ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta jaderná a fyzikálně inženýrská

Katedra matematiky

VÝZKUMNÝ ÚKOL

Konstrukce algoritmů pro paralelní sčítání

Construction of algorithms for parallel addition

Vypracoval:	Jan Legerský
Školitel:	Ing. Štěpán Starosta, Ph.D.
Akademický rok:	2014/2015

Čestné prohlášení

Prohlašuji na tomto místě, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškerou použitou literaturu.

V Praze dne 2. 9. 2015

Jan Legerský

Poděkování

Děkuji Ing. Štěpánu Starostovi, Ph.D., za vedení mého výzkumného úkolu. Dále děkuji Ing. Mileně Svobodové, Ph.D., a prof. Ing. Editě Pelantové, CSc., za podrobné vysvětlení implementované metody a podnětné diskuze.

Jan Legerský

Název práce: **Konstrukce algoritmů pro paralelní sčítání**

Autor: Jan Legerský

Obor: Inženýrská informatika

Zaměření: Matematická informatika

Druh práce: Výzkumný úkol

Vedoucí práce: Ing. Štěpán Starosta, Ph.D., KAM FIT, ČVUT v Praze

Konzultant: —

Abstrakt: Práce je věnována konstrukci algoritmů pro paralelní sčítání v různých numeračních soustavách. Zaměřuje se hlavně na nestandardní numerační systémy s neceločíselnou, obecně komplexní, bází $\beta \in \mathbb{Z}[\omega]$ a obecně neceločíselnou abecedou $\mathcal{A} \subset \mathbb{Z}[\omega]$ pro nějaké algebraické celé číslo ω. Navrhujeme takzvanou *extending window method* s použitím základního přepisovacího pravidla $x - \beta$, která pro danou bázi β a abecedu $\mathcal{A}$ hledá algoritmus pro paralelní sčítání. Metoda se skládá ze dvou fází. Pro první z nich uvádíme postačující podmínku konvergence, pro druhou máme algoritmus, který kontroluje nutnou podmínku konvergence. Tuto metodu implementujeme v programovacím jazyce SageMath a uvádíme množství otestovaných numeračních systémů.

Klíčová slova: Paralelní sčítání, nestandardní numerační systémy.

Title: **Construction of algorithms for parallel addition**

Author: Jan Legerský

Abstract: We focus on the construction of algorithms for parallel addition in different numeration systems, especially non-standard ones. The base β is an element of $\mathbb{Z}[\omega]$ and the alphabet, in general non-integer, is a subset of $\mathbb{Z}[\omega]$ for some algebraic integer ω. We design so-called extending window method with the basic rewriting rule $x - \beta$. The method searchs for a parallel addition algorithm for a given base β and alphabet $\mathcal{A}$. It consists of two phases. We have the sufficient condition of convergence of Phase 1. We introduce the algorithm which verifies the necessary condition of convergence of Phase 2. The method is implemented in SageMath and we provide several tested examples.

Key words: Parallel addition, non-standard numeration systems.

Contents

List of symbols

Symbol	Description
$\mathbb{N}$	set of nonnegative integers $\{0, 1, 2, 3, \dots\}$
$\mathbb{Z}$	set of integers $\{\dots, -2, -1, 0, 1, 2, \dots\}$
$\mathbb{R}$	set of real numbers
$\mathbb{C}$	set of complex numbers
$\#S$	number of elements of the finite set S
$(\beta, \mathcal{A})$	numeration system with the base β and the alphabet $\mathcal{A}$
$(x)_{\beta,\mathcal{A}}$	$(\beta, \mathcal{A})$-representation of the number x
$\mathrm{Fin}_{\mathcal{A}}(\beta)$	set of all complex numbers with a finite $(\beta, \mathcal{A})$-representation
$\mathcal{A}^{\mathbb{Z}}$	set of all bi-infinite sequences of digits in $\mathcal{A}$
$\mathbb{Z}[\omega]$	set of values of all polynomials with integer coefficients evaluated in ω
$\mathcal{B}$	alphabet of input digits
q_j	weight coefficient for the j-th position
$\mathcal{Q}$	weight coefficients set
$\mathcal{Q}_{[w_j,\dots,w_{j-m+1}]}$	set of possible weight coefficiens for input digits $w_j, \dots, w_{j-m+1}$
$\lfloor x \rfloor$	floor function of the number x
$\mathrm{Re}\, x$	real part of the complex number x
$\mathrm{Im}\, x$	imaginary part of the complex number x

Introduction

Algorithms for parallel addition have been studied to improve performance of computer processing units as well as for the theoretical reasons. Addition is an important part of algorithms for multiplication and division. Thus the linear time of the standard addition algorithms is a serious drawback. A parallel algorithm to compute the sum of $x_n x_{n-1} \cdots x_1 x_0 \bullet$ and $y_n y_{n-1} \cdots y_1 y_0 \bullet$ determines the j-th digit of the sum just from the knowledge of fixed number of digits around x_j and y_j. Thus it avoids a carry propagation and all output digits can be determined at the same time. In contrast, the carry propagation in the standard algorithms requires to compute digits one by one.

A parallel addition algorithm for an integer base $\beta \geq 3$ was introduced by A. Avizienis in [1] in 1961. The algorithm works on the alphabet $\{-a, \ldots, 0, \ldots a\}$ where $a \in \mathbb{N}$ is such that $\beta/2 < a \leq \beta - 1$. Later, C. Y. Chow and J. E. Robertson presented an parallel addition algorithm for the base 2 and the alphabet $\{-1, 0, 1\}$ in [2].

So-called non-standard numeration systems, where the base β is not a positive integer, have been extensively studied. The reason of this interest is for instance precise arithmetic in $\mathbb{Q}(\beta)$. Also, complex bases allow to represent any complex number without separating the real and imaginary part. The example of such system is the Penny numeration system with the base $\imath - 1$ and the alphabet $\{0, 1\}$.

Some redundancy of the numeration system is required in order to construct a parallel addition algorithm. It means that numbers may have more than one represenation. P. Kornerup studied the necessary amount of redundancy in [7].

C. Frougny, E. Pelantová and M. Svobodová provide parallel algorithms for all bases β such that $|\beta| > 1$ and no conjugate of β equals 1 in modulus, see [4]. Nevertheless, the integer alphabet is not minimal in general. The parallel addition algorithms for several bases (negative integer, complex numbers $-1 + \imath, 2\imath$ and $\sqrt{2}\imath$, quadratic Pisot unit and the non-integer rational base) with minimal integer alphabet are given in [5].

We focus on the construction of parallel addition algorithm for a given base $\beta, |\beta| > 1$ being an algebraic integer and alphabet $\mathcal{A}$ containing 0. The alphabet $\mathcal{A}$ may be non-integer.

First, we recall the definitions and few previous results in Chapter 1. We include Theorem 1.7 which is an important tool used for the implementation.

The general concept of the construction of parallel addition algorithms is introduced in Chapter 2. We develop so-called extending window method for the construction of parallel addition algorithm for a given base β and alphabet $\mathcal{A}$. The method consists of two phases. We discuss the convergence of both of them in Chapter 3.

We design the implementation of the method in SageMath in Chapter 4. The implementation and provided user interfaces are described. The summary of all tested examples can be found in Chapter 5. See Appendices for images of the iterations of the extending window method for Eisenstein numeration system.

Chapter 1

Preliminaries

In this chapter, we recall few definitions and results connected to numeration systems and parallelism. We define the set $\mathbb{Z}[\omega]$ and we prove that it is isomorphic to $\mathbb{Z}^d$. This property is used in Theorem 1.7 which is an important tool for divisibility in $\mathbb{Z}[\omega]$.

1.1 Numeration systems

Firstly, we give a general definition of numeration system.

Definition 1.1. Let $\beta \in \mathbb{C}, |\beta| > 1$ and $\mathcal{A} \subset \mathbb{C}$ be a finite set containing 0. A pair $(\beta, \mathcal{A})$ is called a *positional numeration system* with *base* β and *digit set* $\mathcal{A}$, usually called *alphabet*.

So-called standard numeration systems have an integer base β and an alphabet $\mathcal{A}$ which is a set of countiguous integers. We restrict ourselves to base β which is an algebraic integer and possibly non-integer alphabet $\mathcal{A}$.

Definition 1.2. Let $(\beta, \mathcal{A})$ be a positional numeration system. We say that a complex number x has a $(\beta, \mathcal{A})$-*representation* if there exist digits $x_n, x_{n-1}, x_{n-2}, \dots \in \mathcal{A}, n \geq 0$ such that $x = \sum_{j=-\infty}^{n} x_j \beta^j$.

We write briefly a *representation* instead of a $(\beta, \mathcal{A})$-representation if the base β and the alphabet $\mathcal{A}$ follow from context.

Definition 1.3. Let $(\beta, \mathcal{A})$ be a positional numeration system. The set of all complex numbers with a finite $(\beta, \mathcal{A})$-representation is defined by

$$\mathrm{Fin}_{\mathcal{A}}(\beta) := \left\{ \sum_{j=-m}^{n} x_j \beta^j : n, m \in \mathbb{N}, x_j \in \mathcal{A} \right\} .$$

For $x \in \mathrm{Fin}_{\mathcal{A}}(\beta)$, we write

$$(x)_{\beta,\mathcal{A}} = 0^\omega x_n x_{n-1} \cdots x_1 x_0 \bullet x_{-1} x_{-2} \cdots x_{-m} 0^\omega ,$$

where 0^ω denotes the infinite sequence of zeros. Notice that indices are decreasing from left to right as it is ussual to write the most significant digits first. In what follows, we omit the starting and ending 0^ω when we work with numbers in $\mathrm{Fin}_{\mathcal{A}}(\beta)$. We remark that existence of an algorithm (standard or parallel) producing a finite $(\beta, \mathcal{A})$-representation of $x + y$ where $x, y \in \mathrm{Fin}_{\mathcal{A}}(\beta)$ implies that the set $\mathrm{Fin}_{\mathcal{A}}(\beta)$ is closed under addition, i.e.,

$$\mathrm{Fin}_{\mathcal{A}}(\beta) + \mathrm{Fin}_{\mathcal{A}}(\beta) \subset \mathrm{Fin}_{\mathcal{A}}(\beta)\,.$$

Designing an algorithm for parallel addition requires some redundancy in numeration system. According to [8], a numeration system $(\beta, \mathcal{A})$ is called *redundant* if there exists $x \in \mathrm{Fin}_{\mathcal{A}}(\beta)$ which has two different $(\beta, \mathcal{A})$-representations. For instance, the number 1 has the $(2, \{-1, 0, 1\})$-representations $1\bullet$ and $1(-1)\bullet$. Redundant numeration system can enable us to avoid carry propagation in addition. On the other hand, there are some disadvantages. For example, comparison is problematic.

1.2 Parallel addition

A local function, which is also often called sliding block code, is used to mathematically formalize parallelism.

Definition 1.4. Let $\mathcal{A}$ and $\mathcal{B}$ be alphabets. A function $\varphi : \mathcal{B}^{\mathbb{Z}} \to \mathcal{A}^{\mathbb{Z}}$ is said to be *p-local* if there exist $r, t \in \mathbb{N}$ satisfying $p = r + t + 1$ and a function $\phi : \mathcal{B}^p \to \mathcal{A}$ such that, for any $w = (w_j)_{j\in\mathbb{Z}} \in \mathcal{B}^{\mathbb{Z}}$ and its image $z = \varphi(w) = (z_j)_{j\in\mathbb{Z}} \in \mathcal{A}^{\mathbb{Z}}$, we have $z_j = \phi(w_{j+t}, \cdots, w_{j-r})$ for every $j \in \mathbb{Z}$. The parameter t, resp. r, is called *anticipation*, resp. *memory.*

This means that each digit of the image $\varphi(w)$ is computed from p digits of w in a sliding window. Suppose that there is a processor on each position with access to t input digits on the left and r input digits on the right. Then computation of $\varphi(w)$, where w finite sequence, can be done in constant time independent on the length of w.

Definition 1.5. Let β be a base and $\mathcal{A}$ and $\mathcal{B}$ two alphabets containing 0. A function $\varphi : \mathcal{B}^{\mathbb{Z}} \to \mathcal{A}^{\mathbb{Z}}$ such that

1. for any $w = (w_j)_{j\in\mathbb{Z}} \in \mathcal{B}^{\mathbb{Z}}$ with finitely many non-zero digits, $z = \varphi(w) = (z_j)_{j\in\mathbb{Z}} \in \mathcal{A}^{\mathbb{Z}}$ has only finite number of non-zero digits, and
2. $\sum_{j\in\mathbb{Z}} w_j\beta^j = \sum_{j\in\mathbb{Z}} z_j\beta^j$

is called *digit set conversion* in base β from $\mathcal{B}$ to $\mathcal{A}$. Such a conversion φ is said to be *computable in parallel* if it is p-local function for some $p \in \mathbb{N}$.

In fact, addition on $\mathrm{Fin}_{\mathcal{A}}(\beta)$ can be performed in parallel if there is digit set conversion from $\mathcal{A} + \mathcal{A}$ to $\mathcal{A}$ computable in parallel as we can easily output digitwise sum of two $(\beta, \mathcal{A})$-representations in parallel.

We recall few results about parallel addition in a numeration system with an integer alphabet. Ch. Frougny, E. Pelantová and M. Svobodová proved in [4] following sufficient condition for existence of the existence of algorithm for parallel addition.

Theorem 1.1. *Let $\beta \in \mathbb{C}$ be an algebraic number such that $|\beta| > 1$ and all its conjugates in modulus differs from 1. There exists an alphabet $\mathcal{A}$ of contiguous integers containing 0 such that addition on $Fin_{\mathcal{A}}(\beta)$ can be performed in parallel.*

The proof of the theorem provides the algorithm for the alphabet of the form $\{-a, -a+1, \dots, 0, \dots, a-1, a\}$. But in general, a is not minimal.

The same authors showed in [3] that the condition on the conjugates of the base β is also necessary:

Theorem 1.2. *Let the base $\beta \in \mathbb{C}, |\beta| > 1$, be an algebraic number with a conjugate β' such that $|\beta'| = 1$. Let $\mathcal{A} \subset \mathbb{Z}$ be an alphabet of contiguous integers containing 0. Then addition on $Fin_{\mathcal{A}}(\beta)$ cannot be computable in parallel.*

The question of minimality of the alphabet is studied in [5]. The following lower bound for the size of the alphabet is provided:

Theorem 1.3. *Let $\beta \in \mathbb{C}, |\beta| > 1$, be an algebraic integer with the minimal polynomial p. Let $\mathcal{A} \subset \mathbb{Z}$ be an alphabet of contiguous integers containing 0 and 1. If addition on $Fin_{\mathcal{A}}(\beta)$ is computable in parallel, then $\#\mathcal{A} \geq |p(1)|$. Moreover, if β is a positive real number, $\beta > 1$, then $\#\mathcal{A} \geq |p(1)| + 2$.*

In this thesis, we work in the more general concept as we consider also non-integer alphabets. First, we recall the following definition.

Definition 1.6. Let ω be a complex number. The set of values of all polynomials with integer coefficients evaluated in ω is denoted by

$$\mathbb{Z}[\omega] = \left\{ \sum_{i=0}^{n} a_i \omega^i : n \in \mathbb{N}, a_i \in \mathbb{Z} \right\} \subset \mathbb{Q}(\omega)\,.$$

Notice that $\mathbb{Z}[\omega]$ is a commutative ring (for our purpose, a ring is associative under multiplication and there is a multiplicative indentity).

From now on, let ω be an algebraic integer which generates the set $\mathbb{Z}[\omega]$ and let the base $\beta \in \mathbb{Z}[\omega]$ be such that $|\beta| > 1$. We remark that β is also the algebraic integer as all elements of $\mathbb{Z}[\omega]$ are algebraic integers. Finally, the alphabet $\mathcal{A}$ is a finite subset of $\mathbb{Z}[\omega]$ containing 0.

