{94085414670058748}\right)^{1099511627776}\\
&\cdot\left(\frac{29814585604326338850796688-27433181\sqrt{\Delta}}{29516162738516093}\right)^{549755813888}\\
&\cdot\left(\frac{4742482370001399770797 90-216237\sqrt{\Delta}}{5562289770843508}\right)^{274877906944}\\
&\cdot\left(\frac{8994292059360835099192150+15709648\sqrt{\Delta}}{1042948738414767}\right)^{137438953472}\\
&\cdot\left(\frac{9763785428043695620084 84-1428721\sqrt{\Delta}}{62890635640999938}\right)^{68719476736}\\
&\cdot\left(\frac{-75312599618541774219463298+122266471\sqrt{\Delta}}{155565239673552804}\right)^{34359738368}\\
&\cdot\left(\frac{34897611814385388175005640+22943939\sqrt{\Delta}}{166487758979078437}\right)^{17179869184}\\
&\cdot\left(\frac{97573138427396428315128328+201821763\sqrt{\Delta}}{122163391652453638}\right)^{8589934592}\\
&\cdot\left(\frac{-24340645884422552620823033+43789211\sqrt{\Delta}}{46778931795994083}\right)^{4294967296}\\
&\cdot\left(\frac{2140342996559015989057 70+315341\sqrt{\Delta}}{260607615595564}\right)^{2147483648}\\
&\cdot\left(\frac{630332948974334427034 24+790705\sqrt{\Delta}}{1217354918074793}\right)^{1073741824}\\
&\cdot\left(\frac{18037613563449867617967584-28106899\sqrt{\Delta}}{350753158822748442}\right)^{536870912}\\
&\cdot\left(\frac{15872715909211439187919 7756-143327819\sqrt{\Delta}}{275989142796778722}\right)^{268435456}\\
&\cdot\left(\frac{22100900861559651630325774+146111305\sqrt{\Delta}}{211394376445069572}\right)^{134217728}\\
&\cdot\left(\frac{2639217760187899951223852+23500639\sqrt{\Delta}}{19150424482400863}\right)^{67108864}\\
&\cdot\left(\frac{1175239001993972071069 0980+8976073\sqrt{\Delta}}{144368047161496354}\right)^{33554432}\\
&\cdot\left(\frac{1808132788247617870343 1292-2231953\sqrt{\Delta}}{31181239623911682}\right)^{16777216}\\
&\cdot\left(\frac{4422080046721331410226983+16758835\sqrt{\Delta}}{381740284019412731}\right)^{8388608}\\
&\cdot\left(\frac{-807616703747278515290743+9691866\sqrt{\Delta}}{253357341637943}\right)^{4194304}\\
&\cdot\left(\frac{-451150517464665816 4516+45245\sqrt{\Delta}}{6219746421254478}\right)^{2097152}\\
&\cdot\left(\frac{-1783013371479982651758 23+2397065\sqrt{\Delta}}{234360862592100251}\right)^{1048576}
\end{aligned}
$$

$$
\begin{aligned}
&\cdot \left(\frac{5987323002366575341016782 4-79109587\sqrt{\Delta}}{9844994969122074}\right)^{524288}\\
&\cdot \left(\frac{1002380191223800117348415 2+14676731\sqrt{\Delta}}{295366319474134394}\right)^{262144}\\
&\cdot \left(\frac{758728936289971039918883 2-8808199\sqrt{\Delta}}{608272077041034}\right)^{131072}\\
&\cdot \left(\frac{558384276098071898945216+777953\sqrt{\Delta}}{376808696535709066}\right)^{65536}\\
&\cdot \left(\frac{-3887423962760378679178415 52+649838479\sqrt{\Delta}}{250665840739697846}\right)^{32768}\\
&\cdot \left(\frac{1579382389432208221988135 74+303288527\sqrt{\Delta}}{377159019447974484}\right)^{16384}\\
&\cdot \left(\frac{806824308505614295314693 4-3458782\sqrt{\Delta}}{15263426199181533}\right)^{8192}\\
&\cdot \left(\frac{-1853310631809914998971958 6+117784301\sqrt{\Delta}}{137932555761686612}\right)^{4096}\\
&\cdot \left(\frac{-3619157994348260550576859 4+80027464\sqrt{\Delta}}{263211958949645031}\right)^{2048}\\
&\cdot \left(\frac{-8748481879799252051322320 0+545102867\sqrt{\Delta}}{802543313127239894}\right)^{1024}\\
&\cdot \left(\frac{1692747821268049837371504 86+596009731\sqrt{\Delta}}{352247649410270052}\right)^{512}\\
&\cdot \left(\frac{1415900414169304841937336 52-157855409\sqrt{\Delta}}{324968962814706257}\right)^{256}\\
&\cdot \left(\frac{7351285674396249521730601 6+142987431\sqrt{\Delta}}{13133973830450662}\right)^{128}\\
&\cdot \left(\frac{2108366485139392272846435 7-33303062\sqrt{\Delta}}{20483284391884161}\right)^{64}\\
&\cdot \left(\frac{2904966152648376137131143 8+44168647\sqrt{\Delta}}{227146304543292724}\right)^{32}\\
&\cdot \left(\frac{9551704201857134168576107 6-105082801\sqrt{\Delta}}{40542316271957009}\right)^{16}\\
&\cdot \left(\frac{4065197043760521383062089 2+12322301\sqrt{\Delta}}{484232828879190546}\right)^{8}\\
&\cdot \left(\frac{2760944696598933105429525 34+204335765\sqrt{\Delta}}{507732433928976732}\right)^{4}\\
&\cdot \left(\frac{-520676375273815010+\sqrt{\Delta}}{2}\right)^{2} \cdot (2)
\end{aligned}
$$

$\underline{\Delta = 4(10^{40}+3)}$

$$\left(\frac{200000000000000000000+\sqrt{\Delta}}{6}\right)^{288230376151711744}$$
$$\cdot\left(\frac{-199999999999999999994+\sqrt{\Delta}}{133333333333333333332}\right)^{144115188075855872}$$
$$\cdot\left(\frac{8230452674897119341975308641975308642 0+411522633744855967\sqrt{\Delta}}{6584362139917695473}\right)^{72057594037927936}$$
$$\cdot\left(\frac{91449474165523557548239597622313671 77-45724737082761769\sqrt{\Delta}}{30544124371284864959}\right)^{36028797018963968}$$
$$\cdot\left(\frac{8992479928493836215336594919289 4+449623995527\sqrt{\Delta}}{536467072302596644}\right)^{18014398509481984}$$
$$\cdot\left(\frac{-40975338131229384223819052742785818 40+20487669065615141\sqrt{\Delta}}{131366934048723521963}\right)^{9007199254740992}$$
$$\cdot\left(\frac{233193412676325149073262368791 0+2557510429\sqrt{\Delta}}{149973952411005781116}\right)^{4503599627370496}$$
$$\cdot\left(\frac{132606523760137231928358897463 4+4916168210\sqrt{\Delta}}{140795671078307286829}\right)^{2251799813685248}$$
$$\cdot\left(\frac{-136745333665010593854169277924+839411201\sqrt{\Delta}}{239241256544449998}\right)^{1125899906842624}$$
$$\cdot\left(\frac{267846369842134218416718423 8+16510235\sqrt{\Delta}}{130313855715135311028}\right)^{562949953421312}$$
$$\cdot\left(\frac{715990041503034540299534980588-855172441\sqrt{\Delta}}{113861136437167338857}\right)^{281474976710656}$$
$$\cdot\left(\frac{-167172142378764919797545015692+947535689\sqrt{\Delta}}{3379678667649997866}\right)^{140737488355328}$$
$$\cdot\left(\frac{1324580597303959291913926856066+6148654015\sqrt{\Delta}}{350593644875886753212}\right)^{70368744177664}$$
$$\cdot\left(\frac{1825534897627498667737112069948+3422981551\sqrt{\Delta}}{93198860858196653697}\right)^{35184372088832}$$
$$\cdot\left(\frac{1110095153622064273994134504 73+2321281046\sqrt{\Delta}}{23395126415774639007}\right)^{17592186044416}$$
$$\cdot\left(\frac{328847053055400471640962232694+2694337879\sqrt{\Delta}}{166478389642300427572}\right)^{8796093022208}$$
$$\cdot\left(\frac{-810513859431085156622401472102+4097498994\sqrt{\Delta}}{21139708099738 76843}\right)^{4398046511104}$$
$$\cdot\left(\frac{6078204084921499439516452 0+381969\sqrt{\Delta}}{208219443758920046}\right)^{2199023255552}$$
$$\cdot\left(\frac{216058614175568044442743726443+4945989046\sqrt{\Delta}}{113845277498153154847}\right)^{1099511627776}$$
$$\cdot\left(\frac{103762234354414577188299400916-393490367\sqrt{\Delta}}{176425573608249666}\right)^{549755813888}$$
$$\cdot\left(\frac{10132831080672208663613907 97+4948351\sqrt{\Delta}}{6077951310502110173}\right)^{274877906944}$$
$$\cdot\left(\frac{-462626208832814919120380627 42+448876011\sqrt{\Delta}}{80117998723167604036}\right)^{137438953472}$$
$$\cdot\left(\frac{382328513209544311000576775218-794989205\sqrt{\Delta}}{75336831457341294726}\right)^{68719476736}$$
$$\cdot\left(\frac{456112455479169789846233730 34+1466113502\sqrt{\Delta}}{44346995211907063601}\right)^{34359738368}$$
$$\cdot\left(\frac{-831082642363404584250814457042+9918486503\sqrt{\Delta}}{91648989955790222308}\right)^{17179869184}$$
$$\cdot\left(\frac{-263423015953857436392023781030+3016788223\sqrt{\Delta}}{140316662376729515578}\right)^{8589934592}$$
$$\cdot\left(\frac{1031911803240372748266236397 10+6754253386\sqrt{\Delta}}{92141333606258060403}\right)^{4294967296}$$
$$\cdot\left(\frac{890281420706861619637685003513+1601671867\sqrt{\Delta}}{81270298746985006589}\right)^{2147483648}$$
$$\cdot\left(\frac{249854927727467994703446371447 8-11418835099\sqrt{\Delta}}{77757625705650604716}\right)^{1073741824}$$
$$\cdot\left(\frac{226096968738495582821150345786+841908698\sqrt{\Delta}}{150621858076704315 57}\right)^{536870912}$$
$$\cdot\left(\frac{770252279751699601940231804 26-97820959\sqrt{\Delta}}{122319870669214063 48}\right)^{268435456}$$
$$\cdot\left(\frac{701771810167360817001048497 98+9247418\sqrt{\Delta}}{131569751558730412341}\right)^{134217728}$$

