The Combinatorics of Barrier Synchronization[a]

Petri Nets[b] 2019 – June 23-28 – Aachen

Olivier Bodini[(1)], Matthieu Dien[(2)], Antoine Genitrini[(3)], **Frédéric Peschanski**[(3)]

(1) LIPN Institut Galilée – (2) Unicaen Greyc – (3) **Sorbonne University – LIP6**

[a]Research partially supported by the MetACOnc project ANR-15-CE40-0014.

[b]and other (less powerful) models of concurrency

Object of study

We study **concurrent systems** from the point of view of **combinatorics** specifically:

- **Enumerative combinatorics**
  $\Rightarrow$ The science of counting "composable things"
- **Order theory**
  $\Rightarrow$ the science of *partially ordered sets* a.k.a. Posets

Object of study

We study **concurrent systems** from the point of view of **combinatorics** specifically:

- **Enumerative combinatorics**
  $\Rightarrow$ The science of counting "composable things"
- **Order theory**
  $\Rightarrow$ the science of *partially ordered sets* a.k.a. Posets

---

Definition (Combinatorial class)
A set of objects associated to a notion of a (finite) **size**, and such that there is a *finite number* of objects of a given size.

Object of study

We study **concurrent systems** from the point of view of **combinatorics** specifically:

- **Enumerative combinatorics**
  $\Rightarrow$ The science of counting "composable things"
- **Order theory**
  $\Rightarrow$ the science of *partially ordered sets* a.k.a. Posets

---

Definition (Combinatorial class)
A set of objects associated to a notion of a (finite) **size**, and such that there is a *finite number* of objects of a given size.

$\Rightarrow$ but what is the size of a concurrent process ?

A simple class: barrier synchronization processes

A very simple calculus of *barrier synchronization*.

Process		**Size** $\lvert . \rvert$	
$P, Q ::=$	0	0	(termination)
	$\mid \alpha.P$	$1 + \lvert P \rvert$	(atomic action and prefixing)
	$\mid \nu(B)P$	$1 + \lvert P \rvert$	(barrier and scope)
	$\mid \langle B \rangle P$	$1 + \lvert P \rvert$	(synchronization)
	$\mid P \parallel Q$	$1 + \lvert P \rvert + \lvert Q \rvert$	(parallel)

A simple class: barrier synchronization processes

A very simple calculus of *barrier synchronization*.

Process		**Size** $|.|$	
$P, Q ::=$	0	0	(termination)
	$\mid \alpha.P$	$1 + |P|$	(atomic action and prefixing)
	$\mid \nu(B)P$	$1 + |P|$	(barrier and scope)
	$\mid \langle B \rangle P$	$1 + |P|$	(synchronization)
	$\mid P \parallel Q$	$1 + |P| + |Q|$	(parallel)

Remark: ✓finite size , ✓finite number of objects of size n

$$0,\ \alpha.0,\ \langle B\rangle\ 0,\ \nu(B)\ 0,\ \alpha.\beta.0,\ \ldots,\ 0 \parallel 0,\ \ldots,\ \alpha.0 \parallel 0,\ \ldots$$

A simple class: barrier synchronization processes

A very simple calculus of *barrier synchronization*.

Process		**Size** $|.|$	
$P, Q ::=$	0	0	(termination)
	$\mid \alpha.P$	$1 + |P|$	(atomic action and prefixing)
	$\mid \nu(B)P$	$1 + |P|$	(barrier and scope)
	$\mid \langle B \rangle P$	$1 + |P|$	(synchronization)
	$\mid P \parallel Q$	$1 + |P| + |Q|$	(parallel)

Remark: ✓finite size , ✓finite number of objects of size n

$$0,\ \alpha.0,\ \langle B \rangle\ 0,\ \nu(B)\ 0,\ \alpha.\beta.0,\ \ldots,\ 0 \parallel 0,\ \ldots,\ \alpha.0 \parallel 0,\ \ldots$$

⇒ what about a **semantic** notion of a size?

Semantics

Process behavior in a nutshell (cf. relatively "unpleasant" proof system in the paper)

$P \stackrel{defs}{=} \nu(B)\ [a_1.\langle B\rangle a_2.0 \parallel \langle B\rangle b_1.0 \parallel \langle B\rangle 0]$

$\Rightarrow$ synchronization on barrier $\langle B\rangle$ is *not* available because the leftmost process is not ready.

Semantics

Process behavior in a nutshell (cf. relatively "unpleasant" proof system in the paper)

$P \stackrel{defs}{=} \nu(B)\ [a_1.\langle B\rangle a_2.0 \parallel \langle B\rangle b_1.0 \parallel \langle B\rangle 0]$

$\Rightarrow$ synchronization on barrier $\langle B\rangle$ is *not* available because the leftmost process is not ready.

$\cdots \xrightarrow{a_1} \nu(B)\ [\langle B\rangle a_2.0 \parallel \langle B\rangle b_1.0 \parallel \langle B\rangle 0]$

$\Rightarrow$ synchronization available

Semantics

Process behavior in a nutshell (cf. relatively "unpleasant" proof system in the paper)

$P \stackrel{defs}{=} \nu(B)\ [a_1.\langle B\rangle a_2.0 \parallel \langle B\rangle b_1.0 \parallel \langle B\rangle 0]$

$\Rightarrow$ synchronization on barrier $\langle B\rangle$ is *not* available because the leftmost process is not ready.