Few parallel addition algorithms for such numeration system with a non-integer alphabet were found ad hoc. We introduce the method for construction of the parallel addition algorithm for a given numeration system $(\beta, \mathcal{A})$ in Chapter 2.

1.3 Isomorphism of $\mathbb{Z}[\omega]$ and $\mathbb{Z}^d$

The goal of this section is to show a connection between the ring $\mathbb{Z}[\omega]$ and the set $\mathbb{Z}^d$.

First we recall notion of companion matrix which we use to define multiplication in $\mathbb{Z}^d$.

Definition 1.7. Let ω be an algebraic integer of degree $d \geq 1$ with the monic minimal polynomial $p(x) = x^d + p_{d-1}x^{d-1} + \cdots + p_1 x + p_0 \in \mathbb{Z}[x]$. A matrix

$$S := \begin{pmatrix} 0 & 0 & \cdots & 0 & -p_0 \\ 1 & 0 & \cdots & 0 & -p_1 \\ 0 & 1 & \cdots & 0 & -p_2 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 1 & -p_{d-1} \end{pmatrix} \in \mathbb{Z}^{d \times d}$$

is called *companion matrix* of the minimal polynomial of ω.

In what follows, the standard basis vectors of $\mathbb{Z}^d$ are denoted by

$$e_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, e_1 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \dots, e_{d-1} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

Definition 1.8. Let ω be an algebraic integer of degree $d \geq 1$, let p be its minimal polynomial and let S be its companion matrix. We define the mapping $\odot_\omega : \mathbb{Z}^d \times \mathbb{Z}^d \to \mathbb{Z}^d$ by

$$u \odot_\omega v := \left(\sum_{i=0}^{d-1} u_i S^i \right) \cdot \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{d-1} \end{pmatrix} \quad \text{for all } u = \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{d-1} \end{pmatrix}, v = \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{d-1} \end{pmatrix} \in \mathbb{Z}^d .$$

and we define powers of $u \in \mathbb{Z}^d$ by

$$\begin{aligned} u^0 &= e_0, \\ u^i &= u^{i-1} \odot_\omega u \text{ for } i \in \mathbb{N} . \end{aligned}$$

We will see later that $\mathbb{Z}^d$ equipped with elementwise addition and multiplication $\odot_\omega$ builds a commutative ring. Let us first recall an important property of a companion matrix – it is a root of its defining polynomial.

Lemma 1.4. *Let ω be an algebraic integer with a minimal polynomial p and let S be its companion matrix. Then*

$$p(S) = 0 .$$

Proof. Following the proof in [6], we have

$$\begin{aligned} e_0 &= S^0 e_0 , \\ S e_0 = e_1 &= S^1 e_0 , \\ S e_1 = e_2 &= S^2 e_0 , \\ S e_2 = e_3 &= S^3 e_0 , \\ &\vdots \\ S e_{d-2} = e_{d-1} &= S^{d-1} e_0 , \\ S e_{d-1} &= S^d e_0 , \end{aligned}$$

where the middle column is obtained by multiplication and the right one by using the previous row. Also by multiplying and substituting

$$\begin{aligned} S^d e_0 = S e_{d-1} &= -p_0 e_0 - p_1 e_1 - \cdots - p_{d-1} e_{d-1} \\ &= -p_0 S^0 e_0 - p_1 S^1 e_0 - \cdots - p_{d-1} S^{d-1} e_0 \\ &= (-p_0 S^0 - p_1 S^1 - \cdots - p_{d-1} S^{d-1}) e_0 \\ &= (S^d - p(S)) e_0 . \end{aligned}$$

Hence

$$p(S)e_0 = 0\,.$$

Moreover,

$$p(S)e_k = p(S)S^k e_0 = S^k p(S)e_0 = 0$$

for $k = \{0, 1, \ldots, d-1\}$ which implies the statement. □

The following lemma summarises basic properties of the mapping $\odot_\omega$ – multiplication by an integer scalar, the identity element, the distributive law and a weaker form of asociativity.

Lemma 1.5. *Let ω be an algebraic integer of degree d. The following statements hold for every $u, v, w \in \mathbb{Z}^d$ and $m \in \mathbb{Z}$:*

(i) $(mu) \odot_\omega v = u \odot_\omega (mv) = m(u \odot_\omega v)$,

(ii) $e_0 \odot_\omega v = v \odot_\omega e_0 = v$,

(iii) $(u \odot_\omega e_1^k) \odot_\omega v = u \odot_\omega (e_1^k \odot_\omega v)$ *for* $k \in \mathbb{N}$,

(iv) $(u + v) \odot_\omega w = u \odot_\omega w + v \odot_\omega w$ *and* $u \odot_\omega (v + w) = u \odot_\omega v + u \odot_\omega w$.

Proof. It is easy to see (i) as multiplication of a matrix by a scalar commutes and a scalar can be factored out of a sum.

The first equality of (ii) follows from definition and

$$v \odot_\omega e_0 = \sum_{i=0}^{d-1} v_i S^i \cdot e_0 = \sum_{i=0}^{d-1} v_i e_i = v\,.$$

For (iii), we use Lemma 1.4 and its proof. Assume $k = 1$:

$$\begin{aligned}
(u \odot_\omega e_1) \odot_\omega v &= \left(\sum_{i=0}^{d-1} u_i S^i \cdot e_1\right) \odot_\omega v = \left(\sum_{i=0}^{d-1} u_i S^i \cdot S e_0\right) \odot_\omega v \\
&= \left(\sum_{i=0}^{d-2} u_i e_{i+1} + u_{d-1} S^d e_0\right) \odot_\omega v = \left(\sum_{i=1}^{d-1} u_{i-1} e_i - u_{d-1} \sum_{i=0}^{d-1} p_i e_i\right) \odot_\omega v \\
&= \left(\sum_{i=1}^{d-1} u_{i-1} S^i - u_{d-1} \sum_{i=0}^{d-1} p_i S^i\right) \cdot v \\
&= \left(\sum_{i=1}^{d-1} u_{i-1} S^i + u_{d-1} S^d\right) \cdot v = \sum_{i=0}^{d-1} u_i S^i \cdot S \cdot v \\
&= u \odot_\omega (S \cdot v) = u \odot_\omega (e_1 \odot_\omega v)\,.
\end{aligned}$$

Now we proceed by induction:

$$\begin{aligned}
\left(u \odot_\omega e_1^k\right) \odot_\omega v &= \left(u \odot_\omega (e_1^{k-1} \odot_\omega e_1)\right) \odot_\omega v = \left((u \odot_\omega e_1^{k-1}) \odot_\omega e_1\right) \odot_\omega v \\
&= (u \odot_\omega e_1^{k-1}) \odot_\omega (e_1 \odot_\omega v) = u \odot_\omega \left(e_1^{k-1} \odot_\omega (e_1 \odot_\omega v)\right) \\
&= u \odot_\omega \left((e_1^{k-1} \odot_\omega e_1) \odot_\omega v\right) = u \odot_\omega \left(e_1^k \odot_\omega v\right)\,.
\end{aligned}$$

The statement (iv) follows easily from distributivity of matrix multiplication with respect to addition. □

Now we can prove that there is a correspondence between elements of $\mathbb{Z}[\omega]$ and $\mathbb{Z}^d$.

Theorem 1.6. *Let ω be an algebraic integer of degree d. Then*

$$\mathbb{Z}[\omega] = \left\{ \sum_{i=0}^{d-1} a_i \omega^i \colon a_i \in \mathbb{Z} \right\},$$

$(\mathbb{Z}^d, +, \odot_\omega)$ is a commutative ring and the mapping $\pi : \mathbb{Z}[\omega] \to \mathbb{Z}^d$ defined by

$$\pi(u) = \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{d-1} \end{pmatrix} \qquad \textit{for every } u = \sum_{i=0}^{d-1} u_i \omega^i \in \mathbb{Z}[\omega]$$

is a ring isomorphism.

Proof. Obviously, $\left\{\sum_{i=0}^{n} a_i\omega^i \colon n \in \mathbb{N}, a_i \in \mathbb{Z}\right\} = \mathbb{Z}[\omega] \supset \left\{\sum_{i=0}^{d-1} a_i\omega^i \colon a_i \in \mathbb{Z}\right\}$. We prove the opposite direction by the induction with respect to n. Assume $u \in \mathbb{Z}[\omega]$, $u = \sum_{i=0}^{n} u_i\omega^i$ for some $n \in \mathbb{N}$. We see that $u \in \left\{\sum_{i=0}^{d-1} a_i\omega^i \colon a_i \in \mathbb{Z}\right\}$ for all $n < d$.

Suppose now that the claim holds for $n-1$ and consider $n \geq d$. Let $p(x) = x^d + p_{d-1}x^{d-1} + \dots p_1 x + p_0$ be the minimal polynomial of ω. By $p(\omega) = 0$, we have an equation $\omega^d = -p_{d-1}\omega^{d-1} - \dots - p_1\omega - p_0$ which enables us to write

$$\begin{aligned} u = u_n\omega^n + \sum_{i=0}^{n-1} u_i\omega^i &= u_n\omega^{n-d}\underbrace{(-p_{d-1}\omega^{d-1} - \dots - p_1\omega - p_0)}_{\omega^d} + \sum_{i=0}^{n-1} u_i\omega^i \\ &= \sum_{i=0}^{n-d-1} u_i\omega^i + \sum_{i=n-d}^{n-1} (u_i - u_n \cdot p_{i-n+d})\omega^i = \sum_{i=0}^{n-1} u_i'\omega^i \,. \end{aligned}$$

Thus $u \in \left\{\sum_{i=0}^{d-1} a_i\omega^i \colon a_i \in \mathbb{Z}\right\}$ by the induction assumption.

Let us check now that the mapping π is well-defined. Assume on contrary that there exists $v \in \mathbb{Z}[\omega]$ and $i_0 \in \{0, 1, \dots, d-1\}$ such that $v = \sum_{i=0}^{d-1} v_i\omega^i = \sum_{i=0}^{d-1} v_i'\omega^i$ and $v_{i_0} \neq v_{i_0}'$. Then

$$\sum_{i=0}^{d-1} (v_i' - v_i)\omega^i = 0$$

and $\sum_{i=0}^{d-1}(v_i' - v_i)x^i \in \mathbb{Z}[x]$ is non-zero polynomial of degree smaller than the degree d of minimal polynomial p, a contradiction.

Clearly, π is bijection.

Let $v = \sum_{i=0}^{d-1} v_i\omega^i$ be element of $\mathbb{Z}[\omega]$. We prove by the induction that

$$\pi(\omega^i v) = (\pi(\omega))^i \odot_\omega \pi(v) \,.$$

For $i = 1$, consider

$$\begin{aligned} \omega v = \omega \sum_{i=0}^{d-1} v_i\omega^i &= \sum_{i=0}^{d-2} v_i\omega^{i+1} + v_{d-1}\underbrace{(-p_{d-1}\omega^{d-1} - \dots - p_1\omega - p_0)}_{=\omega^d} \\ &= -p_0 v_{d-1} + \sum_{i=1}^{d-1} (v_{i-1} - v_{d-1}p_i)\omega^i \,. \end{aligned}$$

Hence

$$\pi(\omega v) = -p_0 v_{d-1} e_0 + \sum_{i=1}^{d-1} (v_{i-1} - v_{d-1} p_i) e_i = S \cdot \pi(v)$$
$$= e_1 \odot_\omega \pi(v) = \pi(\omega) \odot_\omega \pi(v)\,.$$

Suppose now for the induction that

$$\pi(\omega^{i-1} v) = (\pi(\omega))^{i-1} \odot_\omega \pi(v)\,.$$

Then

$$\pi(\omega^i v) = \pi(\omega(\omega^{i-1} v)) = \pi(\omega) \odot_\omega \pi(\omega^{i-1} v) = \pi(\omega) \odot_\omega \left((\pi(\omega))^{i-1} \odot_\omega \pi(v)\right) = (\pi(\omega))^i \odot_\omega \pi(v)\,,$$

where we use (iii) of Lemma 1.5 for the last equality.

Now we multiply v by $m \in \mathbb{Z} \subset \mathbb{Z}[\omega]$:

$$\pi(mv) = \pi\left(m \sum_{i=0}^{d-1} v_i \omega^i\right) = \pi\left(\sum_{i=0}^{d-1} m v_i \omega^i\right) = m\pi(v) = (m e_0) \odot_\omega \pi(v) = \pi(m) \odot_\omega \pi(v)\,.$$

Let $u = \sum_{i=0}^{d-1} u_i \omega^i \in \mathbb{Z}[\omega]$. Since π is obviously additive, we conclude:

$$\pi(uv) = \pi\left(\sum_{i=0}^{d-1} u_i \omega^i v\right) = \sum_{i=0}^{d-1} \pi(\omega^i u_i v) = \sum_{i=0}^{d-1} \pi(\omega)^i \odot_\omega (\pi(u_i) \odot_\omega \pi(v))$$
$$= \sum_{i=0}^{d-1} \pi(\omega^i u_i) \odot_\omega \pi(v) = \pi\left(\sum_{i=0}^{d-1} u_i \omega^i\right) \odot_\omega \pi(v) = \pi(u) \odot_\omega \pi(v)\,.$$

Now we can show that the operation $\odot_\omega$ is associative and commutative. Let $f, g, h \in \mathbb{Z}^d$ and $u, v, w \in \mathbb{Z}[\omega]$ such that $f = \pi(u), g = \pi(v)$ and $h = \pi(w)$. Then

$$f \odot_\omega (g \odot_\omega h) = f \odot_\omega \pi(vw) = \pi(u(vw)) = \pi((uv)w) = \pi(uv) \odot_\omega h = (f \odot_\omega g) \odot_\omega h$$

and

$$g \odot_\omega h = \pi(vw) = \pi(wv) = h \odot_\omega g\,.$$

Thus, $(\mathbb{Z}^d, +, \odot_\omega)$ is the commutative ring. □

Due to this theorem we may work with integer vectors instead of elements of $\mathbb{Z}[\omega]$ and multiplication in $\mathbb{Z}[\omega]$ is replaced by multiplying by an appropriate matrix.

The last theorem of this section is a practical tool for divisibility in $\mathbb{Z}[\omega]$. To check whether an element of $\mathbb{Z}[\omega]$ is divisible by another element, we look for an integer solution of a linear system. Moreover, this solution provides a result of the division in the positive case.

Theorem 1.7. *Let ω be an algebraic integer of degree d and let S be the companion matrix of its minimal polynomial. Let $\beta = \sum_{i=0}^{d-1} b_i \omega^i$ be a nonzero element of $\mathbb{Z}[\omega]$. Then for every $u \in \mathbb{Z}[\omega]$*

$$u \in \beta\mathbb{Z}[\omega] \iff S_\beta^{-1} \cdot \pi(u) \in \mathbb{Z}^d\,,$$

where $S_\beta = \sum_{i=0}^{d-1} b_i S^i$.

Proof. Observe first that S_β is nonsingular. Otherwise, there exists $y = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{d-1} \end{pmatrix} \in \mathbb{Z}^d, y \neq \mathbf{0}$ such that $S_\beta \cdot y = 0$. Thus

$$\pi(\beta) \odot_\omega y = \mathbf{0} \iff \beta\pi^{-1}(y) = 0\,.$$

Since $\beta \neq 0$, we have

$$0 = \pi^{-1}(y) = \sum_{i=0}^{d-1} y_i \omega^i\,,$$

which contradict that degree of ω is d.

Now

$$\begin{aligned} u \in \beta\mathbb{Z}[\omega] &\iff (\exists v \in \mathbb{Z}[\omega])(u = \beta v) \\ &\iff (\exists v \in \mathbb{Z}[\omega])(\pi(u) = \pi(\beta) \odot_\omega \pi(v) = S_\beta \cdot \pi(v)) \\ &\iff \pi(v) = S_\beta^{-1} \cdot \pi(u) \in \mathbb{Z}^d\,. \end{aligned}$$

Clearly, if u is divisible by β, then $v = u/\beta = \pi^{-1}(S_\beta^{-1} \cdot \pi(u)) \in \mathbb{Z}[\omega]$. □

Chapter 2

Design of extending window method

We recall the general concept of addition at the beginning of this chapter and then we describe the so-called *extending window method* which is due to M. Svobodová [9].