$$
\begin{aligned}
&\cdot \left(\frac{4998101520503299398267744589052+5988400417\sqrt{\Delta}}{102650465024693131466}\right)^{67108864}\\
&\cdot \left(\frac{13894770023889380452625299545 22+1820320507\sqrt{\Delta}}{113763154554876859444}\right)^{33554432}\\
&\cdot \left(\frac{-13573589647419750093394886930 38+10063385306\sqrt{\Delta}}{75840455750185526923}\right)^{16777216}\\
&\cdot \left(\frac{8095658668000187173433144192 78+105162459\sqrt{\Delta}}{56934994667203633964}\right)^{8388608}\\
&\cdot \left(\frac{4985713453424867393079883696 94+1724666386\sqrt{\Delta}}{15991445684848906 2141}\right)^{4194304}\\
&\cdot \left(\frac{7590267083784220879360359443 30+6033521621\sqrt{\Delta}}{51618487886459815948}\right)^{2097152}\\
&\cdot \left(\frac{3351730702187975658142481028 82-426138221\sqrt{\Delta}}{17527321970464812998}\right)^{1048576}\\
&\cdot \left(\frac{3322238087259837411722511185 06+1864218036\sqrt{\Delta}}{93226049226012050879}\right)^{524288}\\
&\cdot \left(\frac{-7955198256778285624453934843 0+461134969\sqrt{\Delta}}{28809892905308595 48}\right)^{262144}\\
&\cdot \left(\frac{2175232340432644049247461534 92+505410245\sqrt{\Delta}}{33797192169432951641}\right)^{131072}\\
&\cdot \left(\frac{1856532022643100600553970205 65-212229422\sqrt{\Delta}}{28597463541977196057}\right)^{65536}\\
&\cdot \left(\frac{1385026839959854012646924112 46+611506243\sqrt{\Delta}}{7750038467676895108}\right)^{32768}\\
&\cdot \left(\frac{1284060612293257947597316271 00-567909337\sqrt{\Delta}}{26544500199136753513}\right)^{16384}\\
&\cdot \left(\frac{1472064182689624802480333705 11+716026947\sqrt{\Delta}}{1649061490718960677}\right)^{8192}\\
&\cdot \left(\frac{4902327602152515145591221857 54-2447585943\sqrt{\Delta}}{128903484905756429516}\right)^{4096}\\
&\cdot \left(\frac{4503717136276743234709497190 10+541042269\sqrt{\Delta}}{46009716865432074262}\right)^{2048}\\
&\cdot \left(\frac{1848928029233148591998734241 06+118567082\sqrt{\Delta}}{15883186273975996917}\right)^{1024}\\
&\cdot \left(\frac{4901168905907976402652430486 80-1328438857\sqrt{\Delta}}{33618901390554127 6554}\right)^{512}\\
&\cdot \left(\frac{8401051934760625356632712145 66+2117328769\sqrt{\Delta}}{27947581161218193284}\right)^{256}\\
&\cdot \left(\frac{3784152480936152523610596610 72+2396213983\sqrt{\Delta}}{49206491854581089259}\right)^{128}\\
&\cdot \left(\frac{-2713477700790021885379987350 17+2358567617\sqrt{\Delta}}{61489834765396962259}\right)^{64}\\
&\cdot \left(\frac{5768840520570658932406798955 83-861864098\sqrt{\Delta}}{80159438841052582761}\right)^{32}\\
&\cdot \left(\frac{-1992831411262134723560426188 28+7447314227\sqrt{\Delta}}{169541180778463451342}\right)^{16}\\
&\cdot \left(\frac{1479046611790612987688832207 914+1959192767\sqrt{\Delta}}{141527026682884937004}\right)^{8}\\
&\cdot \left(\frac{-2895866389770772248129061976 42+4017782174\sqrt{\Delta}}{112200768367881011787}\right)^{4}\\
&\cdot \left(\frac{818108984580176186570+41\sqrt{\Delta}}{2644}\right)^{2} \cdot (1322)
\end{aligned}
$$

$\underline{\Delta = 4(10^{45} + 3)}$

$$\left(\frac{632455532033675866399976+\sqrt{\Delta}}{118325387927577279359718}\right)^{73786976294838206464}$$
$$\cdot\left(\frac{-377906127469353025977712561671139570092991+597522052268539851 5\sqrt{\Delta}}{486860968188406125 76499}\right)^{36893488147419103232}$$
$$\cdot\left(\frac{267980647427418183569330483661326 60-268042436289\sqrt{\Delta}}{908624935635961814106 74}\right)^{18446744073709551616}$$
$$\cdot\left(\frac{889526591717909568054050364102899 1-36097467776\sqrt{\Delta}}{358109069986699726516 01}\right)^{9223372036854775808}$$
$$\cdot\left(\frac{312426059552233785175932134480440 70+446241497293\sqrt{\Delta}}{700138733056710472561 56}\right)^{4611686018427387904}$$
$$\cdot\left(\frac{525025098118313751490196196336299 0-62195907967\sqrt{\Delta}}{986695507294257106679 8}\right)^{2305843009213693952}$$
$$\cdot\left(\frac{230272157740873099533020498894321 0+2285816099\sqrt{\Delta}}{542502086273107646522 22}\right)^{1152921504606846976}$$
$$\cdot\left(\frac{652098312285823394588381957857616 8-59669372701\sqrt{\Delta}}{960946878074029398309 3}\right)^{576460752303423488}$$
$$\cdot\left(\frac{180925428561819534641055096202137 1+35915288314\sqrt{\Delta}}{204265979786305065578 39}\right)^{288230376151711744}$$
$$\cdot\left(\frac{575126623441701588832051786832238 4-12812812533\sqrt{\Delta}}{388504599471504717903 14}\right)^{144115188075855872}$$
$$\cdot\left(\frac{844972968685130205012066930928013 6-87247564743\sqrt{\Delta}}{542606526836576372959 46}\right)^{72057594037927936}$$
$$\cdot\left(\frac{139148972944756487992997380662031 98+227055265165\sqrt{\Delta}}{855371796735687360501 2}\right)^{36028797018963968}$$
$$\cdot\left(\frac{237737847928012074833293397833762-2429420636\sqrt{\Delta}}{179924581739246263829 7}\right)^{18014398509481984}$$
$$\cdot\left(\frac{-706262929682502051289691216116 87+1876939768\sqrt{\Delta}}{281209003045688436019 1}\right)^{9007199254740992}$$
$$\cdot\left(\frac{374405906008101548971273358889 58-579803865\sqrt{\Delta}}{148043850037612513852}\right)^{4503599627370496}$$
$$\cdot\left(\frac{124876228313271012066056163895294+3474923538\sqrt{\Delta}}{355093709237468679730 7}\right)^{2251799813685248}$$
$$\cdot\left(\frac{142121836813897762002764258202527 6+9240247381\sqrt{\Delta}}{665521625059293529173 78}\right)^{1125899906842624}$$
$$\cdot\left(\frac{806712158418601973431007163793600+43819382239\sqrt{\Delta}}{317429055978235179840 6}\right)^{562949953421312}$$
$$\cdot\left(\frac{-324143613688922289015663694995839+6062067068\sqrt{\Delta}}{166435270445187472386 63}\right)^{281474976710656}$$
$$\cdot\left(\frac{-252235653364156888271806175557367 8+59248057889\sqrt{\Delta}}{415822371641657999393 08}\right)^{140737488355328}$$
$$\cdot\left(\frac{540923770708605139285728117386571 20+840980891861\sqrt{\Delta}}{249303233787459175745 17}\right)^{70368744177664}$$
$$\cdot\left(\frac{415582471495244280292257141482562 7-29233178049\sqrt{\Delta}}{222881668298696618480 53}\right)^{35184372088832}$$
$$\cdot\left(\frac{673378011545744441099729181493927 0-17541745341\sqrt{\Delta}}{444004468176612368879 96}\right)^{17592186044416}$$
$$\cdot\left(\frac{574271995155086749279884056762360 2+41430784041\sqrt{\Delta}}{529832550416502897465 58}\right)^{8796093022208}$$
$$\cdot\left(\frac{771262300111384440446648938928234+88323945586\sqrt{\Delta}}{109038742841449051107 39}\right)^{4398046511104}$$
$$\cdot\left(\frac{453844379625731570809552492859912 8+10944935707\sqrt{\Delta}}{846057216789981584285 38}\right)^{2199023255552}$$
$$\cdot\left(\frac{225175865888793265247646403331303 00-268385917777\sqrt{\Delta}}{611661870357252894588 66}\right)^{1099511627776}$$
$$\cdot\left(\frac{-484978908386927647389862246737773 9+95148721508\sqrt{\Delta}}{135703144836592257416 23}\right)^{549755813888}$$
$$\cdot\left(\frac{235740884800268785499373283972509 1+50231277728\sqrt{\Delta}}{246280964975873878626 63}\right)^{274877906944}$$
$$\cdot\left(\frac{203549438191299337093600229696194 0+158035132243\sqrt{\Delta}}{789368215195346759173 26}\right)^{137438953472}$$
$$\cdot\left(\frac{615713685032625829184654523999625+77097869792\sqrt{\Delta}}{150198304845835244812 47}\right)^{68719476736}$$
$$\cdot\left(\frac{766312971040814545623401395505422-5088715535\sqrt{\Delta}}{321586121060017826056 4}\right)^{34359738368}$$
$$\cdot\left(\frac{199039464031639788855368534725605 4+29580813505\sqrt{\Delta}}{198364071832665661495 26}\right)^{17179869184}$$

