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

???

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., KM FIT, ČVUT v Praze

Konzultant: —

Abstrakt: ABSTRAKT

Klíčová slova: Paralelní sčítání, nestandardní numerační systémy.

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

Author: Jan Legerský

Abstract: ABSTRACT

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

Contents

Introduction **1**

1 Preliminaries **2**
- 1.1 Numeration systems . . . 2
- 1.2 Parallel addition . . . 3
- 1.3 Isomorphism of $\mathbb{Z}[\omega]$ and $\mathbb{Z}^d$ . . . 4

2 Design of extending window method **10**
- 2.1 Extending window method . . . 12

3 Convergence **18**

4 Design and implementation **21**
- 4.1 Class AlgorithmForParallelAddition . . . 21
- 4.2 Class WeightCoefficientsSetSearch . . . 23
- 4.3 Class WeightFunctionSearch . . . 23
- 4.4 Class WeightFunction . . . 24
- 4.5 User interfaces . . . 25

5 Testing **27**

Summary **28**

References **29**

Appendices **I**
- A Another appendix Subsection . . . I
- B Sample of input file for shell . . . II
- C Interact in SageMath Cloud . . . III

Introduction

DOPSAT PODLE OSTATNIHO

Chapter 1

Preliminaries

CHYBI UVOD, PRECHOZI VYSLEDKY

1.1 Numeration systems

Firstly, we give a general definition of numeration system.

Definition 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 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 associate a number x with its $(\beta, \mathcal{A})$-representation as a bi-infinite string

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

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. 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 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

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

A representation of $x \in \operatorname{Fin}_{\mathcal{A}}(\beta)$ is

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

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 algorithm for parallel addition requires some redundancy in numeration system.

Definition 4. 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.

Redundant numeration system can enable us to avoid carry propagation in addition. On the other hand, thera 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 5. 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 6. 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.

Now we recall known algorithms for parallel addition in different numeration systems. DODELAT Parallel Addition Introduced by Avizienis in 1961:

ROBERTSON For example:

$$\beta \in \mathbb{N}, \beta \geq 3, \mathcal{A} = \{-a, \dots, 0, \dots a\}, b/2 < a \leq b - 1\,.$$

MILENINY A EDITINY ALGORITMY -¿ VETA S VELKOU INTEGEROVOU ABECEDOU

Integer alphabets:

- Base $\beta \in \mathbb{C}, |\beta| > 1$.
- Addition is computable in parallel if and only if β is an algebraic number with no conjugate of modulus 1 [Frougny, Heller, Pelantová, S.].
- Algorithms are known but with large alphabets.

LOWER BOUND VELIKOSTI ABECEDY
ZOBECNENI ABECEDY DO Z[OMEGA]

Definition 7. Let ω be a complex number. The set of values of all polynomials with integer coefficients evaluated in ω is denoted by

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

For our purpose, the multiplicative group of a ring is associative and with an indentity. Notice that $\mathbb{Z}[\omega]$ is a commutative ring.

Non-integer alphabets:

- Base $\beta \in \mathbb{Z}[\omega] = \left\{\sum_{j=0}^{d-1} a_j \omega^j \ : a_j \in \mathbb{Z}\right\}$, where $\omega \in \mathbb{C}$ is an algebraic integer of degree d.
- Alphabet $\mathcal{A} \subset \mathbb{Z}[\omega], 0 \in \mathcal{A}$.
- Only few manually found algorithms.

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 8. 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

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

Definition 9. 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. *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 [1], 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 $\odot_\omega$ – factoring out an integer scalar, the identity element, the distributive law and a weaker form of asociativity.

Lemma 2. *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 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 3. *Let ω be an algebraic integer of degree d. Then*

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

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

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

is a ring isomorphism.

