# Effective Computation of Generalized Abelian Complexity for Pisot Type Substitutive Sequences

(Part 1/2)

*Words and Subwords, Liège*

*May 12, 2025*

JM. Couvreur, M. Delacourt, **N. Ollinger,** P. Popoli, J. Shallit, and M. Stipulanti

LIFO, Université d'Orléans

[arXiv:2504.13584]

# Pisot Type Substitutive **Sequences**

It all starts with a finite **alphabet** $A$ of **letters** $A = \{0, 1, 2\}$

We **concatenate** letters to obtain (finite) **words** denoted by $u, v, w$

The **empty word** $\varepsilon$ is the only word of length $0$.

Let $A^n$ denote the set of words $w$ of **length** $|w| = n$

The set of **all finite words** is $A^* = \bigcup_{n \geqslant 0} A^n$

Concatenation extends to words, $(A^*, \cdot, \varepsilon)$ is a monoid.

Let $A^{\mathbb{N}}$ denote the set of (infinite) **sequences** denoted in bold like $\mathbf{x}$

If $\mathbf{x} \in A^* \cup A^{\mathbb{N}}$, let $\mathbf{x}[i]$ denote the letter in **position** $0 \leqslant i < |\mathbf{x}|$ inside $\mathbf{x}$

# Pisot Type Substitutive Sequences

A **morphism** is a map $\tau : A^* \to B^*$ compatible with concatenation, i.e., such that $\tau(uv) = \tau(u)\tau(v)$ for all $u, v \in A^*$.

A **substitution** $\tau : A \to A^*$ is the restriction of a morphism $\tau : A^* \to A^*$.

A **fixed point** of a substitution $\tau$ is a sequence $\mathbf{x} \in A^{\mathbb{N}}$ such that $\tau(\mathbf{x}) = \mathbf{x}$.

A substitution $\tau$ is **prolongable** on a letter $a \in A$ if $\tau(a) = au$ for some $u \in A^*$ and $\lim_{n\to\infty} |\tau^n(a)| = +\infty$. The **associated fixed point** $\tau^\omega(a)$ is $\lim_{n\to\infty} \tau^n(a) = a \prod_{n\geq 0} \tau^n(u)$.

In this talk, we focus on the famous **Tribonacci sequence**, the associated fixed point $\mathbf{t} = \tau^\omega(0)$ of the substitution $\tau : 0 \mapsto 01,\ 1 \mapsto 02,\ 2 \mapsto 0$.

$\mathbf{t} = 0102010010201010201001020102010010201010201001020102010010\ldots$

# Pisot Type Substitutive Sequences

The **incidence matrix** of a substitution $\tau : A \to A^*$ is the matrix $M_\tau \in \mathbb{N}^{A\times A}$, the $(i, j)$ entry of which is $|\tau(a_i)|_{a_j}$ where $A = \{a_1, \dots, a_n\}$.

A **Pisot-Vijayaraghavan number** $\theta$ is an algebraic integer, which is the dominant root of its minimal monic polynomial $P(X)$ with integer coefficients, where $P(X)$ is irreducible over $\mathbb{Z}$ and admits $n$ complex roots $\theta_1, \dots, \theta_n$, all distinct, satisfying $\theta = \theta_1 > 1 > |\theta_2| \geqslant \cdots \geqslant |\theta_n| > 0$.

A substitution is of **Pisot type** if the characteristic polynomial of its incidence matrix is the minimal polynomial of a Pisot number.

***Remark*** The **Tribonacci substitution** is **Pisot**.

$$M_\tau = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \qquad P(X) = X^3 - X^2 - X - 1$$

## Generalized Abelian Complexity

A **factor** $u$, of length $n$, of $\mathbf{x} \in A^* \cup A^{\mathbb{N}}$ is a subblock of $\mathbf{x}$, formally if $u = \mathbf{x}[i] \cdots \mathbf{x}[i+n-1]$ for some position $i$.

A **prefix** is a factor occuring at position $0$.

The **set of all factors** of $\mathbf{x}$ is denoted by $\mathcal{L}(\mathbf{x})$ and $\mathcal{L}_n(\mathbf{x}) = \mathcal{L}(\mathbf{x}) \cap A^n$.

***Definition*** Let $\mathbf{x}$ be a sequence. The **factor complexity** of $\mathbf{x}$ is the map $p_{\mathbf{x}} : \mathbb{N} \to \mathbb{N}, n \mapsto \#\mathcal{L}_n(\mathbf{x})$ that counts the factors for a given length.

***Theorem (Morse-Hedlund 1938)*** The **factor complexity** of $\mathbf{x}$ is **bounded** if and only if $\mathbf{x}$ is **ultimately periodic**.

$$p_{\mathbf{t}}(n) = 2n + 1$$

## Generalized Abelian Complexity

The **Parikh vector** $\psi(u) \in \mathbb{N}^A$ of a word $u \in A^*$ is defined, for all $a \in A$, by $\psi(u)[a] = |u|_a$, where $|u|_a$ is the **number of occurrences** of $a$ in $u$.

The **abelian complexity** of a sequence counts the number of different Parikh vectors obtained on factors for a given length.