$$
\begin{aligned}
&\cdot\left(\frac{-29375534062274964566632530869461110+85927934762\sqrt{\Delta}}{5312864144613217567360 3}\right)^{8589934592}\\
&\cdot\left(\frac{1188591285586498074932138003027568 3+163054188455\sqrt{\Delta}}{1237426513101263449762 1}\right)^{4294967296}\\
&\cdot\left(\frac{41765729593963893092960783040657 14-61780715063\sqrt{\Delta}}{710651784757363618172 4}\right)^{2147483648}\\
&\cdot\left(\frac{-23493027393428470511121249470043 2+6862660999\sqrt{\Delta}}{1054933734927334255382 7}\right)^{1073741824}\\
&\cdot\left(\frac{11476002076248105091484788415332 54-9140258665\sqrt{\Delta}}{132467697816994129283 56}\right)^{536870912}\\
&\cdot\left(\frac{11454511103628681528575465183150 36+60268334383\sqrt{\Delta}}{334758327333574602944 87}\right)^{268435456}\\
&\cdot\left(\frac{-24614643067380695037607253700856 40+103894460553\sqrt{\Delta}}{165609449978392014961 66}\right)^{134217728}\\
&\cdot\left(\frac{-99566136626007740304344235387563+2242423824\sqrt{\Delta}}{10413742370195405888 09}\right)^{67108864}\\
&\cdot\left(\frac{49175447666028461723593183357126 16+67305323789\sqrt{\Delta}}{570416349847835774684 58}\right)^{33554432}\\
&\cdot\left(\frac{11303279018621102336594908244953 918-123865242823\sqrt{\Delta}}{408106101457829946414 68}\right)^{16777216}\\
&\cdot\left(\frac{68850732841352620087845280591990 84+97604163633\sqrt{\Delta}}{223306157997842067126 73}\right)^{8388608}\\
&\cdot\left(\frac{43545938720263529006523410750577 97-2212513033\sqrt{\Delta}}{379878947883174237628 29}\right)^{4194304}\\
&\cdot\left(\frac{95659393211619647102088921465242 08-143917843481\sqrt{\Delta}}{299976837896936513722 6}\right)^{2097152}\\
&\cdot\left(\frac{25809382906370759423092098803258 1+4558257091\sqrt{\Delta}}{733375738228661055181 9}\right)^{1048576}\\
&\cdot\left(\frac{27870154960478895145613154226376 1+2143756919\sqrt{\Delta}}{110240558104631304914 9}\right)^{524288}\\
&\cdot\left(\frac{15565375338445817449483150003632 4-1922140735\sqrt{\Delta}}{388776679288767285553 8}\right)^{262144}\\
&\cdot\left(\frac{20533168595577890136938037356929 30-32021204387\sqrt{\Delta}}{151746011134218636144 76}\right)^{131072}\\
&\cdot\left(\frac{24706436750399881438505623709342+9928884400\sqrt{\Delta}}{68393295877382222237 19}\right)^{65536}\\
&\cdot\left(\frac{53722781579197920702331590495 80+239869079\sqrt{\Delta}}{346405874676452272446}\right)^{32768}\\
&\cdot\left(\frac{-10637739102236415434175018060063112+239812893613\sqrt{\Delta}}{7515289751657774318523 8}\right)^{16384}\\
&\cdot\left(\frac{89643940175654479309176945385060 44-56308434743\sqrt{\Delta}}{239654051066875674852 34}\right)^{8192}\\
&\cdot\left(\frac{63064170717360207651378493396836 90+51662184893\sqrt{\Delta}}{1013160788808490475214 04}\right)^{4096}\\
&\cdot\left(\frac{57055329370711968108679660649961 24-51698387947\sqrt{\Delta}}{851917152310695671701 7}\right)^{2048}\\
&\cdot\left(\frac{17558374018326182610224802388366 38-286420625\sqrt{\Delta}}{212372188851481737691 48}\right)^{1024}\\
&\cdot\left(\frac{-17049325756388454778683365271086+31109514763\sqrt{\Delta}}{343303759386484555586 82}\right)^{512}\\
&\cdot\left(\frac{48554145131995144586389724358619 70+65780353156\sqrt{\Delta}}{531729604020494133925 7}\right)^{256}\\
&\cdot\left(\frac{-98986757227237031681899621452 57+2436946091\sqrt{\Delta}}{836710221284090758027}\right)^{128}\\
&\cdot\left(\frac{17969839948002399398829534408721 18-27936352361\sqrt{\Delta}}{766996142429109955782 84}\right)^{64}
\end{aligned}
$$

$$
\begin{aligned}
&\cdot \left(\frac{-508253554419129045089449022573942+76689665648\sqrt{\Delta}}{15820205125945321305631}\right)^{32}\\
&\cdot \left(\frac{5534885858953254704410379043510544-50201592107\sqrt{\Delta}}{41062516404577720785834}\right)^{16}\\
&\cdot \left(\frac{479750264626091969493842523000617-3485980913\sqrt{\Delta}}{430695230884581480749}\right)^{8}\\
&\cdot \left(\frac{262309146153190034934369545145442+4356789715\sqrt{\Delta}}{19192561827531103531668}\right)^{4}\\
&\cdot \left(\frac{50260567567175932089254336+169\sqrt{\Delta}}{26190869}\right)^{2} \cdot (26190869)
\end{aligned}
$$

$\underline{\Delta = 10^{11} + 1}$

$$
\begin{aligned}
&\left(\frac{316227+\sqrt{\Delta}}{242236}\right)^{16384} \cdot \left(\frac{23398194193-73991\sqrt{\Delta}}{295966}\right)^{8192} \cdot \left(\frac{-9045113+220\sqrt{\Delta}}{217279}\right)^{4096} \\
&\cdot \left(\frac{199009927+439\sqrt{\Delta}}{215344}\right)^{2048} \cdot \left(\frac{9600337+79\sqrt{\Delta}}{91766}\right)^{1024} \cdot \left(\frac{84497377-145\sqrt{\Delta}}{149546}\right)^{512} \\
&\cdot \left(\frac{10743277+46\sqrt{\Delta}}{17203}\right)^{256} \cdot \left(\frac{-1089025+4\sqrt{\Delta}}{1399}\right)^{128} \cdot \left(\frac{4108471+9\sqrt{\Delta}}{70090}\right)^{64} \\
&\cdot \left(\frac{41563157+107\sqrt{\Delta}}{3706}\right)^{32} \cdot \left(\frac{2687255-7\sqrt{\Delta}}{338032}\right)^{16} \cdot \left(\frac{75525057+577\sqrt{\Delta}}{482890}\right)^{8} \\
&\cdot \left(\frac{49114481+19\sqrt{\Delta}}{20380}\right)^{4} \cdot \left(\frac{2923407+7\sqrt{\Delta}}{17558}\right)^{2} \cdot (8779)
\end{aligned}
$$

$\underline{\Delta = 10^{15} + 1}$

$$
\begin{aligned}
&\left(\frac{31622775+\sqrt{\Delta}}{50649688}\right)^{65536} \cdot \left(\frac{199158105628213-6297931\sqrt{\Delta}}{12595864}\right)^{32768} \cdot \left(\frac{-10442781433+905\sqrt{\Delta}}{17899714}\right)^{16384} \\
&\cdot \left(\frac{53088384293+1463\sqrt{\Delta}}{2116130}\right)^{8192} \cdot \left(\frac{10483195483+367\sqrt{\Delta}}{11072612}\right)^{4096} \cdot \left(\frac{194746222423-3927\sqrt{\Delta}}{22944800}\right)^{2048} \\
&\cdot \left(\frac{-7312107269+1019\sqrt{\Delta}}{7483090}\right)^{1024} \cdot \left(\frac{22471634477-123\sqrt{\Delta}}{4373882}\right)^{512} \cdot \left(\frac{38474886335+1367\sqrt{\Delta}}{40601672}\right)^{256} \\
&\cdot \left(\frac{7712521167+95\sqrt{\Delta}}{979472}\right)^{128} \cdot \left(\frac{59853948011-1131\sqrt{\Delta}}{18756980}\right)^{64} \cdot \left(\frac{7071045597+22\sqrt{\Delta}}{2251837}\right)^{32} \\
&\cdot \left(\frac{6540855167+959\sqrt{\Delta}}{5404124}\right)^{16} \cdot \left(\frac{-1244298715+51\sqrt{\Delta}}{793022}\right)^{8} \cdot \left(\frac{458354963+693\sqrt{\Delta}}{12616840}\right)^{4} \\
&\cdot \left(\frac{3741565729+321\sqrt{\Delta}}{1118722}\right)^{2} \cdot (559361)
\end{aligned}
$$