From now on, let ω be an algebraic integer and $(\beta, \mathcal{A})$ be a numeration system such that a base $\beta \in \mathbb{Z}[\omega]$ and an alphabet $\mathcal{A} \ni 0$ is a finite subset of $\mathbb{Z}[\omega]$.

The general concept of addition (standard or parallel) in any numeration system $(\beta, \mathcal{A})$, such that $\mathrm{Fin}_{\mathcal{A}}(\beta)$ is closed under addition, is following: we add numbers digitwise and then we convert the result into the alphabet $\mathcal{A}$. Obviously, digitwise addition is computable in parallel, thus the crucial point is the digit set conversion of the obtained result. It can be easily done in a standard way but a parallel digit set conversion is nontrivial. However, formulas are basically same but the choice of coefficients differs.

Now we go step by step more precisely. Let $x = \sum_{-m'}^{n'} x_i\beta^i, y = \sum_{-m'}^{n'} y_i\beta^i \in \mathrm{Fin}_{\mathcal{A}}(\beta)$ with $(\beta, \mathcal{A})$-representantions padded by zeros to have the same length. We set

$$\begin{aligned} w = x + y &= \sum_{-m'}^{n'} x_i\beta^i + \sum_{-m'}^{n'} y_i\beta^i = \sum_{-m'}^{n'} (x_i + y_i)\beta^i \\ &= \sum_{-m'}^{n'} w_i\beta^i \,, \end{aligned}$$

where $w_j = x_j + y_j \in \mathcal{A} + \mathcal{A}$. Thus, $w_{n'}w_{n'-1}\cdots w_1w_0 \bullet w_{-1}w_{-2}\cdots w_{-m'}$ is a $(\beta, \mathcal{A} + \mathcal{A})$-representation of $w \in \mathrm{Fin}_{\mathcal{A}+\mathcal{A}}(\beta)$.

We also use column notation of addition in what follows, e.g.,

$$\begin{array}{c} x_{n'}\ x_{n'-1}\cdots x_1\ x_0 \bullet x_{-1}\ x_{-2}\ \cdots x_{-m'} \\ y_{n'}\ y_{n'-1}\cdots y_1\ \ y_0 \bullet y_{-1}\ y_{-2}\ \cdots y_{-m'} \\ \hline w_{n'}w_{n'-1}\cdots w_1w_0 \bullet w_{-1}w_{-2}\cdots w_{-m'} \,. \end{array}$$

As we want to obtain a $(\beta, \mathcal{A})$-representation of w, we search for a sequence

$$z_nz_{n-1}\cdots z_1z_0z_{-1}z_{-2}\cdots z_{-m}$$

such that $z_j \in \mathcal{A}$ and

$$z_nz_{n-1}\cdots z_1z_0 \bullet z_{-1}z_{-2}\cdots z_{-m} = (w)_{\beta,\mathcal{A}} \,.$$

From now on, we consider without lost of generality only β-integers since modification for representations with rational part is obvious:

$$\beta^m \cdot z_n z_{n-1} \cdots z_1 z_0 \bullet z_{-1} z_{-2} \cdots z_{-m} = z_n z_{n-1} \cdots z_1 z_0 z_{-1} z_{-2} \cdots z_{-m} \bullet$$

Particulary, let $(w)_{\beta,\mathcal{A}+\mathcal{A}} = w_{n'} w_{n'-1} \cdots w_1 w_0 \bullet$. We search $n \in \mathbb{N}$ and $z_n, z_{n-1}, \ldots, z_1, z_0 \in \mathcal{A}$ such that $(w)_{\beta,\mathcal{A}} = z_n z_{n-1} \cdots z_1 z_0 \bullet$.

We use suitable representation of zero to convert digits w_j into the alphabet $\mathcal{A}$. For our purpose, we use the simplest possible representation deduced from the polynomial

$$x - \beta \in (\mathbb{Z}[\omega])\,[x]\,.$$

We remark that any polynomial $R(x) = r_s x^s + r_{s-1} x^{s-1} + \cdots + r_1 x + r_0$ with coefficients $r_i \in \mathbb{Z}[\omega]$, such that $R(\beta) = 0$ gives us possible representation of zero. The polynomial R is called a *rewriting rule*.

Within a digit set conversion with an arbitrary rewriting rule R, one of the coefficients of R which is greatest in modulus (so-called *core coefficient*) is used for the conversion of a digit w_j. But using of an arbitrary rewriting rule R is out of scope of this thesis, so we focus on the simplest possible rewriting rule $R(x) = x - \beta$.

As $0 = \beta^j \cdot R(\beta) = 1 \cdot \beta^{j+1} - \beta \cdot \beta^j$, we have a representation of zero

$$1(-\beta)\underbrace{0\cdots 0}_{j}\bullet = (0)_\beta\,.$$

for all $j \in \mathbb{N}$. We multiply this representation by $q_j \in \mathbb{Z}[\omega]$ which is called a *weight coefficient* to obtain representation of zero

$$q_j(-q_j\beta)\underbrace{0\cdots 0}_{j}\bullet = (0)_\beta\,.$$

This is digitwise added to $w_n w_{n-1} \cdots w_1 w_0 \bullet$ to convert the digit w_j into the alphabet $\mathcal{A}$. The conversion of j-th digit causes a *carry* q_j on the $(j+1)$-th position. The digit set conversion runs from the right ($j = 0$) to the left until all digits and carries are converted into the alphabet $\mathcal{A}$:

$$\begin{array}{ccccccc}
w_n w_{n-1} & \cdots & w_{j+1} & w_j & w_{j-1} & \cdots & w_1 w_0 \bullet \\
 & & & & q_{j-2} & \iddots & \\
 & & & q_{j-1} & -\beta q_{j-1} & & \\
 & & q_j & -\beta q_j & & & \\
 & \iddots & -\beta q_{j+1} & & & & \\
\hline
z_{n+s} \cdots z_n z_{n-1} & \cdots & z_{j+1} & z_j & z_{j-1} & \cdots & z_1\ z_0 \bullet
\end{array} \tag{2.1}$$

Hence, the desired formula for conversion on the j-th position is

$$z_j = w_j + q_{j-1} - q_j \beta$$

for $j \in \mathbb{N}_0$. We set $q_{-1} = 0$ as there is no carry from the right on the 0-th position.

Clearly, the value of w is preserved:

$$\begin{aligned}\sum_{j\geq 0} z_j\beta^j &= w_0 - \beta q_0 + \sum_{j>0}(w_j + q_{j-1} - q_j\beta)\beta^j \\ &= \sum_{j\geq 0} w_j\beta^j + \sum_{j>0} q_{j-1}\beta^j - \sum_{j\geq 0} q_j\cdot\beta^{j+1} \\ &= \sum_{j\geq 0} w_j\beta^j + \sum_{j>0} q_{j-1}\beta^j - \sum_{j>0} q_{j-1}\cdot\beta^{j} \\ &= \sum_{j\geq 0} w_j\beta^j = w\,.\end{aligned} \tag{2.2}$$

The weight coefficient q_j must be chosen so that the converted digit is in the alphabet $\mathcal{A}$, i.e.,

$$z_j = w_j + q_{j-1} - q_j\beta \in \mathcal{A}\,. \tag{2.3}$$

The choice of weight coefficients is the crucial part in order to construct addition algorithms which are computable in parallel. The extending window method determining weight coefficients for a given input is decribed in Section 2.1.

On the other hand, the following example shows that determining weight coefficients is trivial for standard numeration systems.

Example 2.1. Assume now a standard numeration system $(\beta, \mathcal{A})$, where

$$\beta \in \mathbb{N}\,, \beta \geq 2\,, \mathcal{A} = \{0, 1, 2, \ldots, \beta - 1\}\,.$$

Notice that

$$z_j \equiv w_j + q_{j-1} \mod \beta\,.$$

There is only one representative of each class modulo β in the standard numeration system $(\beta, \mathcal{A})$. Therefore, the digit z_j is uniquely determined for a given digit $w_j \in \mathcal{A}+\mathcal{A}$ and carry q_{j-1} and thus so is the weight coefficient q_j. This means that $q_j = q_j(w_j, q_{j-1})$ for all $j \geq 0$. Generally,

$$q_j = q_j(w_j, q_{j-1}(w_{j-1}, q_{j-2})) = \cdots = q_j(w_j, \ldots, w_1, w_0)$$

and

$$z_j = z_j(w_j, \ldots, w_1, w_0)\,,$$

which implies that addition runs in linear time.

We require that the digit set conversion from $\mathcal{A}+\mathcal{A}$ into $\mathcal{A}$ is computable in parallel, i.e., there exist constants $r, t \in \mathbb{N}_0$ such that for all $j \geq 0$ is $z_j = z_j(w_{j+r}, \ldots, w_{j-t})$. To avoid the dependency on all less, respectivelly more, significant digits, we need variety in the choice of weight coefficient q_j. This implies that the used numeration system must be redundant.

2.1 Extending window method

In order to construct a digit set conversion in numeration system $(\beta, \mathcal{A})$ which is computable in parallel, we consider more general case of digit set conversion from an *input alphabet* $\mathcal{B}$

such that $\mathcal{A} \subsetneq \mathcal{B} \subset \mathcal{A}+\mathcal{A}$ instead of the alphabet $\mathcal{A}+\mathcal{A}$. As mentioned above, the key problem is to find for every $j \geq 0$ a weight coefficient q_j such that

$$z_j = \underbrace{w_j}_{\in\mathcal{B}} + q_{j-1} - q_j\beta \in \mathcal{A}$$

for any input $w_{n'}w_{n'-1}\dots w_1w_0\bullet = (w)_{\beta,\mathcal{B}}, w \in \mathrm{Fin}_{\mathcal{B}}(\beta)$. We remark that the weight coefficient q_{j-1} is determined by the input w. For a digit set conversion to be computable in parallel we require to digit $z_j = z_j(w_{j+r},\dots,w_{j-t})$ for a fixed anticipation r and memory t in $\mathbb{N}_0$.

We introduce following definitions.

Definition 2.1. Let $\mathcal{B}$ be a set such that $\mathcal{A} \subsetneq \mathcal{B} \subset \mathcal{A}+\mathcal{A}$. Then any finite set $\mathcal{Q} \subset \mathbb{Z}[\omega]$ containing 0 such that

$$\mathcal{B}+\mathcal{Q} \subset \mathcal{A}+\beta\mathcal{Q}$$

is called a *weight coefficients set.*

We see that if $\mathcal{Q}$ is a weight coefficients set, then

$$(\forall w_j \in \mathcal{B})(\forall q_{j-1} \in \mathcal{Q})(\exists q_j \in \mathcal{Q})(w_j + q_{j-1} - q_j\beta \in \mathcal{A})\,.$$

In other words, there is a weight coefficient $q_j \in \mathcal{Q}$ for any carry from the right $q_{j-1} \in \mathcal{Q}$ and any digit w_j in the input alphabet $\mathcal{B}$. I.e., we satisfy the basic digit set conversion formula (2.3). Notice that $q_{-1} = 0$ is in $\mathcal{Q}$ by definition. Thus, all weight coeficients may be chosen from $\mathcal{Q}$.

Definition 2.2. Let M be an integer and $q : \mathcal{B}^M \to \mathcal{Q}$ be a mapping such that

$$w_j + q(w_{j-1},\dots,w_{j-M}) - \beta q(w_j,\dots,w_{j-M+1}) \in \mathcal{A}$$

for all $w_j, w_{j-1},\dots,w_{j-M} \in \mathcal{B}$ and $q(0,0,\dots,0) = 0$. Then q is called a *weight function* and M is called a *length of window.*

Having a weight function q, we define a function $\phi : \mathcal{B}^{M+1} \to \mathcal{A}$ by

$$\phi(w_j,\dots,w_{j-M}) = w_j + \underbrace{q(w_{j-1},\dots,w_{j-M})}_{=q_{j-1}} - \beta\underbrace{q(w_j,\dots,w_{j-M+1})}_{=q_j} =: z_j\,,$$

which verifies that the mapping ϕ is indeed the $(M+1)$-local digit set conversion with anticipation $r = 0$ and memory $t = M$. The requirement of zero output of the weight function q for the input of M zeros guarantees that $\phi(0,0,\dots,0) = 0$. Thus the first condition of Definition 1.5 is satisfied. The second one follows from the equation 2.2.

Let us summarize the construction of digit set conversion by rewriting rule $x - \beta$. We need to find weight coefficients for all possible combinations of digits of the input alphabet $\mathcal{B}$. Their multiples of the rewriting rules are digitwise added to the input sequence. In fact, it means that the equation (2.3) is applied on the each position. If the digit set conversion is computable in parallel, the weight coefficients are determined as the outputs of the weight function q with some fixed length of window M.

We search for the weight function q for a given base β and input alphabet $\mathcal{B}$ by the extending window method. It consists of two phases. First, we find a minimal possible weight coefficients set $\mathcal{Q}$. We know that is possible to convert the input sequence by choosing the weight coefficients from this set $\mathcal{Q}$. The set $\mathcal{Q}$ serves as the starting point for the second phase in which we increment the expected length of the window M until the weight function q is uniquely defined for each $(w_j, w_{j-1}, \ldots, w_{j-M+1}) \in \mathcal{B}^M$. Then, the local conversion is determined – we use the weight function outputs as weight coefficients in the formula(2.3).

We remark that the convergence of both phases is discussed separately in Chapter 3.

2.1.1 Phase 1 – Weight coefficients set

The goal of the first phase is to compute a weight coefficients set $\mathcal{Q}$, i.e., to find a set $\mathcal{Q} \ni 0$ such that

$$\mathcal{B} + \mathcal{Q} \subset \mathcal{A} + \beta\mathcal{Q}\,.$$

We build the sequence $\mathcal{Q}_0, \mathcal{Q}_1, \mathcal{Q}_2, \ldots$ iteratively so that we extend $\mathcal{Q}_k$ to $\mathcal{Q}_{k+1}$ in a way to cover all elements on the left side with the set $\mathcal{Q}_k$ by elements on the right side with the extended set $\mathcal{Q}_{k+1}$, i.e.

$$\mathcal{B} + \mathcal{Q}_k \subset \mathcal{A} + \beta\mathcal{Q}_{k+1}\,.$$

This procedure is repeated until the extended weight coefficients set $\mathcal{Q}_{k+1}$ is the same as the original set $\mathcal{Q}_k$.

In other words, we start with $\mathcal{Q}_0 = \{0\}$ meaning that we search all weight coefficients q_j necessary for digit set conversion for the case where there is no carry from the right, i.e., $q_{j-1} = 0$. We add them to weight coefficient set $\mathcal{Q}_0$ to obtain the set $\mathcal{Q}_1$. Assume now that we have the set $\mathcal{Q}_k$ for some $k \geq 1$. The weight coefficients in $\mathcal{Q}_k$ now may appear as a carry q_{j-1}. If there are no suitable weight coefficients q_j in weight coefficients set $\mathcal{Q}_k$ to cover all sums of added coefficients and digits of input alphabet $\mathcal{B}$, we extend $\mathcal{Q}_k$ to $\mathcal{Q}_{k+1}$ by the suitable coefficients. And so on until there is no need to add more elements, i.e. the extended set $\mathcal{Q}_{k+1}$ equals $\mathcal{Q}_k$. Then the weight coefficients set $\mathcal{Q} := \mathcal{Q}_{k+1}$ satisfies Definition 2.1. We remark that the expression "a weight coefficient q covers an element x" means that there is $a \in \mathcal{A}$ such that $x = a + \beta q$.

The precise description of the semi-algorithm in a pseudocode is in Algorithm 1. For better understanding, see Figures **??**–**??** in Appendix **??** which illustrate the construction of the weight coefficients set $\mathcal{Q}$ for Eisenstein numeration system with a complex alphabet (see Example 5.1 for its description).