***Definition*** Let $\mathbf{x}$ be a sequence. The **abelian complexity** of $\mathbf{x}$ is the map $\rho^{\mathrm{ab}}_{\mathbf{x}} : \mathbb{N} \to \mathbb{N}, n \mapsto \#\{\psi(u) \mid u \in \mathcal{L}_n(\mathbf{x})\}$.

***Remark*** The **abelian complexity** of the **Tribonacci sequence** has been extensively studied **(Richomme et al 2010)**, **(Turek 2013)**, **(Shallit 2021)**.

$$\rho^{\mathrm{ab}}_{\mathbf{t}}(n) \in \{3, 4, 5, 6, 7\} \qquad \forall n \geqslant 1$$

# Generalized Abelian Complexity

A generalization of the abelian complexity is the so-called ***k*-abelian complexity** for some positive integer $k \geq 1$ **(Karhumäki et al 2013)**.

Let $|w|_x$ denote the **number of occurrences** of the factor $x$ in the word $w$.

***Definition*** Two words $u$ and $v$ are ***k*-abelian equivalent**, denoted by $u \sim_k v$, if $|u|_x = |v|_x$ for every word $x$ of length at most $k$.

When $k = 1$, we simply talk about **abelian equivalence**.

***Definition*** The ***k*-abelian complexity** of a sequence $\mathbf{x} \in A^{\mathbb{N}}$ is $\rho^k_{\mathbf{x}}(n) = \#\mathcal{L}_n(\mathbf{x})/\sim_k$ that counts factors of $\mathbf{x}$ for a given length up to $\sim_k$.

## Generalized Exact Abelian Complexity

Two same-length words $u, v$ are **exactly-$k$-abelian equivalent**, denoted by $u \sim_{=k} v$, if $|u|_x = |v|_x$ for every word $x$ of length **exactly** $k$.

We define the **exact $k$-abelian complexity** of $\mathbf{x}$ as $\rho_{\mathbf{x}}^{=k}(n) = \#\mathcal{L}_n(\mathbf{x})/ \sim_{=k}$.

***Lemma*** Let $k \geq 1$. Two words $u, v \in A^*$ are **$k$-abelian equivalent** if and only the following conditions are satisfied:

1. $|u| = |v|$;
2. if $|u| < k$, then $u = v$;
3. if $|u| \geqslant k$, then $u \sim_{=k} v$ and $u[i] = v[i]$ for all $i < k - 1$.

Thanks to this, we can focus on computing the **exact $k$-abelian complexity**!

## Relations between complexities

***Lemma*** Let $\mathbf{x}$ be a sequence. We have $\rho_{\mathbf{x}}^{\mathrm{ab}} = \rho_{\mathbf{x}}^{1} = \rho_{\mathbf{x}}^{=1}$. For each integer $k \in \mathbb{N}$, we have

1. For all $n \in \mathbb{N}$, $\rho_{\mathbf{x}}^{k}(n) \leqslant \rho_{\mathbf{x}}^{k+1}(n)$;
2. For all $n \in \mathbb{N}$, $\rho_{\mathbf{x}}^{=k}(n) \leqslant \rho_{\mathbf{x}}^{k}(n) \leqslant \prod_{i=1}^{k} \rho_{\mathbf{x}}^{=i}(n)$;
3. For all $n < k$, $\rho_{\mathbf{x}}^{k}(n) = p_{\mathbf{x}}(n)$.

## Bounded abelian complexity

Let $C$ be a positive integer.

***Definition*** A sequence $\mathbf{x} \in A^{\mathbb{N}}$ is **$C$-balanced** if, for all factors $u, v$ of $\mathbf{x}$ of equal length and for every letter $a \in A$, we have $||u|_a - |v|_a| \leq C$.

When $C = 1$, we simply call $\mathbf{x}$ **balanced**.

***Lemma (folklore)*** A sequence $\mathbf{x}$ has **bounded abelian complexity** if and only if $\mathbf{x}$ is **$C$-balanced** for some positive integer $C$.

***Corollary (using Adamczewski 2003)*** The **abelian complexity** of the fixed point of a substitution of **Pisot type** is **bounded** by a constant.

# Bounded generalized abelian complexity

Let $k$ and $C_k$ be positive integers.

***Definition*** A sequence $\mathbf{x} \in A^{\mathbb{N}}$ is **$(k, C_k)$-balanced** if, for all factors $u, v$ of $\mathbf{x}$ of equal length and for each $w \in A^k$, we have $||u|_w - |v|_w| \leq C_k$.

The boundedness of the generalized abelian complexity is related to the generalized balancedness as follows.

***Lemma (Karhumaki et al. 2013)*** Let $k$ be a positive integer. A sequence $\mathbf{x}$ has **bounded $k$-abelian complexity** if and only if $\mathbf{x}$ is **$(k, C_k)$-balanced** for some positive integer $C_k$.

If $\rho_{\mathbf{x}}^k$ is bounded by $C_k$, then $\mathbf{x}$ is $(k, C_k - 1)$-balanced

If $\mathbf{x}$ is $(k, C_k)$-balanced, then $\rho_{\mathbf{x}}^k \leq (C_k + 1)^k$.

