Parallel Addition in Non-standard Numeration Systems

Milena Svobodová, Edita Pelantová

TIGR KM FJFI ČVUT

November 9, 2010

Introduction

Introduction

- We work with positional numeration systems, given by
  - base β, $\beta \in \mathbb{C}$, $|\beta| > 1$,
  - finite set of integer digits called alphabet $\mathcal{A} \subset \mathbb{Z}$, and
  - we limit ourselves to base β being an algebraic number (but NOT necessarily algebraic integer!); namely β is a root of an equation

    $$b_d\beta^d + b_{d-1}\beta^{d-1} + \ldots + b_1\beta^1 + b_0\beta^0 = 0$$

    with $d \in \mathbb{N}$, and integer coefficients $b_d, b_{d-1}, \ldots, b_1, b_0 \in \mathbb{Z}$ (wherein $b_d \neq 0$ does NOT have to be equal to 1).

Introduction

- We work with positional numeration systems, given by
  - base β, $\beta \in \mathbb{C}$, $|\beta| > 1$,
  - finite set of integer digits called alphabet $\mathcal{A} \subset \mathbb{Z}$, and
  - we limit ourselves to base β being an algebraic number (but NOT necessarily algebraic integer!); namely β is a root of an equation

$$b_d\beta^d + b_{d-1}\beta^{d-1} + \ldots + b_1\beta^1 + b_0\beta^0 = 0$$

    with $d \in \mathbb{N}$, and integer coefficients $b_d, b_{d-1}, \ldots, b_1, b_0 \in \mathbb{Z}$ (wherein $b_d \neq 0$ does NOT have to be equal to 1).
- In such numeration system, we work with so-called β-representations of real or complex numbers in the form
  - $x = (x_n x_{n-1} \ldots x_{m+1} x_m) = (x_j)_{j=m}^{n}$ for $x \in \mathbb{C}$ or $x \in \mathbb{R}$,
  - meaning that $x = \sum_{j=m}^{n} x_j \beta^j$, with $n \in \mathbb{Z}$ and $m \leq n$,
  - where $m \in \mathbb{Z}$, or $m = -\infty$

Introduction

- We work with positional numeration systems, given by
  - base β, $\beta \in \mathbb{C}$, $|\beta| > 1$,
  - finite set of integer digits called alphabet $\mathcal{A} \subset \mathbb{Z}$, and
  - we limit ourselves to base β being an algebraic number (but NOT necessarily algebraic integer!); namely β is a root of an equation

$$b_d\beta^d + b_{d-1}\beta^{d-1} + \ldots + b_1\beta^1 + b_0\beta^0 = 0$$

  with $d \in \mathbb{N}$, and integer coefficients $b_d, b_{d-1}, \ldots, b_1, b_0 \in \mathbb{Z}$ (wherein $b_d \neq 0$ does NOT have to be equal to 1).
- In such numeration system, we work with so-called β-representations of real or complex numbers in the form
  - $x = (x_n x_{n-1} \ldots x_{m+1} x_m) = (x_j)_{j=m}^{n}$ for $x \in \mathbb{C}$ or $x \in \mathbb{R}$,
  - meaning that $x = \sum_{j=m}^{n} x_j \beta^j$, with $n \in \mathbb{Z}$ and $m \leq n$,
  - where $m \in \mathbb{Z}$, or $m = -\infty$
- Generally, these numeration systems can be redundant (i.e. allowing more than one β-representation for the same number x), or non-redundant (i.e. the opposite).

Introduction

Introduction

- Our aim is to perform addition of two numbers in this numeration system 'in parallel'; which, in terminology of theoretical informatics, means that addition would be a local function.

Introduction

- Our aim is to perform addition of two numbers in this numeration system 'in parallel'; which, in terminology of theoretical informatics, means that addition would be a local function.

Definition

Let $\mathcal{A}, \mathcal{B}$ be two alphabets, let $\mathcal{A}^{\mathbb{Z}}, \mathcal{B}^{\mathbb{Z}}$ be the sets of all bi-infinite words on these two alphabets. Function $\varphi : \mathcal{A}^{\mathbb{Z}} \to \mathcal{B}^{\mathbb{Z}}$ is said to be local with memory r and anticipation t if there exist non-negative integers r, t and a function $\phi : \mathcal{A}^p \to \mathcal{B}$ with $p = r + t + 1$, such that if $u = (u_j)_{j \in \mathbb{Z}} \in \mathcal{A}^{\mathbb{Z}}$ and $v = (v_j)_{j \in \mathbb{Z}} \in \mathcal{B}^{\mathbb{Z}}$, then $v = \varphi(u)$ if and only if for every $j \in \mathbb{Z}$ there is $v_j = \phi(u_{j+t} \dots u_j \dots u_{j-r})$.

Introduction

- Our aim is to perform addition of two numbers in this numeration system 'in parallel'; which, in terminology of theoretical informatics, means that addition would be a local function.

Definition

Let $\mathcal{A}, \mathcal{B}$ be two alphabets, let $\mathcal{A}^{\mathbb{Z}}, \mathcal{B}^{\mathbb{Z}}$ be the sets of all bi-infinite words on these two alphabets. Function $\varphi : \mathcal{A}^{\mathbb{Z}} \to \mathcal{B}^{\mathbb{Z}}$ is said to be local with memory r and anticipation t if there exist non-negative integers r, t and a function $\phi : \mathcal{A}^p \to \mathcal{B}$ with $p = r + t + 1$, such that if $u = (u_j)_{j\in\mathbb{Z}} \in \mathcal{A}^{\mathbb{Z}}$ and $v = (v_j)_{j\in\mathbb{Z}} \in \mathcal{B}^{\mathbb{Z}}$, then $v = \varphi(u)$ if and only if for every $j \in \mathbb{Z}$ there is $v_j = \phi(u_{j+t} \dots u_j \dots u_{j-r})$.
We then say that φ is (r, t)-local or p-local, and that the image of u by φ is obtained through a 'sliding window' of length p.

Introduction

- Our aim is to perform addition of two numbers in this numeration system 'in parallel'; which, in terminology of theoretical informatics, means that addition would be a local function.

Definition

Let $\mathcal{A}, \mathcal{B}$ be two alphabets, let $\mathcal{A}^{\mathbb{Z}}, \mathcal{B}^{\mathbb{Z}}$ be the sets of all bi-infinite words on these two alphabets. Function $\varphi : \mathcal{A}^{\mathbb{Z}} \to \mathcal{B}^{\mathbb{Z}}$ is said to be local with memory r and anticipation t if there exist non-negative integers r, t and a function $\phi : \mathcal{A}^p \to \mathcal{B}$ with $p = r + t + 1$, such that if $u = (u_j)_{j \in \mathbb{Z}} \in \mathcal{A}^{\mathbb{Z}}$ and $v = (v_j)_{j \in \mathbb{Z}} \in \mathcal{B}^{\mathbb{Z}}$, then $v = \varphi(u)$ if and only if for every $j \in \mathbb{Z}$ there is $v_j = \phi(u_{j+t} \dots u_j \dots u_{j-r})$.
We then say that φ is (r, t)-local or p-local, and that the image of u by φ is obtained through a 'sliding window' of length p.

- Parallel addition is not possible on non-redundant numeration systems; therefore, from now onwards, we are going to work with redundant numeration systems.

Introduction

- In our case, in order to be able to do parallel addition in a numeration system with base β:

Introduction

- In our case, in order to be able to do parallel addition in a numeration system with base β:
  - we need to find a convenient alphabet $\mathcal{A}$ and

Introduction

- In our case, in order to be able to do parallel addition in a numeration system with base β:
  - we need to find a convenient alphabet $\mathcal{A}$ and
  - find a convenient function $\phi : (\mathcal{A} + \mathcal{A})^p \to \mathcal{A}$, with suitable non-negative integers r, t, $p = r + t + 1$, allowing to

Introduction

- In our case, in order to be able to do parallel addition in a numeration system with base β:
  - we need to find a convenient alphabet $\mathcal{A}$ and
  - find a convenient function $\phi : (\mathcal{A}+\mathcal{A})^p \to \mathcal{A}$, with suitable non-negative integers r, t, $p = r + t + 1$, allowing to
  - express addition of two numbers in this numeration system in the form of an (r, t)-local function $\varphi : (\mathcal{A}+\mathcal{A})^{\mathbb{Z}} \to \mathcal{A}^{\mathbb{Z}}$, wherein

Introduction

- In our case, in order to be able to do parallel addition in a numeration system with base β:
  - we need to find a convenient alphabet $\mathcal{A}$ and
  - find a convenient function $\phi : (\mathcal{A}+\mathcal{A})^p \to \mathcal{A}$, with suitable non-negative integers r, t, $p = r + t + 1$, allowing to
  - express addition of two numbers in this numeration system in the form of an (r,t)-local function $\varphi : (\mathcal{A}+\mathcal{A})^{\mathbb{Z}} \to \mathcal{A}^{\mathbb{Z}}$, wherein
  - starting from $x, y \in \mathcal{A}^{\mathbb{Z}}$, $x = \sum_j x_j\beta^j$, $y = \sum_j y_j\beta^j$, $x_j, y_j \in \mathcal{A}$,

Introduction

- In our case, in order to be able to do parallel addition in a numeration system with base β:
  - we need to find a convenient alphabet $\mathcal{A}$ and
  - find a convenient function $\phi : (\mathcal{A}+\mathcal{A})^p \to \mathcal{A}$, with suitable non-negative integers r, t, $p = r + t + 1$, allowing to
  - express addition of two numbers in this numeration system in the form of an (r, t)-local function $\varphi : (\mathcal{A}+\mathcal{A})^{\mathbb{Z}} \to \mathcal{A}^{\mathbb{Z}}$, wherein
  - starting from $x, y \in \mathcal{A}^{\mathbb{Z}}$, $x = \sum_j x_j \beta^j$, $y = \sum_j y_j \beta^j$, $x_j, y_j \in \mathcal{A}$,
  - we continue via an interim summation $w = \sum_j w_j \beta^j = \sum_j (x_j + y_j)\beta^j$, $w_j \in (\mathcal{A}+\mathcal{A})$,

Introduction

- In our case, in order to be able to do parallel addition in a numeration system with base β:
  - we need to find a convenient alphabet $\mathcal{A}$ and
  - find a convenient function $\phi : (\mathcal{A}+\mathcal{A})^p \to \mathcal{A}$, with suitable non-negative integers r, t, $p = r + t + 1$, allowing to
  - express addition of two numbers in this numeration system in the form of an (r,t)-local function $\varphi : (\mathcal{A}+\mathcal{A})^{\mathbb{Z}} \to \mathcal{A}^{\mathbb{Z}}$, wherein
  - starting from $x, y \in \mathcal{A}^{\mathbb{Z}}$, $x = \sum_j x_j \beta^j$, $y = \sum_j y_j \beta^j$, $x_j, y_j \in \mathcal{A}$,
  - we continue via an interim summation $w = \sum_j w_j \beta^j = \sum_j (x_j + y_j)\beta^j$, $w_j \in (\mathcal{A}+\mathcal{A})$,
  - upon which we apply the local function $\varphi : (\mathcal{A}+\mathcal{A})^{\mathbb{Z}} \to \mathcal{A}^{\mathbb{Z}}$ as follows: $z = \varphi(w) = \sum_j z_j \beta^j$, where $z_j = \phi(w_{j+t}, \ldots, w_j, \ldots, w_{j-r}) \in \mathcal{A}$ for any $j \in \mathbb{Z}$

Introduction

- In our case, in order to be able to do parallel addition in a numeration system with base β:
  - we need to find a convenient alphabet $\mathcal{A}$ and
  - find a convenient function $\phi : (\mathcal{A}+\mathcal{A})^p \to \mathcal{A}$, with suitable non-negative integers r, t, $p = r + t + 1$, allowing to
  - express addition of two numbers in this numeration system in the form of an (r,t)-local function $\varphi : (\mathcal{A}+\mathcal{A})^{\mathbb{Z}} \to \mathcal{A}^{\mathbb{Z}}$, wherein
  - starting from $x, y \in \mathcal{A}^{\mathbb{Z}}$, $x = \sum_j x_j \beta^j$, $y = \sum_j y_j \beta^j$, $x_j, y_j \in \mathcal{A}$,
  - we continue via an interim summation $w = \sum_j w_j \beta^j = \sum_j (x_j + y_j)\beta^j$, $w_j \in (\mathcal{A}+\mathcal{A})$,
  - upon which we apply the local function $\varphi : (\mathcal{A}+\mathcal{A})^{\mathbb{Z}} \to \mathcal{A}^{\mathbb{Z}}$ as follows: $z = \varphi(w) = \sum_j z_j \beta^j$, where $z_j = \phi(w_{j+t}, \ldots, w_j, \ldots, w_{j-r}) \in \mathcal{A}$ for any $j \in \mathbb{Z}$
- Note: In all the algorithms described further, our alphabet has the form $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$ - i.e. is symmetric with respect to zero; therefore, having addition as a local function, we have at the same time also deduction as a local function.

Previously Known Results

Previously Known Results

A.Avizienis (1961)

Previously Known Results

A.Avizienis (1961)

- Algorithm for parallel addition in numeration system with positive integer base $\beta = b \in \mathbb{N}$, $b \geq 3$, and alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, where $\frac{b}{2} < a \leq b - 1$

Previously Known Results

A.Avizienis (1961)

- Algorithm for parallel addition in numeration system with positive integer base $\beta = b \in \mathbb{N}$, $b \geq 3$, and alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, where $\frac{b}{2} < a \leq b - 1$
- The algorithm is a 2-local function with memory 1 and anticipation 0, i.e. (1,0)-local function

Previously Known Results

A.Avizienis (1961)

- Algorithm for parallel addition in numeration system with positive integer base $\beta = b \in \mathbb{N}$, $b \geq 3$, and alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, where $\frac{b}{2} < a \leq b - 1$
- The algorithm is a 2-local function with memory 1 and anticipation 0, i.e. (1,0)-local function
- The minimal choice of a here is $a = \lceil \frac{b+1}{2} \rceil$, and so the smallest alphabet we can obtain is $\mathcal{A} = \{-\lceil \frac{b+1}{2} \rceil, \ldots, 0, \ldots, +\lceil \frac{b+1}{2} \rceil\}$

Previously Known Results

A.Avizienis (1961)

- Algorithm for parallel addition in numeration system with positive integer base $\beta = b \in \mathbb{N}$, $b \geq 3$, and alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, where $\frac{b}{2} < a \leq b - 1$
- The algorithm is a 2-local function with memory 1 and anticipation 0, i.e. (1,0)-local function
- The minimal choice of a here is $a = \lceil \frac{b+1}{2} \rceil$, and so the smallest alphabet we can obtain is $\mathcal{A} = \{-\lceil \frac{b+1}{2} \rceil, \ldots, 0, \ldots, +\lceil \frac{b+1}{2} \rceil\}$

C.Y.Chow, J.E.Robertson (1978)

Previously Known Results

A.Avizienis (1961)

- Algorithm for parallel addition in numeration system with positive integer base $\beta = b \in \mathbb{N}$, $b \geq 3$, and alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, where $\frac{b}{2} < a \leq b - 1$
- The algorithm is a 2-local function with memory 1 and anticipation 0, i.e. (1,0)-local function
- The minimal choice of a here is $a = \lceil \frac{b+1}{2} \rceil$, and so the smallest alphabet we can obtain is $\mathcal{A} = \{-\lceil \frac{b+1}{2} \rceil, \ldots, 0, \ldots, +\lceil \frac{b+1}{2} \rceil\}$

C.Y.Chow, J.E.Robertson (1978)

- Algorithm for parallel addition in numeration system with even base $\beta = b \in \mathbb{N}$, $b \geq 2$, and alphabet $\mathcal{A} = \{-\frac{b}{2}, \ldots, 0, \ldots, +\frac{b}{2}\}$

Previously Known Results

A.Avizienis (1961)

- Algorithm for parallel addition in numeration system with positive integer base $\beta = b \in \mathbb{N}$, $b \geq 3$, and alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, where $\frac{b}{2} < a \leq b - 1$
- The algorithm is a 2-local function with memory 1 and anticipation 0, i.e. (1,0)-local function
- The minimal choice of a here is $a = \lceil \frac{b+1}{2} \rceil$, and so the smallest alphabet we can obtain is $\mathcal{A} = \{-\lceil \frac{b+1}{2} \rceil, \ldots, 0, \ldots, +\lceil \frac{b+1}{2} \rceil\}$

C.Y.Chow, J.E.Robertson (1978)

- Algorithm for parallel addition in numeration system with even base $\beta = b \in \mathbb{N}$, $b \geq 2$, and alphabet $\mathcal{A} = \{-\frac{b}{2}, \ldots, 0, \ldots, +\frac{b}{2}\}$
- The algorithm is a 3-local function with memory 2 and anticipation 0, i.e. (2,0)-local function

Previously Known Results

Previously Known Results

Comparing the algorithms of Avizienis versus Chow & Robertson:

Previously Known Results

Comparing the algorithms of Avizienis versus Chow & Robertson:

- The 'sliding window' with Chow & Robertson ($p = 3$) is longer than with Avizienis ($p = 2$); on the other hand,

Previously Known Results

Comparing the algorithms of Avizienis versus Chow & Robertson:

- The 'sliding window' with Chow & Robertson ($p = 3$) is longer than with Avizienis ($p = 2$); on the other hand,
- For even bases b, Chow & Robertson works on smaller alphabet ($a = \frac{b}{2}$) than Avizienis ($a = \frac{b}{2} + 1$); and, on top of that,

Previously Known Results

Comparing the algorithms of Avizienis versus Chow & Robertson:

- The 'sliding window' with Chow & Robertson ($p = 3$) is longer than with Avizienis ($p = 2$); on the other hand,
- For even bases b, Chow & Robertson works on smaller alphabet ($a = \frac{b}{2}$) than Avizienis ($a = \frac{b}{2} + 1$); and, on top of that,
- Chow & Robertson works also for $b = 2$, while Avizienis does not

Previously Known Results

Comparing the algorithms of Avizienis versus Chow & Robertson:

- The 'sliding window' with Chow & Robertson ($p = 3$) is longer than with Avizienis ($p = 2$); on the other hand,
- For even bases b, Chow & Robertson works on smaller alphabet ($a = \frac{b}{2}$) than Avizienis ($a = \frac{b}{2} + 1$); and, on top of that,
- Chow & Robertson works also for $b = 2$, while Avizienis does not

Overview of working of Chow & Robertson versus Avizienis algorithms on first few integer bases:

base $b \in \mathbb{N}$	3-local algorithm of Chow & Robertson	2-local algorithm of Avizienis
$b = 2$	$\mathcal{A} = \{-1, 0, +1\}$	not working
$b = 3$	not working	$\mathcal{A} = \{-2, \ldots, +2\}$
$b = 4$	$\mathcal{A} = \{-2, \ldots, +2\}$	$\mathcal{A} = \{-3, \ldots, +3\}$
$b = 5$	not working	$\mathcal{A} = \{-3, \ldots, +3\}$
$b = 6$	$\mathcal{A} = \{-3, \ldots, +3\}$	$\mathcal{A} = \{-4, \ldots, +4\}$

What New Results Do We Bring ?

