# On stricter reachable repetitiveness measures

Gonzalo Navarro [1,2] and Cristian Urbina [1,2]

[1]Departament of Computer Science, University of Chile

[2]CeBiB — Center for Biotechnology and Bioengineering

SPIRE 2021
October 4-6th, 2021 - Lille, France

# Contents

# Motivation

- Text collections: Bioinformatics, Software engineering, etc.

# Motivation

- Text collections: Bioinformatics, Software engineering, etc.
- They are becoming huge, but highly repetitive, which enables handling them efficiently.

# Motivation

- Text collections: Bioinformatics, Software engineering, etc.
- They are becoming huge, but highly repetitive, which enables handling them efficiently.
- Studying their repetitiveness allow us determine how much can they be compressed.

# Motivation

- Text collections: Bioinformatics, Software engineering, etc.
- They are becoming huge, but highly repetitive, which enables handling them efficiently.
- Studying their repetitiveness allow us determine how much can they be compressed.
- Most repetitiveness measures are based on existing compressors.

# Motivation

- Text collections: Bioinformatics, Software engineering, etc.
- They are becoming huge, but highly repetitive, which enables handling them efficiently.
- Studying their repetitiveness allow us determine how much can they be compressed.
- Most repetitiveness measures are based on existing compressors.
- Theoretical lower bounds like $\delta$ (based on substring complexity) and $\gamma$ (size of the smallest string attractor) have been proposed.

# Motivation

- Text collections: Bioinformatics, Software engineering, etc.
- They are becoming huge, but highly repetitive, which enables handling them efficiently.
- Studying their repetitiveness allow us determine how much can they be compressed.
- Most repetitiveness measures are based on existing compressors.
- Theoretical lower bounds like $\delta$ (based on substring complexity) and $\gamma$ (size of the smallest string attractor) have been proposed.
- $\gamma$: NP-hard to compute. Reachable? We don't know yet.

# Motivation

- Text collections: Bioinformatics, Software engineering, etc.
- They are becoming huge, but highly repetitive, which enables handling them efficiently.
- Studying their repetitiveness allow us determine how much can they be compressed.
- Most repetitiveness measures are based on existing compressors.
- Theoretical lower bounds like $\delta$ (based on substring complexity) and $\gamma$ (size of the smallest string attractor) have been proposed.
- $\gamma$: NP-hard to compute. Reachable? We don't know yet.
- $\delta \leq \gamma$: Nice properties, leads to tight lower bounds, apparently the best measure so far.

# Motivation

- Text collections: Bioinformatics, Software engineering, etc.
- They are becoming huge, but highly repetitive, which enables handling them efficiently.
- Studying their repetitiveness allow us determine how much can they be compressed.
- Most repetitiveness measures are based on existing compressors.
- Theoretical lower bounds like $\delta$ (based on substring complexity) and $\gamma$ (size of the smallest string attractor) have been proposed.
- $\gamma$: NP-hard to compute. Reachable? We don't know yet.
- $\delta \leq \gamma$: Nice properties, leads to tight lower bounds, apparently the best measure so far.
- But... is it?

# Motivation

- Thue-Morse words:

# Motivation

- Thue-Morse words:
  - Defined inductively as $T_0 = 0$ and $T_{n+1} = T_n \overline{T_n}$.

# Motivation

- Thue-Morse words:
  - Defined inductively as $T_0 = 0$ and $T_{n+1} = T_n \overline{T_n}$.
  - Or as $T_0 = 0$, $T_n = h^n(0)$ with $h : 0 \to 01, 1 \to 10$.

# Motivation

- Thue-Morse words:
  - Defined inductively as $T_0 = 0$ and $T_{n+1} = T_n \overline{T_n}$.
  - Or as $T_0 = 0$, $T_n = h^n(0)$ with $h : 0 \to 01, 1 \to 10$.
- Recent results:

# Motivation

- Thue-Morse words:
  - Defined inductively as $T_0 = 0$ and $T_{n+1} = T_n \overline{T_n}$.
  - Or as $T_0 = 0$, $T_n = h^n(0)$ with $h : 0 \to 01, 1 \to 10$.
- Recent results:
  - Kutsukake et al. (2020) showed that $\gamma = \Theta(1)$ on Thue-Morse words.

# Motivation

- Thue-Morse words:
  - Defined inductively as $T_0 = 0$ and $T_{n+1} = T_n \overline{T_n}$.
  - Or as $T_0 = 0$, $T_n = h^n(0)$ with $h : 0 \to 01, 1 \to 10$.
- Recent results:
  - Kutsukake et al. (2020) showed that $\gamma = \Theta(1)$ on Thue-Morse words.
  - Bannai et al. (2021) showed that $b = \Theta(\log n)$ on Thue-Morse words.

# Motivation

- Thue-Morse words:
  - Defined inductively as $T_0 = 0$ and $T_{n+1} = T_n \overline{T_n}$.
  - Or as $T_0 = 0$, $T_n = h^n(0)$ with $h : 0 \to 01, 1 \to 10$.
- Recent results:
  - Kutsukake et al. (2020) showed that $\gamma = \Theta(1)$ on Thue-Morse words.
  - Bannai et al. (2021) showed that $b = \Theta(\log n)$ on Thue-Morse words.
- $b$ is the best reachable measure that can be achieved with copy-paste mechanism.

# Motivation

- Thue-Morse words:
  - Defined inductively as $T_0 = 0$ and $T_{n+1} = T_n \overline{T_n}$.
  - Or as $T_0 = 0$, $T_n = h^n(0)$ with $h : 0 \to 01, 1 \to 10$.
- Recent results:
  - Kutsukake et al. (2020) showed that $\gamma = \Theta(1)$ on Thue-Morse words.
  - Bannai et al. (2021) showed that $b = \Theta(\log n)$ on Thue-Morse words.
- $b$ is the best reachable measure that can be achieved with copy-paste mechanism.
- So $b$ is a good characterization of repetitive strings.

