Uniform generation of infinite concurrent runs

The case of trace monoids

Samy Abbes[1] & Vincent Jugé[2]

1: Université Paris Cité (IRIF) — 2: Université Gustave Eiffel (LIGM)

13/06/2022

Contents

Heaps of pieces and trace monoids

Heap of pieces[2]

- Pieces:

Trace monoid[1]

- Alphabet:

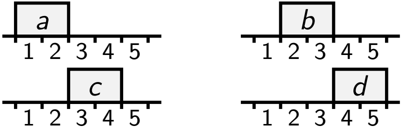
$$\Sigma = \{a, b, c, d\}$$

Heaps of pieces and trace monoids

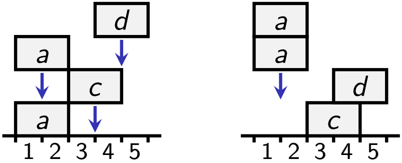
Heap of pieces[2]

- Pieces:

- Horizontal layout:



- Vertical heaps:



Trace monoid[1]

- Alphabet:

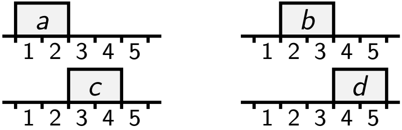
$$\Sigma = \{a, b, c, d\}$$

Heaps of pieces and trace monoids

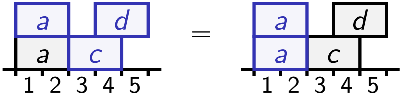
Heap of pieces[2]

- Pieces:

- Horizontal layout:



- Vertical heaps:



Trace monoid[1]

- Alphabet:

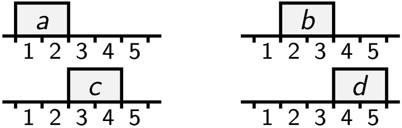
$$\Sigma = \{a, b, c, d\}$$

Heaps of pieces and trace monoids

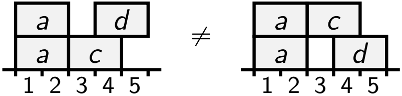
Heap of pieces[2]

- Pieces:

- Horizontal layout:



- Vertical heaps:



Trace monoid[1]

- Alphabet:

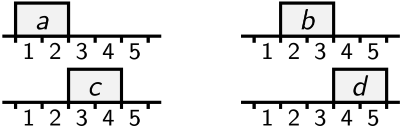
$$\Sigma = \{a, b, c, d\}$$

Heaps of pieces and trace monoids

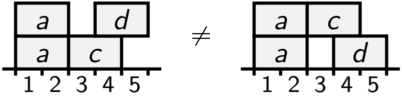
Heap of pieces[2]

- Pieces:

- Horizontal layout:



- Vertical heaps:



Trace monoid[1]

- Alphabet:

$$\Sigma = \{a, b, c, d\}$$

- Dependence relation:

$$\mathrm{D} = \{\{a, b\}, \{b, c\}, \{c, d\}\}$$

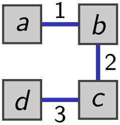
- Trace monoid:

$$\mathcal{M} = \left\langle a, b, c, d \;\middle|\; \begin{array}{l} ac = ca \\ ad = da \\ bd = db \end{array} \right\rangle^{+}$$

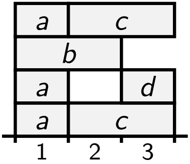
Dependency graph

Heaps of pieces and dependency graph

Dependency graph



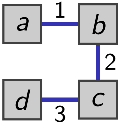
Heap of pieces

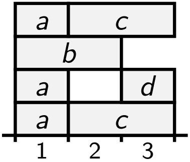


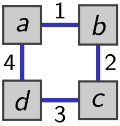
Heaps of pieces and dependency graph

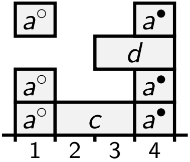
Dependency graph

Heap of (disconnected) **pieces**









Fragments of large heaps

What do **random large** heaps of pieces look like?

Fragments of large heaps

What do **random large** heaps of pieces look like?

1. Define your preferred notion of heap length
2. Focus on finite-horizon / stopping-time events
3. Study uniform distributions μ_k on heaps of length k: what if $k \to \infty$?