Section 3.2 discusses the convergence of Phase 1, i.e. whether it happens that $\mathcal{Q}_{k+1} = \mathcal{Q}_k$ for some k.

Algorithm 1 Search for weight coefficients set (Phase 1)

1: $k := 0$
2: $Q_0 := \{0\}$
3: **repeat**
4: By Algorithm 2, extend $\mathcal{Q}_k$ to $\mathcal{Q}_{k+1}$ in a minimal possible way so that

$$\mathcal{B} + \mathcal{Q}_k \subset \mathcal{A} + \beta\mathcal{Q}_{k+1}$$

5: $k := k + 1$
6: **until** $\mathcal{Q}_k = \mathcal{Q}_{k+1}$
7: $\mathcal{Q} := \mathcal{Q}_k$
8: **return** $\mathcal{Q}$

Possibly, there are more candidating weight coefficients which cover some element of the set $\mathcal{B} + \mathcal{Q}_k$. Let us suppose that we have the list which contains the lists of these candidates for each element of the set $\mathcal{B} + \mathcal{Q}_k$. This list of lists is saved in the variable `candidates` in Algorithm 2. Now, for each element, we check the list of candidates which cover this element and if there is none of them contained in the set $\mathcal{Q}_k$, the smallest (in absolute value) weight coefficient from the list of candidates is added to the set $\mathcal{Q}_k$. The extension $\mathcal{Q}_{k+1}$ of the set $\mathcal{Q}_k$ is obtained in this manner.

We may slightly improve this procedure: for example we may first extend $\mathcal{Q}_k$ by all single-element lists of `candidates`. These elements may be enough to cover also other elements of $\mathcal{B}+\mathcal{Q}_k$. It implies that the resulting $\mathcal{Q}$ is dependent on the way of selection from `candidates`.

Algorithm 2 Extending intermediate weight coefficients set

Input: `candidates` from Algorithm 3, previous weight coefficients set $\mathcal{Q}_k$

$\mathcal{Q}_{k+1} := \mathcal{Q}_k$
for all `cand_for_x` in `candidates` **do**
    if no element of `cand_for_x` in $\mathcal{Q}_k$ **then**
        Add the smallest element (in absolute value) of `cand_for_x` to $\mathcal{Q}_{k+1}$
    end if
end for
return $\mathcal{Q}_{k+1}$

Algorithm 3 describes the search for the list of lists of candidates. For each element $x \in \mathcal{B} + \mathcal{Q}_k$ we build the list of candidates (in the variable `cand_for_x`) so that we test the divisibility of $x - a$ by the base β for all letters $a \in \mathcal{A}$. In the positive case, the result of division is appended to `cand_for_x` as the candidating weight coefficient. We remark that Theorem 1.7 is used to check the divisibility.

Algorithm 3 Search for candidates

Input: previous weight coefficients set $\mathcal{Q}_k$, alternatively also the set $\mathcal{Q}_{k-1}$

1: `candidates`:= empty list of lists
2: **for all** $x \in \mathcal{B} + \mathcal{Q}_k$ **do** {Alternatively, $x \in (\mathcal{B} + \mathcal{Q}_k) \setminus (\mathcal{B} + \mathcal{Q}_{k-1})$}
3:     `cand_for_x`:= empty list
4:     **for all** $a \in \mathcal{A}$ **do**
5:         **if** $(x - a)$ is divisible by β in $\mathbb{Z}[\omega]$ **then** {using Theorem 1.7}
6:             Append $\frac{x-a}{\beta}$ to `cand_for_x`
7:         **end if**
8:     **end for**
9:     Append `cand_for_x` to `candidates`
10: **end for**
11: **return** `candidates`

We can improve the performance of Algorithm 3 by substituting the set $\mathcal{B} + \mathcal{Q}_k$ by $(\mathcal{B} + \mathcal{Q}_k) \setminus (\mathcal{B} + \mathcal{Q}_{k-1})$ on the line 2 because

$$\mathcal{B} + \mathcal{Q}_{k-1} \subset \mathcal{A} + \beta\mathcal{Q}_k \subset \mathcal{A} + \beta\mathcal{Q}_{k+1}$$

for any $\mathcal{Q}_{k+1} \supset \mathcal{Q}_k$. Thus there is no need to check whether the elements of $\mathcal{B} + \mathcal{Q}_{k-1}$ are covered by some weight coefficient on $\mathcal{Q}_k$ in Algorithm 2.

2.1.2 Phase 2 – Weight function

We want to find a length of the window M and a weight function $q : \mathcal{B}^M \to \mathcal{Q}$. We start with the weight coefficients set $\mathcal{Q}$ obtained in Phase 1. The idea is to reduce necessary weight coefficients for the conversion of a given digit up to single value. This is done by enlarging number of considered input digits (extending the length of window) – there are less possible carries from the right if we know which digits on the right are converted.

We introduce the following notation. Let $\mathcal{Q}$ be a weight coefficients set and $w_j \in \mathcal{B}$. Denote by $\mathcal{Q}_{[w_j]}$ any set such that

$$(\forall q_{j-1} \in \mathcal{Q})(\exists q_j \in \mathcal{Q}_{[w_j]})(w_j + q_{j-1} - q_j\beta \in \mathcal{A}) .$$

By induction with respect to $m \in \mathbb{N}, m > 1$, for all $(w_j, \dots, w_{j-m+1}) \in \mathcal{B}^m$ denote by $\mathcal{Q}_{[w_j,\dots,w_{j-m+1}]}$ any subset of $\mathcal{Q}_{[w_j,\dots,w_{j-m+2}]}$ such that

$$(\forall q_{j-1} \in \mathcal{Q}_{[w_{j-1},\dots,w_{j-m+1}]})(\exists q_j \in \mathcal{Q}_{[w_j,\dots,w_{j-m+1}]})(w_j + q_{j-1} - q_j\beta \in \mathcal{A}) .$$

Recall the scheme 2.1 of digit set conversion for better understanding of the definition and method:

$$\begin{array}{ccccc}
\cdots w_{j+1} & w_j & w_{j-1} & \cdots w_{j-M+1} w_{j-M} \cdots \\
 & q_{j-1} & -\beta q_{j-1} & \\
q_j & -\beta q_j & & \\
\hline
\cdots z_{j+1} & z_j & z_{j-1} & \cdots z_{j-M+1}\, z_{j-M} \cdots
\end{array}$$

The idea is to check all possible right carries $q_{j-1} \in \mathcal{Q}$ and determine values $q_j \in \mathcal{Q}$ such that

$$z_j = w_j + q_{j-1} - q_j\beta \in \mathcal{A} .$$

So we obtain a set $\mathcal{Q}_{[w_j]} \subset \mathcal{Q}$ of weight coefficients which are necessary to convert digit w_j with any carry $q_{j-1} \in \mathcal{Q}$. Assuming that we know input digit w_{j-1}, the set of possible carries from the right is also reduced to $\mathcal{Q}_{[w_{j-1}]}$. Thus we may reduce the set $\mathcal{Q}_{[w_j]}$ to a set $\mathcal{Q}_{[w_j,w_{j-1}]} \subset \mathcal{Q}_{[w_j]}$ which is necessary to cover elements of $w_j + \mathcal{Q}_{[w_{j-1}]}$. Prolonging length of window in this manner may lead to a unique weight coefficient q_j for enough given input digits.

Accordingly, the weight function q is found if there is $M \in \mathbb{N}$ such that

$$\#\mathcal{Q}_{[w_j,\dots,w_{j-M+1}]} = 1$$

for all $w_j, \dots, w_{j-M+1} \in \mathcal{B}^M$. Unfortunately, we do not know whether this happens. But we may reveal the nonconvergence of Phase 2 for some cases by Algorithm 6, which is decribed in Section 3.2.

The precise desription of the construction of the weight function is in Algorithm 4. For construction of the set $\mathcal{Q}_{[w_j,\dots,w_{j-m+1}]}$ we first choose elements which are the only possible to cover some value in $x \in w_0 + \mathcal{Q}_{[w_{j-1},\dots,w_{j-m+1}]}$. Then we add to $\mathcal{Q}_{[w_j,\dots,w_{j-m+1}]}$ one by one elements from $\mathcal{Q}_{[w_j,\dots,w_{j-m+2}]}$ covering an uncovered value until each desired value equals $a + \beta q_j$ for some q_j in $\mathcal{Q}_{[w_j,\dots,w_{j-m+1}]}$ and $a \in \mathcal{A}$. The pseudocode is in Algorithm 5.

We remark that this algorithm does not provide the trully minimal set $\mathcal{Q}_{[w_j,\dots,w_{j-m+1}]}$ in the sense of size, but it minimizes the set $\mathcal{Q}_{[w_j,\dots,w_{j-m+1}]}$ enough. Phase 2 can be modified

Algorithm 4 Search for weight function (Phase 2)

Input: weight coefficients set $\mathcal{Q}$

$m := 1$
for all $w_j \in \mathcal{B}$ **do**
  By Algorithm 5, find set $\mathcal{Q}_{[w_j]} \subset \mathcal{Q}$ such that

$$w_j + \mathcal{Q} \subset \mathcal{A} + \beta \mathcal{Q}_{[w_j]}$$

end for
while $\max\{\#\mathcal{Q}_{[w_j,\ldots,w_{j-m+1}]} : (w_j, \ldots, w_{j-m+1}) \in \mathcal{B}^m\} > 1$ **do**
  $m := m + 1$
  for all $(w_j, \ldots, w_{j-m+1}) \in \mathcal{B}^m$ **do**
    By Algorithm 5, find set $\mathcal{Q}_{[w_j,\ldots,w_{j-m+1}]} \subset \mathcal{Q}_{[w_j,\ldots,w_{j-m+2}]}$ such that

$$w_j + \mathcal{Q}_{[w_{j-1},\ldots,w_{j-m+1}]} \subset \mathcal{A} + \beta \mathcal{Q}_{[w_j,\ldots,w_{j-m+1}]} \,,$$

  end for
end while
$M := m$
for all $(w_j, \ldots, w_{j-M+1}) \in \mathcal{B}^M$ **do**
  $q(w_j, \ldots, w_{j-M+1}) :=$ only element of $\mathcal{Q}_{[w_j,\ldots,w_{j-M+1}]}$
end for
return q

Algorithm 5 Search for set $\mathcal{Q}_{[w_j,\ldots,w_{j-m+1}]}$

Input: Input digit w_j, set of possible carries $\mathcal{Q}_{[w_{j-1},\ldots,w_{j-m+1}]}$, previous set of possible weight coefficients $\mathcal{Q}_{[w_j,\ldots,w_{j-m+2}]}$

`list_of_coverings`:=empty list of lists
for all $x \in w_j + \mathcal{Q}_{[w_{j-1},\ldots,w_{j-m+1}]}$ **do**
  Build a list `x_covered_by` of weight coefficients $q_j \in \mathcal{Q}_{[w_j,\ldots,w_{j-m+2}]}$ such that

$$x = a + \beta q_j \qquad \text{for some } a \in \mathcal{A} \,.$$

  Append `x_covered_by` to `list_of_coverings`
end for
$\mathcal{Q}_{[w_j,\ldots,w_{j-m+1}]}$:= empty set
while `list_of_coverings` is nonempty **do**
  Pick any element q of one of the shortest lists of `list_of_coverings`
  Add the element q to $\mathcal{Q}_{[w_j,\ldots,w_{j-m+1}]}$
  Remove lists of `list_of_coverings` containing the element q from `list_of_coverings`
end while
return $\mathcal{Q}_{[w_j,\ldots,w_{j-m+1}]}$

by replacing Algorithm 5 for different one. It is possible that the effort to reduce size of $\mathcal{Q}_{[w_j,\ldots,w_{j-m+1}]}$ as much as possible is not the best for convergence of Phase 2.

Figures **??**–**??** in Appendix **??** illustrate the construction of the set $\mathcal{Q}_{[\omega,1,2]}$ for Eisenstein numeration system with a complex alphabet.

Notice that for given length of the window M, number of calls of Algorithm 5 within Algorithm 4 is

$$\sum_{m=1}^{M} \#\mathcal{B}^m = \#\mathcal{B} \sum_{m=0}^{M-1} \#\mathcal{B}^m = \#\mathcal{B}\frac{\#\mathcal{B}^M - 1}{\#\mathcal{B} - 1}\,.$$

It implies that the time complexity grows exponentially as about $\#\mathcal{B}^M$. The required memory is also exponentional because we have to store at least for $m \in \{M-1, M\}$ sets $\mathcal{Q}_{[w_j,\ldots,w_{j-m+1}]}$ for all $w_j, \ldots, w_{j-m+1} \in \mathcal{B}$.

We my reduce the number of the combinations of the input digits so that if for some $(w_j, \ldots, w_{j-m+1}) \in \mathcal{B}^m, m < M$ is $\#\mathcal{Q}_{[w_j,\ldots,w_{j-m+1}]} = 1$, we do not extend the window for these digits but we set the ouput of $q(w_j, \ldots, w_{j-m+1}, w_{j-m}, \ldots w_{j-M+1})$ to the single element of $\mathcal{Q}_{[w_j,\ldots,w_{j-m+1}]}$ for all $(w_{j-m}, \ldots w_{j-M+1}) \in \mathcal{B}^{M-m}$.

Chapter 3

Convergence

Unfortunately, the extending window method does not always converge. The algorithm may lead to an infinite loop in both phases. In this chapter, we introduce a sufficient condition for convergence of Phase 1 in Theorem 3.1 and we categorize the algebraic numbers ω according to this condition. Theorem 3.4 enables us to construct an algorithm which checks a necessary condition for convergence of Phase 2.

3.1 Convergence of Phase 1

The following theorem gives a condition on ω which is sufficient for convergence of the Phase 1.

Theorem 3.1. *Let ω be an algebraic integer. Let $\mathcal{A}$ and $\mathcal{B}$ be finite subsets of $\mathbb{Z}[\omega]$ such that $\mathcal{A}$ contains at least one representative of each congruence class modulo β in $\mathbb{Z}[\omega]$. Then there exists a set $\mathcal{Q} \subset \mathbb{Z}[\omega]$ such that $\mathcal{B}+\mathcal{Q} \subset \mathcal{A}+\beta\mathcal{Q}$ and all elements of $\mathcal{Q}$ are limited by constant $R \in \mathbb{R}^+$ in modulus.*

Moreover, if ω is such that any complex circle centered at 0 contains only finitely many elements of $\mathbb{Z}[\omega]$, the set $\mathcal{Q}$ is finite.

Proof. Denote $A := \max\{|a| \colon a \in \mathcal{A}\}$ and $B := \max\{|b| \colon b \in \mathcal{B}\}$. Consequently, set $R := \frac{A+B}{|\beta|-1}$ and $\mathcal{Q} := \{q \in \mathbb{Z}[\omega] \colon |q| \leq R\}$. Since $A > 0$ and $|\beta| > 1$, the set $\mathcal{Q}$ is not empty. Any element $x = b + q \in \mathbb{Z}[\omega]$ with $b \in \mathcal{B}$ and $q \in \mathcal{Q}$ can be written as $x = a + \beta q'$ for some $a \in \mathcal{A}$ and $q' \in \mathbb{Z}[\omega]$ due to existence of representative of each congruence class in $\mathcal{A}$. We prove that $|q'| \leq R$:

$$|q'| = \frac{|b+q-a|}{|\beta|} \leq \frac{B+R+A}{|\beta|} \leq \frac{1}{|\beta|}\left(A+B+\frac{A+B}{|\beta|-1}\right) = \frac{A+B}{|\beta|}\left(\frac{|\beta|}{|\beta|-1}\right) = R\,.$$

Hence $q' \in \mathcal{Q}$ and thus $x = b + q \in \mathcal{A} + \beta\mathcal{Q}$.

Obviously, the set $\mathcal{Q}$ is finite if there are only finitely many elements of $\mathbb{Z}[\omega]$ bounded by the constant R. □

We plug in the alphabet $\mathcal{A}$ and input alphabet $\mathcal{B}$. We see from the proof that the candidates obtained by Algorithm 3 for an intermediate weight coefficients set $\mathcal{Q}_k$ with elements bounded by the constant R are also bounded by the constant R.