# Motivation

- Thue-Morse words:
  - Defined inductively as $T_0 = 0$ and $T_{n+1} = T_n \overline{T_n}$.
  - Or as $T_0 = 0$, $T_n = h^n(0)$ with $h : 0 \to 01, 1 \to 10$.
- Recent results:
  - Kutsukake et al. (2020) showed that $\gamma = \Theta(1)$ on Thue-Morse words.
  - Bannai et al. (2021) showed that $b = \Theta(\log n)$ on Thue-Morse words.
- $b$ is the best reachable measure that can be achieved with copy-paste mechanism.
- So $b$ is a good characterization of repetitive strings.
- Is it the only feature to exploit?

# Motivation

- Thue-Morse words:
  - Defined inductively as $T_0 = 0$ and $T_{n+1} = T_n \overline{T_n}$.
  - Or as $T_0 = 0$, $T_n = h^n(0)$ with $h : 0 \to 01, 1 \to 10$.
- Recent results:
  - Kutsukake et al. (2020) showed that $\gamma = \Theta(1)$ on Thue-Morse words.
  - Bannai et al. (2021) showed that $b = \Theta(\log n)$ on Thue-Morse words.
- $b$ is the best reachable measure that can be achieved with copy-paste mechanism.
- So $b$ is a good characterization of repetitive strings.
- Is it the only feature to exploit?
- We exploit **structural repetitiveness**, also known as self-similarity.

# Motivation

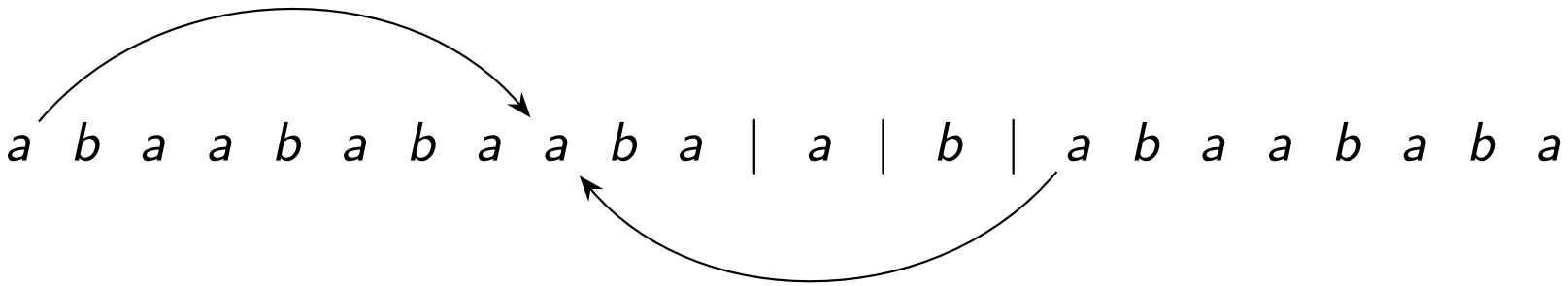
- Thue-Morse words:
  - Defined inductively as $T_0 = 0$ and $T_{n+1} = T_n \overline{T_n}$.
  - Or as $T_0 = 0$, $T_n = h^n(0)$ with $h : 0 \to 01, 1 \to 10$.
- Recent results:
  - Kutsukake et al. (2020) showed that $\gamma = \Theta(1)$ on Thue-Morse words.
  - Bannai et al. (2021) showed that $b = \Theta(\log n)$ on Thue-Morse words.
- $b$ is the best reachable measure that can be achieved with copy-paste mechanism.
- So $b$ is a good characterization of repetitive strings.
- Is it the only feature to exploit?
- We exploit **structural repetitiveness**, also known as self-similarity.
- This form of repetitiveness is not captured by the current measures.

# Motivation

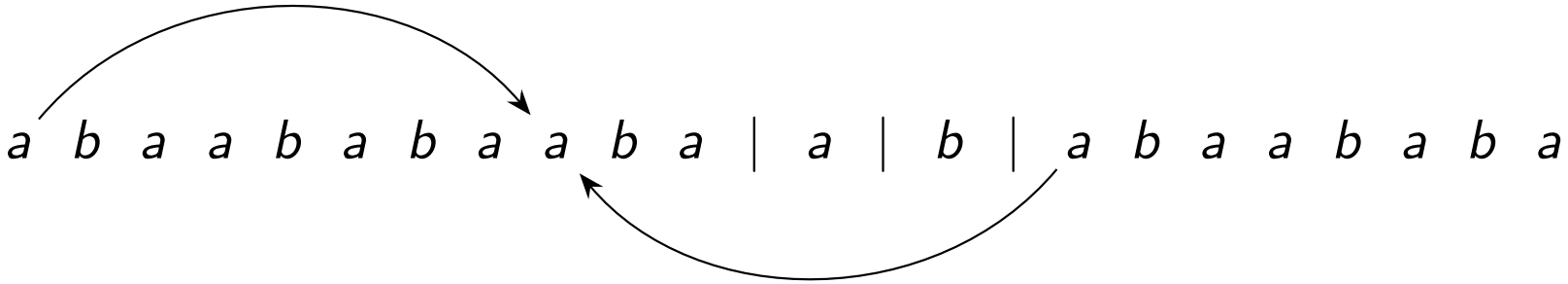
- Thue-Morse words:
  - Defined inductively as $T_0 = 0$ and $T_{n+1} = T_n \overline{T_n}$.
  - Or as $T_0 = 0$, $T_n = h^n(0)$ with $h : 0 \to 01, 1 \to 10$.
- Recent results:
  - Kutsukake et al. (2020) showed that $\gamma = \Theta(1)$ on Thue-Morse words.
  - Bannai et al. (2021) showed that $b = \Theta(\log n)$ on Thue-Morse words.
- $b$ is the best reachable measure that can be achieved with copy-paste mechanism.
- So $b$ is a good characterization of repetitive strings.
- Is it the only feature to exploit?
- We exploit **structural repetitiveness**, also known as self-similarity.
- This form of repetitiveness is not captured by the current measures.
- For Thue-Morse words, for example, we obtain a reachable representation of constant size.