$\underline{\Delta = 10^{21} + 1}$

$$
\begin{aligned}
&\left(\frac{31622776601+\sqrt{\Delta}}{21623443400}\right)^{33554432} \cdot \left(\frac{609763080777342401-19282401\sqrt{\Delta}}{1465462502}\right)^{16777216} \\
&\cdot \left(\frac{-2092282925627531+69433\sqrt{\Delta}}{2442785756}\right)^{8388608} \cdot \left(\frac{116049272963081+3085\sqrt{\Delta}}{5295893908}\right)^{4194304} \\
&\cdot \left(\frac{2899231775395419-9307\sqrt{\Delta}}{37076493176}\right)^{2097152} \cdot \left(\frac{1948592556958649-20327\sqrt{\Delta}}{4923123394}\right)^{1048576} \\
&\cdot \left(\frac{187890982387507+6461\sqrt{\Delta}}{531539504}\right)^{524288} \cdot \left(\frac{41256134082263-745\sqrt{\Delta}}{16239356786}\right)^{262144} \cdot \left(\frac{824981926270519+45303\sqrt{\Delta}}{2600833744}\right)^{131072} \\
&\cdot \left(\frac{114948321723909-5\sqrt{\Delta}}{3906692792}\right)^{65536} \cdot \left(\frac{82371886111397-2011\sqrt{\Delta}}{359187784}\right)^{32768} \cdot \left(\frac{16014355419767-471\sqrt{\Delta}}{536656166}\right)^{16384} \\
&\cdot \left(\frac{7177193010913+1247\sqrt{\Delta}}{10440956480}\right)^{8192} \cdot \left(\frac{206333926302449+4849\sqrt{\Delta}}{349697524}\right)^{4096} \cdot \left(\frac{-93608593139215+3047\sqrt{\Delta}}{8531315618}\right)^{2048} \\
&\cdot \left(\frac{255484276578909+959\sqrt{\Delta}}{1768331300}\right)^{1024} \cdot \left(\frac{133471401368403+5347\sqrt{\Delta}}{6892106956}\right)^{512} \cdot \left(\frac{-578683394012717+24885\sqrt{\Delta}}{11973976402}\right)^{256} \\
&\cdot \left(\frac{1081325852865671+34831\sqrt{\Delta}}{612835160}\right)^{128} \cdot \left(\frac{270335873838779-8379\sqrt{\Delta}}{15304029344}\right)^{64} \cdot \left(\frac{1113997801946117-31739\sqrt{\Delta}}{1994993176}\right)^{32} \\
&\cdot \left(\frac{3016869447226151+95161\sqrt{\Delta}}{23057973010}\right)^{16} \cdot \left(\frac{926719522671943+3482\sqrt{\Delta}}{6370000637}\right)^{8} \cdot (910000091)^4 \cdot (7)^2 \cdot (49)
\end{aligned}
$$

$\underline{\Delta = 10^{25} + 1}$

$$
\begin{aligned}
&\left(\frac{3162277660167+\sqrt{\Delta}}{4361830766056}\right)^{68719476736} \cdot \left(\frac{-344989991700362465934569+109095414375\sqrt{\Delta}}{654572486246}\right)^{34359738368} \\
&\cdot \left(\frac{1432421611714014463-351293\sqrt{\Delta}}{1908589239470}\right)^{17179869184} \cdot \left(\frac{230080107017736554+204829\sqrt{\Delta}}{402570664597}\right)^{8589934592} \\
&\cdot \left(\frac{2275647817138879175+1091527\sqrt{\Delta}}{1298826133808}\right)^{4294967296} \cdot \left(\frac{1004510666609478699+374485\sqrt{\Delta}}{1650671675932}\right)^{2147483648} \\
&\cdot \left(\frac{1465262347548161537-393403\sqrt{\Delta}}{879848508410}\right)^{1073741824} \cdot \left(\frac{600293047451622363+385187\sqrt{\Delta}}{725545823326}\right)^{536870912} \\
&\cdot \left(\frac{-202819738385913093+276379\sqrt{\Delta}}{2745800476384}\right)^{268435456} \cdot \left(\frac{741738827475806115+45533\sqrt{\Delta}}{561787436498}\right)^{134217728} \\
&\cdot \left(\frac{-539985901969985857+499153\sqrt{\Delta}}{1742646295810}\right)^{67108864} \cdot \left(\frac{-183581404772606591+1116191\sqrt{\Delta}}{2045748061976}\right)^{33554432} \\
&\cdot \left(\frac{737290058199282977+664513\sqrt{\Delta}}{1850465889530}\right)^{16777216} \cdot \left(\frac{211764080276499417+43297\sqrt{\Delta}}{76215000952}\right)^{8388608} \\
&\cdot \left(\frac{-25796516384614871+10204\sqrt{\Delta}}{165601725559}\right)^{4194304} \cdot \left(\frac{6578319163379279657+561879\sqrt{\Delta}}{2857136363114}\right)^{2097152} \\
&\cdot \left(\frac{500642088140212964+46123\sqrt{\Delta}}{112391413583}\right)^{1048576} \cdot \left(\frac{62616900828421231+28927\sqrt{\Delta}}{176017135856}\right)^{524288} \\
&\cdot \left(\frac{-28525612194042199+9067\sqrt{\Delta}}{17340298762}\right)^{262144} \cdot \left(\frac{522489286455005329+108399\sqrt{\Delta}}{1010005834430}\right)^{131072} \\
&\cdot \left(\frac{67835197951851828+214397\sqrt{\Delta}}{1784350022969}\right)^{65536} \cdot \left(\frac{-449725674852548085+405767\sqrt{\Delta}}{226799386012}\right)^{32768} \\
&\cdot \left(\frac{117424284041754997-22389\sqrt{\Delta}}{341218392704}\right)^{16384} \cdot \left(\frac{-64444676480372073+58007\sqrt{\Delta}}{1013312936930}\right)^{8192} \\
&\cdot \left(\frac{17077304740886151-976\sqrt{\Delta}}{7692847927}\right)^{4096} \cdot \left(\frac{306601984116342445+37853\sqrt{\Delta}}{1717270442306}\right)^{2048} \\
&\cdot \left(\frac{566032883766010537+1448009\sqrt{\Delta}}{3500641401896}\right)^{1024} \cdot \left(\frac{5052098553991953197+528499\sqrt{\Delta}}{3709756579426}\right)^{512} \\
&\cdot \left(\frac{-1624393155775649065+1321677\sqrt{\Delta}}{2155112792908}\right)^{256} \cdot \left(\frac{3549345250618018901-1076499\sqrt{\Delta}}{434642400650}\right)^{128} \\
&\cdot \left(\frac{385502262413901797+114453\sqrt{\Delta}}{186509212688}\right)^{64} \cdot \left(\frac{-54718670301036005+51173\sqrt{\Delta}}{1333458028232}\right)^{32} \\
&\cdot \left(\frac{-853557642319337769+545303\sqrt{\Delta}}{2525144759404}\right)^{16} \cdot \left(\frac{141836503152097743+86188\sqrt{\Delta}}{135917428295}\right)^{8} \\
&\cdot \left(\frac{409170500679284927-32527\sqrt{\Delta}}{530625083354}\right)^{4} \cdot \left(\frac{-56039436640561+25\sqrt{\Delta}}{5522}\right)^{2} \cdot (2761)
\end{aligned}
$$

$\underline{\Delta = 10^{31} + 1}$

$$
\begin{aligned}
&\left(\frac{3162277660168379+\sqrt{\Delta}}{1049872684256180}\right)^{281474976710656} \\
&\cdot \left(\frac{-18633527374581134464714032509+5892438734691\sqrt{\Delta}}{2910864734933992}\right)^{140737488355328} \\
&\cdot \left(\frac{136471462650017698471461-18505733\sqrt{\Delta}}{3587772710071226}\right)^{70368744177664} \\
&\cdot \left(\frac{-492529088163422818135+24057751\sqrt{\Delta}}{899232509577152}\right)^{35184372088832} \\
&\cdot \left(\frac{63968418699805147871129-13240423\sqrt{\Delta}}{5784851138437456}\right)^{17592186044416} \\
&\cdot \left(\frac{7315644878116848259731+495533\sqrt{\Delta}}{6103557660358}\right)^{8796093022208} \\
&\cdot \left(\frac{3445422864097353934629+1095323\sqrt{\Delta}}{1696301225916496}\right)^{4398046511104} \\
&\cdot \left(\frac{-34899870495317339242617+14039879\sqrt{\Delta}}{1047016272488738}\right)^{2199023255552}
\end{aligned}
$$