Fragments of large heaps

What do **random large** heaps of pieces look like?

1. Heap length $|\xi|$ = #pieces in the heap ξ
2. Consider events $E_x = \{\xi$ **starts with** $x\}$
3. **Weak convergence** of distributions:

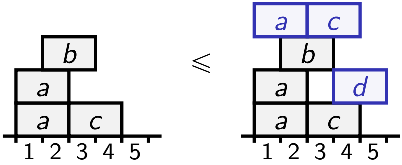
$$\mu_k \to \nu \Leftrightarrow (\forall x \in \mathcal{M}, \mathbb{P}_{\mu_k}[E_x] \to \mathbb{P}_\nu[E_x])$$

Fragments of large heaps

What do **random large** heaps of pieces look like?

1. Heap length $|\xi| = \#$pieces in the heap ξ
2. Consider events $E_x = \{x \leqslant \xi\}$
3. **Weak convergence** of distributions:

$$\mu_k \to \nu \Leftrightarrow (\forall x \in \mathcal{M}, \mathbb{P}_{\mu_k}[E_x] \to \mathbb{P}_\nu[E_x])$$

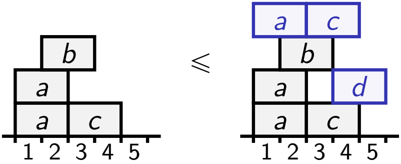


Fragments of large heaps

What do **random large** heaps of pieces look like?

1. Heap length $|\xi| = \#$pieces in the heap ξ
2. Consider events $E_x = \{x \leqslant \xi\}$
3. **Weak convergence** of distributions:

$$\mu_k \to \nu \Leftrightarrow (\forall x \in \mathcal{M}, \mathbb{P}_{\mu_k}[E_x] \to \mathbb{P}_\nu[E_x])$$



Theorem[3] — not constructive!

If μ_k is the unifom measure on $\mathcal{M}_k = \{\xi \in \mathcal{M} : |\xi| = k\}$, ν **exists** and is the **critical Bernoulli** distribution of $\mathcal{M}$.

Bernoulli distributions

Definition

A probability measure μ on $\mathcal{M}$ is:

- **uniform Bernoulli** of parameter p if

$$\forall x \in \mathcal{M}, \forall \sigma \in \Sigma, \mathbb{P}_\mu[x\,\sigma \leqslant \xi \mid x \leqslant \xi] = p$$

Bernoulli distributions

Definition

A probability measure μ on $\mathcal{M}$ is:

- **uniform Bernoulli** of parameter p if

$$\forall x \in \mathcal{M}, \forall \sigma \in \Sigma, \mathbb{P}_\mu[x\,\sigma \leqslant \xi \mid x \leqslant \xi] = p$$

Which are the **possible values** of the parameter p?

Bernoulli distributions

Definition

A probability measure μ on $\mathcal{M}$ is:

- **uniform Bernoulli** of parameter p if

$$\forall x \in \mathcal{M}, \forall \sigma \in \Sigma, \mathbb{P}_\mu[x\,\sigma \leqslant \xi \mid x \leqslant \xi] = p$$

Which are the **possible values** of the parameter p?

$$\text{Uniform Bernoulli} \Rightarrow \mathbb{P}_\mu[x = \xi] \propto p^{|x|}.$$

Bernoulli distributions

Definition

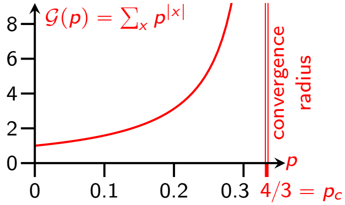
A probability measure μ on $\mathcal{M}$ is:

- **uniform Bernoulli** of parameter p if

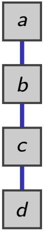
$$\forall x \in \mathcal{M}, \forall \sigma \in \Sigma, \mathbb{P}_\mu[x\,\sigma \leqslant \xi \mid x \leqslant \xi] = p$$

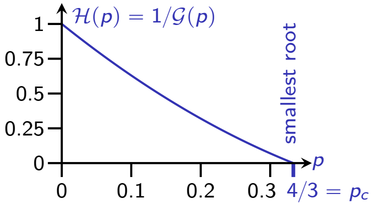
Which are the **possible values** of the parameter p? $\quad 0 \leqslant p < p_c$

$$\text{Uniform Bernoulli} \Rightarrow \mathbb{P}_\mu[x = \xi] = p^{|x|}\mathcal{H}(p).$$