## Effective Computation

Given a **sequence** as an input, the fixed point of a prolongable substitution of **Pisot** type, we want to **compute** its generalized abelian equivalence relations and complexities.

# Effective Computation

Given a **sequence** as an input, the fixed point of a prolongable substitution of **Pisot** type, we want to **compute** its generalized abelian equivalence relations and complexities.

What is an acceptable encoding for the **input**?

## Effective Computation

Given a **sequence** as an input, the fixed point of a prolongable substitution of **Pisot** type, we want to **compute** its generalized abelian equivalence relations and complexities.

What is an acceptable encoding for the **input**? For the **output**?

## Effective Computation

Given a **sequence** as an input, the fixed point of a prolongable substitution of **Pisot** type, we want to **compute** its generalized abelian equivalence relations and complexities.

What is an acceptable encoding for the **input**? For the **output**?

We consider 3 representations for sequences:

- **$\mathcal{S}$-automatic** sequences;
- **$\mathcal{S}$-synchronized** sequences;
- **$\mathcal{S}$-regular** sequences.

where $\mathcal{S}$ denotes a **regular numeration system**.

# Abstract Numeration Systems (with zeros)

***Definition (Lecomte, Rigo 2000)*** A **numeration system (NS)** $\mathcal{S}$ is a tuple $(L, A, <, 0)$ with $A$ the alphabet, ordered by $<$, minimal element $0 \in A$, and $L \subseteq A^*$ such that $\varepsilon \in L$ and $w \in L \Leftrightarrow 0w \in L \quad \forall w \in A^*$.

The **encoding** $\mathrm{rep}_{\mathcal{S}}(n)$ of $n \in \mathbb{N}$ is the $n$th element of $L \setminus 0^+L$ (in radix order).

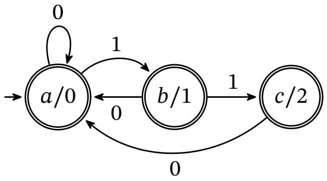
The **valuation** $\mathrm{val}_{\mathcal{S}}(u)$ of $u \in L$ is $\mathrm{rep}_{\mathcal{S}}^{-1}(v)$ where $u \in 0^*v$.

Let $\langle ., . \rangle$ be the **canonical isomorphism** between $\bigcup_{n\geq 0}(A^n \times B^n)$ and $(A \times B)^*$.

In this talk, we only consider **regular** NS, for which both $L$ and the **addition relation** $\{\langle x, y, z\rangle \mid \mathrm{val}_{\mathcal{S}}(x) + \mathrm{val}_{\mathcal{S}}(y) = \mathrm{val}_{\mathcal{S}}(z)\}$ are **regular**.

## Automatic sequences

***Definition*** A sequence $\mathbf{x} \in A^{\mathbb{N}}$ is **$\mathcal{S}$-automatic** if $\mathbf{x}$ can be computed by a **deterministic finite automaton with output (DFAO)** in the NS $\mathcal{S}$: on input $u \in L_{\mathcal{S}}$, its output is equal to $\mathbf{x}[\mathrm{val}_{\mathcal{S}}(u)]$.



The Tribonacci sequence in its DTNS

**(Rigo and Maes 2002)** proved that **morphic sequences** are captured by **automatic sequences** in various NS. We focus on **Dumont-Thomas NS (DTNS)** that capture **substitutive sequences** and are **regular** for substitutions of **Pisot type**.

# Regular sequences

***Definition (Allouche, Shallit 1992)*** A sequence $\mathbf{x} : \mathbb{N} \to \mathbb{Z}$ is **$\mathcal{S}$-regular** if there exist a row vector $\lambda$, a column vector $\gamma$, and a morphism $\mu : A_{\mathcal{S}}^{*} \to \mathbb{Z}^{m\times m}$ such that $\mathbf{x}[n] = \lambda\mu(\mathrm{rep}_{\mathcal{S}}(n))\gamma$ for all $n \in \mathbb{N}$.

The triple $(\lambda, \mu, \gamma)$ is called a **linear representation** of $\mathbf{x}$. They are stable under several operations: sum, external product, Hadmard product, ...

A **reduced representation** is a representation of **minimal dimension**.

***Theorem*** Let $\mathbf{x} : \mathbb{N} \to \mathbb{Z}$ be a sequence of integers.
If $\mathbf{x}$ is $\mathcal{S}$-automatic then $\mathbf{x}$ is $\mathcal{S}$-regular.
If $\mathbf{x}$ is $\mathcal{S}$-regular and $\mathbf{x}(\mathbb{N})$ is **finite** then $\mathbf{x}$ is $\mathcal{S}$-automatic.