$$
\begin{aligned}
&\cdot\left(\frac{318752304044515650885 77+17086809\sqrt{\Delta}}{868219899559054}\right)^{1099511627776}\\
&\cdot\left(\frac{-2955718785501000549086 1+10030205\sqrt{\Delta}}{351344525551888}\right)^{549755813888}\\
&\cdot\left(\frac{1201294615407784099291 1+5529359\sqrt{\Delta}}{2615413852060430}\right)^{274877906944}\\
&\cdot\left(\frac{1056212170603696702386 83+13323333\sqrt{\Delta}}{2742747122753888}\right)^{137438953472}\\
&\cdot\left(\frac{-1345398420789798127344 35+43391773\sqrt{\Delta}}{386826129119914}\right)^{68719476736}\\
&\cdot\left(\frac{2663431880384468309114 9+5434561\sqrt{\Delta}}{2767025868571130}\right)^{34359738368}\\
&\cdot\left(\frac{7565639425510984550620 1+27364841\sqrt{\Delta}}{576134941709188}\right)^{17179869184}\\
&\cdot\left(\frac{1054175692992332575809 33+11883283\sqrt{\Delta}}{2338010249513318}\right)^{8589934592}\\
&\cdot\left(\frac{835411742938780525841+202239\sqrt{\Delta}}{7716450442640}\right)^{4294967296}\cdot\left(\frac{184922234134104322392 27-5376773\sqrt{\Delta}}{333212318379170}\right)^{2147483648}\\
&\cdot\left(\frac{1592045191704934015822 1+6973529\sqrt{\Delta}}{4194169792909220}\right)^{1073741824}\cdot\left(\frac{223405873894315093970 5+372699\sqrt{\Delta}}{20476156362886}\right)^{536870912}\\
&\cdot\left(\frac{5568718686282372742150 9-16889741\sqrt{\Delta}}{474019137758542}\right)^{268435456}\cdot\left(\frac{610321845322253521690 7+2933207\sqrt{\Delta}}{434259272785100}\right)^{134217728}\\
&\cdot\left(\frac{520092993274051755441+815809\sqrt{\Delta}}{270862604213776}\right)^{67108864}\cdot\left(\frac{242956963841372019229 45+4009087\sqrt{\Delta}}{731861241495832}\right)^{33554432}\\
&\cdot\left(\frac{8035968880083443068684 7+23736927\sqrt{\Delta}}{3074049521916590}\right)^{16777216}\cdot\left(\frac{504779688774720736277 12-12853887\sqrt{\Delta}}{379183933593487}\right)^{8388608}\\
&\cdot\left(\frac{-2078199634327989889900 7+8456613\sqrt{\Delta}}{985014483816940}\right)^{4194304}\cdot\left(\frac{119436493565340039431 307+37853393\sqrt{\Delta}}{262684417019738}\right)^{2097152}\\
&\cdot\left(\frac{-2738881811725443912437 51+86959319\sqrt{\Delta}}{4380207538746020}\right)^{1048576}\cdot\left(\frac{626821585131729872293 9-1089589\sqrt{\Delta}}{48588449810948}\right)^{524288}\\
&\cdot\left(\frac{2186796799641539797667 9+8685065\sqrt{\Delta}}{809329591969628}\right)^{262144}\cdot\left(\frac{359515203955026388364 57-221087\sqrt{\Delta}}{3945021810022090}\right)^{131072}\\
&\cdot\left(\frac{-3761528820282441624251 3+20258237\sqrt{\Delta}}{345565469975060}\right)^{65536}\cdot\left(\frac{216880344921265259078 7+464963\sqrt{\Delta}}{42570731848160}\right)^{32768}\\
&\cdot\left(\frac{2273199270063824196099+612349\sqrt{\Delta}}{1564583782697150}\right)^{16384}\cdot\left(\frac{176926887100846614055 539+52156961\sqrt{\Delta}}{3349483259548340}\right)^{8192}\\
&\cdot\left(\frac{3249131341817020464869 3-5648947\sqrt{\Delta}}{656544187291874}\right)^{4096}\cdot\left(\frac{859700771133414532090 97+21104897\sqrt{\Delta}}{545029216378984}\right)^{2048}\\
&\cdot\left(\frac{1626621541475117407499 1-2983583\sqrt{\Delta}}{2364154749323878}\right)^{1024}\cdot\left(\frac{178730558459400087551 433-14345993\sqrt{\Delta}}{2673583819932080}\right)^{512}\\
&\cdot\left(\frac{2125468459861904479974 3-831457\sqrt{\Delta}}{497868238121954}\right)^{256}\cdot\left(\frac{-222611971638931397377 91+8750561\sqrt{\Delta}}{136240391249930}\right)^{128}\\
&\cdot\left(\frac{486074941501843358423+277783\sqrt{\Delta}}{288428581384384}\right)^{64}\cdot\left(\frac{812057008352918915583 7+1260547\sqrt{\Delta}}{240669510287624}\right)^{32}\\
&\cdot\left(\frac{-8271693717492045520676 7+30945233\sqrt{\Delta}}{3776101069279060}\right)^{16}\cdot\left(\frac{-573416155280415313950 7+4145607\sqrt{\Delta}}{38987462528306}\right)^{8}\\
&\cdot\left(\frac{1706061829441632227933+522961\sqrt{\Delta}}{231266210046676}\right)^{4}\cdot\left(\frac{-766032281273985+\sqrt{\Delta}}{22}\right)^{2}\cdot(11)
\end{aligned}
$$

$\underline{\Delta = 10^{35} + 1}$

$$
\begin{aligned}
&\left(\frac{316227766016837933+\sqrt{\Delta}}{63210564006143756}\right)^{1125899906842624}\\
&\cdot\left(\frac{-174124548969931193152165427000043+55063017129523\sqrt{\Delta}}{397775235743413510}\right)^{562949953421312}\\
&\cdot\left(\frac{840740937459652757385367+15446927\sqrt{\Delta}}{1463352177669064}\right)^{281474976710656}\\
&\cdot\left(\frac{249059128420357798274320 9-5473641\sqrt{\Delta}}{119808385933001684}\right)^{140737488355328}\\
&\cdot\left(\frac{4928039838337304980716580 1+37486897\sqrt{\Delta}}{318799366782119014}\right)^{70368744177664}\\
&\cdot\left(\frac{4491575923198326329993197+17485051\sqrt{\Delta}}{2250900229214344}\right)^{35184372088832}\\
&\cdot\left(\frac{-7950734851207787841002867+25308877\sqrt{\Delta}}{2739538464120980}\right)^{17592186044416}\\
&\cdot\left(\frac{6811798103431992850482541+21323291\sqrt{\Delta}}{248450546383641730}\right)^{8796093022208}\\
&\cdot\left(\frac{6876663742100674564645674 7+66992728\sqrt{\Delta}}{277350341230728569}\right)^{4398046511104}\\
&\cdot\left(\frac{-1807572087483974433099080 23+936014155\sqrt{\Delta}}{357103402533930068}\right)^{2199023255552}\\
&\cdot\left(\frac{1156051196570329447460034 9+311069797\sqrt{\Delta}}{149664113755178434}\right)^{1099511627776}\\
&\cdot\left(\frac{1329019870611625293153255+255382409\sqrt{\Delta}}{582182257771261552}\right)^{549755813888}\\
&\cdot\left(\frac{3073617695864803260197868 1-68174375\sqrt{\Delta}}{11328103870880422}\right)^{274877906944}\\
&\cdot\left(\frac{4549121158338208644252843+6825955\sqrt{\Delta}}{62478161014204718}\right)^{137438953472}\\
&\cdot\left(\frac{2351052859927902514826916 83-649631133\sqrt{\Delta}}{418610401115146400}\right)^{68719476736}\\
&\cdot\left(\frac{7159218602718263755224025 29+2232341279\sqrt{\Delta}}{162175117410891188}\right)^{34359738368}\\
&\cdot\left(\frac{-4873194615357993528854151 5+235822517\sqrt{\Delta}}{242306482519608512}\right)^{17179869184}\\
&\cdot\left(\frac{1020087688841467267591678 3-25880559\sqrt{\Delta}}{2526044651762878}\right)^{8589934592}\\
&\cdot\left(\frac{3076244292479440833196 98+737345\sqrt{\Delta}}{25240962610544899}\right)^{4294967296}\\
&\cdot\left(\frac{5002453278012708328360923 7+243063915\sqrt{\Delta}}{167042050463409808}\right)^{2147483648}\\
&\cdot\left(\frac{2086530660657932080852965 5+119402103\sqrt{\Delta}}{70983302552966962}\right)^{1073741824}\\
&\cdot\left(\frac{5906198707161263135273845 1-153766047\sqrt{\Delta}}{446120776908335636}\right)^{536870912}\\
&\cdot\left(\frac{-4389648007738003940546216 1+445496903\sqrt{\Delta}}{180077486351573506}\right)^{268435456}\\
&\cdot\left(\frac{1514420866031660639015852 1+5740949\sqrt{\Delta}}{13941772288800380}\right)^{134217728}\\
&\cdot\left(\frac{3030474745814547375795 85+1317167\sqrt{\Delta}}{21004746704830628}\right)^{67108864}\\
&\cdot\left(\frac{-5850572196835810073936492 69+1918735731\sqrt{\Delta}}{187581260846859526}\right)^{33554432}\\
&\cdot\left(\frac{2066353671233855169955282 33+629705313\sqrt{\Delta}}{173093485771081640}\right)^{16777216}
\end{aligned}
$$

$$
\begin{aligned}
&\cdot\left(\frac{53893368866937254844062583+75428617\sqrt{\Delta}}{155904003399562904}\right)^{8388608}\\
&\cdot\left(\frac{11366671446801355371760235 1-340026337\sqrt{\Delta}}{223537848410374558}\right)^{4194304}\\
&\cdot\left(\frac{14874029858605001969501803 1+775438623\sqrt{\Delta}}{380302791001640356}\right)^{2097152}\\
&\cdot\left(\frac{9609686678706455050304491 1+120944961\sqrt{\Delta}}{107471866276847300}\right)^{1048576}\\
&\cdot\left(\frac{1446000824084087483693499+665251\sqrt{\Delta}}{354394232530586}\right)^{524288}\\
&\cdot\left(\frac{-2488153037785379443933 55+903591\sqrt{\Delta}}{314320974211316572}\right)^{262144}\\
&\cdot\left(\frac{-21109864059398436188159797+72296251\sqrt{\Delta}}{1559721529731476}\right)^{131072}\\
&\cdot\left(\frac{2449244380120984127570067+7682393\sqrt{\Delta}}{159297189769008910}\right)^{65536}\\
&\cdot\left(\frac{28583634759899591495704969+303222319\sqrt{\Delta}}{165067132561635914}\right)^{32768}\\
&\cdot\left(\frac{7770347845393592423833133+62803531\sqrt{\Delta}}{24519991543880264}\right)^{16384}\\
&\cdot\left(\frac{5283832401712555551651667-15193061\sqrt{\Delta}}{64347749946718114}\right)^{8192}\\
&\cdot\left(\frac{3301102714443667079688493+25007641\sqrt{\Delta}}{12471748320835042}\right)^{4096}\\
&\cdot\left(\frac{-12030508015518910962436951+54170985\sqrt{\Delta}}{59756385057217526}\right)^{2048}\\
&\cdot\left(\frac{14601358506563994731201597+188840685\sqrt{\Delta}}{469482576108336508}\right)^{1024}\\
&\cdot\left(\frac{8837653640270181543871497 5+52632133\sqrt{\Delta}}{136713672066648746}\right)^{512}\\
&\cdot\left(\frac{-24324173297382010778287 15+85225819\sqrt{\Delta}}{19272422610606548}\right)^{256}\\
&\cdot\left(\frac{9503673757913378131368937-25388591\sqrt{\Delta}}{139256490241339294}\right)^{128}\\
&\cdot\left(\frac{6549912491987121031291526+9898123\sqrt{\Delta}}{6828266918067083}\right)^{64}\\
&\cdot\left(\frac{1964726620107850722508614 3-60035093\sqrt{\Delta}}{274463316489940100}\right)^{32}\\
&\cdot\left(\frac{13586427506456187740404293+305098043\sqrt{\Delta}}{242237541340853920}\right)^{16}\\
&\cdot\left(\frac{9598022171460295740631277+47899373\sqrt{\Delta}}{4680137763631682}\right)^{8}\\
&\cdot\left(\frac{-1845806645153866784956303+8774293\sqrt{\Delta}}{391880974784813420}\right)^{4}\\
&\cdot\left(\frac{126060145257710537897+1247\sqrt{\Delta}}{1818182}\right)^{2}\cdot(909091)
\end{aligned}
$$