Since we start with $\mathcal{Q}_0 = \{0\}$ which is bounded by any positive constant, all intermediate weight coefficient sets $\mathcal{Q}_k$ in Algorithm 1 have elements bounded by R for all $k \in \mathbb{N}$. Hence, Phase 1 succesfully ends if there are only finitely many elements in $\mathbb{Z}[\omega]$ bounded by the constant R.

The following two lemmas characterize algebraic integers ω according to the number of elements of $\mathbb{Z}[\omega]$ in a complex circle centered around 0. The first lemma deals with real numbers and the second with non-real.

Lemma 3.2. *Let $\omega \in \mathbb{R}$ be an algebraic integer and R be a positive constant. There are only finitely many elements in the set $\{x \in \mathbb{Z}[\omega] \colon |x| < R\}$ if and only if $\omega \in \mathbb{Z}$.*

Proof. If $\omega \in \mathbb{Z}$, then $\mathbb{Z}[\omega] = \mathbb{Z}$. Trivially, there are only finitely many integers bounded by any constant R.

Suppose now that $\omega \notin \mathbb{Z}$. We show that

$$(\forall R > 0)(\exists_\infty x \in \mathbb{Z}[\omega])(|x| < R)\,.$$

The degree of ω is at least two as the only algebraic integers of degree one are integers, i.e. $\omega \in \mathbb{R} \setminus \mathbb{Q}$. Set $x := \omega - \lfloor \omega \rfloor$. We see that $0 < x < 1$ as $\omega \notin \mathbb{Z}$ and $x \in \mathbb{Z}[\omega]$ as $\omega \in \mathbb{Z}[\omega]$, $\lfloor \omega \rfloor \in \mathbb{Z} \subset \mathbb{Z}[\omega]$ and $\mathbb{Z}[\omega]$ is a ring. Hence, the sequence $(x^n)_{n\in\mathbb{N}}$ is strictly decreasing and its limit is 0 which implies the claim. □

Lemma 3.3. *Let $\omega \in \mathbb{C} \setminus \mathbb{R}$ be an algebraic integer and R be a positive constant. There are only finitely many elements in the set $\{x \in \mathbb{Z}[\omega] \colon |x| < R\}$ if and only if the degree of ω is two.*

Proof. If the degree of ω is two, the set $\mathbb{Z}[\omega]$ is generated by integer combinations of 1 and ω, i.e. it is a lattice in $\mathbb{C}$. Thus the number of elements of the set $\{x \in \mathbb{Z}[\omega] \colon |x| < R\}$ is finite.

Suppose now that the degree of ω is at least three as there are no complex algebraic integers of the degree one.

We recall that for all $r, s \in \mathbb{R} \setminus \{0\}, |r| \leq |s|$, there exists $k \in \mathbb{Z} \setminus \{0\}$ such that $|k \cdot r - s| \leq |r|/2$. It follows from the fact that $l \cdot |r| \leq |s| < (l+1)|r|$ for some $l \in \mathbb{Z} \setminus \{0\}$. We choose k from $\{\pm l, \pm(l+1)\}$ accordingly.

Now, if $|\operatorname{Im}\omega| \leq |\operatorname{Im}\omega^2|$, set $z_0 := \omega$ and $w := \omega^2$. Otherwise, set $z_0 := \omega^2$ and $w := \omega$.

We build the sequence $(z_i)_{i\in\mathbb{N}}$ recurrently:

$$z_{i+1} := k_{i+1} \cdot z_i - w$$

where $k_{i+1} \in \mathbb{Z}$ is such that

$$|\operatorname{Im} z_{i+1}| = |k_{i+1} \cdot \operatorname{Im} z_i - \operatorname{Im} w| \leq \frac{|\operatorname{Im} z_i|}{2}\,.$$

We can find such number k_{i+1} under assumption that $\operatorname{Im} z_i \neq 0$.

We prove by induction that $|\operatorname{Im} z_i| \leq |\operatorname{Im} z_0|/2^i$. Clearly, it holds for $i = 0$. Assume now that it holds for i. Then

$$|\operatorname{Im} z_{i+1}| = |k_{i+1} \cdot \operatorname{Im} z_i - \operatorname{Im} w| \leq \frac{1}{2}|\operatorname{Im} z_i| \leq \frac{|\operatorname{Im} z_0|}{2^{i+1}}\,.$$

Hence, $k_i \neq 0$ for all $i \in \mathbb{N}$ as $|\operatorname{Im} z_i| \leq |\operatorname{Im} z_0|/2^i \leq |\operatorname{Im} w|$.

Next, we show that

$$z_i = z_0 \cdot \prod_{j=1}^{i} k_j - l_i \cdot w$$

for some $l_i \in \mathbb{Z}$.

Obviously, $z_0 = z_0 \prod_{j=1}^{0} k_j - 0 \cdot w$. Assume now that it holds for i and consider

$$z_{i+1} = k_{i+1} \cdot z_i - w = k_{i+1} \left(z_0 \cdot \prod_{j=1}^{i} k_j - l_i \cdot w \right) - w = z_0 \cdot \prod_{j=1}^{i+1} k_j - (\underbrace{k_{i+1} \cdot l_i + 1}_{=:l_{i+1}})w\,.$$

Thus, $z_i \in \mathbb{Z}[\omega]$. Moreover, $z_i \notin \mathbb{Z}$. Assume in contrary that

$$z_0 \cdot \prod_{j=1}^{i} \underbrace{k_j}_{\neq 0} - l_i \cdot w = z_i \in \mathbb{Z}\,.$$

Since $z_0, w \in \{\omega, \omega^2\}, z_0 \neq w$, we have the integer polynomial of degree two with zero ω. It contradicts that the degree of ω is at least three.

Now we take $i_0 \in \mathbb{N}$ such that $|\operatorname{Im} z_0|/2^{i_0} < 1/2$. Let $k \in \mathbb{Z}$ be such that $|\operatorname{Re} z_{i_0} - k| \leq 1/2$. Set $x := z_{i_0} - k$. Then

$$|x| \leq |\operatorname{Re} x| + |\operatorname{Im} x| \leq |\operatorname{Re} z_{i_0} - k| + |\operatorname{Im} z_{i_0}| \leq \frac{1}{2} + \frac{|\operatorname{Im} z_0|}{2^{i_0}} < \frac{1}{2} + \frac{1}{2} = 1\,.$$

We know that $\operatorname{Im} z_{i_0} \neq 0$ and thus $0 < |x| < 1$.

Consider now the case that we cannot produce z_{i_1+1} in the sequence $(z_i)_{i \in \mathbb{N}}$ as there is i_1 such that $\operatorname{Im} z_{i_1} = 0$, i.e. $z_{i_1} \in \mathbb{R}$. In the same manner as before, we may prove that $z_{i_1} \in \mathbb{Z}[\omega] \setminus \mathbb{Z}$. Set $x := z_{i_1} - \lfloor z_{i_1} \rfloor$. Obviously, $0 < |x| < 1$ and $x \in \mathbb{Z}[\omega]$.

For both cases, the sequence $(x^n)_{n \in \mathbb{N}}$ has the limit 0 and $0 \neq x^n \in \mathbb{Z}[\omega]$ for all $n \in \mathbb{N}$. Thus there are infinitely many elements of $\mathbb{Z}[\omega]$ bounded by any positive constant R. □

Using Theorem 3.1 and Lemma 3.2 and 3.3, we categorize an algebraic integer ω which generates $\mathbb{Z}[\omega] \ni \beta$ as follows:

- $\omega \in \mathbb{Z} \implies$ Phase 1 converges.
- $\omega \in \mathbb{R} \setminus \mathbb{Z}$ – the sufficient condition does not hold and there is Example 5.14 for which Phase 1 does not converge. Example 5.15 proves that the condition is not necessary.
- $\omega \in \mathbb{C} \setminus \mathbb{R}$, ω being quadratic algebraic integer $\implies$ Phase 1 converges.
- $\omega \in \mathbb{C} \setminus \mathbb{R}$, ω being algebraic integer of degree ≥ 3 – the sufficient condition does not hold and there is Example 5.19 for which Phase 1 does not converge. Example 5.18 proves that the condition is not necessary.

We see that if the sufficient condition does not hold, convergence and non-convergence are both possible.

3.2 Convergence of Phase 2

For shorter notation, set

$$\mathcal{Q}_{[b^m]} := \mathcal{Q}_{\underbrace{[b,\ldots,b]}_{m}}$$

for $m \in \mathbb{N}$ and $b \in \mathcal{B}$.

Obviously, finiteness of Phase 2 implies that there exists a length of window M such that the set $\mathcal{Q}_{[b^m]}$ contains only one element for all $b \in \mathcal{B}$. The following theorem is used for the construction of Algorithm 6 which checks this necessary condition.

Theorem 3.4. *Let $m_0 \in \mathbb{N}$ and $b \in \mathcal{B}$ be such that sets $\mathcal{Q}_{[b^{m_0}]}$ and $\mathcal{Q}_{[b^{m_0-1}]}$ produced by Algorithm 5 within Phase 2 have the same size. Then*

$$\#\mathcal{Q}_{[b^m]} = \#\mathcal{Q}_{[b^{m_0}]} \qquad \forall m \geq m_0 - 1\,.$$

Particularly, if $\#\mathcal{Q}_{[b^{m_0}]} \geq 2$, Phase 2 does not converge.

Proof. We prove the base case of induction with respect to m. For $m = m_0 + 1$, the set $\mathcal{Q}_{[b^{m_0+1}]}$ is found by Algorithm 5 such that

$$b + \mathcal{Q}_{[b^{m_0}]} \subset \mathcal{A} + \beta\mathcal{Q}_{[b^{m_0+1}]}$$

and set $\mathcal{Q}_{[b^{m_0}]}$ is found by the same algorithm such that

$$b + \mathcal{Q}_{[b^{m_0-1}]} \subset \mathcal{A} + \beta\mathcal{Q}_{[b^{m_0}]}\,.$$

As $\mathcal{Q}_{[b^{m_0}]} \subset \mathcal{Q}_{[b^{m_0-1}]}$, the assumption of the same size implies

$$\mathcal{Q}_{[b^{m_0}]} = \mathcal{Q}_{[b^{m_0-1}]}\,.$$

It means that Algorithm 5 runs with the same input and hence

$$\mathcal{Q}_{[b^{m_0+1}]} = \mathcal{Q}_{[b^{m_0}]}\,.$$

The inductive step of the proof for $m + 1$ is analogous to the base case.

Phase 2 ends when there is only one element in $\mathcal{Q}_{[w_j,\ldots,w_{j-m+1}]}$ for all $(w_j, \ldots, w_{j-m+1}) \in \mathcal{B}^m$ for some fixed length of window m. But if $\#\mathcal{Q}_{[b^{m_0}]} \geq 2$, size of $\mathcal{Q}_{[b,\ldots,b]}$ does not decrease despite of extending the length of window. □

Now we describe Algorithm 6 which checks whether Phase 2 stops when it processes input digits $bb\ldots b$. For arbitrary m, sets $\mathcal{Q}_{[b^m]}$ can be easily constructed separately for each $b \in \mathcal{B}$. We build the set $\mathcal{Q}_{[b^m]}$ for input digits $bb\ldots b$ in the same way as in Phase 2. This means that we first search for $\mathcal{Q}_{[b]}$ such that

$$b + \mathcal{Q} \subset \mathcal{A} + \beta\mathcal{Q}_{[b]}\,.$$

Until the set $\mathcal{Q}_{[b^m]}$ contains only one element, we increment the length of the window m and, using Algorithm 5, we build the subset $\mathcal{Q}_{[b^{m+1}]}$ of the set $\mathcal{Q}_{[b^m]}$ such that

$$b + \mathcal{Q}_{[b^m]} \subset \mathcal{A} + \beta\mathcal{Q}_{[b^{m+1}]}\,.$$

In addition, we check whether the set $\mathcal{Q}_{[b^{m+1}]}$ is strictly smaller than the set $\mathcal{Q}_{[b^m]}$. If not, we know by Theorem 3.4 that Phase 2 does not converge because of the input digits $bb\ldots b$.

Thus, running of Algorithm 6 for each input digit $b \in \mathcal{B}$ can reveal non-finiteness of Phase 2.

Algorithm 6 Check input $bb\dots b$

Input: Weight coefficient set $\mathcal{Q}$, digit $b \in \mathcal{B}$

$m := 1$

Find minimal set $\mathcal{Q}_{[b^1]} \subset \mathcal{Q}$ such that

$$b + \mathcal{Q} \subset \mathcal{A} + \beta \mathcal{Q}_{[b^1]} \,.$$

while $\#\mathcal{Q}_{[b^m]} > 1$ **do**

  $m := m + 1$

  By Algorithm 5, find minimal set $\mathcal{Q}_{[b^m]} \subset \mathcal{Q}_{[b^{m-1}]}$ such that

$$b + \mathcal{Q}_{[b^{m-1}]} \subset \mathcal{A} + \beta \mathcal{Q}_{[b^m]} \,.$$

  if $\#\mathcal{Q}_{[b^m]} = \#\mathcal{Q}_{[b^{m-1}]}$ **then**

    return Phase 2 does not converge for input $bb\dots b$.

  end if

end while

return Weight coefficient for input $bb\dots b$ is the only element of $\mathcal{Q}_{[b^m]}$.

Chapter 4

Design and implementation

The designed method requires the arithmetics in $\mathbb{Z}[\omega]$. Therefore, we have chosen Python-based programming language SageMath for the implementation of method as it contains many ready-to-use mathematical structure. Using SageMath is very convenient as it also offers easily usable datastructures or tools for plotting. Thus the code is more readible and we may focus on the algorithmic part of problem. On the other hand, SageMath is considerably slower than for example C++ or other low-level languages. Nevertheless, it is sufficient for our purpose.

The implementation is object-oriented. It consists of four classes. Class *AlgorithmForParallelAddition* contains structures for computations in $\mathbb{Z}[\omega]$. Specifically, we use the provided class *PolynomialQuotientRing* to represent elements of $\mathbb{Z}[\omega]$ and *NumberField* for obtaining numerical complex value of them. The class also links neccessary instances and functions to construct algorithm for parallel addition by the extending window method for an algebraic integer ω given by its minimal polynomial p and approximate complex value, a base $\beta \in \mathbb{Z}[\omega]$, an alphabet $\mathcal{A} \subset \mathbb{Z}[\omega]$ and an input alphabet $\mathcal{B}$. Phase 1 of the extending window method is implemented in class *WeightCoefficientsSetSearch* and Phase 2 in *WeightFunctionSearch*. Class *WeightFunction* holds the weight function q computed in Phase 2. All classes are described in the following sections including lists of the important methods with short desription. Sometimes, notation from Chapter 2 is used for better understanding. For all implemented methods, see commented source code.

Basically, weight function can be found just by creating an instance of *AlgorithmForParallelAddition* and calling **findWeightFunction()**. For more comfortable usage, our implementation includes two interfaces – shell version and graphic user interface using interact in SageMath Cloud. The whole implementation is on the attached CD or it can be downloaded from `https://github.com/Legersky/ParallelAddition`.

4.1 Class AlgorithmForParallelAddition

This class constructs neccessary structures for computation in $\mathbb{Z}[\omega]$. It is *PolynomialQuotientRing* obtained as a *PolynomialRing* over integers factored by polynomial p. This is used for representation of elements of $\mathbb{Z}[\omega]$ and arithmetics. We remark that it is independent on the choice of root of the minimal polynomial p. But as we need also comparisons of numbers in $\mathbb{Z}[\omega]$ in modulus, we specify ω by its approximate complex value and we form a factor ring of rational polynomials by using class *NumberField*. This enables us to get absolute values of elements of $\mathbb{Z}[\omega]$ which can be then compared.

Method **findWeightFunction()** links together both phases of the extending window method to find the weight function q. That is used in the methods for addition and digit set conversion to process them as local functions. There are also verification methods.