## Büchi Arithmetic in base $k$

**Presburger Arithmetic** captures $\mathrm{Th}(\mathbb{N}, +, <)$ with **first-order** syntax

$$t, t' := \overbrace{x \mid y}^{\textit{variables}} \mid \ldots$$

# Büchi Arithmetic in base $k$

**Presburger Arithmetic** captures $\mathrm{Th}(\mathbb{N}, +, <)$ with **first-order** syntax

$$t, t' := \overbrace{x \mid y \mid \dots}^{\textit{variables}} \mid \overbrace{0 \mid 1 \mid \dots}^{\textit{constants}}$$

## Büchi Arithmetic in base $k$

**Presburger Arithmetic** captures $\mathrm{Th}(\mathbb{N}, +, <)$ with **first-order** syntax

$$t, t' := \overbrace{x \mid y \mid \ldots}^{\textit{variables}} \mid \overbrace{0 \mid 1 \mid \ldots}^{\textit{constants}} \mid t + t' \qquad \textit{terms}$$

# Büchi Arithmetic in base $k$

**Presburger Arithmetic** captures $\mathrm{Th}(\mathbb{N}, +, <)$ with **first-order** syntax

$$t, t' \coloneqq \overbrace{x \mid y \mid \ldots}^{\textit{variables}} \mid \overbrace{0 \mid 1 \mid \ldots}^{\textit{constants}} \mid t + t' \qquad \textit{terms}$$

$$\varphi, \psi \coloneqq t = t' \mid t < t' \mid \ldots \qquad \textit{formulas}$$

# Büchi Arithmetic in base $k$

**Presburger Arithmetic** captures $\mathrm{Th}(\mathbb{N}, +, <)$ with **first-order** syntax

$$t, t' \coloneqq \overbrace{x \mid y \mid \dots}^{\textit{variables}} \mid \overbrace{0 \mid 1 \mid \dots}^{\textit{constants}} \mid t + t' \qquad \textit{terms}$$

$$\varphi, \psi \coloneqq t = t' \mid t < t' \mid \dots \qquad \textit{formulas}$$

$$\mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \rightarrow \psi$$

# Büchi Arithmetic in base $k$

**Presburger Arithmetic** captures $\mathrm{Th}(\mathbb{N},+,<)$ with **first-order** syntax

$$t, t' \coloneqq \overbrace{x \mid y \mid \dots}^{\textit{variables}} \mid \overbrace{0 \mid 1 \mid \dots}^{\textit{constants}} \mid t + t' \qquad \textit{terms}$$

$$\varphi, \psi \coloneqq t = t' \mid t < t' \mid \dots \qquad \textit{formulas}$$

$$\mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \to \psi$$

$$\mid \forall x \varphi \mid \exists x \varphi$$

# Büchi Arithmetic in base $k$

**Presburger Arithmetic** captures $\mathrm{Th}(\mathbb{N}, +, <)$ with **first-order** syntax

$$t, t' \coloneqq \underbrace{\boxed{x} \mid \boxed{y} \mid \dots}_{\textit{variables}} \mid \underbrace{\boxed{0} \mid \boxed{1} \mid \dots}_{\textit{constants}} \mid \boxed{t + t'} \qquad \textit{terms}$$

$$\begin{aligned} \varphi, \psi \coloneqq\ & \boxed{t = t'} \mid \boxed{t < t'} \mid \dots & \textit{formulas} \\ & \mid \boxed{\neg\varphi} \mid \boxed{\varphi \wedge \psi} \mid \boxed{\varphi \vee \psi} \mid \boxed{\varphi \rightarrow \psi} \\ & \mid \boxed{\forall x \varphi} \mid \boxed{\exists x \varphi} \end{aligned}$$

**Walnut** implements the classical **Büchi-Bruyère** compilation of **predicates** $\varphi(x_1, \dots, x_n)$ into **deterministic finite automata** (DFA).

# Büchi Arithmetic in base $k$

**Presburger Arithmetic** captures $\mathrm{Th}(\mathbb{N}, +, <)$ with **first-order** syntax

$$
\begin{aligned}
t, t' &:= \underbrace{x \mid y \mid \dots}_{\textit{variables}} \mid \underbrace{0 \mid 1 \mid \dots}_{\textit{constants}} \mid t + t' && \textit{terms} \\
\varphi, \psi &:= t = t' \mid t < t' \mid \dots && \textit{formulas} \\
&\quad \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \rightarrow \psi \\
&\quad \mid \forall x \varphi \mid \exists x \varphi \mid \varphi_{\mathbf{dfa}}
\end{aligned}
$$

**Walnut** implements the classical **Büchi-Bruyère** compilation of **predicates** $\varphi(x_1, \dots, x_n)$ into **deterministic finite automata** (DFA).It also permits to **directly manipulate DFA** and use DFA-defined predicates into formulas.

**Büchi Arithmetic** characterizes the expressivity of **DFA** in base $k$.

# Synchronized sequences

Syncronized relations and sequences capture the expressive power of DFA with input in a given NS. **Walnut** manipulates them in regular NS.

***Definition (Carpi and Maggi 2001)*** A sequence $s : \mathbb{N} \to \mathbb{N}^m$ is **$\mathcal{S}$-synchronized** if the following language is regular:

$$\{\langle x, y_1, \dots, y_m\rangle \mid s(\mathrm{val}_{\mathcal{S}}(x)) = (\mathrm{val}_{\mathcal{S}}(y_1), \dots, \mathrm{val}_{\mathcal{S}}(y_m))\} \quad .$$

***Theorem*** Let $\mathbf{x} : \mathbb{N} \to \mathbb{Z}$ be a sequence of integers.
If $\mathbf{x}$ is $\mathcal{S}$-automatic then $\mathbf{x}$ is $\mathcal{S}$-synchronized.
If $\mathbf{x}$ is $\mathcal{S}$-synchronized then $\mathbf{x}$ is $\mathcal{S}$-regular.