Proof. Obviously, $\mathbb{Z}[\omega] \supset \left\{ \sum_{i=0}^{d-1} a_i \omega^i : a_i \in \mathbb{Z} \right\}$. Let $p(x) = x^d + p_{d-1}x^{d-1} + \dots p_1 x + p_0$ be the minimal polynomial of ω. Assume $u \in \mathbb{Z}[\omega]$, $x = \sum_{i=0}^{n} u_i \omega^i$ for some $n \geq d$. By $p(\omega) = 0$, we have an equation $\omega^d = -p_{d-1}\omega^{d-1} - \cdots - p_1\omega - p_0$ which enables us to write

$$u = u_n \omega^n + \sum_{i=0}^{n-1} u_i \omega^i = u_n \omega^{n-d}(-p_{d-1}\omega^{d-1} - \cdots - p_1\omega - p_0) + \sum_{i=0}^{n-1} u_i \omega^i .$$

Thus $x \in \left\{ \sum_{i=0}^{d-1} a_i \omega^i : a_i \in \mathbb{Z} \right\}$ by induction with respect to n.

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 $u = \sum_{i=0}^{d-1} u_i \omega^i$ and $v = \sum_{i=0}^{d-1} v_i \omega^i$ be elements of $\mathbb{Z}[\omega]$. Consider

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

Hence

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

Suppose for induction with respect to i 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 2 for the last equality.

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

$$\pi(mv) = \pi\left(m \sum_{i=0}^{d-1} v_i \omega^i\right) = \pi\left(\sum_{i=0}^{d-1} m v_i \omega^i\right) = m\pi(v) = (me_0) \odot_\omega \pi(v) = \pi(m) \odot_\omega \pi(v) \,.$$

As π is obviously additive, we conclude:

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

□

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. We also obtained associativity and commutativity of $\odot_\omega$ as $\mathbb{Z}[\omega]$ is an associative and commutative ring.

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 4. *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

NUTNOST REDUNDANCE, PRO VSECHNY DEFINICE A VETY NENI-LI UVEDENO JINAK 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 $\mathcal{A}$ 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

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

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 $x - \beta$.

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 $\mathcal{A}$. It causes a *carry* q_j on the $(j+1)$-th position from the conversion of j-th digit. 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:

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

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

$$z_j = w_j + q_{j-1} - q_j\beta \in \mathcal{A}\,. \tag{2.2}$$

The choice of weight coefficients is the crucial part in order to construct addition algorithms which are computable in parallel. The 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 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 10. 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 (2.2). Notice that $q_{-1} = 0$ is in $\mathcal{Q}$ by definition. Thus, all weight coeficients may be chosen from $\mathcal{Q}$.

Definition 11. 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 $w_j, w_{j-1},\dots,w_{j-M} \in \mathcal{B}$. Then q is called a *weight function* and M is called a *length of window.*

JE TREBA POZADOVAT q(0,...,0)=0 NEBO TO Z NECEHO PLYNE?

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 digit set conversion ϕ is indeed $(M+1)$-local function with anticipation $r = 0$ and memory $t = M$.

The construction of a digit set conversion algorithm which is computable in parallel by so-called *extending window method* consists of two phases. In the first one, we find a minimal possible weight coefficients set $\mathcal{Q}$. It 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 $(w_j, w_{j-1},\dots,w_{j-M+1}) \in \mathcal{B}^M$.

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 $\mathcal{Q}$ iteratively so that we extend $\mathcal{Q}$ in a way to cover all elements on the left side with original $\mathcal{Q}$ by elements on the right side with extended $\mathcal{Q}$. This procedure is repeated until the extended weight coefficients set is the same as original one.

In other words, we start with $\mathcal{Q} = \{0\}$ 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 coeffcient set $\mathcal{Q}$. These weight coefficients now may appear as a carry q_{j-1}. If there are no suitable weight coefficients q_j in weight coefficients set $\mathcal{Q}$ to cover all sums of added coefficients and digits of input alphabet $\mathcal{B}$, we extend $\mathcal{Q}$ by the suitable coefficients. And so on until there is no need to add more elements. Here we mean by expression "a weight coefficient q covers an element x" that there is $a \in \mathcal{A}$ such that $x = a + \beta q$.