Moreover, many methods for printing, plotting and saving outputs are implemented.

The constructor of class *AlgorithmForParallelAddition* is
__init__(*minPol_str, embd, alphabet, base, name='NumerationSystem', inputAlphabet='', printLog=True, printLogLatex=False, verbose=0***)**

> Take *minPol_str* which is symbolic expression in the variable x of minimal polynomial p. The closest root of *minPol_str* to *embd* is used as the ring generator ω. The structures for $\mathbb{Z}[\omega]$ are constructed as described above. Setters **setAlphabet**(*alphabet*), **setInputAlphabet**(A) and **setBase**(*base*) are called. Messages saved to logfile during existence of an instance are printed (using LaTeX) on standard output depending on *printLog* and *printLogLatex*. The level of messages for a development is set by *verbose*.

Methods for the construction of an addition algorithm which is computable in parallel by the designed extending window method are following:
_findWeightCoefSet(*max_iterations, method_number***)**

> Create an instance of *WeightCoefficientsSetSearch(method_number)* and call its method **findWeightCoefficientsSet**(*max_iterations*) to obtain a weight coefficients set $\mathcal{Q}$.

_findWeightFunction(*max_input_length,method_number***)**

> Create an instance of *WeightFunctionSearch(method_number)* and call its methods **check_one_letter_inputs**(*max_input_length*) and **findWeightFunction**(*max_input_length*) to obtain a weight function q.

findWeightFunction(*max_iterations, max_input_length, method_weightCoefSet=2, method_weightFunSearch=4***)**

> Return the weight function q obtaind by calling **_findWeightCoefSet**(*max_iterations, method_weightCoefSet*) and **_findWeightFunction**(*max_input_length, method_weightFunSearch*).

The important function for the searching for possible weight coefficients is
divideByBase(*divided_number***)**

> Using Theorem 1.7, check if *divided_number* is divisible by base β. If it is so, return the result of division, else return *None*.

Methods for the addition and the digit set conversion computable in parallel are following:
addParallel(*a,b***)**

> Sum up numbers represented by lists of digits a and b digitwise and convert the result by **parallelConversion**().

parallelConversion(*_w***)**

> Return $(\beta, \mathcal{A})$-representation of number represented by list *_w* of digits in input alphabet $\mathcal{B}$. It is computed locally according to the equation 2.3 and using weight function q.

localConversion(*w***)**

Return converted digit according to equation 2.3 for list of input digits w.

The correcteness of the implementation of the extending window for a given numeration system can be verified by

sanityCheck_conversion(*num_digits***)**

Check whether the values of all possible numbers of the length *num_digits* with digits in the input alphabet $\mathcal{B}$ are the same as their $(\beta, \mathcal{A})$-representation obtained by **parallelConversion**().

4.2 Class WeightCoefficientsSetSearch

Class *WeightCoefficientsSetSearch* implements Phase 1 of the extending window method described in Section 2.1.1. The most important method is **findWeightCoefficientsSet**() which returns the weight coefficients set $\mathcal{Q}$.

The constructor of the class is

__init__(*algForParallelAdd, method***)**

Initialize the ring generator ω, base β, alphabet $\mathcal{A}$ and input alphabet $\mathcal{B}$ by values obtained from *algForParallelAdd*. The parameter *method* characterizes the way of the choice from the possible candidates for the weight coefficients set.

Methods implementing Phase 1 are following:

_findCandidates(C**)**

Following Algorithm 3, return the list of lists *candidates* such that each element in C is covered by any value of the appropriate list in *candidates*.

_chooseQk_FromCandidates(*candidates***)**

Take the previous intermediate weight coefficients set $\mathcal{Q}_k$ as the class attribute and choose from *candidates* the intermediate weight coefficients set $\mathcal{Q}_{k+1}$ by Algorithm 2.

_getQk(C**)**

Links together methods **_findCandidates**(C) and **_chooseQk_FromCandidates**() to return itermediate weight coefficients set $\mathcal{Q}_k$.

findWeightCoefficientsSet(*maxIterations***)**

According to Algorithm 1, return the weight coefficients set $\mathcal{Q}$ which is build iteratively by using **_getQk**(). The computation is aborted if the number of iterations exceeds *maxIterations*.

4.3 Class WeightFunctionSearch

Phase 2 of the extending window method from Section 2.1.2 is implemented in this class. The weight function q is returned by method **findWeightFunction**(). The constructor of the class is

__init__(*algForParallelAdd, weightCoefSet, method***)**

The ring generator ω, base β, alphabet $\mathcal{A}$ and input alphabet $\mathcal{B}$ are initialised by the values obtained from *algForParallelAdd*. The weight coefficients set $\mathcal{Q}$ is set to *weightCoefSet*. The parameter *method* characterizes the way of the choice of possible weight coefficients set for given input from the previous one. The attribute *_Qw_w* is set to an empty dictionary. It serves for saving possible weight coefficients for possible tuples of input digits.

The following methods are used for search for weight function q:

_find_weightCoef_for_comb_B(*combinations***)**
Take all unsolved inputs $w_j, \dots, w_{j-m+1} \in \mathcal{B}^m$ in *combinations*, extend them by all letters $w_{j-m} \in \mathcal{B}$ and find possible weight coefficients set $\mathcal{Q}_{[w_j,\dots,w_{j-m}]}$ by the method **_findQw**$((w_j, \dots, w_{j-m}))$. If there is only one element in $\mathcal{Q}_{[w_j,\dots,w_{j-m}]}$, it is saved as a solved input of weight function q. Otherwise, the input combination $w_j, \dots, w_{j-m}$ is saved as an unsolved input which requires extending of window. The unsolved combinations are returned.

_findQw(*w_tuple***)**
Return the set $\mathcal{Q}_{[\text{w\_tuple}]}$ obtained by Algorithm 5. The set of possible carries for *w_tuple* without the first digit and the previous set of possible weight coefficients, which are neccessary for computation, are taken from the attribute *_Qw_w* of the class.

findWeightFunction(*max_input_length***)**
Return weight function q unless the length of window exceeds *max_input_length*. Then an exception is raised. It implements Algorithm 4 by repetetive calling of the method **_find_weightCoef_for_comb_B**() which extends length of window. This is done until all possible combinations of input digits are solved for some length of window m, i.e. $\max\{\#\mathcal{Q}_{[w_j,\dots,w_{j-m+1}]} : (w_j, \dots, w_{j-m+1}) \in \mathcal{B}^m\} = 1$.

check_one_letter_inputs(*max_input_length***)**
The method checks by Algorithm 6 if there is a unique weight coefficient for inputs $(b, b, \dots, b) \in \mathcal{B}^m$ for some length of window m. Using Theorem 3.1, an exception is raised in the case of an infinite loop. Otherwise the list of inputs $(b, b, \dots, b)$ which have the largest length of the window is returned.

4.4 Class WeightFunction

This class serves for saving the weight function q. The constructor is

__init__(B)
Set the input alphabet to B and the maximum length of window to 1. Initialise the attribute *_mapping* to an empty dictionary for saving the weight function q.

The methods for saving and calling are following:

addWeightCoefToInput(*_input, coef***)**
Save the weight coefficient *coef* for *_input* to *_mapping*. The digits of *_input* must be in the input alphabet.

getWeightCoef(*w***)**
The digits of the list w are taken from the left until the weight coefficient in the dictionary *_mapping* is found.

The result of the method **getWeightCoef**() is used to make this class callable, i.e. if *_q* is an instance of *WeightFuntion*, then *_q*.**getWeightCoef**(w) is the same as *_q*(w).

4.5 User interfaces

We provide two interfaces for running of the implemented extending window method – the shell version and graphic user interface.

4.5.1 Shell

The implementation of the extending window method is launched in a shell by typping `sage parallelAdd.sage <input_sample.sage>`. The parameters of the numeration system and the setting of outputs and computation is given by SageMath file `input_sample.sage`. See Appendix **??** for the example of such a file with parameters of Eisenstein numeration system.

The name of the numeration system, minimal polynomial of generator ω, an approximate value of ω, the base β, alphabet $\mathcal{A}$ and input alphabet $\mathcal{B}$ are set in the part INPUTS. The maximum number of iterations in Phase 1, maximal length of the window in Phase 2 and the launching of the sanity check are set in SETTING.

The boolean values in the part SAVING determines which formats of the outputs are saved. All outputs are saved in the folder `./ouputs/name/`. The general info about the computatin can be saved in .tex format, the computed weight function and local digit set conversion in .csv format. The inputs setting saved as dictionary can be loaded by the interact interface. The log of the whole computation can be saved as .txt file. There is also an option to save unsolved combinations in Phase 2 in .csv format in the case of the interruption of the program.

According to the boolean values in the part IMAGES, the figures the alphabet, input alphabet, weight coefficients set or part of the lattice of $\mathbb{Z}[\omega]$ with alphabet shifted into points which are divisible by the base β are saved in .png format to folder `./ouputs/name/img/`. Optionally, the weight coefficients set is plotted with bound given by the proof of Theorem 3.1 The images of individual steps of both phases of the extending window method can be saved, too. For Phase 2, the search for the weight coefficient is plotted for digits given by `phase2_input`.

The program print out all inputs and then it computes the weight function q by calling **findWeightFunction**(*max_iterations, max_input_length*). The increments of the weight coefficients set in each iteration of Phase 1 are printed and then also the obtained weight coefficients set $\mathcal{Q}$. The longest inputs given by repetetion of one letter are printed after the computation of **check_one_letter_inputs**(*max_input_length*). During computing of Phase 2, the current length of window and the number of saved combinations are printed. Finally, the length of window, elapsed time and info about saved outputs are printed.

It is possible to batch process all input files in one folder by executing the bash script `parAdd_batch <folder_name>`.

4.5.2 Interact in SageMath Cloud

The graphic user interface is implemented using interact in SageMath Cloud. The parts of the interact are on Figure **??**, **??** and **??** in Appendix **??**. The functionality is basically the same as the shell version. After executing of the cell by Shift+Enter, the parameters of the numeration system are filled in the corresponding spaces or one of the previous settings is loaded by typing its name. By default, the last inputs are shown in the form. The inputs are submitted by pressing the button Update. Using check-boxes, the formats of output are chosen and the search for the weight function is launched by pressing second button Update.

The printed output is similar to the shell output. In addition, it contains the figures and it is formatted using LaTeX. Moreover, the sanity check can be run for a given length, the weight coefficient for a tuple of input digits is returned or the images of individual steps of both phases are shown and saved.

Chapter 5

Testing

We have tested several bases with different alphabets. The bases are chosen such they have no conjugates of modulo 1 according to Theorem 1.1. The sizes of the integer alphabets are given by the lower bound from Theorem 1.3. Since this theorem assumes only an integer alphabet, we have tested also non-integer alphabets of smaller size. The input alphabet $\mathcal{B}$ is always $\mathcal{A} + \mathcal{A}$.

Table 5.1 summarizes the tested numeration systems which are described in the sections below. The first column of the table says if the sufficient condition given by Theorem 3.1 is satisfied. The second one shows whether Phase 1 was successful (✓) or not (✗) for the given numeration system. By not successful we mean that the weight coefficients set of a reasonable size was not found. The third column is the control of the necessary condition for the convergence of Phase 2 by Algorithm 6, i.e. if there is the output of the weight function q for input digits $b, b, \dots, b$ for all $b \in \mathcal{B}$. The results of Phase 2 are in the last column. Notice, that the next step of the extending window method is not processed without the previous one (–).

We see that Phase 1 was successful in all cases when the sufficient condition holds, as we expected. Moreover, we have examples when it converges without the sufficient condition. We remark that the absolute values of all conjugates of the base are greater than 1 in Examples 5.15 and 5.18, whereas there is a conjugate whose modulus is smaller than 1 in Examples 5.14 and 5.19.

Notice that there is no example when necessary condition for convergence of Phase 2 holds, but Phase 2 does not stop. The possible reasoning of non-convergence of Phase 2 for the quadratic bases with integer alphabet is that the coefficients of the rewriting rule $x - \beta$ are not integers.

The complete results including log files and images can be found on the attached CD or `https://github.com/Legersky/ParallelAddition`.

Name	Ex.	Alph.	Suff. c.	Phase 1	Necess. c.	Phase 2
Eisenstein_1-block_complex	5.1	yes	yes	✓	✓	✓
Eisenstein_1-block_integer	5.3	yes	yes	✓	✗	–
Eisenstein_1-block_small_complex	5.2	no	–	–	–	–
Eisenstein_2-block	5.4	no	–	–	–	–
Eisenstein_2-block_4elements	5.5	yes	yes	✓	✗	–
Penney_1-block_complex	5.6	yes	✓	✓	✓	
Penney_1-block_complex_small	5.7	yes	✓	✗	–	
Penney_1-block_integer	5.8	yes	✓	✗	–	
Penney_2-block_integer	5.9	yes	✓	✓	✓	
Quadratic+1-2+2-1_1-block_complex	5.10	yes	✓	✓	✓	
Quadratic+1-2+2-1_1-block_integer	5.11	yes	✓	✗	–	
Quadratic+1+4+5_1-block_complex	5.12	yes	✓	✓	✓	
Quadratic+1+3+5_1-block_complex	5.13	yes	✓	✗	–	
Quadratic+1-5+3_1-block_integer	5.14	no	✗	–	–	
Quadratic+1-5+5_1-block_real	5.15	no	✓	✗	–	
base_2	5.16	yes	✓	✓	✓	
base_4	5.17	yes	✓	✓	✓	
Cubic+1+1-5+5_complex	5.18	no	✓	✗	–	
Cubic+1+1-1+1_complex	5.19	no	✗	–	–	

Table 5.1: Results of extending window method.

5.1 Eisenstein base $\beta = -\frac{3}{2} + \frac{\imath\sqrt{3}}{2}$

Eisenstein base β equals $\omega - 1$ with $\omega = -\frac{1}{2} + \frac{\imath\sqrt{3}}{2}$. The minimal polynomial of the generator ω is $x^2 + x + 1$ and the minimal polynomial of the base β is $x^2 + 3x + 3$. Thus, the lower bound for the size of the integer alphabet given by Theorem 1.3 is 7. We have tested the complex, smaller complex and integer alphabet.

Example 5.1. Eisenstein_1-block_complex
The alphabet $\mathcal{A} = \{0, 1, -1, \omega, -\omega, -\omega - 1, \omega + 1\}$.
The result of the extending window method is:

1. Phase 1 was successful. The number of elements in the weight coefficient set $\mathcal{Q}$ is 19.
2. There is a unique weight coefficient for input $b, b, \ldots, b$ for all $b \in \mathcal{B}$.
3. Phase 2 was successful. The length of window m of the weight function q is 3.

Example 5.2. Eisenstein_1-block_small_complex
The alphabet $\mathcal{A} = \{0, 1, \omega, \omega + 1\}$.
The elements $\{\omega + 2, 2\omega, 2\omega + 1, 2\omega + 2\}$ have no representative modulo $\beta - 1$ in the alphabet $\mathcal{A}$.

Example 5.3. Eisenstein_1-block_integer
The alphabet $\mathcal{A} = \{0, 1, -1, 2, -2, 3, -3\}$.
The result of the extending window method is:

1. Phase 1 was successful. The number of elements in the weight coefficient set $\mathcal{Q}$ is 52.

2. There is not unique weight coefficient for input $b, b, \ldots, b$ for $b \in \{4, 5, 6, -6, -5, -4, -3\}$ for fixed length of window. Thus Phase 2 does not converge.

We may also study so-called 2-block parallel addition. Roughly speaking, we consider two digits after each other in a $(\beta, \mathcal{A})$-representation as one digit of the $(\beta^2, \mathcal{A}+\beta\mathcal{A})$-representation of the same number. So we shift from base β to $\beta^2 = -3 - 3\beta = -3\omega$ which has the minimal polynomial $x^2 - 3x + 9$. We have tested the shifted alphabets $\{0, \pm 1\} + \beta\{0, \pm 1\}$ and $\{0, 1, \omega, \omega + 1\} + \beta\{0, 1, \omega, \omega + 1\}$.