$\underline{\Delta = 10^{41} + 1}$

$$\left(\frac{3162277660168379331990+\sqrt{\Delta}}{281238568793006813200}\right)^{4611686018427387904}$$

$$\cdot\left(\frac{-523748680835043450323552951831621683349+16562387529473165 1\sqrt{\Delta}}{2981229755305169710}\right)^{2305843009213693952}$$

$$\cdot\left(\frac{-1141042183114642624847524931 67+430276817\sqrt{\Delta}}{7727002227280497 7972}\right)^{1152921504606846976}$$

$$\cdot\left(\frac{-1162621151466387255817514052 23+415057739\sqrt{\Delta}}{24857657446359260 02}\right)^{576460752303423488}$$

$$\cdot\left(\frac{67735974563393149471266820043+215483314\sqrt{\Delta}}{35697252970758178747}\right)^{288230376151711744}$$

$$\cdot\left(\frac{377603276146482306252647029099+3174283005\sqrt{\Delta}}{84853173002646811642}\right)^{144115188075855872}$$

$$\cdot\left(\frac{443875938327955040512422833 8169-13565662825\sqrt{\Delta}}{45133787066500941506}\right)^{72057594037927936}$$

$$\cdot\left(\frac{175275462435437856548169361 3291+5985377021\sqrt{\Delta}}{125260265302262240330}\right)^{36028797018963968}$$

$$\cdot\left(\frac{121540789066626756855365735 2369-3154814744\sqrt{\Delta}}{122862107320320152585}\right)^{18014398509481984}$$

$$\cdot\left(\frac{237738255296182583369446798 0877+8960195802\sqrt{\Delta}}{157439399860982029067}\right)^{9007199254740992}$$

$$\cdot\left(\frac{203128330119374484330163690 5534-5763346645\sqrt{\Delta}}{32456128271221684579}\right)^{4503599627370496}$$

$$\cdot\left(\frac{-83533178275446421030068056596+400925921\sqrt{\Delta}}{8635243283318010025}\right)^{2251799813685248}$$

$$\cdot\left(\frac{455705814792511009898746329103+1411500353\sqrt{\Delta}}{56555961313556270128}\right)^{1125899906842624}$$

$$\cdot\left(\frac{85836133625701829337071366 8111-958444017\sqrt{\Delta}}{40325603452978638 6896}\right)^{562949953421312}$$

$$\cdot\left(\frac{27418251099587839604723579 81+15734915\sqrt{\Delta}}{133166235040081874}\right)^{281474976710656}$$

$$\cdot\left(\frac{-987422102759308407977401314137+4346302105\sqrt{\Delta}}{261386868796451391668}\right)^{140737488355328}$$

$$\cdot\left(\frac{401760925510322998080406905 6975+5317637039\sqrt{\Delta}}{389720568832987095992}\right)^{70368744177664}$$

$$\cdot\left(\frac{-453137526201832421218998704003+3840684587\sqrt{\Delta}}{66880929880073572070}\right)^{35184372088832}$$

$$\cdot\left(\frac{711907377566917325419716123509+10270713291\sqrt{\Delta}}{280622934065095096828}\right)^{17592186044416}$$

$$\cdot\left(\frac{118931478995942861818985460 7869+7820088137\sqrt{\Delta}}{238778619695629054898}\right)^{8796093022208}$$

$$\cdot\left(\frac{-320447103200046995068664797025+1212164599\sqrt{\Delta}}{38803628574337 0322}\right)^{4398046511104}$$

$$\cdot\left(\frac{1749040884047517015384738899+14610173\sqrt{\Delta}}{242894458177018941584}\right)^{2199023255552}$$

$$\cdot\left(\frac{57159120687583624677192907519+227985487\sqrt{\Delta}}{523565513256120838}\right)^{1099511627776}$$

$$\cdot\left(\frac{-21432724663467125281866083285+72108219\sqrt{\Delta}}{55265623693973878886}\right)^{549755813888}$$

$$\cdot\left(\frac{-522764816701086597154012025 6321+22344495039\sqrt{\Delta}}{231225451487685810340}\right)^{274877906944}$$

$$\cdot\left(\frac{26210239136923059604913194 6673+211070427\sqrt{\Delta}}{4806309878754716194}\right)^{137438953472}$$

$$\cdot\left(\frac{-87303654674281229548291478 21+121484309\sqrt{\Delta}}{30294095488247415770}\right)^{68719476736}$$

$$\cdot\left(\frac{119908372842199033700865347167+34098108\sqrt{\Delta}}{62160832745727033481}\right)^{34359738368}$$

$$\cdot\left(\frac{126432014504705154223757614 9149+4503136231\sqrt{\Delta}}{555540362954007497 80}\right)^{17179869184}$$

$$\cdot\left(\frac{-233314691039109186392854493377+1089451623\sqrt{\Delta}}{416393481274756103 50}\right)^{8589934592}$$

$$\cdot\left(\frac{39073630501982791756625350 0389-202554139\sqrt{\Delta}}{171379647269196840704}\right)^{4294967296}$$

$$\cdot\left(\frac{-546855106340891893394060142187+4688176747\sqrt{\Delta}}{129301057548988899380}\right)^{2147483648}$$

$$
\begin{aligned}
&\cdot\left(\frac{5091072265026020203357225197653+1241350293\sqrt{\Delta}}{6286057769111477502}\right)^{1073741824}\\
&\cdot\left(\frac{23044152066327516010865276614\,87+5669251537\sqrt{\Delta}}{424408652780017015 54}\right)^{536870912}\\
&\cdot\left(\frac{9220381282245458695191007606 01-2708492018\sqrt{\Delta}}{2588488975470673276 13}\right)^{268435456}\\
&\cdot\left(\frac{124673284286748422992763606661 39+38846259227\sqrt{\Delta}}{338127348221135267 84}\right)^{134217728}\\
&\cdot\left(\frac{-14282325050750653869732045315 09+4547627317\sqrt{\Delta}}{494066426713980974 86}\right)^{67108864}\\
&\cdot\left(\frac{18153010589679350694316627610 3+1343345654\sqrt{\Delta}}{241710098936231571043}\right)^{33554432}\\
&\cdot\left(\frac{3411084409399380648714508354903-7017556810\sqrt{\Delta}}{114865683410296633541}\right)^{16777216}\\
&\cdot\left(\frac{4234593628842599440312297205797+15679508033\sqrt{\Delta}}{252116525371928898740}\right)^{8388608}\\
&\cdot\left(\frac{3390049052208883557294541690549-1841158701\sqrt{\Delta}}{350942883363988644530}\right)^{4194304}\\
&\cdot\left(\frac{-3114171293859876526290980787581+12499037806\sqrt{\Delta}}{192415982828611959787}\right)^{2097152}\\
&\cdot\left(\frac{2786444891851799441252793443183+3186375201\sqrt{\Delta}}{91143484865938622576}\right)^{1048576}\\
&\cdot\left(\frac{-3809005399338801909168463271 47+1274644821\sqrt{\Delta}}{83719458016546592 78}\right)^{524288}\\
&\cdot\left(\frac{4484306326454858216826484324 69+1235861357\sqrt{\Delta}}{344949807965903225434}\right)^{262144}\\
&\cdot\left(\frac{4303594983449779967460077344 99+1893989968\sqrt{\Delta}}{58327577954353686 07}\right)^{131072}\\
&\cdot\left(\frac{-832244847665042421701209688 6+68196661\sqrt{\Delta}}{116344190270807943 25}\right)^{65536}\\
&\cdot\left(\frac{-1003539788314247298713018181 9+62772181\sqrt{\Delta}}{54175180438294520 48}\right)^{32768}\\
&\cdot\left(\frac{-4573120855723356840526797432 1+253322929\sqrt{\Delta}}{117914293351563575 60}\right)^{16384}\\
&\cdot\left(\frac{1344776585935288196873806235 17-126646467\sqrt{\Delta}}{237062255257069315 454}\right)^{8192}\\
&\cdot\left(\frac{3097187695783377528866953639183+13237661589\sqrt{\Delta}}{282249348137239559404}\right)^{4096}\\
&\cdot\left(\frac{3068987246154162132290701932507+1568612123\sqrt{\Delta}}{230280888482241764240}\right)^{2048}\\
&\cdot\left(\frac{6791460269931684593627923324 99-1552262451\sqrt{\Delta}}{33232572304281726158}\right)^{1024}\\
&\cdot\left(\frac{1682864254470690435979066104059+799513893\sqrt{\Delta}}{313303623317912966486}\right)^{512}\\
&\cdot\left(\frac{2245783535918476461620358054 97+10938772625\sqrt{\Delta}}{242773861769278076792}\right)^{256}\\
&\cdot\left(\frac{-2616017589951872536561738239 87+1853983955\sqrt{\Delta}}{93415049170652556 58}\right)^{128}\\
&\cdot\left(\frac{9042428841154421734403444630 8-248597805\sqrt{\Delta}}{91514100213915454879}\right)^{64}\\
&\cdot\left(\frac{6888304878884422364067637801 87+139307646\sqrt{\Delta}}{56424637714064720533}\right)^{32}\\
&\cdot\left(\frac{8463951656370083487675735910 13-558326013\sqrt{\Delta}}{107611180316524505000}\right)^{16}\\
&\cdot\left(\frac{-3447499075481721709047308329+48394796\sqrt{\Delta}}{12286960342856870 95}\right)^{8}\\
&\cdot\left(\frac{1938928676038591447644262415 57+1060905493\sqrt{\Delta}}{96974335562945090636}\right)^{4}\\
&\cdot\left(\frac{7470198077542716686465481383 01+1678661563\sqrt{\Delta}}{58751061190717873934}\right)^{2}\\
&\cdot(29375530595358936967)
\end{aligned}
$$