What New Results Do We Bring ?

The algorithms for parallel addition given by Chow & Robertson or Avizienis only act on numeration systems with positive integer base $\beta = b \in \mathbb{N}$; and still with further restrictions (Avizienis only for $b \geq 3$, Chow & Robertson only for b even).

What New Results Do We Bring ?

The algorithms for parallel addition given by Chow & Robertson or Avizienis only act on numeration systems with positive integer base $\beta = b \in \mathbb{N}$; and still with further restrictions (Avizienis only for $b \geq 3$, Chow & Robertson only for b even).

<u>Our new results</u>:

What New Results Do We Bring ?

The algorithms for parallel addition given by Chow & Robertson or Avizienis only act on numeration systems with positive integer base $\beta = b \in \mathbb{N}$; and still with further restrictions (Avizienis only for $b \geq 3$, Chow & Robertson only for b even).

Our new results:

- We describe two new algorithms for parallel addition in numeration systems with base $\beta \in \mathbb{C}$, $|\beta| > 1$ being an algebraic number, and with integer alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$.

What New Results Do We Bring ?

The algorithms for parallel addition given by Chow & Robertson or Avizienis only act on numeration systems with positive integer base $\beta = b \in \mathbb{N}$; and still with further restrictions (Avizienis only for $b \geq 3$, Chow & Robertson only for b even).

Our new results:

- We describe two new algorithms for parallel addition in numeration systems with base $\beta \in \mathbb{C}$, $|\beta| > 1$ being an algebraic number, and with integer alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$.
- Note that the base β does NOT need to be algebraic integer, but just an algebraic number.

What New Results Do We Bring ?

The algorithms for parallel addition given by Chow & Robertson or Avizienis only act on numeration systems with positive integer base $\beta = b \in \mathbb{N}$; and still with further restrictions (Avizienis only for $b \geq 3$, Chow & Robertson only for b even).

Our new results:

- We describe two new algorithms for parallel addition in numeration systems with base $\beta \in \mathbb{C}$, $|\beta| > 1$ being an algebraic number, and with integer alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$.
- Note that the base β does NOT need to be algebraic integer, but just an algebraic number.
- Both the algorithms do NOT work for all algebraic numbers β, but each of them does work for quite a large class of algebraic numbers β.

What New Results Do We Bring ?

The algorithms for parallel addition given by Chow & Robertson or Avizienis only act on numeration systems with positive integer base $\beta = b \in \mathbb{N}$; and still with further restrictions (Avizienis only for $b \geq 3$, Chow & Robertson only for b even).

<u>Our new results</u>:

- We describe two new algorithms for parallel addition in numeration systems with base $\beta \in \mathbb{C}$, $|\beta| > 1$ being an algebraic number, and with integer alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$.
- Note that the base β does NOT need to be algebraic integer, but just an algebraic number.
- Both the algorithms do NOT work for all algebraic numbers β, but each of them does work for quite a large class of algebraic numbers β.
- The applicability / non-applicability of each of the two algorithms for a particular base β depends on specific properties of β, as described further.

Algorithm I: Base β with a 'Strong Rewriting Rule'

Algorithm I: Base β with a 'Strong Rewriting Rule'

Definition

An algebraic number $\beta \in \mathbb{C}$, $|\beta| > 1$ is said to have a 'strong rewriting rule' if there exist non-negative integers $h, k \in \mathbb{N}$, and a set of integers $b_k, b_{k-1}, \ldots, b_1, b_0, b_{-1}, \ldots, b_{-h} \in \mathbb{Z}$ such that

$$b_k\beta^k + b_{k-1}\beta^{k-1} + \ldots + b_1\beta^1 + b_0 + b_{-1}\beta^{-1} + \ldots + b_{-h}\beta^{-h} = 0$$

and $b_0 > 2\sum_{j=-h,j\neq 0}^{k} |b_j|$.

Algorithm I: Base β with a 'Strong Rewriting Rule'

Definition

An algebraic number $\beta \in \mathbb{C}$, $|\beta| > 1$ is said to have a 'strong rewriting rule' if there exist non-negative integers $h, k \in \mathbb{N}$, and a set of integers $b_k, b_{k-1}, \ldots, b_1, b_0, b_{-1}, \ldots, b_{-h} \in \mathbb{Z}$ such that

$$b_k\beta^k + b_{k-1}\beta^{k-1} + \ldots + b_1\beta^1 + b_0 + b_{-1}\beta^{-1} + \ldots + b_{-h}\beta^{-h} = 0$$

and $b_0 > 2\sum_{j=-h,j\neq 0}^{k} |b_j|$.

Theorem

Let $\beta \in \mathbb{C}$, $|\beta| > 1$ have a 'strong rewriting rule', wherein B, M denote $B = b_0$ and $M = \sum_{j=-h,j\neq 0}^{k} |b_j|$, and let $p = k + h + 1$.

Algorithm I: Base β with a 'Strong Rewriting Rule'

Definition

An algebraic number $\beta \in \mathbb{C}$, $|\beta| > 1$ is said to have a 'strong rewriting rule' if there exist non-negative integers $h, k \in \mathbb{N}$, and a set of integers $b_k, b_{k-1}, \ldots, b_1, b_0, b_{-1}, \ldots, b_{-h} \in \mathbb{Z}$ such that

$$b_k\beta^k + b_{k-1}\beta^{k-1} + \ldots + b_1\beta^1 + b_0 + b_{-1}\beta^{-1} + \ldots + b_{-h}\beta^{-h} = 0$$

and $b_0 > 2\sum_{j=-h,j\neq 0}^{k} |b_j|$.

Theorem

Let $\beta \in \mathbb{C}$, $|\beta| > 1$ have a 'strong rewriting rule', wherein B, M denote $B = b_0$ and $M = \sum_{j=-h,j\neq 0}^{k} |b_j|$, and let $p = k + h + 1$.

Then addition in numeration system with base β can be realized as a p-local function with memory k and anticipation h, in alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, where $a = \lceil \frac{B-1}{2} \rceil + \lceil \frac{B-1}{2(B-2M)} \rceil M$.

Algorithm I: Base β with a 'Strong Rewriting Rule'

This (k,h)-local function of addition is described in Algorithm I,

Algorithm I: Base β with a 'Strong Rewriting Rule'

This (k, h)-local function of addition is described in Algorithm I,

with parameters:

Algorithm I: Base β with a 'Strong Rewriting Rule'

This (k, h)-local function of addition is described in Algorithm I,

with parameters:

- Notation: $a = a' + cM$, where $a' = \lceil \frac{B-1}{2} \rceil$, $c = \lceil \frac{B-1}{2(B-2M)} \rceil$; alphabet $\mathcal{A} = \{-a, \dots, 0, \dots, +a\}$, inner alphabet $\mathcal{A}' = \{-a', \dots, 0, \dots, +a'\}$; integers $m \leq n$

Algorithm I: Base β with a 'Strong Rewriting Rule'

This (k, h)-local function of addition is described in Algorithm I,

with parameters:

- Notation: $a = a' + cM$, where $a' = \lceil \frac{B-1}{2} \rceil$, $c = \lceil \frac{B-1}{2(B-2M)} \rceil$; alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, inner alphabet $\mathcal{A}' = \{-a', \ldots, 0, \ldots, +a'\}$; integers $m \leq n$
- Input: $x = \sum_{j=m}^{n} x_j \beta^j$ and $y = \sum_{j=m}^{n} y_j \beta^j$; digits $x_j, y_j \in \mathcal{A}$

Algorithm I: Base β with a 'Strong Rewriting Rule'

This (k, h)-local function of addition is described in Algorithm I,

with parameters:

- Notation: $a = a' + cM$, where $a' = \lceil \frac{B-1}{2} \rceil$, $c = \lceil \frac{B-1}{2(B-2M)} \rceil$; alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, inner alphabet $\mathcal{A}' = \{-a', \ldots, 0, \ldots, +a'\}$; integers $m \leq n$
- Input: $x = \sum_{j=m}^{n} x_j \beta^j$ and $y = \sum_{j=m}^{n} y_j \beta^j$; digits $x_j, y_j \in \mathcal{A}$
- Output: $z = x + y = \sum_{j=m-h}^{n+k} z_j \beta^j$; digits $z_j \in \mathcal{A}$

Algorithm I: Base β with a 'Strong Rewriting Rule'

This (k, h)-local function of addition is described in Algorithm I,

with parameters:

- Notation: $a = a' + cM$, where $a' = \lceil \frac{B-1}{2} \rceil$, $c = \lceil \frac{B-1}{2(B-2M)} \rceil$; alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, inner alphabet $\mathcal{A}' = \{-a', \ldots, 0, \ldots, +a'\}$; integers $m \leq n$
- Input: $x = \sum_{j=m}^{n} x_j \beta^j$ and $y = \sum_{j=m}^{n} y_j \beta^j$; digits $x_j, y_j \in \mathcal{A}$
- Output: $z = x + y = \sum_{j=m-h}^{n+k} z_j \beta^j$; digits $z_j \in \mathcal{A}$

and with steps:

Algorithm I: Base β with a 'Strong Rewriting Rule'

This (k, h)-local function of addition is described in Algorithm I,

with parameters:

- Notation: $a = a' + cM$, where $a' = \lceil \frac{B-1}{2} \rceil$, $c = \lceil \frac{B-1}{2(B-2M)} \rceil$; alphabet $\mathcal{A} = \{-a, \dots, 0, \dots, +a\}$, inner alphabet $\mathcal{A}' = \{-a', \dots, 0, \dots, +a'\}$; integers $m \leq n$
- Input: $x = \sum_{j=m}^{n} x_j \beta^j$ and $y = \sum_{j=m}^{n} y_j \beta^j$; digits $x_j, y_j \in \mathcal{A}$
- Output: $z = x + y = \sum_{j=m-h}^{n+k} z_j \beta^j$; digits $z_j \in \mathcal{A}$

and with steps:

- Line 0.: for each $j = m, \dots, n$, put $w_j := x_j + y_j$

Algorithm I: Base β with a 'Strong Rewriting Rule'

This (k, h)-local function of addition is described in Algorithm I,

with parameters:

- Notation: $a = a' + cM$, where $a' = \lceil \frac{B-1}{2} \rceil$, $c = \lceil \frac{B-1}{2(B-2M)} \rceil$; alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, inner alphabet $\mathcal{A}' = \{-a', \ldots, 0, \ldots, +a'\}$; integers $m \leq n$
- Input: $x = \sum_{j=m}^{n} x_j \beta^j$ and $y = \sum_{j=m}^{n} y_j \beta^j$; digits $x_j, y_j \in \mathcal{A}$
- Output: $z = x + y = \sum_{j=m-h}^{n+k} z_j \beta^j$; digits $z_j \in \mathcal{A}$

and with steps:

- Line 0.: for each $j = m, \ldots, n$, put $w_j := x_j + y_j$
- Line 1.: for each $j = m, \ldots, n$, find $q_j \in \{-c, \ldots, 0, \ldots, +c\}$ such that $w_j - q_j B \in \mathcal{A}'$

Algorithm I: Base β with a 'Strong Rewriting Rule'

This (k, h)-local function of addition is described in Algorithm I,

with parameters:

- Notation: $a = a' + cM$, where $a' = \lceil \frac{B-1}{2} \rceil$, $c = \lceil \frac{B-1}{2(B-2M)} \rceil$; alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, inner alphabet $\mathcal{A}' = \{-a', \ldots, 0, \ldots, +a'\}$; integers $m \leq n$
- Input: $x = \sum_{j=m}^{n} x_j \beta^j$ and $y = \sum_{j=m}^{n} y_j \beta^j$; digits $x_j, y_j \in \mathcal{A}$
- Output: $z = x + y = \sum_{j=m-h}^{n+k} z_j \beta^j$; digits $z_j \in \mathcal{A}$

and with steps:

- Line 0.: for each $j = m, \ldots, n$, put $w_j := x_j + y_j$
- Line 1.: for each $j = m, \ldots, n$, find $q_j \in \{-c, \ldots, 0, \ldots, +c\}$ such that $w_j - q_j B \in \mathcal{A}'$
- Line 2.: for each $j = m - h, \ldots, n + k$, put $z_j := w_j - \sum_{i=-h}^{k} q_{j-i} b_i$

Algorithm I: Base β with a 'Strong Rewriting Rule'

Algorithm I: Base β with a 'Strong Rewriting Rule'

- Obviously, since $x_j, y_j \in \mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, by application of Line 0., we obtain $w_j = x_j + y_j \in \mathcal{A} + \mathcal{A} = \{-2a, \ldots, 0, \ldots, +2a\}$.

Algorithm I: Base β with a 'Strong Rewriting Rule'

- Obviously, since $x_j, y_j \in \mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, by application of Line 0., we obtain $w_j = x_j + y_j \in \mathcal{A} + \mathcal{A} = \{-2a, \ldots, 0, \ldots, +2a\}$.
- The steps listed in Lines 1.+2. in fact describe how to deduct a convenient q_j-multiple of the 'strong rewriting rule' on the j-th position of the β-representation $(w_j)_{j=m}^{n}$:

Algorithm I: Base β with a 'Strong Rewriting Rule'

- Obviously, since $x_j, y_j \in \mathcal{A} = \{-a, \dots, 0, \dots, +a\}$, by application of Line 0., we obtain $w_j = x_j + y_j \in \mathcal{A} + \mathcal{A} = \{-2a, \dots, 0, \dots, +2a\}$.
- The steps listed in Lines 1.+2. in fact describe how to deduct a convenient q_j-multiple of the 'strong rewriting rule' on the j-th position of the β-representation $(w_j)_{j=m}^{n}$:

$w_j \in \mathcal{A} + \mathcal{A}$

Algorithm I: Base β with a 'Strong Rewriting Rule'

- Obviously, since $x_j, y_j \in \mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, by application of Line 0., we obtain $w_j = x_j + y_j \in \mathcal{A} + \mathcal{A} = \{-2a, \ldots, 0, \ldots, +2a\}$.
- The steps listed in Lines 1.+2. in fact describe how to deduct a convenient q_j-multiple of the 'strong rewriting rule' on the j-th position of the β-representation $(w_j)_{j=m}^n$:

$$w_j \in \mathcal{A} + \mathcal{A} \qquad \underbrace{-2a, \ldots, -a, \ldots, -a', \ldots, 0, \ldots, +a', \ldots, +a, \ldots, +2a}$$

Algorithm I: Base β with a 'Strong Rewriting Rule'

- Obviously, since $x_j, y_j \in \mathcal{A} = \{-a, \dots, 0, \dots, +a\}$, by application of Line 0., we obtain $w_j = x_j + y_j \in \mathcal{A} + \mathcal{A} = \{-2a, \dots, 0, \dots, +2a\}$.
- The steps listed in Lines 1.+2. in fact describe how to deduct a convenient q_j-multiple of the 'strong rewriting rule' on the j-th position of the β-representation $(w_j)_{j=m}^{n}$:

$$w_j \in \mathcal{A} + \mathcal{A} \qquad \underbrace{-2a, \dots, -a, \dots, -a', \dots, 0, \dots, +a', \dots, +a, \dots, +2a}$$

$$w_j - q_j B \in \mathcal{A}'$$

Algorithm I: Base β with a 'Strong Rewriting Rule'

- Obviously, since $x_j, y_j \in \mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, by application of Line 0., we obtain $w_j = x_j + y_j \in \mathcal{A} + \mathcal{A} = \{-2a, \ldots, 0, \ldots, +2a\}$.
- The steps listed in Lines 1.+2. in fact describe how to deduct a convenient q_j-multiple of the 'strong rewriting rule' on the j-th position of the β-representation $(w_j)_{j=m}^{n}$:

$$w_j \in \mathcal{A} + \mathcal{A} \qquad \underbrace{-2a, \ldots, -a, \ldots, -a', \ldots, 0, \ldots, +a', \ldots, +a, \ldots, +2a}$$

$$w_j - q_j B \in \mathcal{A}' \qquad \underbrace{\quad} \overbrace{-a', \ldots, 0, \ldots, +a'} \underbrace{\quad}$$