# Long standing conjecture

***Conjecture (Parreau, Rigo, Rowland and Vandomme 2015)***
The abelian complexity of a $\mathcal{S}$-automatic sequence is $\mathcal{S}$-regular.

**Some examples:**

- The abelian complexity of the Thue-Morse sequence is **automatic**;
- The abelian complexity of the Rudin-Shapiro sequence is **regular**;
- The abelian complexity of the paperfolding sequence is **2-regular** and **unbounded**. **(Madill and Rampersad 2013)**

## Our contribution

**Part 1 (this talk)**

***Theorem*** Let $\mathbf{x}$ be a **uniformly factor-balanced** $\mathcal{S}$-automatic sequence. Its **abelian equivalence relation** is $\mathcal{S}$-synchronized and its **2D generalized abelian complexity** $(k, n) \mapsto \rho_{\mathbf{x}}^{k}(n)$ is $\mathcal{S}$-regular.

**Part 2 (Pierre on Wednesday)**

***Theorem*** The **abelian complexity** of the fixed point of a prolongable primitive substitution of ultimately **Pisot** type is an automatic sequence in its DTNS and the DFAO computing it is **effectively computable**.

***Theorem*** Previous theorem extends to the computation of the **generalized** abelian complexity under some **sliding-block code** condition.

## Uniform factor-balancedness

***Definition*** A sequence $\mathbf{x}$ is **uniformly factor-($C$-)balanced** for some uniform bound $C$ if $||u|_w - |v|_w| \leqslant C$ for all $u, v \in \mathcal{L}_n(\mathbf{x})$, for all $w \in \mathcal{L}(\mathbf{x})$, and for all $n \in \mathbb{N}$.

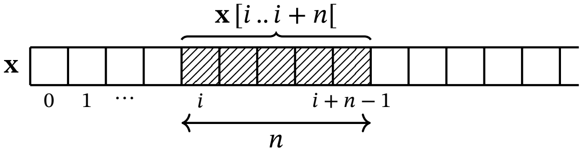
**How common is this property?**

For **Sturmian sequences**, in the general case, **(Fagnot and Vuillon 2002)** proved $(k,k)$**-balancedness**, which is unbounded.

Combining **(Fagnot and Vuillon 2002)** with **(Vandeth 2000)**, a Sturmian sequence of slope $\alpha \in (0,1)$ is **uniformly factor-balanced** when the continued fraction of $\alpha/(1-\alpha)$ has bounded partial quotients.

# Identifying factors and their relations

The **factors** of an automatic sequence are well captured by their appearance inside the sequence given by an **index** and a **length**.



Let $\mathbf{x}[i \mathinner{..} i+n[$ denote the length-$n$ factor of $\mathbf{x}$ starting at position $i$.

Let's formalize a few relations:

$$
\begin{aligned}
\mathrm{feq}_{\mathbf{x}} &= \{(i,j,n) \mid \mathbf{x}[i \mathinner{..} i+n[ \, = \mathbf{x}[j \mathinner{..} j+n[\} \\
\mathrm{abexeq}_{\mathbf{x}} &= \{(i,j,k,n) \mid \mathbf{x}[i \mathinner{..} i+n+k[ \, \sim_{=k} \mathbf{x}[j \mathinner{..} j+n+k[\} \\
\mathrm{abeq}_{\mathbf{x}} &= \{(i,j,k,n) \mid \mathbf{x}[i \mathinner{..} i+n[ \, \sim_{k} \mathbf{x}[j \mathinner{..} j+n[\}
\end{aligned}
$$

# Computing the balance function

The **balance function** compares **factor occurences**:

$$\Delta_{\mathbf{x}}(i, j_1, j_2, k, n) = \left|\mathbf{x}\,[j_1 \mathinner{..} j_1 + n + k[\right|_{\mathbf{x}[i..i+k[} - \left|\mathbf{x}\,[j_2 \mathinner{..} j_2 + n + k[\right|_{\mathbf{x}[i..i+k[}$$

***Lemma*** The balance function $\Delta_{\mathbf{x}}(i, j_1, j_2, k, n)$ of a **uniformly factor-balanced** $\mathcal{S}$-automatic sequence $\mathbf{x}$ is $\mathcal{S}$-automatic.

1. As $\mathrm{feq}_{\mathbf{x}}$ is $\mathcal{S}$-synchronized, $\mathrm{occ}_{\mathbf{x}}(i, j, k, n, u)$ that tests if $j \leq u \leq j + n$ and $\mathrm{feq}_{\mathbf{x}}(i, u, k)$ is also $\mathcal{S}$-synchronized;
2. Count accepting paths in a DFA computing $\mathrm{occ}_{\mathbf{x}}$ to obtain a $\mathcal{S}$-regular linear representation for $\left|\mathbf{x}\,[j \mathinner{..} j + n + k[\right|_{\mathbf{x}[i..i+k[}$;
3. Combine to get a $\mathcal{S}$-regular linear representation for $\Delta_{\mathbf{x}}(i, j_1, j_2, k, n)$;
4. Apply the **semigroup trick** to obtain $\mathcal{S}$-automaticity of $\Delta_{\mathbf{x}}$.