$\underline{\Delta = 10^{45} + 1}$

$$
\begin{aligned}
&\left(\frac{31622776601683793319987+\sqrt{\Delta}}{61204123583578113159916}\right)^{576460752303423488}\\
&\cdot\left(\frac{-27270753947833010867205204646867727+871101410169\sqrt{\Delta}}{8074676456919295194394}\right)^{288230376151711744}\\
&\cdot\left(\frac{-112871524055854515189242034232979+6153454333\sqrt{\Delta}}{770701386198674873936}\right)^{144115188075855872}\\
&\cdot\left(\frac{63786082748338981695992212093481+766866135\sqrt{\Delta}}{11719509111845418719732}\right)^{72057594037927936}\\
&\cdot\left(\frac{-341745771675563941347717294242783+18149304329\sqrt{\Delta}}{3095913846338541756146}\right)^{36028797018963968}\\
&\cdot\left(\frac{1486024196369092252016674906707799+47687545029\sqrt{\Delta}}{13737343599355158586180}\right)^{18014398509481984}\\
&\cdot\left(\frac{-20848179047350281494588343383709+3525440984\sqrt{\Delta}}{1016908027632671241623}\right)^{9007199254740992}\\
&\cdot\left(\frac{-56122840253679417928072327274203+1807930957\sqrt{\Delta}}{14365260198903063770}\right)^{4503599627370496}\\
&\cdot\left(\frac{6030205986902776531659597431299+184498251\sqrt{\Delta}}{5630383149173266870898}\right)^{2251799813685248}\\
&\cdot\left(\frac{655513984078371714470116359820099+72810311075\sqrt{\Delta}}{19209212118219234546482}\right)^{1125899906842624}\\
&\cdot\left(\frac{1084828652700862156177839451880062-10026809987\sqrt{\Delta}}{11667580312808932709675}\right)^{562949953421312}\\
&\cdot\left(\frac{412907709094306640509185889353283+10067263283\sqrt{\Delta}}{253954866006324405808}\right)^{281474976710656}\\
&\cdot\left(\frac{-63280451654650954627935742967147+2006425131\sqrt{\Delta}}{1322699842295379622322}\right)^{140737488355328}\\
&\cdot\left(\frac{60151225593008765224895848932633+1720582613\sqrt{\Delta}}{751931753297754183260}\right)^{70368744177664}\\
&\cdot\left(\frac{-2999197853205337910607320865037+1211913037\sqrt{\Delta}}{645443272948637133130}\right)^{35184372088832}\\
&\cdot\left(\frac{-178806523675553925524042220225661+5949611189\sqrt{\Delta}}{4112007813804329023886}\right)^{17592186044416}\\
&\cdot\left(\frac{29696435250563208856981202428940-134672963\sqrt{\Delta}}{1430322383519963093633}\right)^{8796093022208}\\
&\cdot\left(\frac{-644528442828337435029018816915148+47186996549\sqrt{\Delta}}{18067640280834061349977}\right)^{4398046511104}\\
&\cdot\left(\frac{10129706461312322789226133494067319-263157815405\sqrt{\Delta}}{51095079451771795155892}\right)^{2199023255552}\\
&\cdot\left(\frac{59090274932755588967196198521679+5101304957\sqrt{\Delta}}{379741050545291461918}\right)^{1099511627776}\\
&\cdot\left(\frac{624058020163749346070868121876507-11889405935\sqrt{\Delta}}{14218361840136604875806}\right)^{549755813888}\\
&\cdot\left(\frac{3168838858113674443759100426627567+40410112129\sqrt{\Delta}}{20796614366201423854084}\right)^{274877906944}\\
&\cdot\left(\frac{1614667306921337177798223458547731+30778128823\sqrt{\Delta}}{15351310862981911488038}\right)^{137438953472}\\
&\cdot\left(\frac{7458152136879769950280085382822583-197686029073\sqrt{\Delta}}{35101575328159692340870}\right)^{68719476736}\\
&\cdot\left(\frac{1017172947247860783139850070360401+26156163586\sqrt{\Delta}}{5689311805556635416649}\right)^{34359738368}\\
&\cdot\left(\frac{-150222701326827935947005122988357+13277211121\sqrt{\Delta}}{2374508751278632135396}\right)^{17179869184}\\
&\cdot\left(\frac{4866938003885411252818648284269337+191294536837\sqrt{\Delta}}{14650127067048555210218}\right)^{8589934592}\\
&\cdot\left(\frac{79023318369107526024480727281409+89296695\sqrt{\Delta}}{58116951943782697688}\right)^{4294967296}\\
&\cdot\left(\frac{-18120768424904842189370223368867+1273866077\sqrt{\Delta}}{47903102748852877519370}\right)^{2147483648}
\end{aligned}
$$

$$
\begin{aligned}
&\cdot\left(\frac{6580263899140129960573237748963043+323840509993\sqrt{\Delta}}{536650615023621499797 16}\right)^{1073741824}\\
&\cdot\left(\frac{1645887499543903086327535350422353+101222180157\sqrt{\Delta}}{1046825582043717876761 0}\right)^{536870912}\\
&\cdot\left(\frac{1500444895022298809046228042434247-34000030453\sqrt{\Delta}}{999533704504253284175 8}\right)^{268435456}\\
&\cdot\left(\frac{-1022906466645470677302492008426273+53997457099\sqrt{\Delta}}{187113232796308574075 98}\right)^{134217728}\\
&\cdot\left(\frac{6720833114678999694131465449237 83+88429157047\sqrt{\Delta}}{105223268123182473051 40}\right)^{67108864}\\
&\cdot\left(\frac{752422322664473876282830546663911+7545305461\sqrt{\Delta}}{183963386362549345003 66}\right)^{33554432}\\
&\cdot\left(\frac{882096528829306416381940096023 15+2157785461\sqrt{\Delta}}{133888105957587964768}\right)^{16777216}\\
&\cdot\left(\frac{1048631108383021729393998315397 11+106811647\sqrt{\Delta}}{291457111179709355552 2}\right)^{8388608}\\
&\cdot\left(\frac{472368811510150020127109118694167-8383531069\sqrt{\Delta}}{179933670740454551902 42}\right)^{4194304}\\
&\cdot\left(\frac{5998561575185318506288373361569843+136158634211\sqrt{\Delta}}{269389354569611777268 76}\right)^{2097152}\\
&\cdot\left(\frac{1538368082213792131869374889055133+31325490323\sqrt{\Delta}}{381777068807616824422 0}\right)^{1048576}\\
&\cdot\left(\frac{-641577024690518106800626633112 31+2420568256\sqrt{\Delta}}{191329843033250406220 7}\right)^{524288}\\
&\cdot\left(\frac{468455436282560729465751296992 21+2920792985\sqrt{\Delta}}{432738811192173835054}\right)^{262144}\\
&\cdot\left(\frac{-766884375226026139093943561046 31+11021658953\sqrt{\Delta}}{192903704443535874064 04}\right)^{131072}\\
&\cdot\left(\frac{-4573561243124063830684644947940 25+41480088381\sqrt{\Delta}}{162466907374502557663 34}\right)^{65536}\\
&\cdot\left(\frac{4994171030504289408917060253991495+196493762353\sqrt{\Delta}}{258908837438342609137 82}\right)^{32768}\\
&\cdot\left(\frac{2422451658857449676649525808022617-27277717263\sqrt{\Delta}}{152884044845564359739 60}\right)^{16384}\\
&\cdot\left(\frac{-2096634168042757833012938546105 69+26156028169\sqrt{\Delta}}{547781211679817591693 6}\right)^{8192}\\
&\cdot\left(\frac{695065073299692805908702023922963+25161636179\sqrt{\Delta}}{999735730926462410676 4}\right)^{4096}\\
&\cdot\left(\frac{566368363312474056103732662955563+20595897355\sqrt{\Delta}}{827780374866435822428 8}\right)^{2048}\\
&\cdot\left(\frac{791596898188979081121326530692657-12408587825\sqrt{\Delta}}{603559166522585640598 4}\right)^{1024}\\
&\cdot\left(\frac{1328235943845303305859298512363467+24810596811\sqrt{\Delta}}{128700443951814910699 4}\right)^{512}\\
&\cdot\left(\frac{235118001496853625360039535969354-7210901645\sqrt{\Delta}}{792902895933080156669 9}\right)^{256}\\
&\cdot\left(\frac{490366587769221692980942564566220+28255778733\sqrt{\Delta}}{887440872436627961948 9}\right)^{128}\\
&\cdot\left(\frac{1095201522425336921965784465219225+15742073719\sqrt{\Delta}}{604185078004258218779 2}\right)^{64}\\
&\cdot\left(\frac{493205875991599175320619448800683-10542603861\sqrt{\Delta}}{723787429488307682302 4}\right)^{32}\\
&\cdot\left(\frac{201132824382179037939241771143215-3357361487\sqrt{\Delta}}{445646969821155738209 8}\right)^{16}\\
&\cdot\left(\frac{1830076097168953969686839518449489+47527256015\sqrt{\Delta}}{686261171835621364257 8}\right)^{8}\\
&\cdot\left(\frac{131685151987455419343709329285717+1483906327\sqrt{\Delta}}{450035119211290723886 0}\right)^{4}\\
&\cdot\left(\frac{434359442329636393989 2279+271\sqrt{\Delta}}{769886}\right)^{2}\\
&\cdot(2694601)
\end{aligned}
$$

Bibliography

[Abe94] Christine Abel. *Ein Algorithmus zur Berechnung der Klassenzahl und des Regulators reellquadratischer Ordnungen*. PhD thesis, Universität des Saarlandes, 1994.