# Contents

# Bidirectional macro schemes

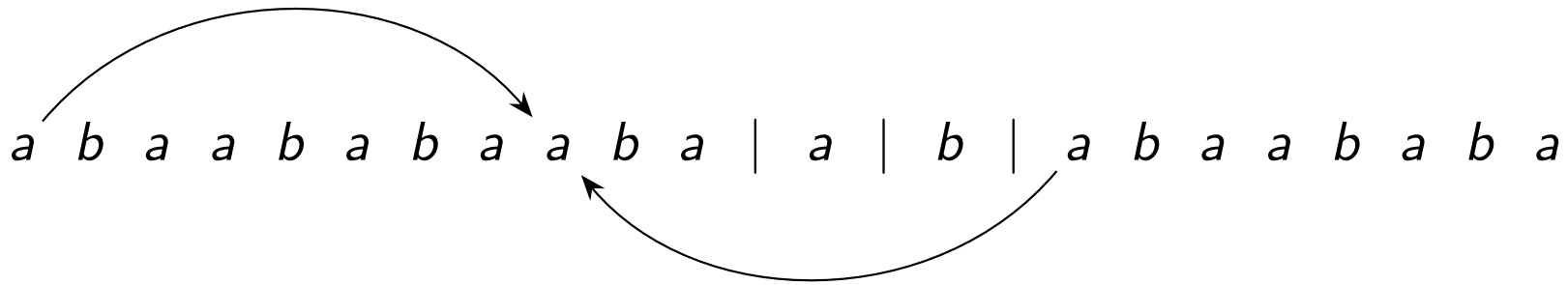


# Bidirectional macro schemes



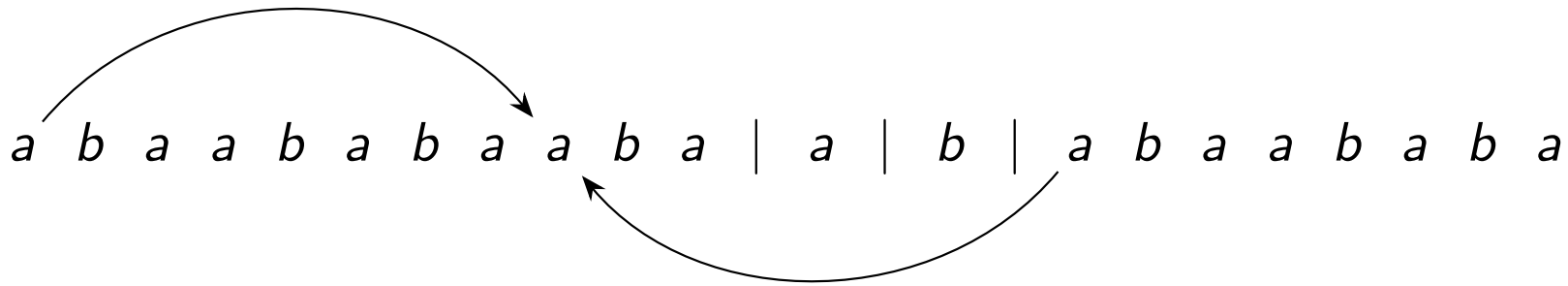

- A BMS for $w$ is a mapping from phrases to sources avoiding cycles.

# Bidirectional macro schemes



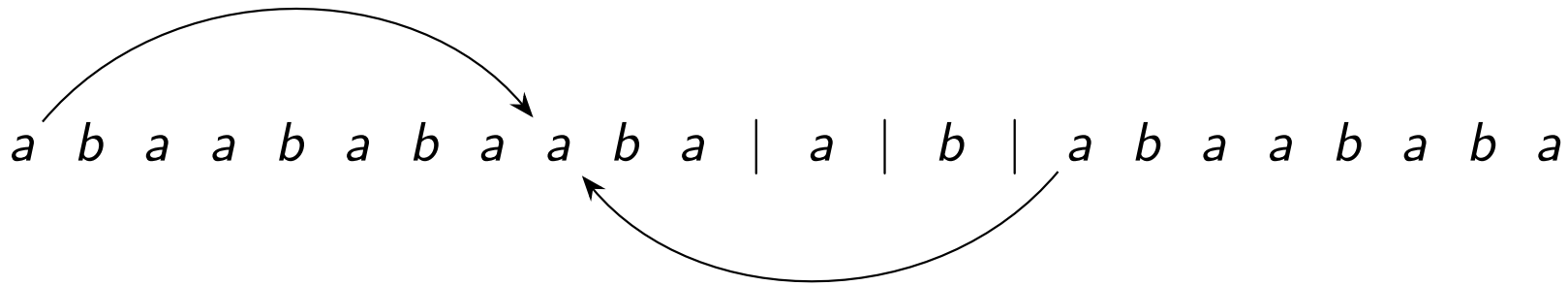

- A BMS for $w$ is a mapping from phrases to sources avoiding cycles.
- In the example, $B = (abaababaaba, 9), (a, \perp), (b, \perp), (abaababa, 9)$ is a minimal BMS of size 4.