# Using balancedness to conclude

The **balancedness relation** identifies zeros the **balance function**:

$$\mathrm{bal}_{\mathbf{x}} = \{(i, j_1, j_2, k, n) \mid \Delta_{\mathbf{x}}(i, j_1, j_2, k, n) = 0\}$$

***Lemma*** If the balance function $\Delta_{\mathbf{x}}$ of a sequence $\mathbf{x}$ is $\mathcal{S}$-automatic then its balancedness relation $\mathrm{bal}_{\mathbf{x}}(i, j_1, j_2, k, n)$ is $\mathcal{S}$-synchronized.

***Lemma*** If the balancedness relation of a sequence $\mathbf{x}$ is $\mathcal{S}$-synchronized, then $\mathrm{abeq}_{\mathbf{x}}(i, j, k, n)$ and $\mathrm{abexeq}_{\mathbf{x}}(i, j, k, n)$ are $\mathcal{S}$-synchronized, and the 2D function $(k, n) \mapsto \rho^k_{\mathbf{x}}(n)$ is $\mathcal{S}$-regular.

1. Write first-order formulaes for $\mathrm{abexeq}_{\mathbf{x}}$ and $\mathrm{abeq}_{\mathbf{x}}$ using $\mathrm{bal}_{\mathbf{x}}$ and $\mathrm{feq}_{\mathbf{x}}$;
2. Write a first-order formula for first occurences of equivalent factors and derive a linear representation using accepting paths counting.

# Effective computation

***Theorem*** Let $\mathbf{x}$ be a **uniformly factor-balanced** $\mathcal{S}$-automatic sequence. Its **abelian equivalence relation** $\mathrm{abeq}_{\mathbf{x}}(i, j, k, n)$ is $\mathcal{S}$-synchronized and its **2D generalized abelian complexity** $(k, n) \mapsto \rho_{\mathbf{x}}^{k}(n)$ is $\mathcal{S}$-regular.

This approach is quite **naive** and direct. It is quite **computer-intensive** too!

We applied it to a limited number of sequences, proving a **tight bound** on their uniform factor-balancedness in the process:

- The Fibonacci sequence $\mathbf{f} = \varphi^{\omega}(0)$ $\qquad \varphi : 0 \mapsto 01,\ 1 \mapsto 0$
- The Pell sequence $\mathbf{p} = \pi^{\omega}(0)$ $\qquad \pi : 0 \mapsto 001,\ 1 \mapsto 0$

## Effective computation

***Theorem*** Let $\mathbf{x}$ be a **uniformly factor-balanced** $\mathcal{S}$-automatic sequence. Its **abelian equivalence relation** $\mathrm{abeq}_{\mathbf{x}}(i, j, k, n)$ is $\mathcal{S}$-synchronized and its **2D generalized abelian complexity** $(k, n) \mapsto \rho_{\mathbf{x}}^{k}(n)$ is $\mathcal{S}$-regular.

This approach is quite **naive** and direct. It is quite **computer-intensive** too!

We applied it to a limited number of sequences, proving a **tight bound** on their uniform factor-balancedness in the process:

- The Fibonacci sequence $\mathbf{f} = \varphi^{\omega}(0)$ $\qquad \varphi : 0 \mapsto 01,\ 1 \mapsto 0$
- The Pell sequence $\mathbf{p} = \pi^{\omega}(0)$ $\qquad \pi : 0 \mapsto 001,\ 1 \mapsto 0$
- Some $k$-uniform fixed point $\mathbf{b} = \beta^{\omega}(0)$ $\qquad \beta : 0 \mapsto 001,\ 1 \mapsto 010$

# Effective computation

***Theorem*** Let $\mathbf{x}$ be a **uniformly factor-balanced** $\mathcal{S}$-automatic sequence. Its **abelian equivalence relation** $\mathrm{abeq}_{\mathbf{x}}(i, j, k, n)$ is $\mathcal{S}$-synchronized and its **2D generalized abelian complexity** $(k, n) \mapsto \rho^k_{\mathbf{x}}(n)$ is $\mathcal{S}$-regular.

This approach is quite **naive** and direct. It is quite **computer-intensive** too!