[ANS85] ANSI/IEEE Std 754-1995. IEEE Standard for Binary Floating-Point Arithmetic, 1985.

[AS64] M. Abramowitz and I.A. Stegun. *Handbook of Mathematical Functions*, volume 55 of *Applied Mathematical Series*. reprinted 1968 by Dover Publications, Ney York, Washington, National Bureau of Standards, 1964.

[Bac95] E. Bach. Improved approximations for Euler products. In *Fourth Conference of the Canadian Number Theory Association*, pages 13–28, 1995.

[BB94] Ingrid Biehl and Johannes Buchmann. Algorithms for quadratic orders. In *Proceedings of Symposia in Applied Mathematics*, volume 48, pages 425 – 448, 1994.

[BB97] Ingrid Biehl and Johannes Buchmann. An analysis of the reduction algorithm for binary quadratic forms. Technical report, Technical University Darmstadt, 1997.

[BBT94] Ingrid Biehl, Johannes Buchmann, and Christoph Thiel. Cryptographic protocols based on discrete logarithms in real-quadratic orders. In Yvo G. Desmedt, editor, *Advances in Cryptology — CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 56–60. Springer–Verlag, 1994.

[BK92] J. Buchmann and V. Kessler. Computing a reduced lattice basis from a generating system. manuscript, 1992.

[BM98] J. Buchmann and M. Maurer. Approximate Evaluation of $L(1, \chi_\Delta)$. Technical report, TU Darmstadt, 1998.

[BMM00] Johannes Buchmann, Markus Maurer, and Bodo Möller. Cryptography based on number fields with large regulator. *Journal de Theorie des Nombres de Bordeaux*, 2000. to appear.

[BMT96] Ingrid Biehl, Bernd Meyer, and Christoph Thiel. Cryptographic Protocols Based on Real-Quadratic A-Fields (Extended Abstract). In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology — ASIACRYPT '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 15–25. Springer–Verlag, 1996.

[Bre76] Richard P. Brent. Fast multiple-precision evaluation of elementary functions. *Journal of the ACM*, 23(2):242 – 251, 1976.

[BS66] Z.I. Borevitch and I.R. Shafarevitch. *Number Theory*. Academic Press, Ney York, 1966.

[BS96] Eric Bach and Jeff Shallit. *Algorithmic number theory*. MIT Press, Cambridge, Massachusetts and London, England, 1996.

[BTW95] Johannes Buchmann, Christoph Thiel, and Hugh Williams. Short representation of quadratic integers. *Computational Algebra and Number Theory, Mathematics and its Applications*, 325:159 – 185, 1995.

[Buc] Johannes Buchmann. *Algorithms for binary quadratic forms*. Springer Verlag. to appear.

[Buc90] Johannes Buchmann. A subexponential algorithm for the determination of class groups and regulators of algebraic number fields. In Catherine Goldstein, editor, *Seminaire de Theorie des Nombres*, pages 27 – 41. Birkhäuser, Paris, 1988 - 1989 edition, 1990.

[Buc99] Johannes Buchmann. *Einführung in die Kryptographie*. Springer Verlag, 1999.

[Bue89] D.A. Buell. *Binary Quadratic Forms*. Springer Verlag, 1989.

[Coh78] Harvey Cohn. *A Classical Inivitation to Algebraic Numbers and Class Fields*. Springer-Verlag, 1978.

[Coh95] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Graduate Texts in Mathematics. Springer Verlag, 1995.

[Gau86] C.F. Gauss. *Disquisitiones Arithmeticae (English edition: translated by A.A. Clark)*. Springer Verlag, 1986.

[Ge93] Guoqiang Ge. *Algorithms Related to Multiplicative Representations of Algebraic Numbers.* PhD thesis, University of California at Berkeley, 1993.

[HP00] Detlef Hühnlein and Sachar Paulus. On the implementation of cryptosystems based on real quadratic number fields (extended abstract). In *Seventh Annual Workshop on Selected Areas in Cryptography SAC 2000.* Centre for Applied Cryptographic Research (CACR), University of Waterloo, Waterloo, Ontario, Canada, 2000.

[Hua42] Loo Keng Hua. On the least solution of Pell's equation. *Bull. Amer. Math. Soc.*, 48:731 – 735, 1942.

[Hua82] L.K. Hua. *Introduction to number theory.* Springer Verlag, 1982.

[Jac99] Micheal J. Jacobson, Jr. *Subexponential Class Group Computation in Quadratic Orders.* PhD thesis, Technische Universität Darmstadt, 1999.

[JLW95] Michael J. Jacobson, Jr., Richard F. Lukes, and Hugh C. Williams. An investigation of bounds for the regulator of quadratic fields. *Experimental Mathematics*, 4(3):211 – 225, 1995.

[Kog60] E.G. Kogbetliantz. Generation of elementary functions. In Anthony Ralston and Herbert S. Wilf, editors, *Mathematical Methods for Digital Computers*, volume 1. Wiley and Sons, 1960.

[Kön84] Heinz König. *Analysis I.* Birkhäuser Verlag, 1984.

[Lag80] J.C. Lagarias. On the computational complexity of determining the solvability or unsolvability of the equation $x^2 - dy^2 = -1$. *Transactions of the American Mathematical Society*, 260(2):485 – 507, August 1980.

[Len82] H.W. Lenstra, Jr. On the calculation of regulators and class numbers of quadratic fields. In *London Math. Soc., Lecture Note Series*, volume 56, pages 123 – 150. 1982.

[LiD] LiDIA. A library for computational number theory. http://www.informatik.tu-darmstadt.de/TI/LiDIA. Technische Universität Darmstadt.

[LL90] A.K. Lenstra and H.W. Lenstra, Jr. Algorithms in number theory. In J. van Leeuwen, editor, *Handbook of TCS*, volume A, pages 675–715. Elsevier, 1990.

[Mol96] Richard A. Mollin. *Quadratics.* Discrete Mathematics. CRC Press, 1996.

[MW92] R.A. Mollin and H.C. Williams. Computation of the class number of a real quadratic field. In *Utilitas Mathematica*, volume 41, pages 259 – 308. 1992.

[Pap93] Thomas Papanikolaou. libF, eine lange Gleitpunktarithmetik, Diplomarbeit, Universität des Saarlandes. *http://www.informatik.tu-darmstadt.de TI/Veroeffentlichung/reports/papa.diplom.ps.gz*, 1993.

[Pap97] Thomas Papanikolaou. *Entwurf und Entwicklung einer objektorientierten Bibliothek für algorithmische Zahlentheorie.* PhD thesis, Univerity of Saarland, 1997.

[Per77] Oskar Perron. *Die Lehre von den Kettenbrüchen, I.* Teubner Verlag, 1977.

[SBW91] Renate Scheidler, Johannes Buchmann, and Hugh C. Williams. Implementation of a key-exchange protocol using real quadratic fields (extended abstract). In Ivan Damgård, editor, *Advances in Cryptology — EUROCRYPT '90*, volume 473 of *Lecture Notes in Computer Science*, pages 98–109. Springer–Verlag, 1991.

[SBW94] Renate Scheidler, Johannes Buchmann, and Hugh C. Williams. A key-exchange protocol using real quadratic fields. *Journal of Cryptology*, 7(3):171–199, 1994.

[Sch71] A. Schönhage. Schnelle Berechnung von Kettenbruchentwicklungen. *Acta Informatica*, 1:139 – 144, 1971.

[Sch82] R.J. Schoof. Quadratic Fields and Factorization. In H.W. Lenstra jr. and R. Tijdeman, editors, *Computational methods in number theory II*, Mathematical Centre Tracts, pages 235 – 286. 1982.

[Sch90] A. Schönhage. Numerik analytischer Funktionen und Komplexität. In *Jahresbericht der Dt. Math.-Verein.*, volume 92, pages 1–20. 1990.

[Sch91] Arnold Schönhage. Fast Reduction and Composition of Binary Quadratic Forms. In *Proceedings of ISSAC'91*, pages 128–133, 1991.

[SGV94] A. Schönhage, A.F.W. Grotefeld, and E. Vetter. *Fast Algorithms, A Multitape Turing machine implementation.* BI, Wissenschaftsverlag, 1994.

[Sha73] Daniel Shanks. The Infrastructure of a Real Quadratic Field and its Applications. In *Proceedings of the 1972 Boulder Conference on Number Theory*, pages 217 – 224, Boulder, Colorado, 1973.

[Sri98] Anitha Srinivasan. Computations of class numbers of real quadratic fields. *Mathematics of computation*, 67(223):1285 – 1308, July 1998.

[SS71] A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7:281 – 292, 1971.

[Vol00] Ulrich Vollmer. Asymptotically Fast Discrete Logarithms in Quadratic Fields. In *Proceedings of ANTS IV*, LNCS. Springer Verlag, 2000. to appear.

[Wil66] J.H. Wilkinson. *Rundungsfehler (german translation of: Rounding Errors in Algebraic Processes)*. Springer Verlag, 1966.

[Zag81] D.B. Zagier. *Zetafunktionen und quadratische Zahlkörper*. Springer-Verlag, 1981.

Index

Curriculum Vitae (Academic Education)

05/09/69	Born in Püttlingen (Saarland), Germany
08/75 - 06/88	School education
06/88	*Abitur*, Warndt-Gymnasium Völklingen
10/89 - 07/95	Studies in computer science at the Universität des Saarlandes
07/95	Diploma in computer science (*Dipl.-Inform.*) Universität des Saarlandes
08/95 - 04/96	Research associate at Prof. J. Buchmann at the Dept. of Computer Science of the Universität des Saarlandes
since 05/96	Research and teaching associate at Prof. J. Buchmann at the Dept. of Computer Science of the Technische Universität Darmstadt