# Bidirectional macro schemes




- A BMS for $w$ is a mapping from phrases to sources avoiding cycles.
- In the example, $B = (abaababaaba, 9), (a, \perp), (b, \perp), (abaababa, 9)$ is a minimal BMS of size 4.
- We store a pair of $O(\log n)$ bits per phrase, containing length and source, to recover $w$. In the example, $(11, 9), (1, a), (1, b), (8, 9)$.

# Bidirectional macro schemes




- A BMS for $w$ is a mapping from phrases to sources avoiding cycles.
- In the example, $B = (abaababaaba, 9), (a, \perp), (b, \perp), (abaababa, 9)$ is a minimal BMS of size 4.
- We store a pair of $O(\log n)$ bits per phrase, containing length and source, to recover $w$. In the example, $(11, 9), (1, a), (1, b), (8, 9)$.
- The size $b(w)$ of the smallest of them is NP-hard to compute.

# Macro systems - Definition

A *macro system* is a *context free grammar* where the rules are extended with *extraction symbols*:

# Macro systems - Definition

A *macro system* is a *context free grammar* where the rules are extended with *extraction symbols*:

$$R : V \to (V \cup \Sigma \cup \{A[i,j] \mid A \in V, i,j \in \mathbb{N}\})^*,$$

# Macro systems - Definition

A *macro system* is a *context free grammar* where the rules are extended with *extraction symbols*:

$$R : V \to (V \cup \Sigma \cup \{A[i,j] \mid A \in V, i,j \in \mathbb{N}\})^*,$$

- The size of a macro system is the sum of the right hand side of its rules.

# Macro systems - Definition

A *macro system* is a *context free grammar* where the rules are extended with *extraction symbols*:

$$R : V \to (V \cup \Sigma \cup \{A[i,j] \mid A \in V, i,j \in \mathbb{N}\})^*,$$

- The size of a macro system is the sum of the right hand side of its rules.
- For a macro system to be valid, the expansion of its initial symbol must be deterministic and also must not loop. $exp(S) = w$ is the string generated by the macro system.

# Macro systems - Definition

A *macro system* is a *context free grammar* where the rules are extended with *extraction symbols*:

$$R : V \to (V \cup \Sigma \cup \{A[i,j] \mid A \in V, i,j \in \mathbb{N}\})^*,$$

- The size of a macro system is the sum of the right hand side of its rules.
- For a macro system to be valid, the expansion of its initial symbol must be deterministic and also must not loop. $exp(S) = w$ is the string generated by the macro system.
- The extraction symbols $A[i,j]$ mean to extract the explicit symbols of $exp(A)[i,j]$.

# Macro systems - Definition

A *macro system* is a *context free grammar* where the rules are extended with *extraction symbols*:

$$R : V \to (V \cup \Sigma \cup \{A[i,j] \mid A \in V, i,j \in \mathbb{N}\})^*,$$

- The size of a macro system is the sum of the right hand side of its rules.
- For a macro system to be valid, the expansion of its initial symbol must be deterministic and also must not loop. $exp(S) = w$ is the string generated by the macro system.
- The extraction symbols $A[i,j]$ mean to extract the explicit symbols of $exp(A)[i,j]$.
- We denote by $m$ the size of the smallest macro system generating $w$.

# Macro systems - Results

**Theorem 1**

*It always holds that* $m = O(b)$.

# Contents

# L-systems - Definition

An *L-system* is a tuple $L = (V, R, S, \tau, d, n)$, where:

# L-systems - Definition

An *L-system* is a tuple $L = (V, R, S, \tau, d, n)$, where:

- $V$ is a finite set of symbols called *variables*.

# L-systems - Definition

An *L-system* is a tuple $L = (V, R, S, \tau, d, n)$, where:

- $V$ is a finite set of symbols called *variables*.
- $R : V \to V^+$ is the set of *rules*.

# L-systems - Definition

An *L-system* is a tuple $L = (V, R, S, \tau, d, n)$, where:

- $V$ is a finite set of symbols called *variables*.
- $R : V \to V^+$ is the set of *rules*.
- $S \in V^*$ is a string of variables called the *axiom*.

# L-systems - Definition

An *L-system* is a tuple $L = (V, R, S, \tau, d, n)$, where:

- $V$ is a finite set of symbols called *variables*.
- $R : V \to V^+$ is the set of *rules*.
- $S \in V^*$ is a string of variables called the *axiom*.
- $\tau : V \to V$ is a coding.

# L-systems - Definition

An *L-system* is a tuple $L = (V, R, S, \tau, d, n)$, where:

- $V$ is a finite set of symbols called *variables*.
- $R : V \to V^+$ is the set of *rules*.
- $S \in V^*$ is a string of variables called the *axiom*.
- $\tau : V \to V$ is a coding.
- $d \in \mathbb{N}$ is the level where to stop.

# L-systems - Definition

An *L-system* is a tuple $L = (V, R, S, \tau, d, n)$, where:

- $V$ is a finite set of symbols called *variables*.
- $R : V \to V^+$ is the set of *rules*.
- $S \in V^*$ is a string of variables called the *axiom*.
- $\tau : V \to V$ is a coding.
- $d \in \mathbb{N}$ is the level where to stop.
- $n \in \mathbb{N}$ is the length of the string to generate.