The precise description of the semi-algorithm in a pseudocode in Algorithm 1.

Algorithm 1 Search for weight coefficients set (Phase 1)

```
$k := 0$
$Q_0 := \{0\}$
repeat
    By Algorithm 3, extend $\mathcal{Q}_k$ to $\mathcal{Q}_{k+1}$ in a minimal possible way so that
        $$\mathcal{B} + \mathcal{Q}_k \subset \mathcal{A} + \beta\mathcal{Q}_{k+1}$$
    $k := k + 1$
until $\mathcal{Q}_k = \mathcal{Q}_{k+1}$
$\mathcal{Q} := \mathcal{Q}_k$
return $\mathcal{Q}$
```

To extend $\mathcal{Q}_k$ to $\mathcal{Q}_{k+1}$, we first find all possible candidates to cover each element of $\mathcal{B}+\mathcal{Q}_k$ (Algorithm 2). We can improve the performance of Algorithm 2 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 $\mathcal{Q}_{k+1} \supset \mathcal{Q}_k$.

An added element from each list of `candidates` is chosen as the smallest one unless there is already a covering element contained in $\mathcal{Q}_k$ (Algorithm 3).

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`.

2.1.2 Phase 2 – Weight function

We want to find a length of the window M and a weight function $q : \mathcal{B}^M \to \mathcal{Q}$. The idea of the extending window method is to reduce necessary weight coefficients from weight coefficients

Algorithm 2 Search for candidates

```
Input: previous weight coefficients set $\mathcal{Q}_k$, alternatively also the set $\mathcal{Q}_{k-1}$
candidates:= empty list of lists
for all $x \in \mathcal{B} + \mathcal{Q}_k$ do {Alternatively, $x \in (\mathcal{B} + \mathcal{Q}_k) \setminus (\mathcal{B} + \mathcal{Q}_{k-1})$}
    cand_for_x:= empty list
    for all $a \in \mathcal{A}$ do
        if $(x - a)$ is divisible by $\beta$ in $\mathbb{Z}[\omega]$ then {using Theorem 4}
            Append $\frac{x-a}{\beta}$ to cand_for_x
        end if
    end for
    Append cand_for_x to candidates
end for
return candidates
```

Algorithm 3 Extending intermediate weight coefficients set

```
Input: candidates from Algorithm 2, 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 $\mathcal{Q}_{k+1}$
```

set $\mathcal{Q}$ obtained in the previous section up to single value by enlarging number of considered input digits. The following notation is used: 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 $w_j, \dots, w_{j-M+1} \in \mathcal{B}^M$.

The precise desription of the construction of the weight function is in Algorithm 4.

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 $\max\{\#\mathcal{Q}_{[w_j, \dots, w_{j-m+1}]} : (w_j, \dots, w_{j-m+1}) \in \mathcal{B}^m\} > 1$ do
    $m := m + 1$
    for all $(w_j, \dots, w_{j-m+1}) \in \mathcal{B}^m$ do
        By Algorithm 5, find set $\mathcal{Q}_{[w_j, \dots, w_{j-m+1}]} \subset \mathcal{Q}_{[w_j, \dots, w_{j-m+2}]}$ such that
            $w_j + \mathcal{Q}_{[w_{j-1}, \dots, w_{j-m+1}]} \subset \mathcal{A} + \beta\mathcal{Q}_{[w_j, \dots, w_{j-m+1}]}$ ,
    end for
end while
$M := m$
for all $(w_j, \dots, w_{j-M+1}) \in \mathcal{B}^M$ do
    $q(w_j, \dots, w_{j-M+1})$ := only element of $\mathcal{Q}_{[w_j, \dots, w_{j-M+1}]}$
end for
return $q$
```

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

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

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