Algorithm I: Base β with a 'Strong Rewriting Rule'

- Obviously, since $x_j, y_j \in \mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, by application of Line 0., we obtain $w_j = x_j + y_j \in \mathcal{A} + \mathcal{A} = \{-2a, \ldots, 0, \ldots, +2a\}$.
- The steps listed in Lines 1.+2. in fact describe how to deduct a convenient q_j-multiple of the 'strong rewriting rule' on the j-th position of the β-representation $(w_j)_{j=m}^{n}$:

$w_j \in \mathcal{A} + \mathcal{A}$ $\qquad$ $\underbrace{-2a, \ldots, -a, \ldots, -a', \ldots, 0, \ldots, +a', \ldots, +a, \ldots, +2a}$

$w_j - q_j B \in \mathcal{A}'$ $\qquad$ $\underbrace{}\overbrace{-a', \ldots, 0, \ldots, +a'}\underbrace{}$

$-\sum_{i=-h, i \neq j}^{k} q_{j-i} b_i$

Algorithm I: Base β with a 'Strong Rewriting Rule'

- Obviously, since $x_j, y_j \in \mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, by application of Line 0., we obtain $w_j = x_j + y_j \in \mathcal{A} + \mathcal{A} = \{-2a, \ldots, 0, \ldots, +2a\}$.
- The steps listed in Lines 1.+2. in fact describe how to deduct a convenient q_j-multiple of the 'strong rewriting rule' on the j-th position of the β-representation $(w_j)_{j=m}^{n}$:

$w_j \in \mathcal{A} + \mathcal{A}$ $\qquad \underbrace{-2a, \ldots, -a, \ldots, -a', \ldots, 0, \ldots, +a', \ldots, +a, \ldots, +2a}$

$w_j - q_j B \in \mathcal{A}'$ $\qquad \underbrace{}_{-cM} \overbrace{-a', \ldots, 0, \ldots, +a'} \underbrace{}_{+cM}$

$-\sum_{i=-h, i \neq j}^{k} q_{j-i} b_i$

Algorithm I: Base β with a 'Strong Rewriting Rule'

- Obviously, since $x_j, y_j \in \mathcal{A} = \{-a, \dots, 0, \dots, +a\}$, by application of Line 0., we obtain $w_j = x_j + y_j \in \mathcal{A} + \mathcal{A} = \{-2a, \dots, 0, \dots, +2a\}$.
- The steps listed in Lines 1.+2. in fact describe how to deduct a convenient q_j-multiple of the 'strong rewriting rule' on the j-th position of the β-representation $(w_j)_{j=m}^{n}$:

$w_j \in \mathcal{A} + \mathcal{A}$ $\qquad \underbrace{-2a, \dots, -a, \dots, -a', \dots, 0, \dots, +a', \dots, +a, \dots, +2a}$

$w_j - q_j B \in \mathcal{A}'$ $\qquad \overbrace{-a', \dots, 0, \dots, +a'}$

$- \sum_{i=-h, i \neq j}^{k} q_{j-i} b_i$ $\qquad \underbrace{\quad}_{-cM} \qquad \underbrace{\quad}_{+cM}$

$z_j \in \mathcal{A}$

Algorithm I: Base β with a 'Strong Rewriting Rule'

- Obviously, since $x_j, y_j \in \mathcal{A} = \{-a, \dots, 0, \dots, +a\}$, by application of Line 0., we obtain $w_j = x_j + y_j \in \mathcal{A} + \mathcal{A} = \{-2a, \dots, 0, \dots, +2a\}$.
- The steps listed in Lines 1.+2. in fact describe how to deduct a convenient q_j-multiple of the 'strong rewriting rule' on the j-th position of the β-representation $(w_j)_{j=m}^{n}$:

$w_j \in \mathcal{A} + \mathcal{A}$ $\qquad \underbrace{-2a, \dots, -a, \dots, -a', \dots, 0, \dots, +a', \dots, +a, \dots, +2a}$

$w_j - q_j B \in \mathcal{A}'$ $\qquad \overbrace{-a', \dots, 0, \dots, +a'}$

$-\sum_{i=-h, i\neq j}^{k} q_{j-i} b_i$ $\qquad \underbrace{\quad}_{-cM} \qquad \underbrace{\quad}_{+\,cM}$

$z_j \in \mathcal{A}$ $\qquad \overbrace{-a = -cM - a', \dots, +a' + cM = a}$

Algorithm I: Base β with a 'Strong Rewriting Rule'

- Obviously, since $x_j, y_j \in \mathcal{A} = \{-a, \dots, 0, \dots, +a\}$, by application of Line 0., we obtain $w_j = x_j + y_j \in \mathcal{A} + \mathcal{A} = \{-2a, \dots, 0, \dots, +2a\}$.
- The steps listed in Lines 1.+2. in fact describe how to deduct a convenient q_j-multiple of the 'strong rewriting rule' on the j-th position of the β-representation $(w_j)_{j=m}^{n}$:

$w_j \in \mathcal{A} + \mathcal{A}$ $\qquad \underbrace{-2a, \dots, -a, \dots, -a', \dots, 0, \dots, +a', \dots, +a, \dots, +2a}$

$w_j - q_j B \in \mathcal{A}'$ $\qquad \overbrace{-a', \dots, 0, \dots, +a'}$

$-\sum_{i=-h, i\neq j}^{k} q_{j-i} b_i$ $\qquad \underbrace{\quad}_{-cM} \qquad \underbrace{\quad}_{+\,cM}$

$z_j \in \mathcal{A}$ $\qquad \overbrace{-a = -cM - a', \dots, +a' + cM = a}$

Parameters $a' = \lceil \frac{B-1}{2} \rceil$, $c = \lceil \frac{B-1}{2(B-2M)} \rceil$, $a = a' + cM = \lceil \frac{B-1}{2} \rceil + \lceil \frac{B-1}{2(B-2M)} \rceil M$ are directly derived from the parameters B and M of the 'strong rewriting rule', as we want them to be as small as possible, and to fulfil the following inequalities:

$$2a' + 1 \geq B \qquad a' + cM \leq a \qquad 2a - cB \leq a'$$

Algorithm I: Base β with a 'Strong Rewriting Rule'

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: integer base $\beta = b \in \mathbb{N}$, $b \geq 3$:

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: integer base $\beta = b \in \mathbb{N}$, $b \geq 3$:

- 'strong rewriting rule' $-\beta + b = 0$

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: integer base $\beta = b \in \mathbb{N}$, $b \geq 3$:

- 'strong rewriting rule' $-\beta + b = 0$
- memory $k = 1$, anticipation $h = 0$, and so $p = 2$

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: integer base $\beta = b \in \mathbb{N}$, $b \geq 3$:

- 'strong rewriting rule' $-\beta + b = 0$
- memory $k = 1$, anticipation $h = 0$, and so $p = 2$
- $B = b$, $M = 1$, $c = 1$, $a' = \lceil \frac{b-1}{2} \rceil$, $a = \lceil \frac{b+1}{2} \rceil$

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: integer base $\beta = b \in \mathbb{N}$, $b \geq 3$:

- 'strong rewriting rule' $-\beta + b = 0$
- memory $k = 1$, anticipation $h = 0$, and so $p = 2$
- $B = b$, $M = 1$, $c = 1$, $a' = \lceil \frac{b-1}{2} \rceil$, $a = \lceil \frac{b+1}{2} \rceil$
- ... the same result as Avizienis: 2-local, in alphabet $\mathcal{A} = \{-\lceil \frac{b+1}{2} \rceil, \ldots, 0, \ldots, +\lceil \frac{b+1}{2} \rceil\}$

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: integer base $\beta = b \in \mathbb{N}$, $b \geq 3$:

- 'strong rewriting rule' $-\beta + b = 0$
- memory $k = 1$, anticipation $h = 0$, and so $p = 2$
- $B = b$, $M = 1$, $c = 1$, $a' = \lceil \frac{b-1}{2} \rceil$, $a = \lceil \frac{b+1}{2} \rceil$
- ... the same result as Avizienis: 2-local, in alphabet $\mathcal{A} = \{-\lceil \frac{b+1}{2} \rceil, \ldots, 0, \ldots, +\lceil \frac{b+1}{2} \rceil\}$

Example: integer base $\beta = 2$:

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: integer base $\beta = b \in \mathbb{N}$, $b \geq 3$:

- 'strong rewriting rule' $-\beta + b = 0$
- memory $k = 1$, anticipation $h = 0$, and so $p = 2$
- $B = b$, $M = 1$, $c = 1$, $a' = \lceil \frac{b-1}{2} \rceil$, $a = \lceil \frac{b+1}{2} \rceil$
- ... the same result as Avizienis: 2-local, in alphabet $\mathcal{A} = \{-\lceil \frac{b+1}{2} \rceil, \ldots, 0, \ldots, +\lceil \frac{b+1}{2} \rceil\}$

Example: integer base $\beta = 2$:

- $-\beta + 2 = 0$ is NOT 'strong rewriting rule', however,

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: integer base $\beta = b \in \mathbb{N},\ b \geq 3$:

- 'strong rewriting rule' $-\beta + b = 0$
- memory $k = 1$, anticipation $h = 0$, and so $p = 2$
- $B = b$, $M = 1$, $c = 1$, $a' = \lceil \frac{b-1}{2} \rceil$, $a = \lceil \frac{b+1}{2} \rceil$
- ... the same result as Avizienis: 2-local, in alphabet $\mathcal{A} = \{-\lceil \frac{b+1}{2} \rceil, \dots, 0, \dots, +\lceil \frac{b+1}{2} \rceil\}$

Example: integer base $\beta = 2$:

- $-\beta + 2 = 0$ is NOT 'strong rewriting rule', however,
- instead, we can use the 'strong rewriting rule' $-\beta^2 + 4 = 0$

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: integer base $\beta = b \in \mathbb{N}$, $b \geq 3$:

- 'strong rewriting rule' $-\beta + b = 0$
- memory $k = 1$, anticipation $h = 0$, and so $p = 2$
- $B = b$, $M = 1$, $c = 1$, $a' = \lceil \frac{b-1}{2} \rceil$, $a = \lceil \frac{b+1}{2} \rceil$
- ... the same result as Avizienis: 2-local, in alphabet $\mathcal{A} = \{-\lceil \frac{b+1}{2} \rceil, \dots, 0, \dots, +\lceil \frac{b+1}{2} \rceil\}$

Example: integer base $\beta = 2$:

- $-\beta + 2 = 0$ is NOT 'strong rewriting rule', however,
- instead, we can use the 'strong rewriting rule' $-\beta^2 + 4 = 0$
- memory $k = 2$, anticipation $h = 0$, and so $p = 3$

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: integer base $\beta = b \in \mathbb{N}$, $b \geq 3$:

- 'strong rewriting rule' $-\beta + b = 0$
- memory $k = 1$, anticipation $h = 0$, and so $p = 2$
- $B = b$, $M = 1$, $c = 1$, $a' = \lceil \frac{b-1}{2} \rceil$, $a = \lceil \frac{b+1}{2} \rceil$
- ... the same result as Avizienis: 2-local, in alphabet $\mathcal{A} = \{-\lceil \frac{b+1}{2} \rceil, \ldots, 0, \ldots, +\lceil \frac{b+1}{2} \rceil\}$

Example: integer base $\beta = 2$:

- $-\beta + 2 = 0$ is NOT 'strong rewriting rule', however,
- instead, we can use the 'strong rewriting rule' $-\beta^2 + 4 = 0$
- memory $k = 2$, anticipation $h = 0$, and so $p = 3$
- $B = 4$, $M = 1$, $c = 1$, $a' = 2$, $a = 3$

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: integer base $\beta = b \in \mathbb{N}$, $b \geq 3$:

- 'strong rewriting rule' $-\beta + b = 0$
- memory $k = 1$, anticipation $h = 0$, and so $p = 2$
- $B = b$, $M = 1$, $c = 1$, $a' = \lceil \frac{b-1}{2} \rceil$, $a = \lceil \frac{b+1}{2} \rceil$
- ... the same result as Avizienis: 2-local, in alphabet $\mathcal{A} = \{-\lceil \frac{b+1}{2} \rceil, \ldots, 0, \ldots, +\lceil \frac{b+1}{2} \rceil\}$

Example: integer base $\beta = 2$:

- $-\beta + 2 = 0$ is NOT 'strong rewriting rule', however,
- instead, we can use the 'strong rewriting rule' $-\beta^2 + 4 = 0$
- memory $k = 2$, anticipation $h = 0$, and so $p = 3$
- $B = 4$, $M = 1$, $c = 1$, $a' = 2$, $a = 3$
- ... this algorithm is 3-local (like Chow & Robertson), but it has bigger alphabet $\mathcal{A} = \{-3, \ldots, 0, \ldots, +3\}$ than Chow & Robertson (where $\mathcal{A} = \{-1, 0, +1\}$)

Algorithm I: Base β with a 'Strong Rewriting Rule'

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: irrational base $\beta = \tau =$ the Golden Mean:

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: irrational base $\beta = \tau =$ the Golden Mean:

- τ is the bigger of the two roots of equation $\tau^2 = \tau + 1$, but this is NOT 'strong rewriting rule', however,

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: irrational base $\beta = \tau =$ the Golden Mean:

- τ is the bigger of the two roots of equation $\tau^2 = \tau + 1$, but this is NOT 'strong rewriting rule', however,
- instead, we can use the 'strong rewriting rule' $-\tau^4 + 7 - \tau^{-4} = 0$

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: irrational base $\beta = \tau =$ the Golden Mean:

- τ is the bigger of the two roots of equation $\tau^2 = \tau + 1$, but this is NOT 'strong rewriting rule', however,
- instead, we can use the 'strong rewriting rule' $-\tau^4 + 7 - \tau^{-4} = 0$
- memory $k = 4$, anticipation $h = 4$, and so $p = 9$

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: irrational base $\beta = \tau =$ the Golden Mean:

- τ is the bigger of the two roots of equation $\tau^2 = \tau + 1$, but this is NOT 'strong rewriting rule', however,
- instead, we can use the 'strong rewriting rule' $-\tau^4 + 7 - \tau^{-4} = 0$
- memory $k = 4$, anticipation $h = 4$, and so $p = 9$
- $B = 7$, $M = 2$, $c = 1$, $a' = 3$, $a = 5$

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: irrational base $\beta = \tau =$ the Golden Mean:

- τ is the bigger of the two roots of equation $\tau^2 = \tau + 1$, but this is NOT 'strong rewriting rule', however,
- instead, we can use the 'strong rewriting rule' $-\tau^4 + 7 - \tau^{-4} = 0$
- memory $k = 4$, anticipation $h = 4$, and so $p = 9$
- $B = 7$, $M = 2$, $c = 1$, $a' = 3$, $a = 5$
- ... we have a 9-local algorithm on alphabet $\mathcal{A} = \{-5, \ldots, 0, \ldots, +5\}$

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: irrational base $\beta = \tau =$ the Golden Mean:

- τ is the bigger of the two roots of equation $\tau^2 = \tau + 1$, but this is NOT 'strong rewriting rule', however,
- instead, we can use the 'strong rewriting rule' $-\tau^4 + 7 - \tau^{-4} = 0$
- memory $k = 4$, anticipation $h = 4$, and so $p = 9$
- $B = 7$, $M = 2$, $c = 1$, $a' = 3$, $a = 5$
- ... we have a 9-local algorithm on alphabet $\mathcal{A} = \{-5, \ldots, 0, \ldots, +5\}$

Example: complex base $\beta = -1 + \imath$:

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: irrational base $\beta = \tau =$ the Golden Mean:

- τ is the bigger of the two roots of equation $\tau^2 = \tau + 1$, but this is NOT 'strong rewriting rule', however,
- instead, we can use the 'strong rewriting rule' $-\tau^4 + 7 - \tau^{-4} = 0$
- memory $k = 4$, anticipation $h = 4$, and so $p = 9$
- $B = 7$, $M = 2$, $c = 1$, $a' = 3$, $a = 5$
- ... we have a 9-local algorithm on alphabet $\mathcal{A} = \{-5, \dots, 0, \dots, +5\}$

Example: complex base $\beta = -1 + \imath$:

- 'strong rewriting rule' $\beta^4 + 4 = 0$

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: irrational base $\beta = \tau =$ the Golden Mean:

- τ is the bigger of the two roots of equation $\tau^2 = \tau + 1$, but this is NOT 'strong rewriting rule', however,
- instead, we can use the 'strong rewriting rule' $-\tau^4 + 7 - \tau^{-4} = 0$
- memory $k = 4$, anticipation $h = 4$, and so $p = 9$
- $B = 7$, $M = 2$, $c = 1$, $a' = 3$, $a = 5$
- ... we have a 9-local algorithm on alphabet $\mathcal{A} = \{-5, \ldots, 0, \ldots, +5\}$

Example: complex base $\beta = -1 + \imath$:

- 'strong rewriting rule' $\beta^4 + 4 = 0$
- memory $k = 4$, anticipation $h = 0$, and so $p = 5$

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: irrational base $\beta = \tau =$ the Golden Mean:

- τ is the bigger of the two roots of equation $\tau^2 = \tau + 1$, but this is NOT 'strong rewriting rule', however,
- instead, we can use the 'strong rewriting rule' $-\tau^4 + 7 - \tau^{-4} = 0$
- memory $k = 4$, anticipation $h = 4$, and so $p = 9$
- $B = 7,\ M = 2,\ c = 1,\ a' = 3,\ a = 5$
- ... we have a 9-local algorithm on alphabet $\mathcal{A} = \{-5, \ldots, 0, \ldots, +5\}$

Example: complex base $\beta = -1 + \imath$:

- 'strong rewriting rule' $\beta^4 + 4 = 0$
- memory $k = 4$, anticipation $h = 0$, and so $p = 5$
- $B = 4,\ M = 1,\ c = 1,\ a' = 2,\ a = 3$

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: irrational base $\beta = \tau =$ the Golden Mean:

- τ is the bigger of the two roots of equation $\tau^2 = \tau + 1$, but this is NOT 'strong rewriting rule', however,
- instead, we can use the 'strong rewriting rule' $-\tau^4 + 7 - \tau^{-4} = 0$
- memory $k = 4$, anticipation $h = 4$, and so $p = 9$
- $B = 7$, $M = 2$, $c = 1$, $a' = 3$, $a = 5$
- ... we have a 9-local algorithm on alphabet $\mathcal{A} = \{-5, \ldots, 0, \ldots, +5\}$

Example: complex base $\beta = -1 + \imath$:

- 'strong rewriting rule' $\beta^4 + 4 = 0$
- memory $k = 4$, anticipation $h = 0$, and so $p = 5$
- $B = 4$, $M = 1$, $c = 1$, $a' = 2$, $a = 3$
- ... we have a 5-local algorithm on alphabet $\mathcal{A} = \{-3, \ldots, 0, \ldots, +3\}$

Algorithm I: Base β with a 'Strong Rewriting Rule'

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: irrational base $\beta = \frac{-17-\sqrt{265}}{4} \doteq -8.3197$:

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: irrational base $\beta = \frac{-17-\sqrt{265}}{4} \doteq -8.3197$:

- root of the 'strong rewriting rule' $2\beta + 17 + 3\beta^{-1} = 0$

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: irrational base $\beta = \frac{-17-\sqrt{265}}{4} \doteq -8.3197$:

- root of the 'strong rewriting rule' $2\beta + 17 + 3\beta^{-1} = 0$
- memory $k = 1$, anticipation $h = 1$, and so $p = 3$

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: irrational base $\beta = \frac{-17-\sqrt{265}}{4} \doteq -8.3197$:

- root of the 'strong rewriting rule' $2\beta + 17 + 3\beta^{-1} = 0$
- memory $k = 1$, anticipation $h = 1$, and so $p = 3$
- $B = 17$, $M = 5$, $c = 2$, $a' = 8$, $a = 18$

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: irrational base $\beta = \frac{-17-\sqrt{265}}{4} \doteq -8.3197$:

- root of the 'strong rewriting rule' $2\beta + 17 + 3\beta^{-1} = 0$
- memory $k = 1$, anticipation $h = 1$, and so $p = 3$
- $B = 17$, $M = 5$, $c = 2$, $a' = 8$, $a = 18$
- ... 3-local algorithm on alphabet $\mathcal{A} = \{-18, \ldots, 0, \ldots, +18\}$

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: irrational base $\beta = \frac{-17-\sqrt{265}}{4} \doteq -8.3197$:

- root of the 'strong rewriting rule' $2\beta + 17 + 3\beta^{-1} = 0$
- memory $k = 1$, anticipation $h = 1$, and so $p = 3$
- $B = 17$, $M = 5$, $c = 2$, $a' = 8$, $a = 18$
- ... 3-local algorithm on alphabet $\mathcal{A} = \{-18, \dots, 0, \dots, +18\}$

See the concrete action of parallel addition in this numeration system:

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: irrational base $\beta = \frac{-17-\sqrt{265}}{4} \doteq -8.3197$:

- root of the 'strong rewriting rule' $2\beta + 17 + 3\beta^{-1} = 0$
- memory $k = 1$, anticipation $h = 1$, and so $p = 3$
- $B = 17$, $M = 5$, $c = 2$, $a' = 8$, $a = 18$
- ... 3-local algorithm on alphabet $\mathcal{A} = \{-18, \dots, 0, \dots, +18\}$

See the concrete action of parallel addition in this numeration system:

x	$=$	(3)	(-7)	(17)	(-1)	(9)
y	$=$			(13)	(-9)	(-3)

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: irrational base $\beta = \frac{-17-\sqrt{265}}{4} \doteq -8.3197$:

- root of the 'strong rewriting rule' $2\beta + 17 + 3\beta^{-1} = 0$
- memory $k = 1$, anticipation $h = 1$, and so $p = 3$
- $B = 17$, $M = 5$, $c = 2$, $a' = 8$, $a = 18$
- ... 3-local algorithm on alphabet $\mathcal{A} = \{-18, \dots, 0, \dots, +18\}$

See the concrete action of parallel addition in this numeration system:

x	$=$	(3)	(-7)	(17)	(-1)	(9)
y	$=$			(13)	(-9)	(-3)
w	$=$	(3)	(-7)	(30)	(-10)	(6)

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: irrational base $\beta = \frac{-17-\sqrt{265}}{4} \doteq -8.3197$:

- root of the 'strong rewriting rule' $2\beta + 17 + 3\beta^{-1} = 0$
- memory $k = 1$, anticipation $h = 1$, and so $p = 3$
- $B = 17,\ M = 5,\ c = 2,\ a' = 8,\ a = 18$
- ... 3-local algorithm on alphabet $\mathcal{A} = \{-18, \ldots, 0, \ldots, +18\}$

See the concrete action of parallel addition in this numeration system:

x	$=$	(3)	(-7)	(17)	(-1)	(9)
y	$=$			(13)	(-9)	(-3)
w	$=$	(3)	(-7)	(30)	(-10)	(6)
$q_2 = 2$	$\mapsto$		(-4)	(-34)	(-6)	
$q_1 = -1$	$\mapsto$			(2)	(17)	(3)

Algorithm I: Base β with a 'Strong Rewriting Rule'

Example: irrational base $\beta = \frac{-17-\sqrt{265}}{4} \doteq -8.3197$:

- root of the 'strong rewriting rule' $2\beta + 17 + 3\beta^{-1} = 0$
- memory $k = 1$, anticipation $h = 1$, and so $p = 3$
- $B = 17,\ M = 5,\ c = 2,\ a' = 8,\ a = 18$
- ... 3-local algorithm on alphabet $\mathcal{A} = \{-18, \ldots, 0, \ldots, +18\}$

See the concrete action of parallel addition in this numeration system:

x	$=$	(3)	(−7)	(17)	(−1)	(9)
y	$=$			(13)	(−9)	(−3)
w	$=$	(3)	(−7)	(30)	(−10)	(6)
$q_2 = 2$	$\mapsto$		(−4)	(−34)	(−6)	
$q_1 = -1$	$\mapsto$			(2)	(17)	(3)
z	$=$	(3)	(−11)	(−2)	(1)	(9)

...q_j are the convenient coefficients indicating which multiples of the 'strong rewriting rule' need to be deducted for j-th position.

Algorithm I: Base β with a 'Strong Rewriting Rule'

?! Which algebraic numbers actually do have a 'strong rewriting rule'!?

Algorithm I: Base β with a 'Strong Rewriting Rule'

?! Which algebraic numbers actually do have a 'strong rewriting rule'!?

Remember e.g. the case of the Golden Mean τ, whose minimal polynomial $\tau^2 = \tau + 1$ is NOT 'strong rewriting rule'.

Algorithm I: Base β with a 'Strong Rewriting Rule'

?! Which algebraic numbers actually do have a 'strong rewriting rule'!?

Remember e.g. the case of the Golden Mean τ, whose minimal polynomial $\tau^2 = \tau + 1$ is NOT 'strong rewriting rule'.

However, τ is also root of another equation $-\tau^4 + 7 - \tau^{-4} = 0$, which already IS a 'strong rewriting rule'.

Algorithm I: Base β with a 'Strong Rewriting Rule'

?! Which algebraic numbers actually do have a 'strong rewriting rule'!?

Remember e.g. the case of the Golden Mean τ, whose minimal polynomial $\tau^2 = \tau + 1$ is NOT 'strong rewriting rule'.

However, τ is also root of another equation $-\tau^4 + 7 - \tau^{-4} = 0$, which already IS a 'strong rewriting rule'.

It was not only by chance that we found this 'strong rewriting rule' for τ:

Algorithm I: Base β with a 'Strong Rewriting Rule'

?! Which algebraic numbers actually do have a 'strong rewriting rule'!?

Remember e.g. the case of the Golden Mean τ, whose minimal polynomial $\tau^2 = \tau + 1$ is NOT 'strong rewriting rule'.

However, τ is also root of another equation $-\tau^4 + 7 - \tau^{-4} = 0$, which already IS a 'strong rewriting rule'.

It was not only by chance that we found this 'strong rewriting rule' for τ:

Theorem

Let α be an algebraic number of degree d with algebraic conjugates $\alpha_1, \ldots, \alpha_d$ (including α itself). Let $|\alpha_j| \neq 1$ for all $j = 1, \ldots, d$, and let $|\alpha| > 1$.

Algorithm I: Base β with a 'Strong Rewriting Rule'

?! Which algebraic numbers actually do have a 'strong rewriting rule'!?

Remember e.g. the case of the Golden Mean τ, whose minimal polynomial $\tau^2 = \tau + 1$ is NOT 'strong rewriting rule'.

However, τ is also root of another equation $-\tau^4 + 7 - \tau^{-4} = 0$, which already IS a 'strong rewriting rule'.

It was not only by chance that we found this 'strong rewriting rule' for τ:

Theorem

Let α be an algebraic number of degree d with algebraic conjugates $\alpha_1, \ldots, \alpha_d$ (including α itself). Let $|\alpha_j| \neq 1$ for all $j = 1, \ldots, d$, and let $|\alpha| > 1$.

Then there exists a polynomial $Q(X) \in \mathbb{Z}[X]$,

$$Q(X) = a_m X^m + a_{m-1} X^{m-1} + \cdots + a_1 X^1 + a_0$$

and an index $j_0 \in \{0, \ldots, m\}$ such that

$$Q(\alpha) = 0 \quad \text{and} \quad |a_{j_0}| > 2 \sum_{j=0, j \neq j_0}^{m} |a_j|.$$

Algorithm I: Base β with a 'Strong Rewriting Rule'

We can read this Theorem, in other words, that equation $\frac{1}{X^{j_0}} Q(X) = 0$ is a 'strong rewriting rule' for base α.

Algorithm I: Base β with a 'Strong Rewriting Rule'

We can read this Theorem, in other words, that equation $\frac{1}{X^{j_0}}Q(X) = 0$ is a 'strong rewriting rule' for base α.

It is important that we have proved this Theorem in a constructive way, so it gives a direct prescription leading to the concrete form of the 'strong rewriting rule' for a given base α, or β, ...

Algorithm I: Base β with a 'Strong Rewriting Rule'

We can read this Theorem, in other words, that equation $\frac{1}{X^{j_0}} Q(X) = 0$ is a 'strong rewriting rule' for base α.

It is important that we have proved this Theorem in a constructive way, so it gives a direct prescription leading to the concrete form of the 'strong rewriting rule' for a given base α, or β, ...

For some of the previously mentioned examples of bases, the application of this Theorem (and its constructive proof) provides the following results:

base $\beta \in \mathbb{C}$	minimal polynomial	polynomial $Q(X) \in \mathbb{Z}[X]$ obtained from the Theorem
$\beta = 2$	$-\beta + 2 = 0$	$Q(X) = X^2 - 4$
$\beta = \tau$	$-\beta^2 + \beta + 1 = 0$	$Q(X) = X^8 - 7X^4 + 1$
$\beta = -1 + \imath$	$\beta^4 + 4 = 0$	$Q(X) = X^4 + 4$

Algorithm I: Base β with a 'Strong Rewriting Rule'

As an implication of the previous Theorem, we have the following corollary:

Algorithm I: Base β with a 'Strong Rewriting Rule'

As an implication of the previous Theorem, we have the following corollary:

Theorem

Let β be an algebraic number of degree d, and let $|\beta| > 1$.

Algorithm I: Base β with a 'Strong Rewriting Rule'

As an implication of the previous Theorem, we have the following corollary:

Theorem

Let β be an algebraic number of degree d, and let $|\beta| > 1$.

- *If d is odd, or*

Algorithm I: Base β with a 'Strong Rewriting Rule'

As an implication of the previous Theorem, we have the following corollary:

Theorem

Let β be an algebraic number of degree d, and let $|\beta| > 1$.

- *If d is odd, or*
- *if d is even and the minimal polynomial of β is not reciprocal,*

Algorithm I: Base β with a 'Strong Rewriting Rule'

As an implication of the previous Theorem, we have the following corollary:

Theorem

Let β be an algebraic number of degree d, and let $|\beta| > 1$.

- *If d is odd, or*
- *if d is even and the minimal polynomial of β is not reciprocal,*

then β has a 'strong rewriting rule'.

Algorithm I: Base β with a 'Strong Rewriting Rule'

As an implication of the previous Theorem, we have the following corollary:

Theorem

Let β be an algebraic number of degree d, and let $|\beta| > 1$.

- *If d is odd, or*
- *if d is even and the minimal polynomial of β is not reciprocal,*

then β has a 'strong rewriting rule'.

Thereby we see that the class of algebraic numbers β that do have a 'strong rewriting rule' is quite large.

And for all such algebraic numbers β, the Algorithm I is working; i.e. there exists an alphabet $\mathcal{A} = \{-a, \ldots, +a\}$ in which addition can be done in parallel in the numeration system with base β.

From Algorithm I to Algorithm II

From Algorithm I to Algorithm II

- Having a numeration system with base β, which has a 'strong rewriting rule', we've seen how we can find an alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$ such that addition (and also deduction) is possible 'in parallel' (by means of a local function with a 'sliding window').

From Algorithm I to Algorithm II

- Having a numeration system with base β, which has a 'strong rewriting rule', we've seen how we can find an alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$ such that addition (and also deduction) is possible 'in parallel' (by means of a local function with a 'sliding window').
- The number of steps needed to do the parallel addition within Algorithm I is quite low - in fact only three steps.

From Algorithm I to Algorithm II

- Having a numeration system with base β, which has a 'strong rewriting rule', we've seen how we can find an alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$ such that addition (and also deduction) is possible 'in parallel' (by means of a local function with a 'sliding window').
- The number of steps needed to do the parallel addition within Algorithm I is quite low - in fact only three steps.
- The one disadvantage there is that the alphabet $\mathcal{A}$ is rather big.

From Algorithm I to Algorithm II

- Having a numeration system with base β, which has a 'strong rewriting rule', we've seen how we can find an alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$ such that addition (and also deduction) is possible 'in parallel' (by means of a local function with a 'sliding window').
- The number of steps needed to do the parallel addition within Algorithm I is quite low - in fact only three steps.
- The one disadvantage there is that the alphabet $\mathcal{A}$ is rather big.

That is why we are introducing another method, Algorithm II, wherein

From Algorithm I to Algorithm II

- Having a numeration system with base β, which has a 'strong rewriting rule', we've seen how we can find an alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$ such that addition (and also deduction) is possible 'in parallel' (by means of a local function with a 'sliding window').
- The number of steps needed to do the parallel addition within Algorithm I is quite low - in fact only three steps.
- The one disadvantage there is that the alphabet $\mathcal{A}$ is rather big.

That is why we are introducing another method, Algorithm II, wherein

- instead of 'strong rewriting rule', we need only 'weak rewriting rule',

From Algorithm I to Algorithm II

- Having a numeration system with base β, which has a 'strong rewriting rule', we've seen how we can find an alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$ such that addition (and also deduction) is possible 'in parallel' (by means of a local function with a 'sliding window').
- The number of steps needed to do the parallel addition within Algorithm I is quite low - in fact only three steps.
- The one disadvantage there is that the alphabet $\mathcal{A}$ is rather big.

That is why we are introducing another method, Algorithm II, wherein

- instead of 'strong rewriting rule', we need only 'weak rewriting rule',
- the alphabet $\mathcal{A}$ becomes smaller; however, on the other hand,

From Algorithm I to Algorithm II

- Having a numeration system with base β, which has a 'strong rewriting rule', we've seen how we can find an alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$ such that addition (and also deduction) is possible 'in parallel' (by means of a local function with a 'sliding window').
- The number of steps needed to do the parallel addition within Algorithm I is quite low - in fact only three steps.
- The one disadvantage there is that the alphabet $\mathcal{A}$ is rather big.

That is why we are introducing another method, Algorithm II, wherein

- instead of 'strong rewriting rule', we need only 'weak rewriting rule',
- the alphabet $\mathcal{A}$ becomes smaller; however, on the other hand,
- the number of steps needed to do the parallel addition within Algorithm II is generally higher, compared to Algorithm I.

From Algorithm I to Algorithm II

- Having a numeration system with base β, which has a 'strong rewriting rule', we've seen how we can find an alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$ such that addition (and also deduction) is possible 'in parallel' (by means of a local function with a 'sliding window').
- The number of steps needed to do the parallel addition within Algorithm I is quite low - in fact only three steps.
- The one disadvantage there is that the alphabet $\mathcal{A}$ is rather big.

That is why we are introducing another method, Algorithm II, wherein

- instead of 'strong rewriting rule', we need only 'weak rewriting rule',
- the alphabet $\mathcal{A}$ becomes smaller; however, on the other hand,
- the number of steps needed to do the parallel addition within Algorithm II is generally higher, compared to Algorithm I.