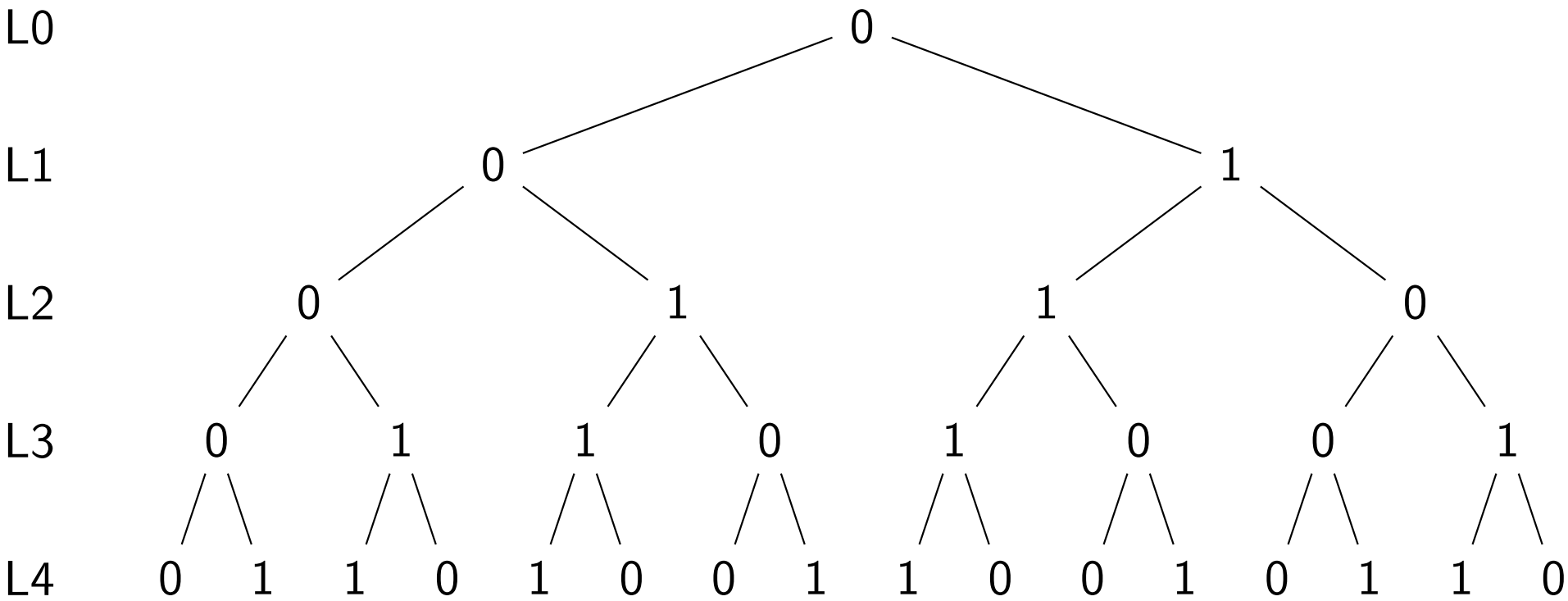
# L-systems - Definition

An *L-system* is a tuple $L = (V, R, S, \tau, d, n)$, where:

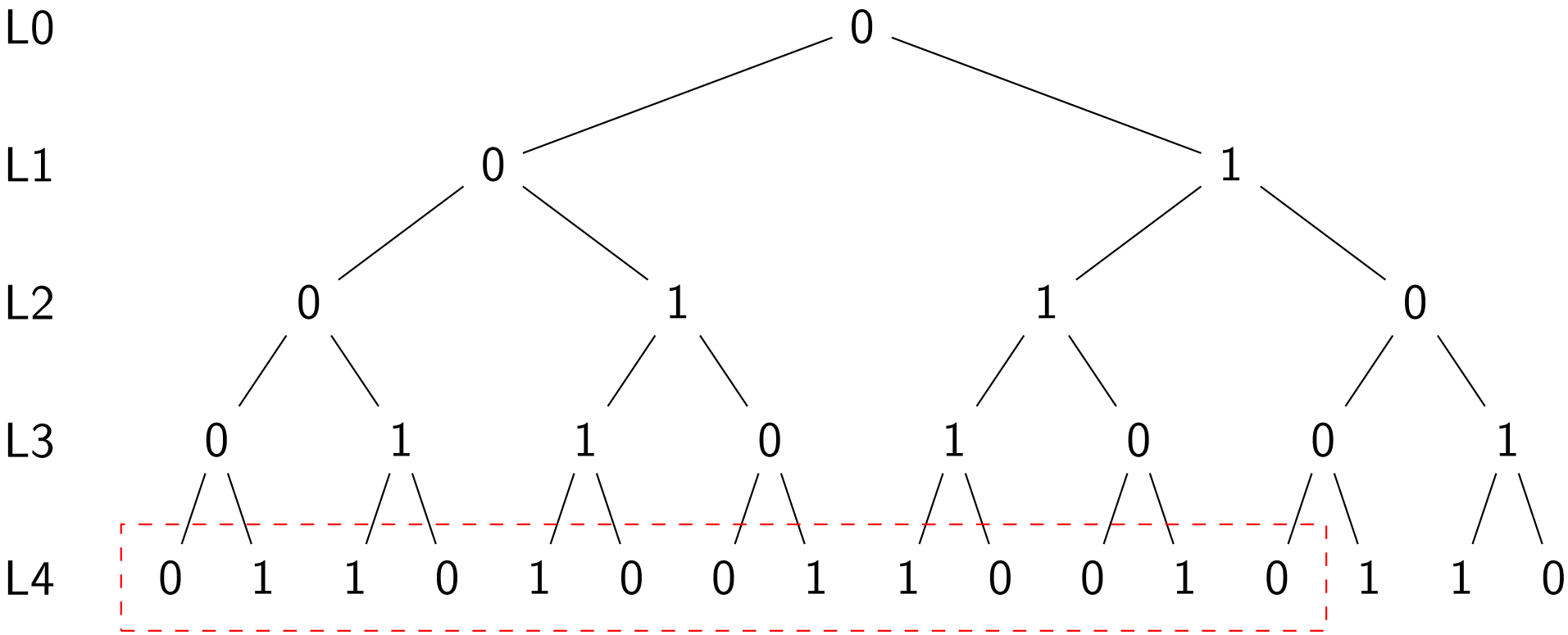
- $V$ is a finite set of symbols called *variables*.
- $R : V \to V^+$ is the set of *rules*.
- $S \in V^*$ is a string of variables called the *axiom*.
- $\tau : V \to V$ is a coding.
- $d \in \mathbb{N}$ is the level where to stop.
- $n \in \mathbb{N}$ is the length of the string to generate.