Example 5.4. Eisenstein_2-block
The alphabet $\mathcal{A} = \{0, 1, -1, \omega, -\omega, \omega - 1, -\omega + 1, \omega - 2, -\omega + 2\}$.
The elements $\{2\omega - 1, 2\omega, \omega + 1, -\omega - 1, -2\omega, -2\omega + 1\}$ have no representative modulo $\beta - 1$ in the alphabet $\mathcal{A}$.

Example 5.5. Eisenstein_2-block_4elements
The alphabet $\mathcal{A} = \{0, 1, -1, \omega, -\omega, \omega + 1, -\omega - 1, \omega - 1, 2\omega - 1, 2\omega, -2\omega, -2\omega - 1, -2, -\omega - 2\}$.
The result of the extending window method is:

1. Phase 1 was successful. The number of elements in the weight coefficient set $\mathcal{Q}$ is 17.

2. There is not unique weight coefficient for input $b, b, \ldots, b$ for $b \in \{2\omega - 1, \omega + 1, -2\omega, -\omega - 2, -4\}$ for fixed length of window. Thus Phase 2 does not converge.

5.2 Penney base $\beta = -1 + \imath$

Penney base $\beta = -1 + \omega$ where $\omega = \imath$. The minimal polynomial of the base β is $x^2 + 2x + 2$. We have tested the complex, smaller complex and integer alphabet. The lower bound for the size of the integer alphabet is 5.

Example 5.6. Penney_1-block_complex
The alphabet $\mathcal{A} = \{0, 1, -1, \omega, -\omega\}$.
The result of the extending window method is:

1. Phase 1 was succesful. The number of elements in the weight coefficient set $\mathcal{Q}$ is 45.

2. There is a unique weight coefficient for input $b, b, \ldots, b$ for all $b \in \mathcal{B}$.

3. Phase 2 was succesful. The length of window m of the weight function q is 6.

Example 5.7. Penney_1-block_complex_small
The alphabet $\mathcal{A} = \{0, 1, \omega\}$.
The result of the extending window method is:

1. Phase 1 was succesful. The number of elements in the weight coefficient set $\mathcal{Q}$ is 22.

2. There is not unique weight coefficient for input $b, b, \ldots, b$ for the $b = \omega + 1$ for fixed length of window. Thus Phase 2 does not converge.

Example 5.8. Penney_1-block_integer
The alphabet $\mathcal{A} = \{0, 1, -1, 2, -2\}$.
The result of the extending window method is:

1. Phase 1 was succesful. The number of elements in the weight coefficient set $\mathcal{Q}$ is 47.
2. There is not unique weight coefficient for input $b, b, \ldots, b$ for the $b = 1$ for fixed length of window. Thus Phase 2 does not converge.

For 2-block parallel addition, we shift from base β to $\beta^2 = -2 - 2\beta = -2\omega$ which has the minimal polynomial $x^2 + 4$. We have tested the shifted alphabet $\{0, \pm 1\} + \beta\{0, \pm 1\}$.

Example 5.9. Penney_2-block_integer
The alphabet $\mathcal{A} = \{0, 1, -1, \omega, -\omega, \omega - 1, -\omega + 1, \omega - 2, -\omega + 2\}$.
The result of the extending window method is:

1. Phase 1 was succesful. The number of elements in the weight coefficient set $\mathcal{Q}$ is 27.
2. There is a unique weight coefficient for input $b, b, \ldots, b$ for all $b \in \mathcal{B}$.
3. Phase 2 was succesful. The lenght of window m of the weight function q is 5.

5.3 Base $\beta = 1 + \imath$

The following numeration systems have the base $\beta = 1 + \imath$ with the minimal polynomial $x^2 - 2x + 2$. We have tested the complex and integer alphabet.

Example 5.10. Quadratic+1-2+2_1-block_complex
The alphabet $\mathcal{A} = \{0, 1, -1, \omega - 1, -\omega + 1\}$.
The result of the extending window method is:

1. Phase 1 was succesful. The number of elements in the weight coefficient set $\mathcal{Q}$ is 45.
2. There is a unique weight coefficient for input $b, b, \ldots, b$ for all $b \in \mathcal{B}$.
3. Phase 2 was succesful. The lenght of window m of the weight function q is 6.

Example 5.11. Quadratic+1-2+2_1-block_integer
The alphabet $\mathcal{A} = \{0, 1, -1, 2, -2\}$.
The result of the extending window method is:

1. Phase 1 was succesful. The number of elements in the weight coefficient set $\mathcal{Q}$ is 46.
2. There is not unique weight coefficient for input $b, b, \ldots, b$ for the $b = 0$ for fixed length of window. Thus Phase 2 does not converge.

5.4 Base $\beta = -2 + \imath$

The base $\beta = -2 + \imath$ has the minimal polynomial $x^2 + 4x + 5$.

Example 5.12. Quadratic+1+4+5_1-block_complex
The alphabet $\mathcal{A} = \{0, 1, -1, \omega, -\omega, \omega + 1, -\omega - 1, \omega - 1, -\omega - 2, -2\}$.
The result of the extending window method is:

1. Phase 1 was succesful. The number of elements in the weight coefficient set $\mathcal{Q}$ is 17.
2. There is a unique weight coefficient for input $b, b, \ldots, b$ for all $b \in \mathcal{B}$.
3. Phase 2 was succesful. The lenght of window m of the weight function q is 3.

5.5 Base $\beta = -\frac{3}{2} + \frac{\imath\sqrt{11}}{2}$

The base $\beta = -\frac{3}{2} + \frac{\imath\sqrt{11}}{2}$ has the minimal polynomial $x^2 + 3x + 5$.

Example 5.13. Quadratic+1+3+5_1-block_complex
The alphabet $\mathcal{A} = \{0, 1, -1, \omega + 1, -\omega - 1, \omega + 2, -\omega - 2, \omega + 3, -\omega - 3\}$.
The result of the extending window method is:

1. Phase 1 was succesful. The number of elements in the weight coefficient set $\mathcal{Q}$ is 11.
2. There is not unique weight coefficient for input $b, b, \ldots, b$ for the $b = 2\omega + 2$ for fixed length of window. Thus Phase 2 does not converge.

5.6 Base $\beta = \frac{5}{2} + \frac{\imath\sqrt{13}}{2}$

The minimal polynomial of the base $\beta = \frac{5}{2} + \frac{\imath\sqrt{13}}{2}$ is $x^2 - 5x + 3$.

Example 5.14. Quadratic+1-5+3_1-block_integer
The alphabet $\mathcal{A} = \{0, 1, 2, 3, 4, 5, 6\}$.
The result of the extending window method is:

1. Phase 1 was not succesful.

5.7 Base $\beta = \frac{5}{2} + \frac{\sqrt{5}}{2}$

The minimal polynomial of the base $\beta = \frac{5}{2} + \frac{\sqrt{5}}{2}$ is $x^2 - 5x + 5$.

Example 5.15. Quadratic+1-5+5_1-block_real
The alphabet $\mathcal{A} = \{0, \omega + 1, -\omega - 1, \omega + 2, -\omega - 2\}$.
The result of the extending window method is:

1. Phase 1 was succesful. The number of elements in the weight coefficient set $\mathcal{Q}$ is 127.
2. There is not unique weight coefficient for input $b, b, \ldots, b$ for the $b = 0$ for fixed length of window. Thus Phase 2 does not converge.

5.8 Integer bases

We have also tested integer bases 2, resp. 4 with the alphabets $\{0, \pm 1\}$, resp. $\{0, \pm 1, \pm 2\}$. Since the minimal polynomial is $x - 2$, resp. $x - 4$, the minimal size of the integer alphabet given by Theorem 1.3 is $|1 - 2| + 2 = 3$, resp. 4.

Example 5.16. base_2
The alphabet $\mathcal{A} = \{0, 1, -1\}$.
The result of the extending window method is:

1. Phase 1 was succesful. The number of elements in the weight coefficient set $\mathcal{Q}$ is 3.
2. There is a unique weight coefficient for input $b, b, \ldots, b$ for all $b \in \mathcal{B}$.

3. Phase 2 was succesful. The lenght of window m of the weight function q is 2.

Example 5.17. base_4
The alphabet $\mathcal{A} = \{0, 1, -1, 2, -2\}$.
The result of the extending window method is:

1. Phase 1 was succesful. The number of elements in the weight coefficient set $\mathcal{Q}$ is 3.
2. There is a unique weight coefficient for input $b, b, \ldots, b$ for all $b \in \mathcal{B}$.
3. Phase 2 was succesful. The lenght of window m of the weight function q is 2.

5.9 Cubic bases

The base is chosen as the zero of cubic polynomial $x^3 + x - 5x + 5$, resp. $x^3 + x - x + 1$, which is the greatest one in modulus.

Example 5.18. Cubic+1+1-5+5_complex
The alphabet $\mathcal{A} = \{0, \omega + 1, \omega + 2, -\omega - 1, -\omega - 2\}$.
The result of the extending window method is:

1. Phase 1 was succesful. The number of elements in the weight coefficient set $\mathcal{Q}$ is 345.
2. There is not unique weight coefficient for input $b, b, \ldots, b$ for the $b = 0$ for fixed length of window. Thus Phase 2 does not converge.

Example 5.19. Cubic+1+1-1+1_complex
The alphabet $\mathcal{A} = \{0, \omega + 1, \omega + 2, -\omega - 1, -\omega - 2\}$.
The result of the extending window method is:

1. Phase 1 was not succesful.

Summary

The main goal of this thesis was to implement the extending window method in SageMath. In order to do that,first, we have recalled the definitions and the previous results in the field of parallel addition algorithms. We have proved Theorem 1.7 which is necessary tool for computation in $\mathbb{Z}[\omega]$.

From the general concept of construction of parallel addition algorithm, we have designed both phases of the extending window method for rewriting rule $x-\beta$. The sufficient condition for the convergence of Phase 1, i.e. the search for the weight coefficients set $\mathcal{Q}$, have been introduced in Theorem 3.1. The algebraic integer ω have been categorized according to this sufficient condition. Next, we have developed Algorithm 6 which checks the necessary condition for the convergence of Phase 2, i.e. the search for the weight function q. This algorithm is based on Theorem 3.4 which we have proved.

Both phases were implemented in SageMath. The graphic user interface is provided for comfortable using in SageMath Cloud and the shell iterface enables us to compute the weight function for the more demanding numeration systems.

We have tested several examples of numeration systems. Our program found the weight function successfuly for many of them. We have also examples for which Phase 1 converges, although the sufficient condition given by Theorem 3.1 does not hold.

Many questions remains open:

- What is the necessary condition of convergence of Phase 1? Is there any connection with modulus of the conjugates of the base?
- Can we ensure the convergence of Phase 1 for wider class of numeration systems, for instance by using different metric in $\mathbb{C}$?
- Is there any example when necessary condition of Phase 2 is satisfied, but Phase 2 does not converge?
- How can we improve Phase 2 to converge for more numeration systems?

We focus on these questions in the future work.

Bibliography

[1] A. Avizienis. Signed-digit numbe representations for fast parallel arithmetic. *Electronic Computers, IRE Transactions on*, EC-10(3):389–400, Sept 1961.

[2] C.Y. Chow and J.E. Robertson. Logical design of a redundant binary adder. In *Computer Arithmetic (ARITH), 1978 IEEE 4th Symposium on*, pages 109–115, Oct 1978.

[3] C. Frougny, P. Heller, E. Pelantová, and M. Svobodová. k-block parallel addition versus 1-block parallel addition in non-standard numeration systems. *Theoretical Computer Science*, 543:52 – 67, 2014.

[4] C. Frougny, E. Pelantová, and M. Svobodová. Parallel addition in non-standard numeration systems. *Theoretical Computer Science*, 412(41):5714–5727, 2011.

[5] C. Frougny, E. Pelantová, and M. Svobodová. Minimal digit sets for parallel addition in non-standard numeration systems. *J. Integer Seq.*, 16(2):Article 13.2.17, 36, 2013.

[6] R.A. Horn and C.R. Johnson. *Matrix Analysis.* Cambridge University Press, 1990.

[7] P. Kornerup. Necessary and sufficient conditions for parallel, constant time conversion and addition. In *IEEE Computer Society*, pages 152–156, 1999.

[8] A. M. Nielsen and P. Kornerup. Redundant radix representations of rings. *IEEE Transactions on Computers*, 48(11):1153–1165, 1999.

[9] M. Svobodová. Private communication, 2014–2015.

Appendices

A Illustration of Phase 1

Figures **??** – **??** illustrates the construction of the weight coefficients set $\mathcal{Q}$ for the Eisenstein numeration system with complex alphabet (see Example 5.1).

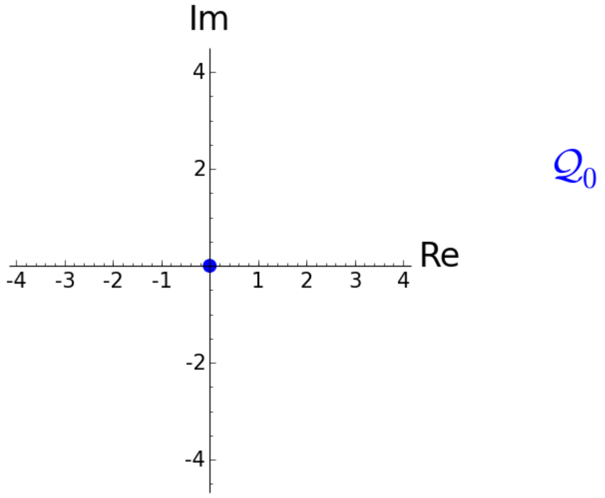


Figure 1: skjvfdn

B Illustration of Phase 2

The construction of set $\mathcal{Q}_{[\omega,1,2]}$ for the Eisenstein numeration system (see Example 5.1) is illustrated on Figures ?? – ??.