Cases where the two Algorithms I and II coincide are also specified.

Algorithm II: Base β with a 'Weak Rewriting Rule'

Algorithm II: Base β with a 'Weak Rewriting Rule'

Definition

An algebraic number $\beta \in \mathbb{C}$, $|\beta| > 1$ is said to have a 'weak rewriting rule' if there exist non-negative integers $h, k \in \mathbb{N}$, and a set of integers $b_k, b_{k-1}, \ldots, b_1, b_0, b_{-1}, \ldots, b_{-h} \in \mathbb{Z}$ such that

$$b_k\beta^k + b_{k-1}\beta^{k-1} + \ldots + b_1\beta^1 + b_0 + b_{-1}\beta^{-1} + \ldots + b_{-h}\beta^{-h} = 0$$

and $b_0 > \sum_{j=-h, j\neq 0}^{k} |b_j|$.

Algorithm II: Base β with a 'Weak Rewriting Rule'

Definition

An algebraic number $\beta \in \mathbb{C}$, $|\beta| > 1$ is said to have a 'weak rewriting rule' if there exist non-negative integers $h, k \in \mathbb{N}$, and a set of integers $b_k, b_{k-1}, \ldots, b_1, b_0, b_{-1}, \ldots, b_{-h} \in \mathbb{Z}$ such that

$$b_k\beta^k + b_{k-1}\beta^{k-1} + \ldots + b_1\beta^1 + b_0 + b_{-1}\beta^{-1} + \ldots + b_{-h}\beta^{-h} = 0$$

and $b_0 > \sum_{j=-h, j\neq 0}^{k} |b_j|$.

Theorem

Let $\beta \in \mathbb{C}$, $|\beta| > 1$ have a 'weak rewriting rule', wherein B, M denote $B = b_0$ and $M = \sum_{j=-h, j\neq 0}^{k} |b_j|$.

Algorithm II: Base β with a 'Weak Rewriting Rule'

Definition

An algebraic number $\beta \in \mathbb{C}$, $|\beta| > 1$ is said to have a 'weak rewriting rule' if there exist non-negative integers $h, k \in \mathbb{N}$, and a set of integers $b_k, b_{k-1}, \dots, b_1, b_0, b_{-1}, \dots, b_{-h} \in \mathbb{Z}$ such that

$$b_k\beta^k + b_{k-1}\beta^{k-1} + \dots + b_1\beta^1 + b_0 + b_{-1}\beta^{-1} + \dots + b_{-h}\beta^{-h} = 0$$

and $b_0 > \sum_{j=-h, j\neq 0}^{k} |b_j|$.

Theorem

Let $\beta \in \mathbb{C}$, $|\beta| > 1$ have a 'weak rewriting rule', wherein B, M denote $B = b_0$ and $M = \sum_{j=-h, j\neq 0}^{k} |b_j|$.

Then addition in numeration system with base β can be realized as a p-local function with memory sk and anticipation sh, in alphabet $\mathcal{A} = \{-a, \dots, 0, \dots, +a\}$, where $a = \lceil \frac{B-1}{2} \rceil + M$, $s = \lceil \frac{a}{B-M} \rceil$, and $p = sk + sh + 1$.

Algorithm II: Base β with a 'Weak Rewriting Rule'

This (sk, sh)-local function of addition is described in Algorithm II,

Algorithm II: Base β with a 'Weak Rewriting Rule'

This (sk, sh)-local function of addition is described in Algorithm II, with parameters:

Algorithm II: Base β with a 'Weak Rewriting Rule'

This (sk, sh)-local function of addition is described in Algorithm II, with parameters:

- Notation: $a = a' + M$, where $a' = \lceil \frac{B-1}{2} \rceil$, $s = \lceil \frac{a}{B-M} \rceil$; alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, inner alphabet $\mathcal{A}' = \{-a', \ldots, 0, \ldots, +a'\}$; integers $m \leq n$

Algorithm II: Base β with a 'Weak Rewriting Rule'

This (sk, sh)-local function of addition is described in Algorithm II, with parameters:

- Notation: $a = a' + M$, where $a' = \lceil \frac{B-1}{2} \rceil$, $s = \lceil \frac{a}{B-M} \rceil$; alphabet $\mathcal{A} = \{-a, \dots, 0, \dots, +a\}$, inner alphabet $\mathcal{A}' = \{-a', \dots, 0, \dots, +a'\}$; integers $m \leq n$
- Input: $x = \sum_{j=m}^{n} x_j \beta^j$ and $y = \sum_{j=m}^{n} y_j \beta^j$; digits $x_j, y_j \in \mathcal{A}$

Algorithm II: Base β with a 'Weak Rewriting Rule'

This (sk, sh)-local function of addition is described in Algorithm II, with parameters:

- Notation: $a = a' + M$, where $a' = \lceil \frac{B-1}{2} \rceil$, $s = \lceil \frac{a}{B-M} \rceil$; alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, inner alphabet $\mathcal{A}' = \{-a', \ldots, 0, \ldots, +a'\}$; integers $m \leq n$
- Input: $x = \sum_{j=m}^{n} x_j \beta^j$ and $y = \sum_{j=m}^{n} y_j \beta^j$; digits $x_j, y_j \in \mathcal{A}$
- Output: $z = x + y = \sum_{j=m-sh}^{n+sk} z_j \beta^j$; digits $z_j \in \mathcal{A}$

Algorithm II: Base β with a 'Weak Rewriting Rule'

This (sk, sh)-local function of addition is described in Algorithm II,

with parameters:

- Notation: $a = a' + M$, where $a' = \lceil \frac{B-1}{2} \rceil$, $s = \lceil \frac{a}{B-M} \rceil$; alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, inner alphabet $\mathcal{A}' = \{-a', \ldots, 0, \ldots, +a'\}$; integers $m \leq n$
- Input: $x = \sum_{j=m}^{n} x_j \beta^j$ and $y = \sum_{j=m}^{n} y_j \beta^j$; digits $x_j, y_j \in \mathcal{A}$
- Output: $z = x + y = \sum_{j=m-sh}^{n+sk} z_j \beta^j$; digits $z_j \in \mathcal{A}$

and with steps:

Algorithm II: Base β with a 'Weak Rewriting Rule'

This (sk, sh)-local function of addition is described in Algorithm II, with parameters:

- Notation: $a = a' + M$, where $a' = \lceil \frac{B-1}{2} \rceil$, $s = \lceil \frac{a}{B-M} \rceil$; alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, inner alphabet $\mathcal{A}' = \{-a', \ldots, 0, \ldots, +a'\}$; integers $m \leq n$
- Input: $x = \sum_{j=m}^{n} x_j \beta^j$ and $y = \sum_{j=m}^{n} y_j \beta^j$; digits $x_j, y_j \in \mathcal{A}$
- Output: $z = x + y = \sum_{j=m-sh}^{n+sk} z_j \beta^j$; digits $z_j \in \mathcal{A}$

and with steps:

- Line 0.: for each $j = m, \ldots, n$, put $w_j := x_j + y_j$

Algorithm II: Base β with a 'Weak Rewriting Rule'

This (sk, sh)-local function of addition is described in Algorithm II, with parameters:

- Notation: $a = a' + M$, where $a' = \lceil \frac{B-1}{2} \rceil$, $s = \lceil \frac{a}{B-M} \rceil$; alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, inner alphabet $\mathcal{A}' = \{-a', \ldots, 0, \ldots, +a'\}$; integers $m \leq n$
- Input: $x = \sum_{j=m}^{n} x_j \beta^j$ and $y = \sum_{j=m}^{n} y_j \beta^j$; digits $x_j, y_j \in \mathcal{A}$
- Output: $z = x + y = \sum_{j=m-sh}^{n+sk} z_j \beta^j$; digits $z_j \in \mathcal{A}$

and with steps:

- Line 0.: for each $j = m, \ldots, n$, put $w_j := x_j + y_j$
- Line 1.: for each $l = 1, \ldots, s$ do

Algorithm II: Base β with a 'Weak Rewriting Rule'

This (sk, sh)-local function of addition is described in Algorithm II,

with parameters:

- Notation: $a = a' + M$, where $a' = \lceil \frac{B-1}{2} \rceil$, $s = \lceil \frac{a}{B-M} \rceil$; alphabet $\mathcal{A} = \{-a, \dots, 0, \dots, +a\}$, inner alphabet $\mathcal{A}' = \{-a', \dots, 0, \dots, +a'\}$; integers $m \leq n$
- Input: $x = \sum_{j=m}^{n} x_j \beta^j$ and $y = \sum_{j=m}^{n} y_j \beta^j$; digits $x_j, y_j \in \mathcal{A}$
- Output: $z = x + y = \sum_{j=m-sh}^{n+sk} z_j \beta^j$; digits $z_j \in \mathcal{A}$

and with steps:

- Line 0.: for each $j = m, \dots, n$, put $w_j := x_j + y_j$
- Line 1.: for each $l = 1, \dots, s$ do
  - Line 1.a: for each $j = m - (l-1)h, \dots, n + (l-1)k$, put

Algorithm II: Base β with a 'Weak Rewriting Rule'

This (sk, sh)-local function of addition is described in Algorithm II, with parameters:

- Notation: $a = a' + M$, where $a' = \lceil \frac{B-1}{2} \rceil$, $s = \lceil \frac{a}{B-M} \rceil$; alphabet $\mathcal{A} = \{-a, \dots, 0, \dots, +a\}$, inner alphabet $\mathcal{A}' = \{-a', \dots, 0, \dots, +a'\}$; integers $m \leq n$
- Input: $x = \sum_{j=m}^{n} x_j \beta^j$ and $y = \sum_{j=m}^{n} y_j \beta^j$; digits $x_j, y_j \in \mathcal{A}$
- Output: $z = x + y = \sum_{j=m-sh}^{n+sk} z_j \beta^j$; digits $z_j \in \mathcal{A}$

and with steps:

- Line 0.: for each $j = m, \dots, n$, put $w_j := x_j + y_j$
- Line 1.: for each $l = 1, \dots, s$ do
  - Line 1.a: for each $j = m - (l-1)h, \dots, n + (l-1)k$, put
    - ⋆ $q_j := 0$ if $w_j \in \mathcal{A}'$

Algorithm II: Base β with a 'Weak Rewriting Rule'

This (sk, sh)-local function of addition is described in Algorithm II,

with parameters:

- Notation: $a = a' + M$, where $a' = \lceil \frac{B-1}{2} \rceil$, $s = \lceil \frac{a}{B-M} \rceil$; alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, inner alphabet $\mathcal{A}' = \{-a', \ldots, 0, \ldots, +a'\}$; integers $m \leq n$
- Input: $x = \sum_{j=m}^{n} x_j \beta^j$ and $y = \sum_{j=m}^{n} y_j \beta^j$; digits $x_j, y_j \in \mathcal{A}$
- Output: $z = x + y = \sum_{j=m-sh}^{n+sk} z_j \beta^j$; digits $z_j \in \mathcal{A}$

and with steps:

- Line 0.: for each $j = m, \ldots, n$, put $w_j := x_j + y_j$
- Line 1.: for each $l = 1, \ldots, s$ do
  - Line 1.a: for each $j = m - (l-1)h, \ldots, n + (l-1)k$, put
    - ★ $q_j := 0$ if $w_j \in \mathcal{A}'$
    - ★ $q_j := \operatorname{sgn}(w_j)$ if $w_j \notin \mathcal{A}'$

Algorithm II: Base β with a 'Weak Rewriting Rule'

This (sk, sh)-local function of addition is described in Algorithm II, with parameters:

- Notation: $a = a' + M$, where $a' = \lceil \frac{B-1}{2} \rceil$, $s = \lceil \frac{a}{B-M} \rceil$; alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, inner alphabet $\mathcal{A}' = \{-a', \ldots, 0, \ldots, +a'\}$; integers $m \leq n$
- Input: $x = \sum_{j=m}^{n} x_j \beta^j$ and $y = \sum_{j=m}^{n} y_j \beta^j$; digits $x_j, y_j \in \mathcal{A}$
- Output: $z = x + y = \sum_{j=m-sh}^{n+sk} z_j \beta^j$; digits $z_j \in \mathcal{A}$

and with steps:

- Line 0.: for each $j = m, \ldots, n$, put $w_j := x_j + y_j$
- Line 1.: for each $l = 1, \ldots, s$ do
  - Line 1.a: for each $j = m - (l-1)h, \ldots, n + (l-1)k$, put
    - $q_j := 0$ if $w_j \in \mathcal{A}'$
    - $q_j := \mathrm{sgn}(w_j)$ if $w_j \notin \mathcal{A}'$
  - Line 1.b: for each $j = m - lh, \ldots, n + lk$, put $w_j := w_j - \sum_{i=-h}^{k} q_{j-i} b_i$

Algorithm II: Base β with a 'Weak Rewriting Rule'

This (sk, sh)-local function of addition is described in Algorithm II, with parameters:

- Notation: $a = a' + M$, where $a' = \lceil \frac{B-1}{2} \rceil$, $s = \lceil \frac{a}{B-M} \rceil$; alphabet $\mathcal{A} = \{-a, \ldots, 0, \ldots, +a\}$, inner alphabet $\mathcal{A}' = \{-a', \ldots, 0, \ldots, +a'\}$; integers $m \leq n$
- Input: $x = \sum_{j=m}^{n} x_j \beta^j$ and $y = \sum_{j=m}^{n} y_j \beta^j$; digits $x_j, y_j \in \mathcal{A}$
- Output: $z = x + y = \sum_{j=m-sh}^{n+sk} z_j \beta^j$; digits $z_j \in \mathcal{A}$

and with steps:

- Line 0.: for each $j = m, \ldots, n$, put $w_j := x_j + y_j$
- Line 1.: for each $l = 1, \ldots, s$ do
  - Line 1.a: for each $j = m - (l-1)h, \ldots, n + (l-1)k$, put
    - $q_j := 0$ if $w_j \in \mathcal{A}'$
    - $q_j := \mathrm{sgn}(w_j)$ if $w_j \notin \mathcal{A}'$
  - Line 1.b: for each $j = m - lh, \ldots, n + lk$, put $w_j := w_j - \sum_{i=-h}^{k} q_{j-i} b_i$
- Line 2.: for each $j = m - sh, \ldots, n + sk$, put $z_j := w_j$

Comparison of Algorithm I versus Algorithm II

The basic idea of Algorithm II:

Comparison of Algorithm I versus Algorithm II

The basic idea of Algorithm II:

- the 'weak rewriting rule' decreases the maximum digit value by $B - M$ in each step,

Comparison of Algorithm I versus Algorithm II

The basic idea of Algorithm II:

- the 'weak rewriting rule' decreases the maximum digit value by $B - M$ in each step,
- by applying this action repeatedly, we get the minimal possible digits,

Comparison of Algorithm I versus Algorithm II

The basic idea of Algorithm II:

- the 'weak rewriting rule' decreases the maximum digit value by $B - M$ in each step,
- by applying this action repeatedly, we get the minimal possible digits,
- parameters a' and a are delimiting the alphabets (a for alphabet $\mathcal{A}$, a' for the inner alphabet $\mathcal{A}'$), and

Comparison of Algorithm I versus Algorithm II

The basic idea of Algorithm II:

- the 'weak rewriting rule' decreases the maximum digit value by $B - M$ in each step,
- by applying this action repeatedly, we get the minimal possible digits,
- parameters a' and a are delimiting the alphabets (a for alphabet $\mathcal{A}$, a' for the inner alphabet $\mathcal{A}'$), and
- s is the number of steps carried out in Line 1.

Comparison of Algorithm I versus Algorithm II

The basic idea of Algorithm II:

- the 'weak rewriting rule' decreases the maximum digit value by $B - M$ in each step,
- by applying this action repeatedly, we get the minimal possible digits,
- parameters a' and a are delimiting the alphabets (a for alphabet $\mathcal{A}$, a' for the inner alphabet $\mathcal{A}'$), and
- s is the number of steps carried out in Line 1.

Having a 'strong rewriting rule' for a base β, we can use both Algorithms I and II; then, they coincide if and only if $B \geq 4M - 1$; in such case:

Comparison of Algorithm I versus Algorithm II

The basic idea of Algorithm II:

- the 'weak rewriting rule' decreases the maximum digit value by $B - M$ in each step,
- by applying this action repeatedly, we get the minimal possible digits,
- parameters a' and a are delimiting the alphabets (a for alphabet $\mathcal{A}$, a' for the inner alphabet $\mathcal{A}'$), and
- s is the number of steps carried out in Line 1.

Having a 'strong rewriting rule' for a base β, we can use both Algorithms I and II; then, they coincide if and only if $B \geq 4M - 1$; in such case:

- in Algorithm I, the paremeter $c = 1$, and so $a = a' + cM = a' + M$ is the same as $a = a' + M$ in Algorithm II;

Comparison of Algorithm I versus Algorithm II

The basic idea of Algorithm II:

- the 'weak rewriting rule' decreases the maximum digit value by $B - M$ in each step,
- by applying this action repeatedly, we get the minimal possible digits,
- parameters a' and a are delimiting the alphabets (a for alphabet $\mathcal{A}$, a' for the inner alphabet $\mathcal{A}'$), and
- s is the number of steps carried out in Line 1.