The *size* of an L-system is $|S| + \sum_{A \in V} |R(A)|$. We call $\ell$ the size of the smallest L-system generating a string $w$.

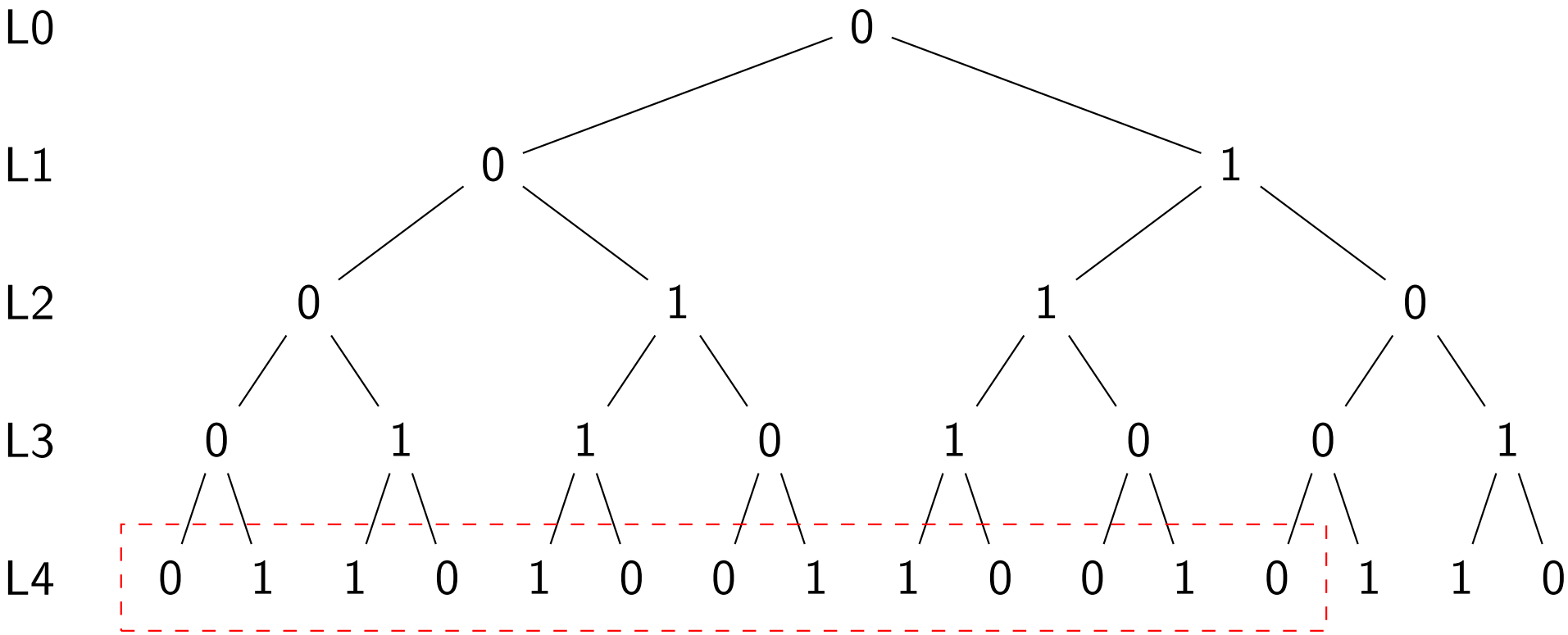
# L-systems - Example: Thue-Morse words



# L-systems - Example: Thue-Morse words



# L-systems - Example: Thue-Morse words




$$V = \{0, 1\}, R = \{0 \to 01, 1 \to 10\}, S = 0$$
$$d = 4, n = 13, \tau = \{0 \to 0, 1 \to 1\}$$

# L-systems - Results

**Theorem 2**

*There exist string families where $\delta = \Omega(\ell \log n)$, where $\delta$ is the maximum of the substring complexity divided by length, for each length.*

# L-systems - Results

## Theorem 2

*There exist string families where $\delta = \Omega(\ell \log n)$, where $\delta$ is the maximum of the substring complexity divided by length, for each length.*

## Proof.

The L-system $L_d = (V, R, S, \tau, d, n)$ where $V = \{0, 1\}$, $S = 0$, $R(0) = 001$, $R(1) = 1$, $\tau = id$, and $n = 2^{d+1} - 1$, trivially has constant size as $d$ (and $n$) grows. The first strings of the family generated by this system by growing $d$, are 0, 001, 0010011, 001001100100111, and so on.
For $d \geq 4$, in $exp(L_d)$ there exist $d^2/8 + d/4$ distinct substrings of length $d$, and $\delta(exp(L_d)) \geq (d^2/8 + d/4)/d = d/8 + 1/4$.
Hence, on the family $\{exp(L_d) \mid d \geq 0\}$, it holds that $\delta = \Omega(\ell \log n)$. □

# L-systems - Results

**Theorem 3**

*There exist string families where* $\ell = \Omega(\delta \log n)$.