Generating function $\mathcal{G}(p)$





Möbius polynomial[2] $\mathcal{H}(p)$

Bernoulli distributions

Definition

A probability measure μ on $\overline{\mathcal{M}}$ is:

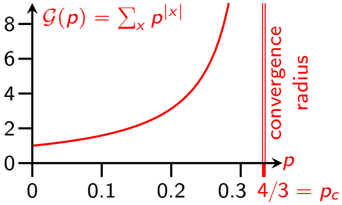
- **uniform Bernoulli** of parameter p if

$$\forall x \in \mathcal{M}, \forall \sigma \in \Sigma, \mathbb{P}_\mu[x\,\sigma \leqslant \xi \mid x \leqslant \xi] = p$$

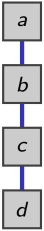
- **critical Bernoulli** if $p = p_c$ (requires **infinite** heaps)

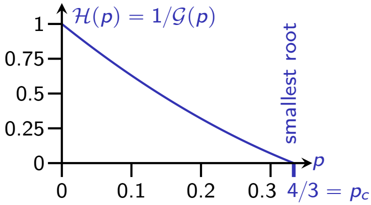
Which are the **possible values** of the parameter p? $0 \leqslant p \leqslant p_c$

$$\text{Uniform Bernoulli} \Rightarrow \mathbb{P}_\mu[x = \xi] = p^{|x|}\mathcal{H}(p).$$



Generating function $\mathcal{G}(p)$

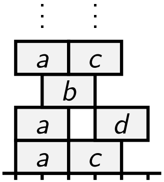




Möbius polynomial[2] $\mathcal{H}(p)$

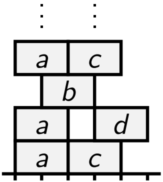
Infinite heaps

Heap of pieces

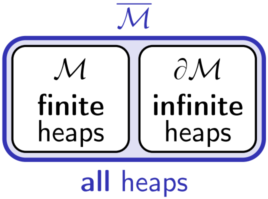


Infinite heaps

Heap of pieces



Sets of interest



Fact #1[3]

The limit ν is a distribution **on the set** $\overline{\mathcal{M}}$ with **support** $\partial\mathcal{M}$.⚠

Contents

Simulating the limit ν

Idea #1: Pick $\xi_k \sim \mu_k$, pick a piece x **wisely** and set $\xi_{k+1} = \xi_k \cdot x$

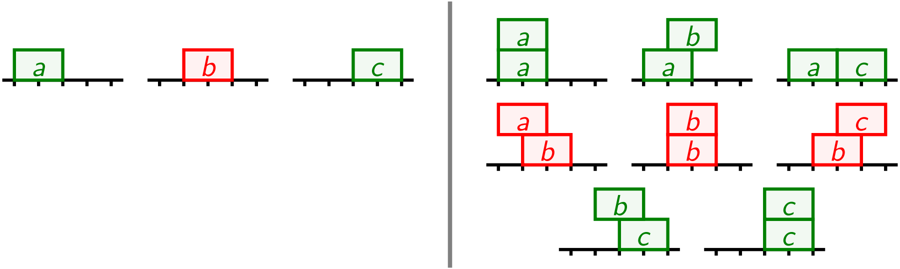
Simulating the limit ν

Idea #1: Pick $\xi_k \sim \mu_k$, pick a piece x **wisely** and set $\xi_{k+1} = \xi_k \cdot x$
Problem: In general, ξ_{k+1} **cannot** be distributed according to μ_{k+1}

Example in the monoid $\langle a, b, c \mid ac = ca \rangle^+$

Dependency graph

$\mu_1(\Uparrow a \cup \Uparrow c) = 2/3 \quad > \quad \mu_2(\Uparrow a \cup \Uparrow c) = 5/8$