Having a 'strong rewriting rule' for a base β, we can use both Algorithms I and II; then, they coincide if and only if $B \geq 4M - 1$; in such case:

- in Algorithm I, the paremeter $c = 1$, and so $a = a' + cM = a' + M$ is the same as $a = a' + M$ in Algorithm II;
- in Algorithm II, the parameter $s = 1$, and so the number of steps is the same as in Algorithm I, and also the p-locality is the same (with $p = sk + sh + 1 = k + h + 1$)

Comparison of Algorithm I versus Algorithm II

The basic idea of Algorithm II:

- the 'weak rewriting rule' decreases the maximum digit value by $B - M$ in each step,
- by applying this action repeatedly, we get the minimal possible digits,
- parameters a' and a are delimiting the alphabets (a for alphabet $\mathcal{A}$, a' for the inner alphabet $\mathcal{A}'$), and
- s is the number of steps carried out in Line 1.

Having a 'strong rewriting rule' for a base β, we can use both Algorithms I and II; then, they coincide if and only if $B \geq 4M - 1$; in such case:

- in Algorithm I, the paremeter $c = 1$, and so $a = a' + cM = a' + M$ is the same as $a = a' + M$ in Algorithm II;
- in Algorithm II, the parameter $s = 1$, and so the number of steps is the same as in Algorithm I, and also the p-locality is the same (with $p = sk + sh + 1 = k + h + 1$)

If $4M - 1 > B > 2M$, then Algorithm II uses a strictly smaller alphabet than Algorithm I, but, at the same time, with strictly higher number of steps, and with strictly longer 'sliding window' (wider p-locality).

Comparison of Algorithm I versus Algorithm II

Example: integer base $\beta = b \in \mathbb{N}$, $b \geq 3$:

Comparison of Algorithm I versus Algorithm II

Example: integer base $\beta = b \in \mathbb{N}$, $b \geq 3$:

- $-\beta + b = 0$ is a 'strong rewriting rule', wherein

Comparison of Algorithm I versus Algorithm II

Example: integer base $\beta = b \in \mathbb{N}$, $b \geq 3$:

- $-\beta + b = 0$ is a 'strong rewriting rule', wherein
- $B = b$, $M = 1$, and $B \geq 4M - 1 = 3$,

Comparison of Algorithm I versus Algorithm II

Example: integer base $\beta = b \in \mathbb{N}$, $b \geq 3$:

- $-\beta + b = 0$ is a 'strong rewriting rule', wherein
- $B = b$, $M = 1$, and $B \geq 4M - 1 = 3$,
- so Algorithm II gives the same result as Algorithm I, namely again the Avizienis algorithm

Comparison of Algorithm I versus Algorithm II

Example: integer base $\beta = b \in \mathbb{N}$, $b \geq 3$:

- $-\beta + b = 0$ is a 'strong rewriting rule', wherein
- $B = b$, $M = 1$, and $B \geq 4M - 1 = 3$,
- so Algorithm II gives the same result as Algorithm I, namely again the Avizienis algorithm

Example: integer base $\beta = 2$:

Comparison of Algorithm I versus Algorithm II

Example: integer base $\beta = b \in \mathbb{N}$, $b \geq 3$:

- $-\beta + b = 0$ is a 'strong rewriting rule', wherein
- $B = b$, $M = 1$, and $B \geq 4M - 1 = 3$,
- so Algorithm II gives the same result as Algorithm I, namely again the Avizienis algorithm

Example: integer base $\beta = 2$:

- $-\beta + 2 = 0$ is NOT 'strong rewriting rule', but

Comparison of Algorithm I versus Algorithm II

Example: integer base $\beta = b \in \mathbb{N}$, $b \geq 3$:

- $-\beta + b = 0$ is a 'strong rewriting rule', wherein
- $B = b$, $M = 1$, and $B \geq 4M - 1 = 3$,
- so Algorithm II gives the same result as Algorithm I, namely again the Avizienis algorithm

Example: integer base $\beta = 2$:

- $-\beta + 2 = 0$ is NOT 'strong rewriting rule', but
- it is a 'weak rewriting rule', which we can use for Algorithm II,

Comparison of Algorithm I versus Algorithm II

Example: integer base $\beta = b \in \mathbb{N},\ b \geq 3$:

- $-\beta + b = 0$ is a 'strong rewriting rule', wherein
- $B = b$, $M = 1$, and $B \geq 4M - 1 = 3$,
- so Algorithm II gives the same result as Algorithm I, namely again the Avizienis algorithm

Example: integer base $\beta = 2$:

- $-\beta + 2 = 0$ is NOT 'strong rewriting rule', but
- it is a 'weak rewriting rule', which we can use for Algorithm II,
- with parameters $B = 2$, $M = 1$, $a' = 1$, $a = 2$, $s = 2$, $k = 1$, $h = 0$, and therefore memory $sk = 2$, anticipation $sh = 0$, and so $p = sk + sh + 1 = 3$

Comparison of Algorithm I versus Algorithm II

Example: integer base $\beta = b \in \mathbb{N},\ b \geq 3$:

- $-\beta + b = 0$ is a 'strong rewriting rule', wherein
- $B = b$, $M = 1$, and $B \geq 4M - 1 = 3$,
- so Algorithm II gives the same result as Algorithm I, namely again the Avizienis algorithm

Example: integer base $\beta = 2$:

- $-\beta + 2 = 0$ is NOT 'strong rewriting rule', but
- it is a 'weak rewriting rule', which we can use for Algorithm II,
- with parameters $B = 2$, $M = 1$, $a' = 1$, $a = 2$, $s = 2$, $k = 1$, $h = 0$, and therefore memory $sk = 2$, anticipation $sh = 0$, and so $p = sk + sh + 1 = 3$
- ... again 3-local (like Chow & Robertson), but still with bigger alphabet $\mathcal{A} = \{-2, -1, 0, +1, +2\}$ than Chow & Robertson (where $\mathcal{A} = \{-1, 0, +1\}$)

Comparison of Algorithm I versus Algorithm II

Example: irrational base $\beta = \tau =$ the Golden Mean:

Comparison of Algorithm I versus Algorithm II

Example: irrational base $\beta = \tau =$ the Golden Mean:

- the minimal polynomial $\tau^2 = \tau + 1$ of the Golden Mean τ is NEITHER 'strong' NOR 'weak rewriting rule',

Comparison of Algorithm I versus Algorithm II

Example: irrational base $\beta = \tau =$ the Golden Mean:

- the minimal polynomial $\tau^2 = \tau + 1$ of the Golden Mean τ is NEITHER 'strong' NOR 'weak rewriting rule',
- there exists the 'strong rewriting rule' $-\tau^4 + 7 - \tau^{-4} = 0$, used for Algorithm I, and

Comparison of Algorithm I versus Algorithm II

Example: irrational base $\beta = \tau =$ the Golden Mean:

- the minimal polynomial $\tau^2 = \tau + 1$ of the Golden Mean τ is NEITHER 'strong' NOR 'weak rewriting rule',
- there exists the 'strong rewriting rule' $-\tau^4 + 7 - \tau^{-4} = 0$, used for Algorithm I, and
- there exists also a 'weak rewriting rule' which we can use for Algorithm II, namely $-\tau^2 + 3 - \tau^{-2} = 0$:

Comparison of Algorithm I versus Algorithm II

Example: irrational base $\beta = \tau =$ the Golden Mean:

- the minimal polynomial $\tau^2 = \tau + 1$ of the Golden Mean τ is NEITHER 'strong' NOR 'weak rewriting rule',
- there exists the 'strong rewriting rule' $-\tau^4 + 7 - \tau^{-4} = 0$, used for Algorithm I, and
- there exists also a 'weak rewriting rule' which we can use for Algorithm II, namely $-\tau^2 + 3 - \tau^{-2} = 0$:
- with parameters $B = 3$, $M = 2$, $a' = 1$, $a = 3$, $s = 3$, $k = 2$, $h = 2$, and therefore memory $sk = 6$, anticipation $sh = 6$, and so $p = sk + sh + 1 = 13$

Comparison of Algorithm I versus Algorithm II

Example: irrational base $\beta = \tau =$ the Golden Mean:

- the minimal polynomial $\tau^2 = \tau + 1$ of the Golden Mean τ is NEITHER 'strong' NOR 'weak rewriting rule',
- there exists the 'strong rewriting rule' $-\tau^4 + 7 - \tau^{-4} = 0$, used for Algorithm I, and
- there exists also a 'weak rewriting rule' which we can use for Algorithm II, namely $-\tau^2 + 3 - \tau^{-2} = 0$:
- with parameters $B = 3$, $M = 2$, $a' = 1$, $a = 3$, $s = 3$, $k = 2$, $h = 2$, and therefore memory $sk = 6$, anticipation $sh = 6$, and so $p = sk + sh + 1 = 13$
- ... Algorithm II is 13-local, with alphabet $\mathcal{A} = \{-3, \ldots, 0, \ldots, +3\}$

Comparison of Algorithm I versus Algorithm II

Example: irrational base $\beta = \tau =$ the Golden Mean:

- the minimal polynomial $\tau^2 = \tau + 1$ of the Golden Mean τ is NEITHER 'strong' NOR 'weak rewriting rule',
- there exists the 'strong rewriting rule' $-\tau^4 + 7 - \tau^{-4} = 0$, used for Algorithm I, and
- there exists also a 'weak rewriting rule' which we can use for Algorithm II, namely $-\tau^2 + 3 - \tau^{-2} = 0$:
- with parameters $B = 3$, $M = 2$, $a' = 1$, $a = 3$, $s = 3$, $k = 2$, $h = 2$, and therefore memory $sk = 6$, anticipation $sh = 6$, and so $p = sk + sh + 1 = 13$
- ... Algorithm II is 13-local, with alphabet $\mathcal{A} = \{-3, \ldots, 0, \ldots, +3\}$
- ... compare with Algorithm I for τ (based on the 'strong rewriting rule' $-\tau^4 + 7 - \tau^{-4} = 0$), which is 9-local on alphabet $\mathcal{A} = \{-5, \ldots, 0, \ldots, +5\}$

Algorithm II: Base β with a 'Weak Rewriting Rule'

See the concrete action of parallel addition in this numeration system:

Algorithm II: Base β with a 'Weak Rewriting Rule'

See the concrete action of parallel addition in this numeration system:

position	j	:	8	7	6	5	4	3	2	1	0.	-1	-2	-3	-4
	x	$=$					3	-1	3	0	3.				
	y	$=$					2	0	3	-2	3.				
	$w_{initial}$	$=$					5	-1	6	-2	6.				

Algorithm II: Base β with a 'Weak Rewriting Rule'

See the concrete action of parallel addition in this numeration system:

position	j	:	8	7	6	5	4	3	2	1	0.	-1	-2	-3	-4
	x	$=$					3	-1	3	0	3.				
	y	$=$					2	0	3	-2	3.				
	$w_{initial}$	$=$					5	-1	6	-2	6.				
$l = 1$	$q_0 = 1$	$\mapsto$							1	0	-3.	0	1		
	$q_1 = -1$	$\mapsto$						-1	0	3	0.	-1			
	$q_2 = 1$	$\mapsto$					1	0	-3	0	1.				
	$q_4 = 1$	$\mapsto$			1	0	-3	0	1						
	$w_{l=1}$	$=$			1	0	3	-2	5	1	4.	-1	1		

Algorithm II: Base β with a 'Weak Rewriting Rule'

See the concrete action of parallel addition in this numeration system:

position	j	:	8	7	6	5	4	3	2	1	0.	-1	-2	-3	-4
	x	$=$					3	-1	3	0	3.				
	y	$=$					2	0	3	-2	3.				
	$w_{initial}$	$=$					5	-1	6	-2	6.				
$l=1$	$q_0 = 1$	$\mapsto$							1	0	-3.	0	1		
	$q_1 = -1$	$\mapsto$						-1	0	3	0.	-1			
	$q_2 = 1$	$\mapsto$					1	0	-3	0	1.				
	$q_4 = 1$	$\mapsto$			1	0	-3	0	1						
	$w_{l=1}$	$=$			1	0	3	-2	5	1	4.	-1	1		
$l=2$	$q_0 = 1$	$\mapsto$							1	0	-3.	0	1		
	$q_2 = 1$	$\mapsto$					1	0	-3	0	1.				
	$q_3 = -1$	$\mapsto$				-1	0	3	0	-1					
	$q_4 = 1$	$\mapsto$			1	0	-3	0	1						
	$w_{l=2}$	$=$			2	-1	1	1	4	0	2.	-1	2		

Algorithm II: Base β with a 'Weak Rewriting Rule'

See the concrete action of parallel addition in this numeration system:

position	j	:	8	7	6	5	4	3	2	1	0.	-1	-2	-3	-4
	x	$=$					3	-1	3	0	3.				
	y	$=$					2	0	3	-2	3.				
	$w_{initial}$	$=$					5	-1	6	-2	6.				
$l = 1$	$q_0 = 1$	$\mapsto$							1	0	-3.	0	1		
	$q_1 = -1$	$\mapsto$						-1	0	3	0.	-1			
	$q_2 = 1$	$\mapsto$					1	0	-3	0	1.				
	$q_4 = 1$	$\mapsto$			1	0	-3	0	1						
	$w_{l=1}$	$=$			1	0	3	-2	5	1	4.	-1	1		
$l = 2$	$q_0 = 1$	$\mapsto$							1	0	-3.	0	1		
	$q_2 = 1$	$\mapsto$					1	0	-3	0	1.				
	$q_3 = -1$	$\mapsto$				-1	0	3	0	-1					
	$q_4 = 1$	$\mapsto$			1	0	-3	0	1						
	$w_{l=2}$	$=$			2	-1	1	1	4	0	2.	-1	2		
$l = 3$	$q_{-2} = 1$	$\mapsto$									1.	0	-3	0	1
	$q_0 = 1$	$\mapsto$							1	0	-3.	0	1		
	$q_2 = 1$	$\mapsto$					1	0	-3	0	1.				
	$q_6 = 1$	$\mapsto$	1	0	-3	0	1								
$z =$	$w_{l=3}$	$=$	1	0	-1	-1	3	1	2	0	1.	-1	0	0	1

Algorithm II: Base β with a 'Weak Rewriting Rule'

See the concrete action of parallel addition in this numeration system:

position	j	:	8	7	6	5	4	3	2	1	0.	-1	-2	-3	-4
	x	$=$					3	-1	3	0	3.				
	y	$=$					2	0	3	-2	3.				
	$w_{initial}$	$=$					5	-1	6	-2	6.				
$l = 1$	$q_0 = 1$	$\mapsto$							1	0	-3.	0	1		
	$q_1 = -1$	$\mapsto$						-1	0	3	0.	-1			
	$q_2 = 1$	$\mapsto$					1	0	-3	0	1.				
	$q_4 = 1$	$\mapsto$			1	0	-3	0	1						
	$w_{l=1}$	$=$			1	0	3	-2	5	1	4.	-1	1		
$l = 2$	$q_0 = 1$	$\mapsto$							1	0	-3.	0	1		
	$q_2 = 1$	$\mapsto$					1	0	-3	0	1.				
	$q_3 = -1$	$\mapsto$				-1	0	3	0	-1					
	$q_4 = 1$	$\mapsto$			1	0	-3	0	1						
	$w_{l=2}$	$=$			2	-1	1	1	4	0	2.	-1	2		
$l = 3$	$q_{-2} = 1$	$\mapsto$									1.	0	-3	0	1
	$q_0 = 1$	$\mapsto$							1	0	-3.	0	1		
	$q_2 = 1$	$\mapsto$					1	0	-3	0	1.				
	$q_6 = 1$	$\mapsto$	1	0	-3	0	1								
$z =$	$w_{l=3}$	$=$	1	0	-1	-1	3	1	2	0	1.	-1	0	0	1

We run 3-times through the formulas on Line 1., because $s = 3$.

Algorithm II: Base β with a 'Weak Rewriting Rule'

See the concrete action of parallel addition in this numeration system:

position	j	:	8	7	6	5	4	3	2	1	0.	-1	-2	-3	-4
	x	$=$					3	-1	3	0	3.				
	y	$=$					2	0	3	-2	3.				
	$w_{initial}$	$=$					5	-1	6	-2	6.				
$l = 1$	$q_0 = 1$	$\mapsto$							1	0	-3.	0	1		
	$q_1 = -1$	$\mapsto$						-1	0	3	0.	-1			
	$q_2 = 1$	$\mapsto$					1	0	-3	0	1.				
	$q_4 = 1$	$\mapsto$			1	0	-3	0	1						
	$w_{l=1}$	$=$			1	0	3	-2	5	1	4.	-1	1		
$l = 2$	$q_0 = 1$	$\mapsto$							1	0	-3.	0	1		
	$q_2 = 1$	$\mapsto$					1	0	-3	0	1.				
	$q_3 = -1$	$\mapsto$				-1	0	3	0	-1					
	$q_4 = 1$	$\mapsto$			1	0	-3	0	1						
	$w_{l=2}$	$=$			2	-1	1	1	4	0	2.	-1	2		
$l = 3$	$q_{-2} = 1$	$\mapsto$									1.	0	-3	0	1
	$q_0 = 1$	$\mapsto$							1	0	-3.	0	1		
	$q_2 = 1$	$\mapsto$					1	0	-3	0	1.				
	$q_6 = 1$	$\mapsto$	1	0	-3	0	1								
$z =$	$w_{l=3}$	$=$	1	0	-1	-1	3	1	2	0	1.	-1	0	0	1

We run 3-times through the formulas on Line 1., because $s = 3$.
Notice how the length of the τ-representation is prolonged in each run...