```
Input: Input digit $w_j$, set of possible carries $\mathcal{Q}_{[w_{j-1},\dots,w_{j-m+1}]}$, previous set of possible weight
    coefficients $\mathcal{Q}_{[w_j,\dots,w_{j-m+2}]}$
list_of_coverings:=empty list of lists
for all $x \in w_j + \mathcal{Q}_{[w_{j-1},\dots,w_{j-m+1}]}$ do
    Build a list x_covered_by of weight coefficients $q_j \in \mathcal{Q}_{[w_j,\dots,w_{j-m+2}]}$ 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
$\mathcal{Q}_{[w_j,\dots,w_{j-m+1}]}$:= 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 $\mathcal{Q}_{[w_j,\dots,w_{j-m+1}]}$
    Remove lists of list_of_coverings containing the element $q$ from
        list_of_coverings
end while
return $\mathcal{Q}_{[w_j,\dots,w_{j-m+1}]}$
```

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 $\mathcal{Q}_{[w_j,\dots,w_{j-m+1}]}$ for all $w_j, \dots, w_{j-m+1} \in \mathcal{B}$.

Chapter 3

Convergence

UVOD, ZAVISLOST NA HUSTYCH MNOZINACH Unfortunately, the extending window method does not always converge. The algorithm may lead to an infinite loop in both phases. However, Theorem 6 gives a sufficient condition for convergence of the Phase 1.

Lemma 5. *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 subset $\mathcal{Q} \subset \mathbb{Z}[\omega]$ 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.*

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 $a \in \mathcal{A}$ 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 $x = b + q \in \mathcal{A} + \beta\mathcal{Q}$. □

We plug in the alphabet $\mathcal{A}$ and input alphabet $\mathcal{B}$. Because of the choice of the smallest elements in Algorithm 3, we know by the lemma that the weight coefficient set $\mathcal{Q}$ constructed in Phase 1 has elements bounded by the constant R.

Theorem 6. *Let ω be an algebraic integer such that any complex circle centered at 0 contains only finitely many elements of $\mathbb{Z}[\omega]$. Let $\beta \in \mathbb{Z}[\omega], |\beta| > 1$. 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 $\mathbb{Z}[\omega]$. There exists a finite subset $\mathcal{Q} \subset \mathbb{Z}[\omega]$ such that $\mathcal{B} + \mathcal{Q} \subset \mathcal{A} + \beta\mathcal{Q}$.*

Proof. Directly from Lemma 5 and assumption on $\mathbb{Z}[\omega]$. □

Therefore, Phase 1 succesfully ends if there can be only finitely many elements in $\mathbb{Z}[\omega]$ NEBO Zbeta??? bounded by constant R as intermediate weight coefficient sets $\mathcal{Q}_k$ have elements bounded by R for all k. We categorize an algebraic integer ω which generates $\mathbb{Z}[\omega] \ni \beta$ as follows:

- $\omega \in \mathbb{Z}$ implies $\mathbb{Z}[\omega] = \mathbb{Z}$ which has the required property and thus Phase 1 converges.

- $\omega \in \mathbb{R} \setminus \mathbb{Z}$ implies $\mathbb{Z}[\omega]$ being dense in $\mathbb{R}$. CITACE NEBO DUKAZ??? Thus the convergence of Phase 1 is not guaranteed and we have an example for which it does not converge.
- $\omega \in \mathbb{C} \setminus \mathbb{R}$, ω being quadratic algebraic integer implies that $\mathbb{Z}[\omega]$ is not dense in $\mathbb{C}$ and thus Phase 1 can converge. CO NEJAKY DUKAZ NEBO LEPSI ZDUVODNENI?????? NOT DENSE PORAD JESTE NEZNAMENA, ZE JICH NEMUZE BYT NEKONECNE...
- $\omega \in \mathbb{C} \setminus \mathbb{R}$, ω being algebraic integer of degree ≥ 3 implies $\mathbb{Z}[\omega]$ is dense in $\mathbb{C}$ and therefore the convergence of Phase 1 is not guaranteed.

We focus on Phase 2 now. For shorter notation, set

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

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

Obviously, finitness 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 $b \in \mathcal{B}$.

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 $\mathcal{Q}_{[b^m]}$.
```

For arbitrary m, sets $\mathcal{Q}_{[b^m]}$ can be easily constructed separately for each $b \in \mathcal{B}$. Using Lemma 7, Algorithm 6 checks whether Phase 2 stops processing input digits $bb \dots b$. Thus, non-finitness of Phase 2 can be revealed by running it for each input digit $b \in \mathcal{B}$.