We applied it to a limited number of sequences, proving a **tight bound** on their uniform factor-balancedness in the process:

- The Fibonacci sequence $\mathbf{f} = \varphi^\omega(0)$ $\quad \varphi : 0 \mapsto 01,\ 1 \mapsto 0$
- The Pell sequence $\mathbf{p} = \pi^\omega(0)$ $\quad \pi : 0 \mapsto 001,\ 1 \mapsto 0$
- Some $k$-uniform fixed point $\mathbf{b} = \beta^\omega(0)$ $\quad \beta : 0 \mapsto 001,\ 1 \mapsto 010$
- !!! The **Tribonacci sequence** $\mathbf{t} = \tau^\omega(0)$ $\quad \tau : 0 \mapsto 01,\ 1 \mapsto 02,\ 2 \mapsto 0$

## Implementation and experiments

Our **implementation** combines several tools:

1. `licofage` for DTNS and fixpoints `https://pypi.org/project/licofage/`
2. `Walnut` for first-order predicates `https://github.com/Walnut-Theorem-Prover`
3. `Awali` C++ weighted automata library `http://vaucanson-project.org/Awali/`

We ported the exact rational representation of `GMP` to `Awali` and wrote an **efficient `OpenMP` parallel reduction** to reduce regular sequences in parallel.

Experiments were conducted on a cluster where nodes with 96 OpenMP threads were available to parallelize the computations.

(two 24-core Intel Xeon Gold 6248R @3GHz processors with 256 GB of RAM)

# Computing the balance function

`Walnut` is first used to produce a DFA as follows (notice the `j2` trick):

```
def occ_tri "?msd_tri j1<=u & u<=j1+n & $feq_tri(i,u,k) & j2=j2":
```

The first C++ program applies path counting to obtain a linear representation for $\left|\mathbf{t}\left[j_1 .. j_1+n+k\right[\right|_{\mathbf{t}[i..i+k[}$.

Combining the linear representation with itself, permuting `j1` and `j2`, it computes a **reduced** linear representation for $\Delta_{\mathbf{t}}(i, j_1, j_2, k, n)$.

The semigroup trick is applied. When the sequence is **uniformly factor-balanced**, the program terminates with an automatic representation and a **tight bound** is obtained.

For the Tribonacci sequence $\mathbf{t}$, it took **16 hours** on our cluster, produced a **920 931-state** DFAO, proving that $\mathbf{t}$ is uniformly factor-**2**-balanced.

## Computing the equivalence relations

Walnut is used to capture the zeros:

```
def sametri "?msd_tri Dequitri[i][j1][j2][k][n] = @0":
```

For the Tribonacci sequence **t**, it took Walnut **75 seconds** to compute the corresponding **487 964-state** DFA.

Walnut is used to derive the 2D equivalence relations:

```
def abeqextri "?msd_tri Ai $sametri(i,j1,j2,k,n)":
def abeqtri "?msd_tri (n<k & $feq_tri(i,j,n))
    | (n>=k & $feq_tri(i,j,k-1) & $abeqextri(i,j,k,n-k))":
```

# Computing the 2D complexity function

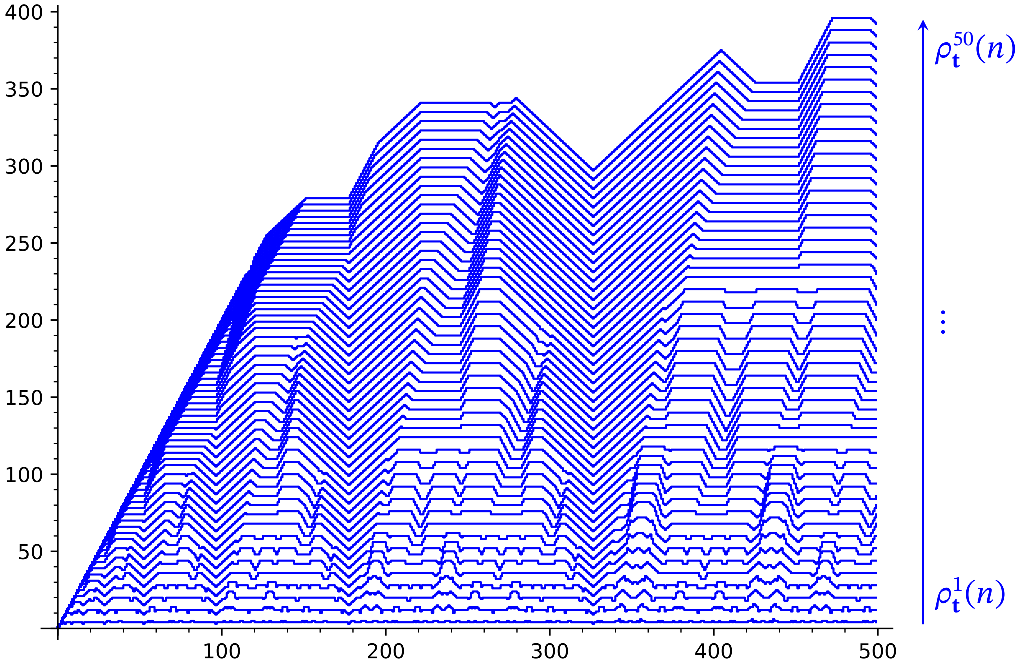
`Walnut` is used to identify the first occurence of each equivalence class:

```
def abfirsttri "?msd_tri k>0 & ~Ej j<i & $abeqtri(i,j,k,n)":
```

The second C++ program applies path counting to obtain a reduced linear representation for the 2D generalized abelian complexity function $\rho_{\mathbf{t}}^{k}(n)$.