Algorithms I and II versus Chow & Robertson

Algorithms I and II versus Chow & Robertson

- One feature, which is in common for both the Algorithms I and II, is that the decision about application of the rewriting rule at position j depends only on the actual value of the digit at the j-th position, and not on values of digits on any of the neighboring positions.

Algorithms I and II versus Chow & Robertson

- One feature, which is in common for both the Algorithms I and II, is that the decision about application of the rewriting rule at position j depends only on the actual value of the digit at the j-th position, and not on values of digits on any of the neighboring positions.
- This is a crucial difference against the algorithm of Chow & Robertson, in which the decision whether or not to apply the rewriting rule on position j depends not only on the actual digit on position j itself, but also on the value of digit of its right neighbor on position $j-1$.

Algorithms I and II versus Chow & Robertson

- One feature, which is in common for both the Algorithms I and II, is that the decision about application of the rewriting rule at position j depends only on the actual value of the digit at the j-th position, and not on values of digits on any of the neighboring positions.
- This is a crucial difference against the algorithm of Chow & Robertson, in which the decision whether or not to apply the rewriting rule on position j depends not only on the actual digit on position j itself, but also on the value of digit of its right neighbor on position $j-1$.
- Although this idea - to check the values on neighboring positions when operating on j-th position - is not easy to generalize, we took it as inspiration when searching how to further decrease the alphabet for parallel addition in base τ, the Golden Mean. Further we provide the result: parallel addition in base τ with alphabet $\mathcal{A} = \{-1, 0, +1\}$.

Algorithms I and II versus Chow & Robertson

- One feature, which is in common for both the Algorithms I and II, is that the decision about application of the rewriting rule at position j depends only on the actual value of the digit at the j-th position, and not on values of digits on any of the neighboring positions.
- This is a crucial difference against the algorithm of Chow & Robertson, in which the decision whether or not to apply the rewriting rule on position j depends not only on the actual digit on position j itself, but also on the value of digit of its right neighbor on position $j-1$.
- Although this idea - to check the values on neighboring positions when operating on j-th position - is not easy to generalize, we took it as inspiration when searching how to further decrease the alphabet for parallel addition in base τ, the Golden Mean. Further we provide the result: parallel addition in base τ with alphabet $\mathcal{A} = \{-1, 0, +1\}$.
- This result, with just minor modifications, is valid also for calculating in the Fibonacci numeration system , which is based on the same basic rewriting rule $F_{j+2} = F_{j+1} + F_j$.

Special Algorithms for Base τ, the Golden Mean

For base τ, the Golden Mean, we are able to do parallel addition

Special Algorithms for Base τ, the Golden Mean

For base τ, the Golden Mean, we are able to do parallel addition

- via Algorithm I, using the 'strong rewriting rule' $-\tau^4 + 7 - \tau^{-4} = 0$, with alphabet $\mathcal{A} = \{-5, \ldots, 0, \ldots, +5\}$; or

Special Algorithms for Base τ, the Golden Mean

For base τ, the Golden Mean, we are able to do parallel addition

- via Algorithm I, using the 'strong rewriting rule' $-\tau^4 + 7 - \tau^{-4} = 0$, with alphabet $\mathcal{A} = \{-5, \ldots, 0, \ldots, +5\}$; or
- via Algorithm II, using the 'weak rewriting rule' $-\tau^2 + 3 - \tau^{-2} = 0$, with alphabet $\mathcal{A} = \{-3, \ldots, 0, \ldots, +3\}$;

Special Algorithms for Base τ, the Golden Mean

For base τ, the Golden Mean, we are able to do parallel addition

- via Algorithm I, using the 'strong rewriting rule' $-\tau^4 + 7 - \tau^{-4} = 0$, with alphabet $\mathcal{A} = \{-5, \ldots, 0, \ldots, +5\}$; or
- via Algorithm II, using the 'weak rewriting rule' $-\tau^2 + 3 - \tau^{-2} = 0$, with alphabet $\mathcal{A} = \{-3, \ldots, 0, \ldots, +3\}$;

We introduce still two additional algorithms, developed specifically for the base τ, which enable us to further decrease the alphabet as follows:

Special Algorithms for Base τ, the Golden Mean

For base τ, the Golden Mean, we are able to do parallel addition

- via Algorithm I, using the 'strong rewriting rule' $-\tau^4 + 7 - \tau^{-4} = 0$, with alphabet $\mathcal{A} = \{-5, \ldots, 0, \ldots, +5\}$; or
- via Algorithm II, using the 'weak rewriting rule' $-\tau^2 + 3 - \tau^{-2} = 0$, with alphabet $\mathcal{A} = \{-3, \ldots, 0, \ldots, +3\}$;

We introduce still two additional algorithms, developed specifically for the base τ, which enable us to further decrease the alphabet as follows:

- Algorithm A: using the rewriting rule $-\tau^2 + 3 - \tau^{-2} = 0$ on j-th position, and checking the values of digits on neighboring positions $j+2$ and $j-2$, we get to alphabet $\mathcal{A} = \{-2, -1, 0, +1, +2\}$;

Special Algorithms for Base τ, the Golden Mean

For base τ, the Golden Mean, we are able to do parallel addition

- via Algorithm I, using the 'strong rewriting rule' $-\tau^4 + 7 - \tau^{-4} = 0$, with alphabet $\mathcal{A} = \{-5, \ldots, 0, \ldots, +5\}$; or
- via Algorithm II, using the 'weak rewriting rule' $-\tau^2 + 3 - \tau^{-2} = 0$, with alphabet $\mathcal{A} = \{-3, \ldots, 0, \ldots, +3\}$;

We introduce still two additional algorithms, developed specifically for the base τ, which enable us to further decrease the alphabet as follows:

- Algorithm A: using the rewriting rule $-\tau^2 + 3 - \tau^{-2} = 0$ on j-th position, and checking the values of digits on neighboring positions $j+2$ and $j-2$, we get to alphabet $\mathcal{A} = \{-2, -1, 0, +1, +2\}$;
- plus, we do one additional trick, taking advantage of the fact that the rewriting rule $-\tau^2 + 3 - \tau^{-2} = 0$ operates only on positions j, $j+2$, $j-2$, but leaves positions $j+1$ and $j-1$ untouched; and

Special Algorithms for Base τ, the Golden Mean

For base τ, the Golden Mean, we are able to do parallel addition

- via Algorithm I, using the 'strong rewriting rule' $-\tau^4 + 7 - \tau^{-4} = 0$, with alphabet $\mathcal{A} = \{-5, \ldots, 0, \ldots, +5\}$; or
- via Algorithm II, using the 'weak rewriting rule' $-\tau^2 + 3 - \tau^{-2} = 0$, with alphabet $\mathcal{A} = \{-3, \ldots, 0, \ldots, +3\}$;

We introduce still two additional algorithms, developed specifically for the base τ, which enable us to further decrease the alphabet as follows:

- Algorithm A: using the rewriting rule $-\tau^2 + 3 - \tau^{-2} = 0$ on j-th position, and checking the values of digits on neighboring positions $j+2$ and $j-2$, we get to alphabet $\mathcal{A} = \{-2, -1, 0, +1, +2\}$;
- plus, we do one additional trick, taking advantage of the fact that the rewriting rule $-\tau^2 + 3 - \tau^{-2} = 0$ operates only on positions j, $j+2$, $j-2$, but leaves positions $j+1$ and $j-1$ untouched; and
- Algorithm B: using the rewriting rules $2 = \tau + 1 - \tau^{-1}$ and $2 = 1 + \tau^{-1} + \tau^{-2}$, we limit the alphabet to $\mathcal{A} = \{-1, 0, +1\}$, which is the minimal possible alphabet for parallel addition in base τ.

Special Algorithms for Base τ, the Golden Mean

Algorithm A uses the rewriting rule $-\tau^2 + 3 - \tau^{-2} = 0$:

Special Algorithms for Base τ, the Golden Mean

Algorithm A uses the rewriting rule $-\tau^2 + 3 - \tau^{-2} = 0$:

notation:

Special Algorithms for Base τ, the Golden Mean

Algorithm A uses the rewriting rule $-\tau^2 + 3 - \tau^{-2} = 0$:

notation:

- Input: $w = \sum_j w_j \tau^j$; digits $w_j \in \{-3, \ldots, 0, \ldots, +3\}$

Special Algorithms for Base τ, the Golden Mean

Algorithm A uses the rewriting rule $-\tau^2 + 3 - \tau^{-2} = 0$:

notation:

- Input: $w = \sum_j w_j \tau^j$; digits $w_j \in \{-3, \dots, 0, \dots, +3\}$
- Output: $z = w = \sum_j z_j \tau^j$; digits $z_j \in \{-2, -1, 0, +1, +2\}$

Special Algorithms for Base τ, the Golden Mean

Algorithm A uses the rewriting rule $-\tau^2 + 3 - \tau^{-2} = 0$:

notation:

- Input: $w = \sum_j w_j \tau^j$; digits $w_j \in \{-3, \dots, 0, \dots, +3\}$
- Output: $z = w = \sum_j z_j \tau^j$; digits $z_j \in \{-2, -1, 0, +1, +2\}$

steps:

Special Algorithms for Base τ, the Golden Mean

Algorithm A uses the rewriting rule $-\tau^2 + 3 - \tau^{-2} = 0$:

notation:

- Input: $w = \sum_j w_j \tau^j$; digits $w_j \in \{-3, \dots, 0, \dots, +3\}$
- Output: $z = w = \sum_j z_j \tau^j$; digits $z_j \in \{-2, -1, 0, +1, +2\}$

steps:

- Line 1.: for each j do
  - if $[w_j = 3]$ or $[w_j = 2$ and $(w_{j+2} \geq 2$ or $w_{j-2} \geq 2)]$ or $[(w_j = 1)$ and $(w_{j+2} > 0$ and $w_{j-2} > 0)]$, put $q_j := 1$;
  - if $[w_j = -3]$ or $[w_j = -2$ and $(w_{j+2} \leq -2$ or $w_{j-2} \leq -2)]$ or $[(w_j = -1)$ and $(w_{j+2} < 0$ and $w_{j-2} < 0)]$, put $q_j := -1$;
  - else put $q_j := 0$

Special Algorithms for Base τ, the Golden Mean

Algorithm A uses the rewriting rule $-\tau^2 + 3 - \tau^{-2} = 0$:

notation:

- Input: $w = \sum_j w_j \tau^j$; digits $w_j \in \{-3, \ldots, 0, \ldots, +3\}$
- Output: $z = w = \sum_j z_j \tau^j$; digits $z_j \in \{-2, -1, 0, +1, +2\}$

steps:

- Line 1.: for each j do
  - if $[w_j = 3]$ or $[w_j = 2$ and $(w_{j+2} \geq 2$ or $w_{j-2} \geq 2)]$ or $[(w_j = 1)$ and $(w_{j+2} > 0$ and $w_{j-2} > 0)]$, put $q_j := 1$;
  - if $[w_j = -3]$ or $[w_j = -2$ and $(w_{j+2} \leq -2$ or $w_{j-2} \leq -2)]$ or $[(w_j = -1)$ and $(w_{j+2} < 0$ and $w_{j-2} < 0)]$, put $q_j := -1$;
  - else put $q_j := 0$
- Line 2.: for each j, put $z_j := w_j - 3q_j + q_{j+2} + q_{j-2}$

Special Algorithms for Base τ, the Golden Mean

Algorithm A uses the rewriting rule $-\tau^2 + 3 - \tau^{-2} = 0$:

notation:

- Input: $w = \sum_j w_j \tau^j$; digits $w_j \in \{-3, \ldots, 0, \ldots, +3\}$
- Output: $z = w = \sum_j z_j \tau^j$; digits $z_j \in \{-2, -1, 0, +1, +2\}$

steps:

- Line 1.: for each j do
  - if $[w_j = 3]$ or $[w_j = 2$ and $(w_{j+2} \geq 2$ or $w_{j-2} \geq 2)]$ or $[(w_j = 1)$ and $(w_{j+2} > 0$ and $w_{j-2} > 0)]$, put $q_j := 1$;
  - if $[w_j = -3]$ or $[w_j = -2$ and $(w_{j+2} \leq -2$ or $w_{j-2} \leq -2)]$ or $[(w_j = -1)$ and $(w_{j+2} < 0$ and $w_{j-2} < 0)]$, put $q_j := -1$;
  - else put $q_j := 0$
- Line 2.: for each j, put $z_j := w_j - 3q_j + q_{j+2} + q_{j-2}$

Application of Algorithm A onto τ-representation in alphabet $\{-3, \ldots, 0, \ldots, +3\}$ results in τ-representation in alphabet $\{-2, -1, 0, +1, +2\}$.

Special Algorithms for Base τ, the Golden Mean

Algorithm B uses rewriting rules $2 = \tau + 1 - \tau^{-1}$ and $2 = 1 + \tau^{-1} + \tau^{-2}$:

Special Algorithms for Base τ, the Golden Mean

Algorithm B uses rewriting rules $2 = \tau + 1 - \tau^{-1}$ and $2 = 1 + \tau^{-1} + \tau^{-2}$:

notation:

Special Algorithms for Base τ, the Golden Mean

Algorithm B uses rewriting rules $2 = \tau + 1 - \tau^{-1}$ and $2 = 1 + \tau^{-1} + \tau^{-2}$:

notation:

- Input: $w = \sum_j w_j \tau^j$; digits $w_{2j} \in \{-2, -1, 0, +1, +2\}, w_{2j+1} = 0$

Special Algorithms for Base τ, the Golden Mean

Algorithm B uses rewriting rules $2 = \tau + 1 - \tau^{-1}$ and $2 = 1 + \tau^{-1} + \tau^{-2}$:

notation:

- Input: $w = \sum_j w_j \tau^j$; digits $w_{2j} \in \{-2, -1, 0, +1, +2\}, w_{2j+1} = 0$
- Output: $z = w = \sum_j z_j \tau^j$; digits $z_j \in \{-1, 0, +1\}$

Special Algorithms for Base τ, the Golden Mean

Algorithm B uses rewriting rules $2 = \tau + 1 - \tau^{-1}$ and $2 = 1 + \tau^{-1} + \tau^{-2}$:

notation:

- Input: $w = \sum_j w_j \tau^j$; digits $w_{2j} \in \{-2, -1, 0, +1, +2\}, w_{2j+1} = 0$
- Output: $z = w = \sum_j z_j \tau^j$; digits $z_j \in \{-1, 0, +1\}$

steps:

Special Algorithms for Base τ, the Golden Mean

Algorithm B uses rewriting rules $2 = \tau + 1 - \tau^{-1}$ and $2 = 1 + \tau^{-1} + \tau^{-2}$:

notation:

- Input: $w = \sum_j w_j \tau^j$; digits $w_{2j} \in \{-2, -1, 0, +1, +2\}, w_{2j+1} = 0$
- Output: $z = w = \sum_j z_j \tau^j$; digits $z_j \in \{-1, 0, +1\}$

steps:

- Line 1.: for each j do
  - if $w_j = 2$ and $w_{j-2} \geq 0$, put $q_j := -1$, $l_j := 1$, $m_j := -1$, $r_j := 0$;
  - if $w_j = 2$ and $w_{j-2} \leq -1$, put $q_j := -1$, $l_j := 0$, $m_j := 1$, $r_j := 1$;
  - if $w_j = -2$ and $w_{j-2} \leq 0$, put $q_j := 1$, $l_j := -1$, $m_j := 1$, $r_j := 0$;
  - if $w_j = -2$ and $w_{j-2} \geq 1$, put $q_j := 1$, $l_j := 0$, $m_j := -1$, $r_j := -1$;
  - if $w_j \neq \pm 2$, put $q_j := 0$, $l_j := 0$, $m_j := 0$, $r_j := 0$

Special Algorithms for Base τ, the Golden Mean

Algorithm B uses rewriting rules $2 = \tau + 1 - \tau^{-1}$ and $2 = 1 + \tau^{-1} + \tau^{-2}$:

notation:

- Input: $w = \sum_j w_j \tau^j$; digits $w_{2j} \in \{-2, -1, 0, +1, +2\}, w_{2j+1} = 0$
- Output: $z = w = \sum_j z_j \tau^j$; digits $z_j \in \{-1, 0, +1\}$

steps:

- Line 1.: for each j do
  - if $w_j = 2$ and $w_{j-2} \geq 0$, put $q_j := -1,\ l_j := 1,\ m_j := -1,\ r_j := 0$;
  - if $w_j = 2$ and $w_{j-2} \leq -1$, put $q_j := -1,\ l_j := 0,\ m_j := 1,\ r_j := 1$;
  - if $w_j = -2$ and $w_{j-2} \leq 0$, put $q_j := 1,\ l_j := -1,\ m_j := 1,\ r_j := 0$;
  - if $w_j = -2$ and $w_{j-2} \geq 1$, put $q_j := 1,\ l_j := 0,\ m_j := -1,\ r_j := -1$;
  - if $w_j \neq \pm 2$, put $q_j := 0,\ l_j := 0,\ m_j := 0,\ r_j := 0$
- Line 2.: for each j, put $z_j := w_j + q_j + m_{j+1} + r_{j+2} + l_{j-1}$

Special Algorithms for Base τ, the Golden Mean

Algorithm B uses rewriting rules $2 = \tau + 1 - \tau^{-1}$ and $2 = 1 + \tau^{-1} + \tau^{-2}$:

notation:

- Input: $w = \sum_j w_j \tau^j$; digits $w_{2j} \in \{-2, -1, 0, +1, +2\}, w_{2j+1} = 0$
- Output: $z = w = \sum_j z_j \tau^j$; digits $z_j \in \{-1, 0, +1\}$

steps:

- Line 1.: for each j do
  - if $w_j = 2$ and $w_{j-2} \geq 0$, put $q_j := -1$, $l_j := 1$, $m_j := -1$, $r_j := 0$;
  - if $w_j = 2$ and $w_{j-2} \leq -1$, put $q_j := -1$, $l_j := 0$, $m_j := 1$, $r_j := 1$;
  - if $w_j = -2$ and $w_{j-2} \leq 0$, put $q_j := 1$, $l_j := -1$, $m_j := 1$, $r_j := 0$;
  - if $w_j = -2$ and $w_{j-2} \geq 1$, put $q_j := 1$, $l_j := 0$, $m_j := -1$, $r_j := -1$;
  - if $w_j \neq \pm 2$, put $q_j := 0$, $l_j := 0$, $m_j := 0$, $r_j := 0$
- Line 2.: for each j, put $z_j := w_j + q_j + m_{j+1} + r_{j+2} + l_{j-1}$

Application of Algorithm B onto τ-representation with even digits in alphabet $\{-2, -1, 0, +1, +2\}$ and with zero odd digits results in τ-representation in alphabet $\{-1, 0, +1\}$.