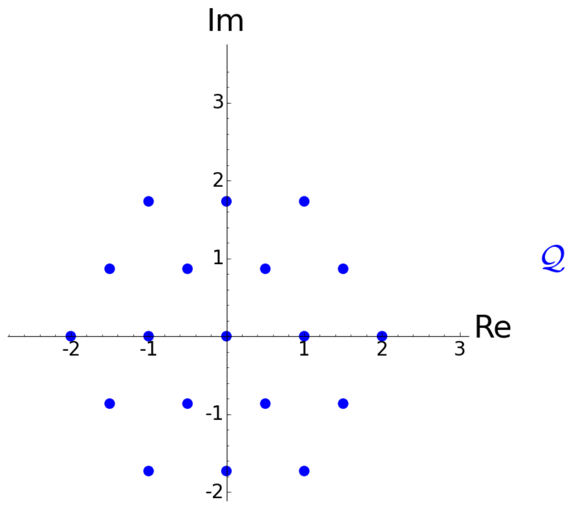


Figure 2: Construction of the set $\mathcal{Q}_{[\omega,1,2]}$ for Eisenstein numeration system

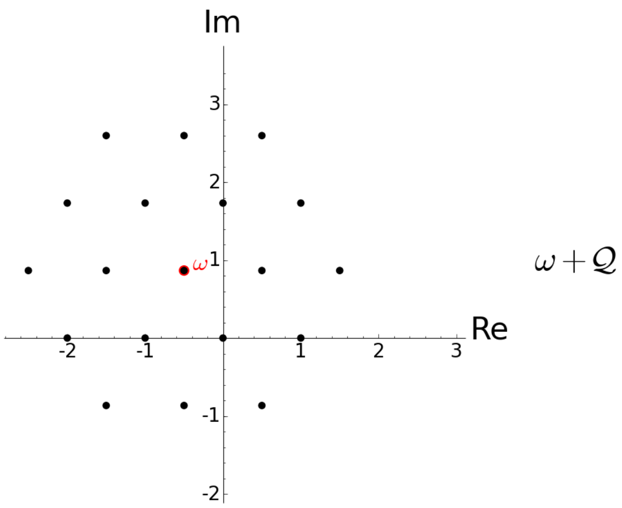


Figure 3: Construction of the set $\mathcal{Q}_{[\omega,1,2]}$ for Eisenstein numeration system

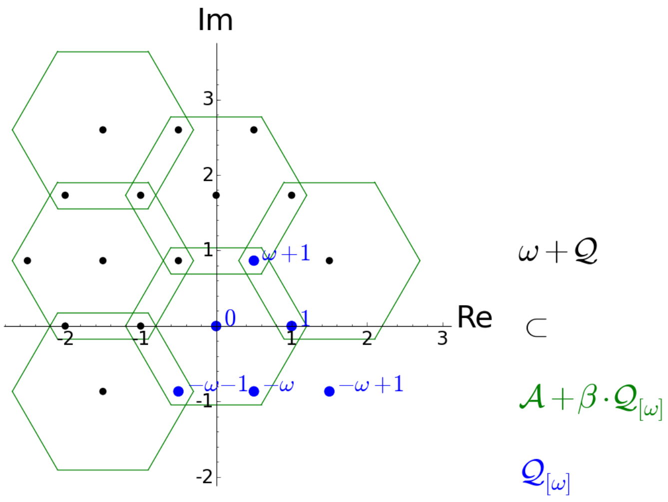


Figure 4: Construction of the set $\mathcal{Q}_{[\omega,1,2]}$ for Eisenstein numeration system

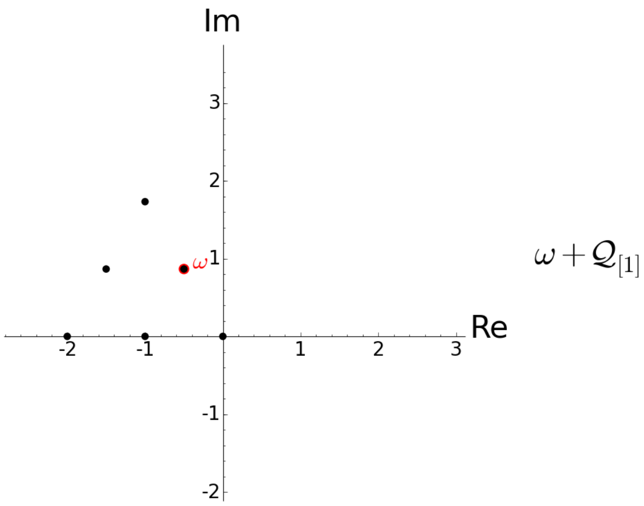


Figure 5: Construction of the set $\mathcal{Q}_{[\omega,1,2]}$ for Eisenstein numeration system

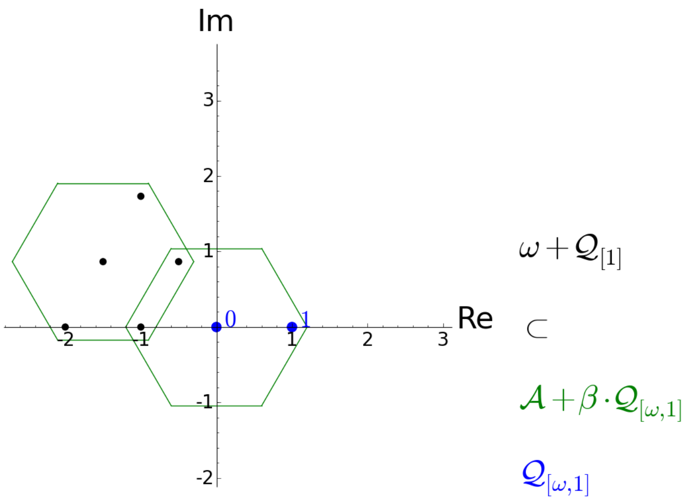


Figure 6: Construction of the set $\mathcal{Q}_{[\omega,1,2]}$ for Eisenstein numeration system

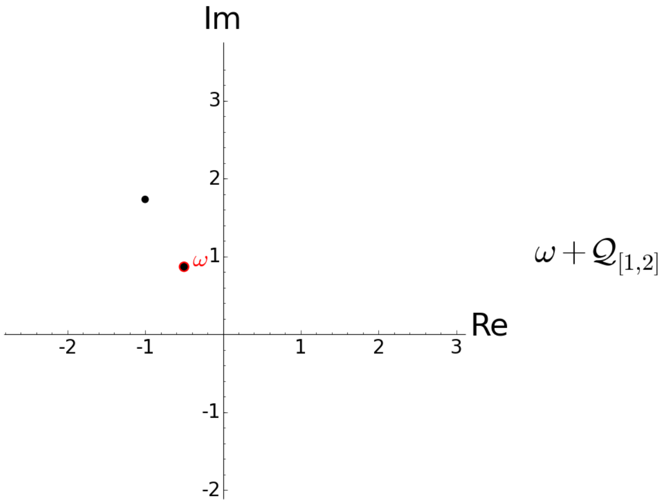


Figure 7: Construction of the set $\mathcal{Q}_{[\omega,1,2]}$ for Eisenstein numeration system

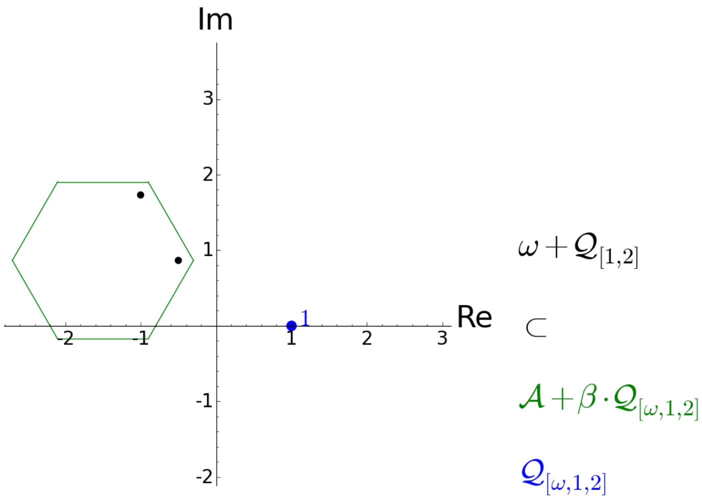


Figure 8: Construction of the set $\mathcal{Q}_{[\omega,1,2]}$ for Eisenstein numeration system

C Sample of input file for shell

File `input_sample.sage`:

```
#------------INPUTS---------------------
#Name of the numeration system:
name = 'Eisenstein_1-block_complex'
#Minimal polynomial of ring generator (use variable x):
minPol ='x^2 + x + 1'
#Embedding (the closest root of the minimal polynomial to this
   value is taken as the ring generator):
omegaCC= -0.5 + 0.8*I
#Alphabet (use 'omega' as ring generator):
alphabet = '[0, 1, -1, omega, -omega, -omega - 1, omega + 1]'
#Input alphabet (if empty, A + A is used):
inputAlphabet = ''
#Base (use 'omega' as ring generator):
base ='omega - 1'

#------------SETTING--------------------
max_iterations = 20      #maximum of iterations in searching for
    the weight coefficient set
max_input_length = 10    #maximal length of the input of the
   weight function
sanityCheck=False        #run sanity check

#------------SAVING---------------------
info=True                #save general info to .tex file
WFcsv=False              #save weight function to .csv file
localConversionCsv=False #save local conversion to .csv file
saveSetting=False        #save inputs setting as a dictionary
saveLog=True             #save log file
saveUnsolved=False       #save unsolved combinations after
   interruption

#------------IMAGES--------------------
alphabet_img=True        #save image of alphabet and input
   alphabet
lattice_img=True         #save image of lattice
phase1_images=True       #save images of steps of phase 1
weightCoefSet_img=True   #save image of the weight coefficient
   set with the estimation given by lemma:
estimation=True
phase2_images=True       #save images of steps of phase 2 for
   the input:
phase2_input='(omega,1,omega,1,omega,1,omega,1)'
```

D Interact in SageMath Cloud

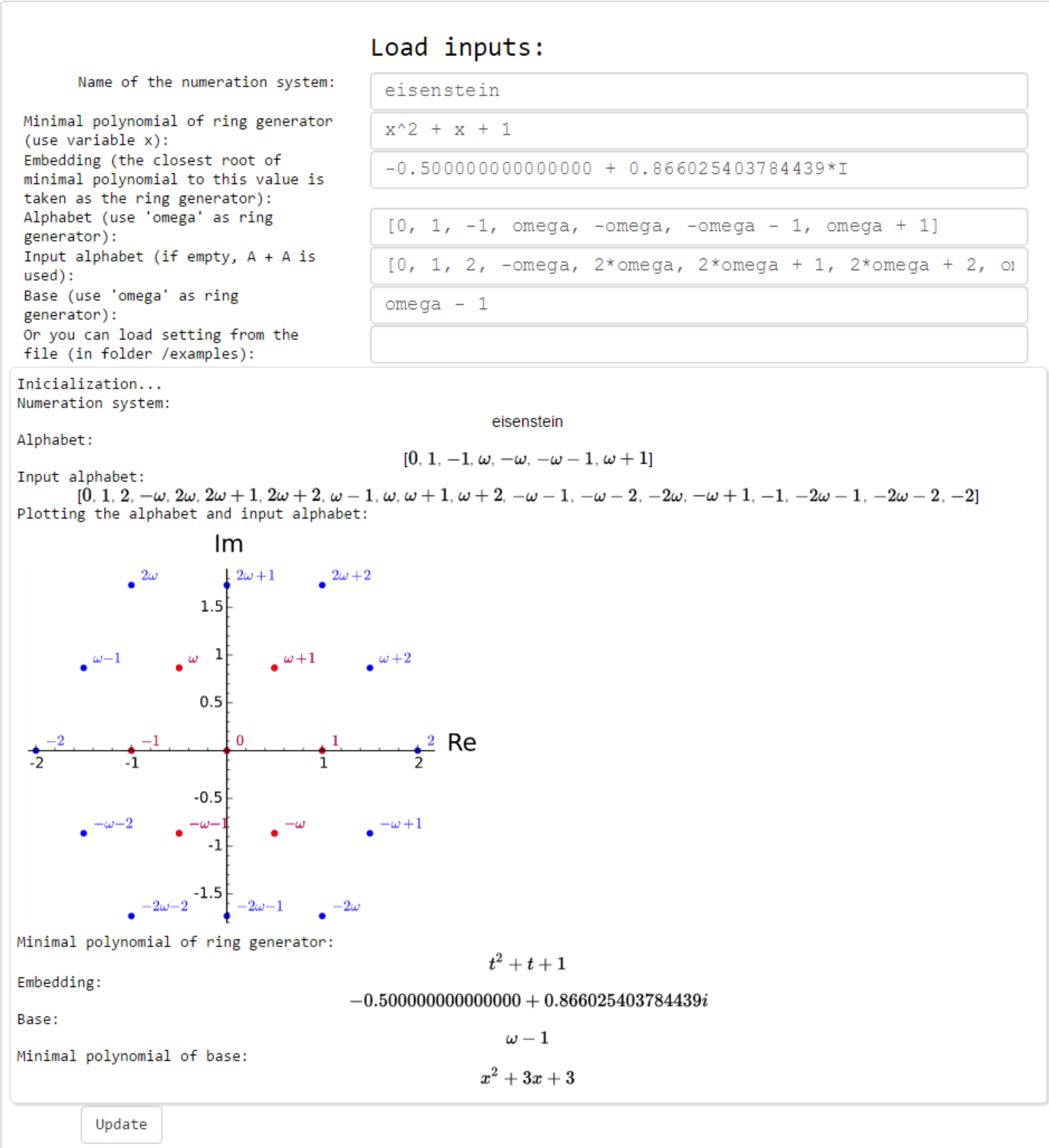


Figure 9: The interact after loading inputs.

Find weight function:

Maximum of iterations to get Weight coefficient set: 20

Maximal length of weight function input: 10

Choose required outputs to save:

- General info to .tex file: ☑
- Weight function to .csv file: ☐
- Local conversion to .csv file: ☐
- Inputs setting: ☑
- Log file: ☑
- Unsolved inputs after interruption: ☐

```
The following setting was saved to ./examples/eisenstein
{'name': 'eisenstein', 'inputAlphabet': '[0, 1, 2, -omega, 2*omega, 2*omega + 1, 2*omega + 2, omega - 1, omega, omega
+ 1, omega + 2, -omega - 1, -omega - 2, -2*omega, -omega + 1, -1, -2*omega - 1, -2*omega - 2, -2]', 'alphabet': '[0, 1
, -1, omega, -omega, -omega - 1, omega + 1]', 'base': 'omega - 1', 'minPol_alpGen': 'x^2 + x + 1', 'embedding': -0.500
000000000000 + 0.866025403784439*I}

Phase 1 - Searching for the Weight Coefficient Set using method 2...
Starting Q_0:
```

$$[0]$$

```
Number of elements in Qk: 1
Added coefficients:
```

$$[-\omega, 1, -\omega-1, -1, \omega, \omega+1]$$

```
Number of elements in Qk: 7
Added coefficients:
```

$$[-\omega-2, 2\omega+1, \omega-1, -\omega+1, -2\omega-1, \omega+2]$$

```
Number of elements in Qk: 13
Added coefficients:
```

$$[2\omega, 2, -2\omega-2, 2\omega+2, -2\omega, -2]$$

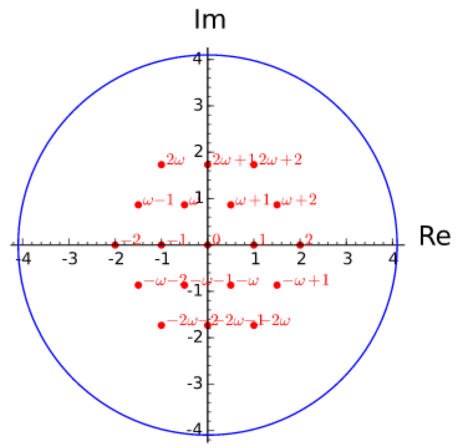
```
Number of elements in Qk: 19
Added coefficients:
```

$$[]$$

```
The Weight Coefficient Set is:
```

$$[0, 1, 2, -\omega, 2\omega, 2\omega+1, 2\omega+2, \omega-1, \omega, \omega+1, \omega+2, -\omega-1, -\omega-2, -2\omega, -\omega+1, -1, -2\omega-2, -2\omega-1, -2]$$

```
Number of elements: 19
Plotting the weight coefficients set with the estimation:
```




```
Phase 2 is starting...
Checking one letter inputs...
The longest inputs are:
```

$$[(1,1,1), (2,2,2), (-\omega,-\omega,-\omega), (2\omega,2\omega,2\omega), (2\omega+1,2\omega+1,2\omega+1), (2\omega+2,2\omega+2,2\omega+2), (\omega-1,\omega-1,\omega-1), (\omega,\omega,\omega), (\omega+1,\omega+1,\omega+1), (\omega+2,\omega+2,\omega+2), (-\omega-1,-\omega-1,-\omega-1), (-\omega-2,-\omega-2,-\omega-2), (-2\omega,-2\omega,-2\omega), (-\omega+1,-\omega+1,-\omega+1), (-1,-1,-1), (-2\omega-1,-2\omega-1,-2\omega-1), (-2\omega-2,-2\omega-2,-2\omega-2), (-2,-2,-2)]$$

```
Length of one letter input: 3:
Number of letters with longest input: 18
Searching the Weight Function using method 4...
Length of the window: 1, Number of saved combinations of input digits: 0, To next iteration: 19
Length of the window: 2, Number of saved combinations of input digits: 43, To next iteration: 318
Length of the window: 3, Number of saved combinations of input digits: 6042, To next iteration: 0
Info about Weight Function:
Maximal input length: 3
Number of inputs: 6085
Elapsed time: 51.410392
Saving...
Info about algorithm for parallel addition saved to ./outputs/eisenstein/eisenstein.tex
Log saved to ./outputs/eisenstein/eisenstein_log.txt
```

Figure 10: The output of the extending window method in the interact

Figure 11: The part of the interact for the sanity check, calling of the weight function and plotting of images of steps of both phases.