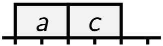


Simulating the limit ν

Idea #2: Simulate $\xi \sim \nu$, **floor by floor**

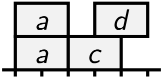
Simulating the limit ν

Idea #2: Simulate $\xi \sim \nu$, **floor by floor**



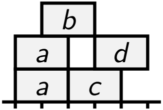
Simulating the limit ν

Idea #2: Simulate $\xi \sim \nu$, **floor by floor**



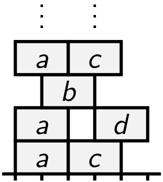
Simulating the limit ν

Idea #2: Simulate $\xi \sim \nu$, **floor by floor**



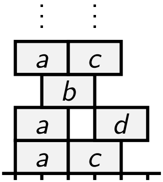
Simulating the limit ν

Idea #2: Simulate $\xi \sim \nu$, **floor by floor**



Simulating the limit ν

Idea #2: Simulate $\xi \sim \nu$, **floor by floor**

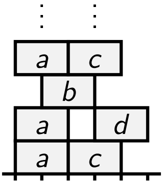


Fact #2[3]

This approach works because ν is **Bernoulli**!

Simulating the limit ν

Idea #2: Simulate $\xi \sim \nu$, **floor by floor**



Fact #2[3]

This approach works because ν is **Bernoulli**!

Problem: Huge state space (exponential number of possible floors)

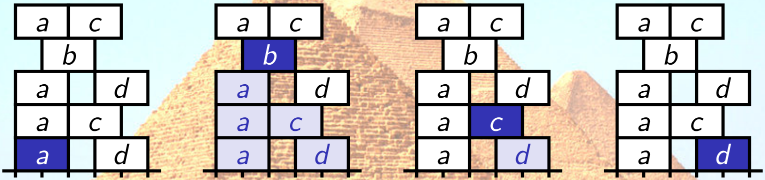
Contents

Simulating the limit ν piece by piece⚠

Idea #3: Decompose heaps recursively by using **pyramids**

Simulating the limit ν piece by piece⚠

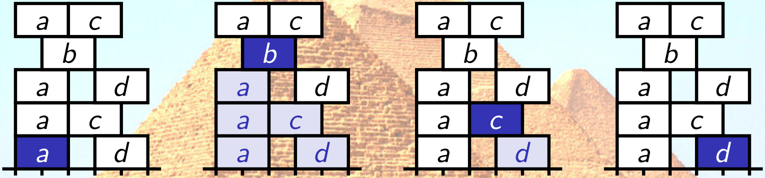
Idea #3: Decompose heaps recursively by using **pyramids**