Special Algorithms for Base τ, the Golden Mean

Now we compile the methods explained earlier into the final Algorithm III which carries out parallel addition in base τ with the minimal possible alphabet $\mathcal{A} = \{-1, 0, +1\}$. (The minimality of this alphabet has been proved by Ch.Frougny.)

Special Algorithms for Base τ, the Golden Mean

Now we compile the methods explained earlier into the final Algorithm III which carries out parallel addition in base τ with the minimal possible alphabet $\mathcal{A} = \{-1, 0, +1\}$. (The minimality of this alphabet has been proved by Ch.Frougny.)

Algorithm III combines Algorithm II with Algorithm A and Algorithm B, in the following way:

Special Algorithms for Base τ, the Golden Mean

Now we compile the methods explained earlier into the final Algorithm III which carries out parallel addition in base τ with the minimal possible alphabet $\mathcal{A} = \{-1, 0, +1\}$. (The minimality of this alphabet has been proved by Ch.Frougny.)

Algorithm III combines Algorithm II with Algorithm A and Algorithm B, in the following way:

notation:

Special Algorithms for Base τ, the Golden Mean

Now we compile the methods explained earlier into the final Algorithm III which carries out parallel addition in base τ with the minimal possible alphabet $\mathcal{A} = \{-1, 0, +1\}$. (The minimality of this alphabet has been proved by Ch.Frougny.)

Algorithm III combines Algorithm II with Algorithm A and Algorithm B, in the following way:

notation:

- Input: $x = \sum_j x_j \tau^j, y = \sum_j y_j \tau^j$; digits $x_j, y_j \in \mathcal{A} = \{-1, 0, +1\}$

Special Algorithms for Base τ, the Golden Mean

Now we compile the methods explained earlier into the final Algorithm III which carries out parallel addition in base τ with the minimal possible alphabet $\mathcal{A} = \{-1, 0, +1\}$. (The minimality of this alphabet has been proved by Ch.Frougny.)

Algorithm III combines Algorithm II with Algorithm A and Algorithm B, in the following way:

notation:

- Input: $x = \sum_j x_j \tau^j, y = \sum_j y_j \tau^j$; digits $x_j, y_j \in \mathcal{A} = \{-1, 0, +1\}$
- Output: $z = x + y = \sum_j z_j \tau^j$; digits $z_j \in \mathcal{A} = \{-1, 0, +1\}$

Special Algorithms for Base τ, the Golden Mean

Now we compile the methods explained earlier into the final Algorithm III which carries out parallel addition in base τ with the minimal possible alphabet $\mathcal{A} = \{-1, 0, +1\}$. (The minimality of this alphabet has been proved by Ch.Frougny.)

Algorithm III combines Algorithm II with Algorithm A and Algorithm B, in the following way:

notation:

- Input: $x = \sum_j x_j \tau^j, y = \sum_j y_j \tau^j$; digits $x_j, y_j \in \mathcal{A} = \{-1, 0, +1\}$
- Output: $z = x + y = \sum_j z_j \tau^j$; digits $z_j \in \mathcal{A} = \{-1, 0, +1\}$

steps:

Special Algorithms for Base τ, the Golden Mean

Now we compile the methods explained earlier into the final Algorithm III which carries out parallel addition in base τ with the minimal possible alphabet $\mathcal{A} = \{-1, 0, +1\}$. (The minimality of this alphabet has been proved by Ch.Frougny.)

Algorithm III combines Algorithm II with Algorithm A and Algorithm B, in the following way:

notation:

- Input: $x = \sum_j x_j \tau^j, y = \sum_j y_j \tau^j$; digits $x_j, y_j \in \mathcal{A} = \{-1, 0, +1\}$
- Output: $z = x + y = \sum_j z_j \tau^j$; digits $z_j \in \mathcal{A} = \{-1, 0, +1\}$

steps:

- Phase 0.: for each j put $w_j := x_j + y_j$

Special Algorithms for Base τ, the Golden Mean

Now we compile the methods explained earlier into the final Algorithm III which carries out parallel addition in base τ with the minimal possible alphabet $\mathcal{A} = \{-1, 0, +1\}$. (The minimality of this alphabet has been proved by Ch.Frougny.)

Algorithm III combines Algorithm II with Algorithm A and Algorithm B, in the following way:

notation:

- Input: $x = \sum_j x_j \tau^j, y = \sum_j y_j \tau^j$; digits $x_j, y_j \in \mathcal{A} = \{-1, 0, +1\}$
- Output: $z = x + y = \sum_j z_j \tau^j$; digits $z_j \in \mathcal{A} = \{-1, 0, +1\}$

steps:

- Phase 0.: for each j put $w_j := x_j + y_j$
- Phase 1.: for each j put $w_{2j}^{new} := w_{2j} + w_{2j-1} - w_{2j+1}$, $w_{2j+1}^{new} := 0$

Special Algorithms for Base τ, the Golden Mean

Now we compile the methods explained earlier into the final Algorithm III which carries out parallel addition in base τ with the minimal possible alphabet $\mathcal{A} = \{-1, 0, +1\}$. (The minimality of this alphabet has been proved by Ch.Frougny.)

Algorithm III combines Algorithm II with Algorithm A and Algorithm B, in the following way:

notation:

- Input: $x = \sum_j x_j \tau^j, y = \sum_j y_j \tau^j$; digits $x_j, y_j \in \mathcal{A} = \{-1, 0, +1\}$
- Output: $z = x + y = \sum_j z_j \tau^j$; digits $z_j \in \mathcal{A} = \{-1, 0, +1\}$

steps:

- Phase 0.: for each j put $w_j := x_j + y_j$
- Phase 1.: for each j put $w_{2j}^{new} := w_{2j} + w_{2j-1} - w_{2j+1}$, $w_{2j+1}^{new} := 0$
- Phase 2.: apply Algorithm II

Special Algorithms for Base τ, the Golden Mean

Now we compile the methods explained earlier into the final Algorithm III which carries out parallel addition in base τ with the minimal possible alphabet $\mathcal{A} = \{-1, 0, +1\}$. (The minimality of this alphabet has been proved by Ch.Frougny.)

Algorithm III combines Algorithm II with Algorithm A and Algorithm B, in the following way:

notation:

- Input: $x = \sum_j x_j \tau^j, y = \sum_j y_j \tau^j$; digits $x_j, y_j \in \mathcal{A} = \{-1, 0, +1\}$
- Output: $z = x + y = \sum_j z_j \tau^j$; digits $z_j \in \mathcal{A} = \{-1, 0, +1\}$

steps:

- Phase 0.: for each j put $w_j := x_j + y_j$
- Phase 1.: for each j put $w_{2j}^{new} := w_{2j} + w_{2j-1} - w_{2j+1}$, $w_{2j+1}^{new} := 0$
- Phase 2.: apply Algorithm II
- Phase 3.: apply Algorithm A

Special Algorithms for Base τ, the Golden Mean

Now we compile the methods explained earlier into the final Algorithm III which carries out parallel addition in base τ with the minimal possible alphabet $\mathcal{A} = \{-1, 0, +1\}$. (The minimality of this alphabet has been proved by Ch.Frougny.)

Algorithm III combines Algorithm II with Algorithm A and Algorithm B, in the following way:

notation:

- Input: $x = \sum_j x_j \tau^j, y = \sum_j y_j \tau^j$; digits $x_j, y_j \in \mathcal{A} = \{-1, 0, +1\}$
- Output: $z = x + y = \sum_j z_j \tau^j$; digits $z_j \in \mathcal{A} = \{-1, 0, +1\}$

steps:

- Phase 0.: for each j put $w_j := x_j + y_j$
- Phase 1.: for each j put $w_{2j}^{new} := w_{2j} + w_{2j-1} - w_{2j+1}$, $w_{2j+1}^{new} := 0$
- Phase 2.: apply Algorithm II
- Phase 3.: apply Algorithm A
- Phase 4.: apply Algorithm B

Special Algorithms for Base τ, the Golden Mean

All Phases 0.-4. maintain the total value $x + y$ of the τ-representation, only the digits are being transformed as needed.

Special Algorithms for Base τ, the Golden Mean

All Phases 0.-4. maintain the total value $x + y$ of the τ-representation, only the digits are being transformed as needed.

- We start from $x_j, y_j \in \mathcal{A} = \{-1, 0, +1\}$

Special Algorithms for Base τ, the Golden Mean

All Phases 0.-4. maintain the total value $x + y$ of the τ-representation, only the digits are being transformed as needed.

- We start from $x_j, y_j \in \mathcal{A} = \{-1, 0, +1\}$
- After Phase 0.: $w_j \in \{-2, -1, 0, +1, +2\}$, $w = x + y$;

Special Algorithms for Base τ, the Golden Mean

All Phases 0.-4. maintain the total value $x + y$ of the τ-representation, only the digits are being transformed as needed.

- We start from $x_j, y_j \in \mathcal{A} = \{-1, 0, +1\}$
- After Phase 0.: $w_j \in \{-2, -1, 0, +1, +2\}$, $w = x + y$;
- After Phase 1.: $w_{2j} \in \{-6, \ldots, 0, \ldots, +6\}$ and $w_{2j+1} = 0$

Special Algorithms for Base τ, the Golden Mean

All Phases 0.-4. maintain the total value $x + y$ of the τ-representation, only the digits are being transformed as needed.

- We start from $x_j, y_j \in \mathcal{A} = \{-1, 0, +1\}$
- After Phase 0.: $w_j \in \{-2, -1, 0, +1, +2\}$, $w = x + y$;
- After Phase 1.: $w_{2j} \in \{-6, \ldots, 0, \ldots, +6\}$ and $w_{2j+1} = 0$
- After Phase 2.: $w_{2j} \in \{-3, \ldots, 0, \ldots, +3\}$ and $w_{2j+1} = 0$

Special Algorithms for Base τ, the Golden Mean

All Phases 0.-4. maintain the total value $x + y$ of the τ-representation, only the digits are being transformed as needed.

- We start from $x_j, y_j \in \mathcal{A} = \{-1, 0, +1\}$
- After Phase 0.: $w_j \in \{-2, -1, 0, +1, +2\}$, $w = x + y$;
- After Phase 1.: $w_{2j} \in \{-6, \ldots, 0, \ldots, +6\}$ and $w_{2j+1} = 0$
- After Phase 2.: $w_{2j} \in \{-3, \ldots, 0, \ldots, +3\}$ and $w_{2j+1} = 0$
- After Phase 3.: $w_{2j} \in \{-2, -1, 0, +1, +2\}$ and $w_{2j+1} = 0$

Special Algorithms for Base τ, the Golden Mean

All Phases 0.-4. maintain the total value $x + y$ of the τ-representation, only the digits are being transformed as needed.

- We start from $x_j, y_j \in \mathcal{A} = \{-1, 0, +1\}$
- After <u>Phase 0.</u>: $w_j \in \{-2, -1, 0, +1, +2\}$, $w = x + y$;
- After <u>Phase 1.</u>: $w_{2j} \in \{-6, \dots, 0, \dots, +6\}$ and $w_{2j+1} = 0$
- After <u>Phase 2.</u>: $w_{2j} \in \{-3, \dots, 0, \dots, +3\}$ and $w_{2j+1} = 0$
- After <u>Phase 3.</u>: $w_{2j} \in \{-2, -1, 0, +1, +2\}$ and $w_{2j+1} = 0$
- After <u>Phase 4.</u>: $z_j \in \mathcal{A} = \{-1, 0, +1\}$

Special Algorithms for Base τ, the Golden Mean

All Phases 0.-4. maintain the total value $x + y$ of the τ-representation, only the digits are being transformed as needed.

- We start from $x_j, y_j \in \mathcal{A} = \{-1, 0, +1\}$
- After Phase 0.: $w_j \in \{-2, -1, 0, +1, +2\}$, $w = x + y$;
- After Phase 1.: $w_{2j} \in \{-6, \dots, 0, \dots, +6\}$ and $w_{2j+1} = 0$
- After Phase 2.: $w_{2j} \in \{-3, \dots, 0, \dots, +3\}$ and $w_{2j+1} = 0$
- After Phase 3.: $w_{2j} \in \{-2, -1, 0, +1, +2\}$ and $w_{2j+1} = 0$
- After Phase 4.: $z_j \in \mathcal{A} = \{-1, 0, +1\}$

Link between the numeration systems with base τ and Fibonacci:

Special Algorithms for Base τ, the Golden Mean

All Phases 0.-4. maintain the total value $x + y$ of the τ-representation, only the digits are being transformed as needed.

- We start from $x_j, y_j \in \mathcal{A} = \{-1, 0, +1\}$
- After Phase 0.: $w_j \in \{-2, -1, 0, +1, +2\}$, $w = x + y$;
- After Phase 1.: $w_{2j} \in \{-6, \dots, 0, \dots, +6\}$ and $w_{2j+1} = 0$
- After Phase 2.: $w_{2j} \in \{-3, \dots, 0, \dots, +3\}$ and $w_{2j+1} = 0$
- After Phase 3.: $w_{2j} \in \{-2, -1, 0, +1, +2\}$ and $w_{2j+1} = 0$
- After Phase 4.: $z_j \in \mathcal{A} = \{-1, 0, +1\}$

Link between the numeration systems with base τ and Fibonacci:

- All methods developed here for the numeration system with base τ are derived only from the basic equality / rewriting rule $\tau^2 = \tau + 1$.

Special Algorithms for Base τ, the Golden Mean

All Phases 0.-4. maintain the total value $x + y$ of the τ-representation, only the digits are being transformed as needed.

- We start from $x_j, y_j \in \mathcal{A} = \{-1, 0, +1\}$
- After Phase 0.: $w_j \in \{-2, -1, 0, +1, +2\}$, $w = x + y$;
- After Phase 1.: $w_{2j} \in \{-6, \ldots, 0, \ldots, +6\}$ and $w_{2j+1} = 0$
- After Phase 2.: $w_{2j} \in \{-3, \ldots, 0, \ldots, +3\}$ and $w_{2j+1} = 0$
- After Phase 3.: $w_{2j} \in \{-2, -1, 0, +1, +2\}$ and $w_{2j+1} = 0$
- After Phase 4.: $z_j \in \mathcal{A} = \{-1, 0, +1\}$

Link between the numeration systems with base τ and Fibonacci:

- All methods developed here for the numeration system with base τ are derived only from the basic equality / rewriting rule $\tau^2 = \tau + 1$.
- The same rewriting rule holds for the Fibonacci numeration system too, because $F_{j+2} = F_{j+1} + F_j$.

Special Algorithms for Base τ, the Golden Mean

All Phases 0.-4. maintain the total value $x + y$ of the τ-representation, only the digits are being transformed as needed.

- We start from $x_j, y_j \in \mathcal{A} = \{-1, 0, +1\}$
- After Phase 0.: $w_j \in \{-2, -1, 0, +1, +2\}$, $w = x + y$;
- After Phase 1.: $w_{2j} \in \{-6, \ldots, 0, \ldots, +6\}$ and $w_{2j+1} = 0$
- After Phase 2.: $w_{2j} \in \{-3, \ldots, 0, \ldots, +3\}$ and $w_{2j+1} = 0$
- After Phase 3.: $w_{2j} \in \{-2, -1, 0, +1, +2\}$ and $w_{2j+1} = 0$
- After Phase 4.: $z_j \in \mathcal{A} = \{-1, 0, +1\}$

Link between the numeration systems with base τ and Fibonacci:

- All methods developed here for the numeration system with base τ are derived only from the basic equality / rewriting rule $\tau^2 = \tau + 1$.
- The same rewriting rule holds for the Fibonacci numeration system too, because $F_{j+2} = F_{j+1} + F_j$.
- So all the algorithms valid for base τ work in the same way also for Fibonacci; we just need to modify the formulas for several last positions ($j = 0, 1, 2$) at the end of the Fibonacci representations.

Konec

TAK TO JE VŠECHNO, MILÉ DĚTIČKY...

Konec

TAK TO JE VŠECHNO, MILÉ DĚTIČKY...

... A TEĎ UŽ PĚKNĚ DO PRÁCE...

Konec

TAK TO JE VŠECHNO, MILÉ DĚTIČKY...

... A TEĎ UŽ PĚKNĚ DO PRÁCE...

... ANEBO RADŠI NA OBÍDEK !!