Lemma 7. *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*

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

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

Proof. It is enough to prove the base case of induction as the inductive step is analogous. 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

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

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. □

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 4, 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 2.2 and using weight function q.

localConversion(*w***)**

Return converted digit according to equation 2.2 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 2, 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 3.

_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 lenght 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 Lemma 7, 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 B 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 Lemma 5 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 EVALUATED interact are on Figure 1 and 2 in Appendix C. 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

USPESNE A NEUSPESNE PRIKLADY - otestovat ty z Mileniných poznámek? Pridat nejake kubické?

5.0.3 Eisenstein base $\beta = -1 + \omega$ with $\omega = \exp \frac{2\pi\imath}{3} = -\frac{1}{2} + \frac{\imath\sqrt{3}}{2}$

KAZDEMU NEJAKY TAKOVYTO VYSTUP:

Numeration System: eisenstein
Minimal polynomial of ω: $t^2 + t + 1$
Base $\beta = \omega - 1$
Minimal polynomial of base: $x^2 + 3x + 3$
Alphabet $\mathcal{A} = \{0, 1, -1, \omega, -\omega, -\omega - 1, \omega + 1\}$
Input alphabet $\mathcal{B} = \mathcal{A} + \mathcal{A}$
Phase 1 was succesfull.
Weight Coefficient Set:

$$\mathcal{Q} = \{0, 1, 2, -\omega, 2\omega, 2\omega + 1, 2\omega + 2, \omega - 1, \omega, \omega + 1, \omega + 2, -\omega - 1, -\omega - 2, -2\omega, -\omega + 1, -1, -2\omega - 2, -2\omega -$$

Number of elements in the weight coefficient set $\mathcal{Q}$ is: 19
Weight function Info:
Phase 2 was succesfull.
Maximal input length: 3
Number of inputs: 6085

5.0.4 Penney base $\beta = -1 + \imath$ with $\imath = \exp \frac{2\pi\imath}{4}$

5.0.5 Base $\beta = 1 + \imath$

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

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

Summary

DOPSAT PODLE OSTATNIHO

Bibliography

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

Appendices

A Another appendix Subsection

B Sample of input file for shell

File `input_sample.sage`:

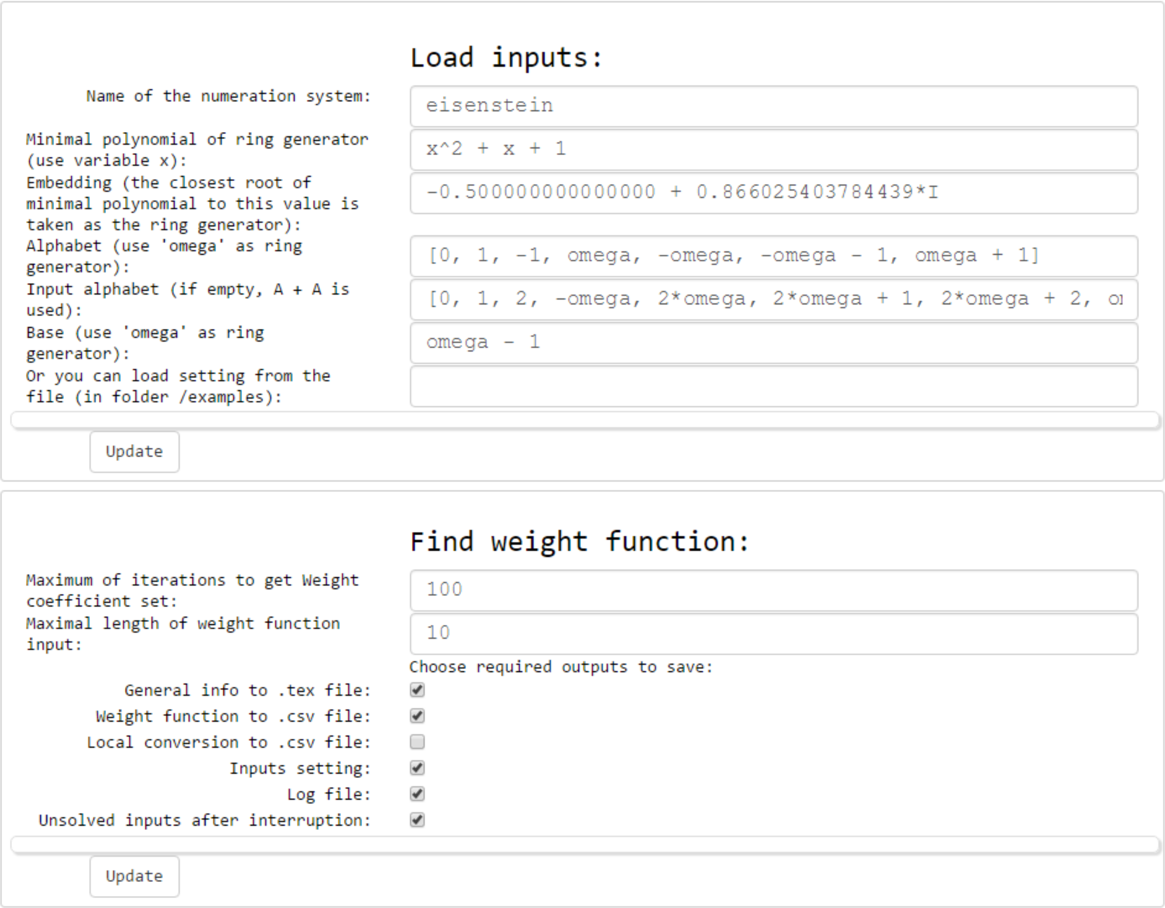
```
#------------INPUTS---------------------
#Name of the numeration system:
name = ’Eisenstein_1-block_complex’
#Minimal polynomial of ring generator (use variable x):
minPol =’x^2 + x + 1’
#Embedding (the closest root of the minimal polynomial to this
   value is taken as the ring generator):
omegaCC= -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)’
```

C Interact in SageMath Cloud



Load inputs:

Name of the numeration system: eisenstein

Minimal polynomial of ring generator (use variable x): x^2 + x + 1

Embedding (the closest root of minimal polynomial to this value is taken as the ring generator): -0.500000000000000 + 0.866025403784439*I

Alphabet (use 'omega' as ring generator): [0, 1, -1, omega, -omega, -omega - 1, omega + 1]

Input alphabet (if empty, A + A is used): [0, 1, 2, -omega, 2*omega, 2*omega + 1, 2*omega + 2, o

Base (use 'omega' as ring generator): omega - 1

Or you can load setting from the file (in folder /examples):

Find weight function:

Maximum of iterations to get Weight coefficient set: 100

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: □

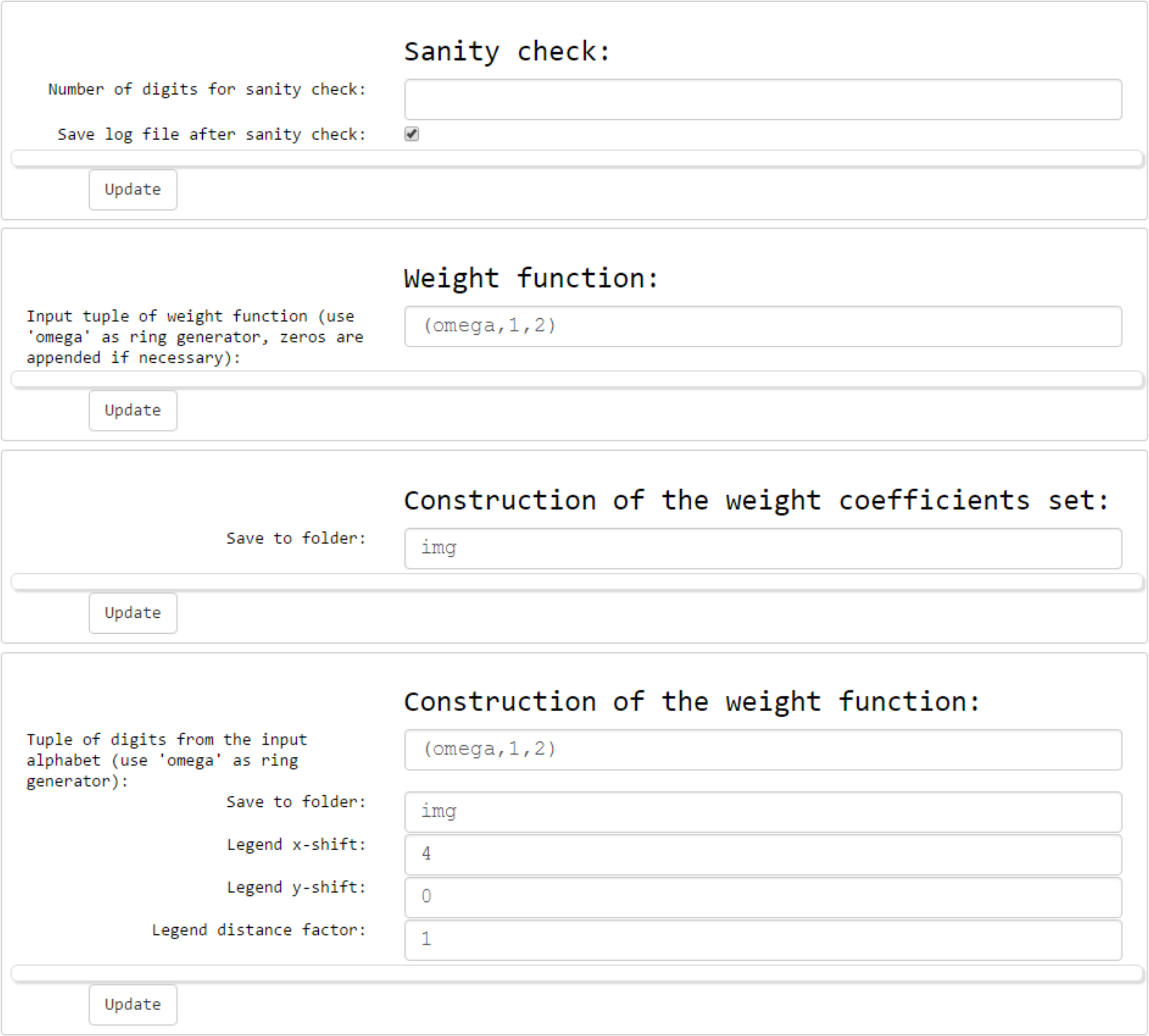
Inputs setting: ☑

Log file: ☑

Unsolved inputs after interruption: ☑

Figure 1: POPISEK INTERACTU

OBRAZKY KROKU ODOU FAZI, OBRAZKY INTERACTU



Sanity check:

Number of digits for sanity check:

Save log file after sanity check: ☑

Weight function:

Input tuple of weight function (use 'omega' as ring generator, zeros are appended if necessary): (omega,1,2)

Construction of the weight coefficients set:

Save to folder: img

Construction of the weight function:

Tuple of digits from the input alphabet (use 'omega' as ring generator): (omega,1,2)

Save to folder: img

Legend x-shift: 4

Legend y-shift: 0

Legend distance factor: 1

Figure 2: POPISEK INTERACTU