a-pyramid　*b*-pyramid　*c*-pyramid　*d*-pyramid

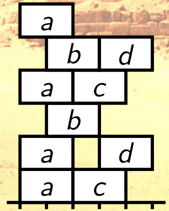
Simulating the limit ν piece by piece⚠

Idea #3: Decompose heaps recursively by using **pyramids**



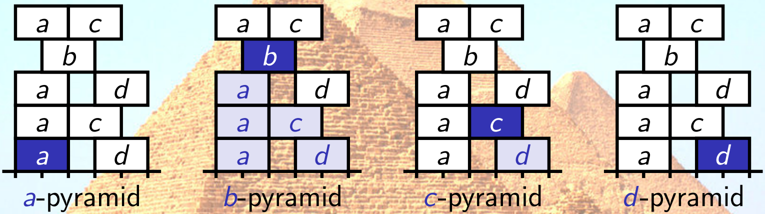
a-pyramid b-pyramid c-pyramid d-pyramid

Example: Recursive decomposition using b-pyramids

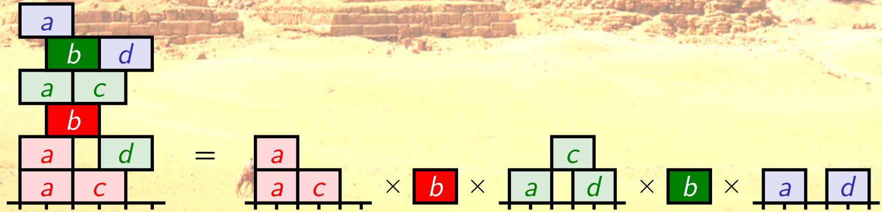


Simulating the limit ν piece by piece⚠

Idea #3: Decompose heaps recursively by using **pyramids**

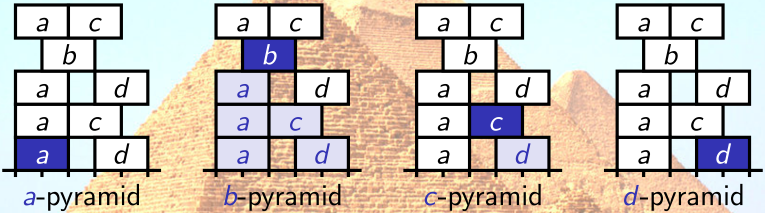


Example: Recursive decomposition using *b*-pyramids

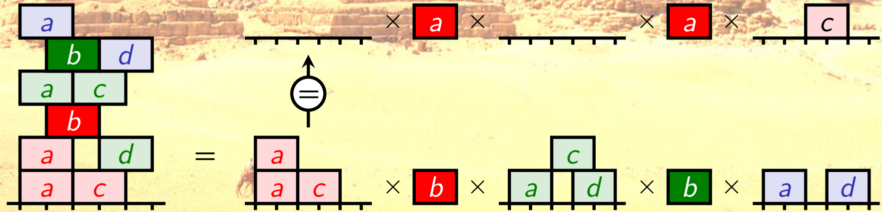


Simulating the limit ν piece by piece ⚠

Idea #3: Decompose heaps recursively by using **pyramids**



Example: Recursive decomposition using b-pyramids, then a-pyramids. . .



Simulating the limit ν piece by piece

Idea #3: Decompose heaps recursively by using **independent pyramids**

Theorem[4]

If a $\in \Sigma$ and ν is **Bernoulli** on $\overline{\mathcal{M}}(\Sigma)$, then

Heap

$=$

a-pyramid

$\times$

Heap

$+$

a-free heap

where right-hand side random variables are **independent**.

Simulating the limit ν piece by piece ⚠

Idea #3: Decompose heaps recursively by using **independent pyramids**

Theorem[4]

If $\mathsf{a} \in \Sigma$ and ν is **Bernoulli** on $\overline{\mathcal{M}}(\Sigma)$, then

 $=$  $\times$  $+$ 

Heap = a-pyramid × Heap + a-free heap

where right-hand side random variables are **independent**.

 $=$  $\times$ 

a-pyramid = a-free heap ξ s.t. $\mathsf{R}(\xi) \subseteq \mathsf{D}(\mathsf{a})$

- $\mathsf{R}(\xi) = \{\sigma \in \Sigma : \xi \in \mathcal{M} \cdot \sigma\}$
- $\mathsf{D}(\mathsf{a}) = \{\sigma \in \Sigma : \sigma \cdot \mathsf{a} \neq \mathsf{a} \cdot \sigma\}$

Simulating the limit ν piece by piece⚠

Idea #3: Decompose heaps recursively by using **independent pyramids**

Theorem[4]

If $X \subseteq \Sigma$, $\mathsf{a} \in \Sigma$ and ν is **Bernoulli** on $\overline{\mathcal{M}}(\Sigma)$, then

Heap ξ s.t. $\mathsf{R}(\xi) \subseteq X$

$= \sum_{k=1}^{\infty}$

a-pyramid

$^{k} \times$

a-free heap ξ s.t. $\mathsf{R}(\mathsf{a} \cdot \xi) \subseteq X$

$+$

a-free heap ξ s.t. $\mathsf{R}(\xi) \subseteq X$

where right-hand side random variables are **independent**.

a-pyramid

$=$

a-free heap ξ s.t. $\mathsf{R}(\xi) \subseteq \mathsf{D}(\mathsf{a})$

$\times$ Ⓐ

- $\mathsf{R}(\xi) = \{\sigma \in \Sigma : \xi \in \mathcal{M} \cdot \sigma\}$
- $\mathsf{D}(\mathsf{a}) = \{\sigma \in \Sigma : \sigma \cdot \mathsf{a} \neq \mathsf{a} \cdot \sigma\}$

Our algorithm: Generating heaps distributed according to ν

Refined goal: Generate a heap $\xi \in \mathcal{M}$ with $\mathrm{R}(\xi) \subseteq X$:

1. Generate an a-free heap $\xi \in \mathcal{M}$ with $\mathrm{R}(\xi) \subseteq X$
2. How many pieces a should ξ contain? ($k \leftarrow$ **Geometric law**)
   - if $k = 0$: output $\xi = \xi$
   - if $k \geqslant 1$ and $\mathrm{R}(\mathrm{a}\cdot\xi) \nsubseteq X$: go back to step #1 (anticipated rejection)
   - if $k \geqslant 1$ and $\mathrm{R}(\mathrm{a}\cdot\xi) \subseteq X$: generate a-pyramids $\xi_1, \ldots, \xi_k$ and output $\xi = \xi_1 \cdot \xi_2 \cdots \xi_k \cdot \xi$

Our algorithm: Generating heaps distributed according to ν

Refined goal: Generate a heap $\xi \in \mathcal{M}$ with $\mathrm{R}(\xi) \subseteq X$:

1. Generate an a-free heap $\xi \in \mathcal{M}$ with $\mathrm{R}(\xi) \subseteq X$
2. How many pieces a should ξ contain? ($k \leftarrow$ **Geometric law**)
   - if $k = 0$: output $\xi = \xi$
   - if $k \geqslant 1$ and $\mathrm{R}(\mathrm{a}\cdot\xi) \nsubseteq X$: go back to step #1 (anticipated rejection)
   - if $k \geqslant 1$ and $\mathrm{R}(\mathrm{a}\cdot\xi) \subseteq X$: generate a-pyramids $\xi_1, \ldots, \xi_k$ and output $\xi = \xi_1 \cdot \xi_2 \cdots \xi_k \cdot \xi$

Variant: Choose $\mathrm{a} \in X$ and avoid rejection

Our algorithm: Generating heaps distributed according to ν

Refined goal: Generate a heap $\xi \in \mathcal{M}$ with $\mathrm{R}(\xi) \subseteq X$:

1. Generate an a-free heap $\xi \in \mathcal{M}$ with $\mathrm{R}(\xi) \subseteq X$
2. How many pieces a should ξ contain? ($k \leftarrow$ **Geometric law**)
   - if $k = 0$: output $\xi = \xi$
   - if $k \geqslant 1$ and $\mathrm{R}(\mathrm{a}\cdot\xi) \nsubseteq X$: go back to step #1 (anticipated rejection)
   - if $k \geqslant 1$ and $\mathrm{R}(\mathrm{a}\cdot\xi) \subseteq X$: generate a-pyramids $\xi_1, \ldots, \xi_k$ and output $\xi = \xi_1 \cdot \xi_2 \cdots \xi_k \cdot \xi$

Variant: Choose $\mathrm{a} \in X$ and avoid rejection

Running time/piece: $n\,q$ or n **Read-only memory usage:** n or 2^n

where $\mathcal{M}' = \mathcal{M}(\Sigma \backslash \{a\})$ and $q = \mathcal{G}_{\mathcal{M}'}(p_c) \leqslant 1/p_c^n$ ($q = n^{\Theta(n)}$ is possible)

Contents

A distributed simulation algorithm

Algorithm based on:

- precomputing and storing **Möbius polynomials** of sub-monoids
- decomposing heaps into **independent** pyramids/heaps in sub-monoids
- outputting pieces **one by one** with little synchronisation

A distributed simulation algorithm

Algorithm based on:

- precomputing and storing **Möbius polynomials** of sub-monoids
- decomposing heaps into **independent** pyramids/heaps in sub-monoids
- outputting pieces **one by one** with little synchronisation

Two variants (mixtures are possible):

- small storage – anticipated rejection – rather low efficiency
- huge storage – no rejection – high, guaranteed efficiency

A distributed simulation algorithm

Algorithm based on:

- precomputing and storing **Möbius polynomials** of sub-monoids
- decomposing heaps into **independent** pyramids/heaps in sub-monoids
- outputting pieces **one by one** with little synchronisation

Two variants (mixtures are possible):

- small storage – anticipated rejection – rather low efficiency
- huge storage – no rejection – high, guaranteed efficiency

Very efficient on graphs with:

- few cycles (small storage/high efficiency for a mix between variants)
- small tree-width (no preprocessing/storage)

Some references

[1] *Problèmes combinatoires de commutation et réarrangements*, Cartier & Foata (1969)
[2] *Heaps of pieces I*, Viennot (1986)
[3] *Uniform generation in trace monoids*, Abbes & Mairesse (2015)
[4] *Uniform generation of infinite traces*, Abbes & Jugé (2022)


Thank you