# L-systems - Results

**Theorem 3**

*There exist string families where* $\ell = \Omega(\delta \log n)$.

**Proof.**

Kociumaka et al. showed a family of strings with $\delta = O(1)$ that needs $\Omega(\log n) = \Omega(\delta \log n)$ space, to be represented with any method. On the other hand, an L-system of size $\ell$ is described in $O(\ell)$ space. Therefore $\ell = \Omega(\delta \log n)$ on this family. □

# L-systems - Results

**Theorem 4**

*It always holds that $\ell = O(g)$ where $g(w)$ is the size of the smallest context free grammar generating only $w$.*

# L-systems - Results

**Theorem 4**

*It always holds that $\ell = O(g)$ where $g(w)$ is the size of the smallest context free grammar generating only $w$.*

**Proof.**

Consider the smallest grammar $G = (V, \Sigma, R, S)$ of size $g$ and height $h$ generating the string $w[1, n]$. We can easily construct an L-system of size $O(g)$ simulating $G$ by defining rules $a \to a$ for the terminals, keeping the rules for the variables of the grammar, letting $R(S)$ be the axiom, and $d = h$. □

# L-systems - Results

**Theorem 5**

*For any L-system $L = (V, R, S, \tau, d, n)$ of size $\ell$ generating $w[1, n]$, there is a context-free grammar of size $(d + 1)\ell$ generating $w$. If the morphism represented by $R$ is expanding, then the grammar is of size $O(\ell \log n)$.*

# L-systems - Results

**Theorem 5**

*For any L-system $L = (V, R, S, \tau, d, n)$ of size $\ell$ generating $w[1, n]$, there is a context-free grammar of size $(d + 1)\ell$ generating $w$. If the morphism represented by $R$ is expanding, then the grammar is of size $O(\ell \log n)$.*

**Proof.**

We create a variant of each terminal sub indicated by the level where it is used in the parse tree. This allows us control the level where to stop but multiplies the size by the expansion height. □

# L-systems - Results

**Theorem 6**

*Let $w \in \Sigma^*$, $\phi : \Sigma^* \to \Sigma^*$ be a non-erasing automorphism, and let $\tau : \Sigma \to \Sigma$ be a coding. Then $\ell = \Theta(1)$ on the family $\{\tau(\phi^d(w)) \mid d > 0\}$.*

# L-systems - Results

**Theorem 6**

*Let $w \in \Sigma^*$, $\phi : \Sigma^* \to \Sigma^*$ be a non-erasing automorphism, and let $\tau : \Sigma \to \Sigma$ be a coding. Then $\ell = \Theta(1)$ on the family $\{\tau(\phi^d(w)) \mid d > 0\}$.*

**Proof.**

We can easily simulate the automorphism $\phi$ on the L-system $L = (\Sigma, R, w, \tau, d, n)$ of fixed size, with $R(a) = \phi(a)$ and $n = |\tau(\phi^d(w))|$. This system generates $\tau(\phi^d(w))$, and as $d$ grows, it does not change its size. □

# L-systems - Summary

The measure $\ell$ is:

# L-systems - Summary

The measure $\ell$ is:

- computable.

# L-systems - Summary

The measure $\ell$ is:

- computable.
- reachable.

# L-systems - Summary

The measure $\ell$ is:

- computable.
- reachable.
- sometimes $o(\delta)$.

# L-systems - Summary

The measure $\ell$ is:

- computable.
- reachable.
- sometimes $o(\delta)$.
- decompressible in polynomial time.

# L-systems - Summary

The measure $\ell$ is:

- computable.
- reachable.
- sometimes $o(\delta)$.
- decompressible in polynomial time.

On the other hand, it is probably NP-hard to compute $\ell$.

# Contents

# NU-systems - Definition

A *NU-system* is a tuple $N = (V, R, S, \tau, d, n)$, which works on the same way that an L-system, but with the addition of *extractions rules*, that is, $R : V \to (V \cup E)^+$ with $E = \{A(l)[i, j] \mid A \in V, l, i, j \in \mathbb{N}\}$.

# NU-systems - Definition

A *NU-system* is a tuple $N = (V, R, S, \tau, d, n)$, which works on the same way that an L-system, but with the addition of *extractions rules*, that is, $R : V \to (V \cup E)^+$ with $E = \{A(l)[i, j] \mid A \in V, l, i, j \in \mathbb{N}\}$.

- Semantics for $A(l)[i, j]$: expand $A$ for $l$ levels and then extract $\tau(A_l[i, j])$.

# NU-systems - Definition

A *NU-system* is a tuple $N = (V, R, S, \tau, d, n)$, which works on the same way that an L-system, but with the addition of *extractions rules*, that is, $R : V \to (V \cup E)^+$ with $E = \{A(l)[i,j] \mid A \in V, l, i, j \in \mathbb{N}\}$.

- Semantics for $A(l)[i,j]$: expand $A$ for $l$ levels and then extract $\tau(A_l[i,j])$.
- $\nu(w)$ is the size of the smallest NU-system for $w$.