For the Tribonacci sequence $\mathbf{t}$, we obtain a linear representation of dimension **264** with integer coefficients.

`https://github.com/nopid/abcomp/blob/main/section3/out/matri.sage`



$\rho_{\mathbf{t}}^{50}(n)$
$\vdots$
$\rho_{\mathbf{t}}^{1}(n)$
400
350
300
250
200
150
100
50
100
200
300
400
500

## Trust issues

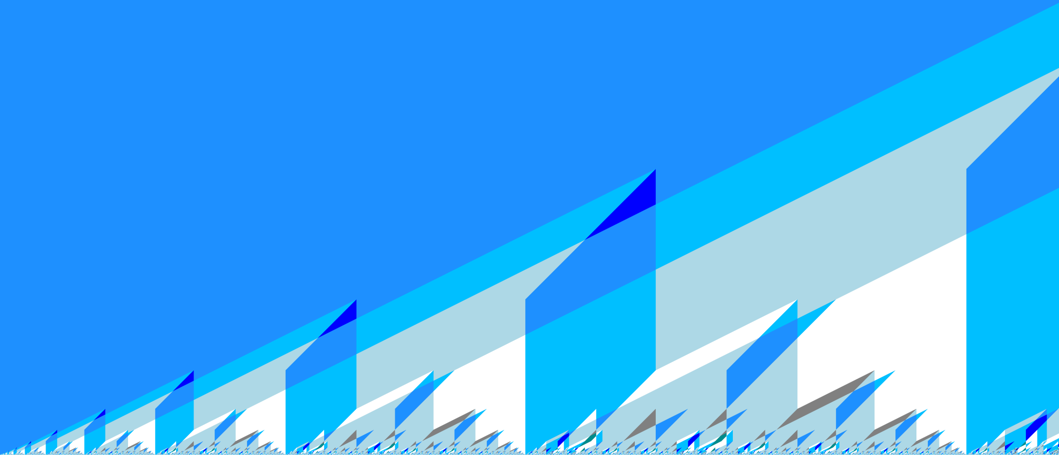
For the Tribonacci sequence $\mathbf{t}$, it took **16 hours** on our cluster, produced a **920 931-state** DFAO, proving that $\mathbf{t}$ is uniformly factor-**2**-balanced.

Can we trust this result? Maybe there is a **bug** 🐞 somewhere? Maybe the computation is incorrect?

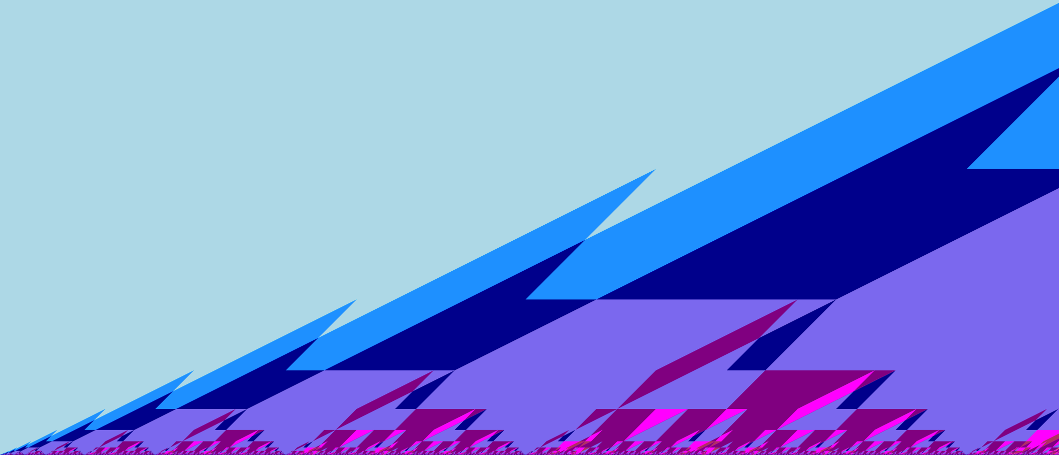
***Theorem*** The validity of the DFAO computing $\Delta_{\mathbf{x}}(i, j_1, j_2, k, n)$ can be **checked inductively** on $n$ with first-order predicates.

For the Tribonacci sequence $\mathbf{t}$, it took only **45 minutes** to check the **920 931-state** DFAO.

$\Delta_n \rho_{\mathbf{t}} : (n,k) \mapsto \rho_{\mathbf{t}}^k(n+1) - \rho_{\mathbf{t}}^k(n)$ is **automatic**



$\Delta_k \rho_{\mathbf{t}} : (n,k) \mapsto \rho_{\mathbf{t}}^{k+1}(n) - \rho_{\mathbf{t}}^{k}(n)$ is **automatic**



# Open Problems and Questions

More Problems to come at the end of **Pierre's talk**!

***Open Problem*** Find a **5 minutes blackboard proof** to replace the **16 hours** and **920 931-state** DFAO proof of the Tribonacci sequence uniform factor-**2**-balancedness!

***Open Problem*** Prove that the generalized abelian complexity of the Tribonacci sequence is **not synchronized**.

# Table of contents