# NU-systems - Example

Consider a NU-system with rules $a \to a$, $b \to b$, $A \to ab$, and $S \to A\ S(2)[1,4]$, $d = 2$, $n = 6$, and $\tau = id$. The derivation is then generated as follows:

# NU-systems - Example

Consider a NU-system with rules $a \to a$, $b \to b$, $A \to ab$, and $S \to A\ S(2)[1,4]$, $d = 2$, $n = 6$, and $\tau = id$. The derivation is then generated as follows:

$$
\begin{array}{lcll}
L_0 & = & S & \longrightarrow \quad A\ S(2)[1]\ S(2)[2]\ S(2)[3]\ S(2)[4] \\
 & & & \quad\ A\ A(1)[1]\ A(1)[2]\ (S(2)[1])(1)[1]\ (S(2)[2])(1)[1] \\
 & & & \quad\ A\ a\ b\ (A(1)[1])(1)[1]\ (A(1)[2])(1)[1] \\
L_1 & = & A\ a\ b\ a\ b & \longleftarrow \quad A\ a\ b\ a(1)[1]\ b(1)[1] \\
L_2 & = & a\ b\ a\ b\ a\ b. &
\end{array}
$$

# NU-systems - Example

Consider a NU-system with rules $a \to a$, $b \to b$, $A \to ab$, and $S \to A\ S(2)[1,4]$, $d = 2$, $n = 6$, and $\tau = id$. The derivation is then generated as follows:

$$
\begin{array}{lcll}
L_0 & = & S & \longrightarrow \quad A\ S(2)[1]\ S(2)[2]\ S(2)[3]\ S(2)[4] \\
 & & & \quad\ A\ A(1)[1]\ A(1)[2]\ (S(2)[1])(1)[1]\ (S(2)[2])(1)[1] \\
 & & & \quad\ A\ a\ b\ (A(1)[1])(1)[1]\ (A(1)[2])(1)[1] \\
L_1 & = & A\ a\ b\ a\ b & \longleftarrow \quad A\ a\ b\ a(1)[1]\ b(1)[1] \\
L_2 & = & a\ b\ a\ b\ a\ b. &
\end{array}
$$

The number of extraction symbols is bounded by $|V| \cdot max(l) \cdot n$, so it is computable if the derivation of a NU-system eventually ends.

# NU-systems - Results

**Theorem 7**

*It always holds that* $\nu = O(\min(\ell, m))$.

# NU-systems - Results

**Theorem 7**

*It always holds that* $\nu = O(\min(\ell, m))$.

**Proof.**

It is trivial that $\nu \leq \ell$ because L-systems are a particular case of NU-systems.

With respect to $m$, we do the same that we did for simulating grammars with L-systems. We handle the extractions rules of the form $A \to B[i, j]$ by replacing them with rules $A \to B(h)[i, j]$, where $h$ is the height of the macro system. □

# NU-systems - Results

**Theorem 8**

*Let $N_1$ and $N_2$ be NU-systems generating $w_1$ and $w_2$, respectively. Then there are NU-systems of size $O(size(N_1) + size(N_2))$ that generate $w_1 \cdot w_2$ and the* composition *of $N_1 \circ N_2$ ($N_2$ with axiom $w_1$).*

# NU-systems - Results

**Theorem 8**

*Let $N_1$ and $N_2$ be NU-systems generating $w_1$ and $w_2$, respectively. Then there are NU-systems of size $O(size(N_1) + size(N_2))$ that generate $w_1 \cdot w_2$ and the* composition *of $N_1 \circ N_2$ ($N_2$ with axiom $w_1$).*

**Proof.**

We can use the the NU-systems $N_1$ and $N_2$ to construct new NU-systems satisfying these claims. □

# NU-systems - Summary

The measure $\nu$ is:

# NU-systems - Summary

The measure $\nu$ is:

- computable.

# NU-systems - Summary

The measure $\nu$ is:

- computable.
- reachable.

# NU-systems - Summary

The measure $\nu$ is:

- computable.
- reachable.
- always in $O(b)$.

# NU-systems - Summary

The measure $\nu$ is:

- computable.
- reachable.
- always in $O(b)$.
- sometimes $o(\delta)$.

# NU-systems - Summary

The measure $\nu$ is:

- computable.
- reachable.
- always in $O(b)$.
- sometimes $o(\delta)$.

On the other hand $\nu$ is probably NP-hard and does not have a guaranteed polynomial decompression time.

# Contents

# Open questions

- Is the size $\ell$ of the smallest L-system upper bounded by the size $c$ of the smallest collage system?

# Open questions

- Is the size $\ell$ of the smallest L-system upper bounded by the size $c$ of the smallest collage system? Or at least by the size $g_{rl}$ of the smallest RLSLP?

# Open questions

- Is the size $\ell$ of the smallest L-system upper bounded by the size $c$ of the smallest collage system? Or at least by the size $g_{rl}$ of the smallest RLSLP?
- Is the size $\nu$ of the smallest NU-system upper bounded by the size $\gamma$ of the smallest string attractor?

# Open questions

- Is the size $\ell$ of the smallest L-system upper bounded by the size $c$ of the smallest collage system? Or at least by the size $g_{rl}$ of the smallest RLSLP?
- Is the size $\nu$ of the smallest NU-system upper bounded by the size $\gamma$ of the smallest string attractor? Or at least is $\nu = o(\gamma \log n)$?

# Open questions

- Is the size $\ell$ of the smallest L-system upper bounded by the size $c$ of the smallest collage system? Or at least by the size $g_{rl}$ of the smallest RLSLP?
- Is the size $\nu$ of the smallest NU-system upper bounded by the size $\gamma$ of the smallest string attractor? Or at least is $\nu = o(\gamma \log n)$?
- Is the size $g$ of the smallest SLP upper bounded by $\ell \log n$?

# Open questions

- Is the size $\ell$ of the smallest L-system upper bounded by the size $c$ of the smallest collage system? Or at least by the size $g_{rl}$ of the smallest RLSLP?
- Is the size $\nu$ of the smallest NU-system upper bounded by the size $\gamma$ of the smallest string attractor? Or at least is $\nu = o(\gamma \log n)$?
- Is the size $g$ of the smallest SLP upper bounded by $\ell \log n$?
- Can we find a good theoretical lower bound for self-similarity schemes?

# Closing

Thank you very much for your attention.