$\cdots \xrightarrow{a_1} \nu(B)\ [\langle B\rangle a_2.0 \parallel \langle B\rangle b_1.0 \parallel \langle B\rangle 0]$

$\Rightarrow$ synchronization available

$\cdots \longrightarrow a_2.0 \parallel b_1.0 \parallel 0$ (in the paper synchronization is not a transition, but it could)

$\Rightarrow$ interleaving semantics

Semantics

Process behavior in a nutshell (cf. relatively "unpleasant" proof system in the paper)

$P \stackrel{defs}{=} \nu(B)\ [a_1.\langle B\rangle a_2.0 \parallel \langle B\rangle b_1.0 \parallel \langle B\rangle 0]$

$\Rightarrow$ synchronization on barrier $\langle B\rangle$ is *not* available because the leftmost process is not ready.

$\cdots \xrightarrow{a_1} \nu(B)\ [\langle B\rangle a_2.0 \parallel \langle B\rangle b_1.0 \parallel \langle B\rangle 0]$

$\Rightarrow$ synchronization available

$\cdots \rightarrow a_2.0 \parallel b_1.0 \parallel 0$ (in the paper synchronization is not a transition, but it could)

$\Rightarrow$ interleaving semantics

$\cdots \xrightarrow{a_2}\xrightarrow{b_1} 0$ or $\cdots \xrightarrow{b_1}\xrightarrow{a_2} 0$

Semantics

Process behavior in a nutshell (cf. relatively “unpleasant” proof system in the paper)

$P \stackrel{defs}{=} \nu(B)\ [a_1.\langle B\rangle a_2.0 \parallel \langle B\rangle b_1.0 \parallel \langle B\rangle 0]$

$\Rightarrow$ synchronization on barrier $\langle B\rangle$ is *not* available because the leftmost process is not ready.

$\cdots \xrightarrow{a_1} \nu(B)\ [\langle B\rangle a_2.0 \parallel \langle B\rangle b_1.0 \parallel \langle B\rangle 0]$

$\Rightarrow$ synchronization available

$\cdots \rightarrow a_2.0 \parallel b_1.0 \parallel 0$ (in the paper synchronization is not a transition, but it could)

$\Rightarrow$ interleaving semantics

$\cdots \xrightarrow{a_2}\xrightarrow{b_1} 0$ or $\cdots \xrightarrow{b_1}\xrightarrow{a_2} 0$

Definition (**Execution**) A maximal path of transitions

2 paths: $P \xrightarrow{a_1}\rightarrow\xrightarrow{a_2}\xrightarrow{b_1} 0$ and $P \xrightarrow{a_1}\rightarrow\xrightarrow{b_1}\xrightarrow{a_2} 0$

Semantics

Process behavior in a nutshell (cf. relatively “unpleasant” proof system in the paper)

$P \stackrel{defs}{=} \nu(B)\ [a_1.\langle B\rangle a_2.0 \parallel \langle B\rangle b_1.0 \parallel \langle B\rangle 0]$

$\Rightarrow$ synchronization on barrier $\langle B\rangle$ is *not* available because the leftmost process is not ready.

$\cdots \xrightarrow{a_1} \nu(B)\ [\langle B\rangle a_2.0 \parallel \langle B\rangle b_1.0 \parallel \langle B\rangle 0]$

$\Rightarrow$ synchronization available

$\cdots \rightarrow a_2.0 \parallel b_1.0 \parallel 0$ (in the paper synchronization is not a transition, but it could)

$\Rightarrow$ interleaving semantics

$\cdots \xrightarrow{a_2}\xrightarrow{b_1} 0$ or $\cdots \xrightarrow{b_1}\xrightarrow{a_2} 0$

Definition (**Execution**) A maximal path of transitions

2 paths: $P \xrightarrow{a_1}\rightarrow\xrightarrow{a_2}\xrightarrow{b_1} 0$ and $P \xrightarrow{a_1}\rightarrow\xrightarrow{b_1}\xrightarrow{a_2} 0$

$\Rightarrow$ (semantic) size 2

Counting executions?

Why taking the number of (interleaved) executions as size?

Counting executions?

Why taking the number of (interleaved) executions as size?

▷ **Intuitively** it lets us observe/reason about *combinatorial explosion*.

Counting executions?

Why taking the number of (interleaved) executions as size?

▷ **Intuitively** it lets us observe/reason about *combinatorial explosion*.

▷ **More concretely** it is a very effective *enumerative combinatorics* tool by discriminating process terms in a very sharp way.

Counting executions?

Why taking the number of (interleaved) executions as size?

▷ **Intuitively** it lets us observe/reason about *combinatorial explosion*.

▷ **More concretely** it is a very effective *enumerative combinatorics* tool by discriminating process terms in a very sharp way.

e.g. $\alpha_1.\alpha_2.\alpha_3.\alpha_4.\alpha_5.0$ has syntactic size 5 and semantic size 1

Counting executions?

Why taking the number of (interleaved) executions as size?

▷ **Intuitively** it lets us observe/reason about *combinatorial explosion*.

▷ **More concretely** it is a very effective *enumerative combinatorics* tool by discriminating process terms in a very sharp way.

e.g. $\alpha_1.\alpha_2.\alpha_3.\alpha_4.\alpha_5.0$ has syntactic size 5 and semantic size 1

whereas $\alpha_1.\alpha_2.0 \parallel \alpha_3.\alpha_4.0$ has syntactic size 5 and semantic size 6

1. $\xrightarrow{\alpha_1} \cdot \xrightarrow{\alpha_2} \cdot \xrightarrow{\alpha_3} \cdot \xrightarrow{\alpha_4}$
2. $\xrightarrow{\alpha_1} \cdot \xrightarrow{\alpha_3} \cdot \xrightarrow{\alpha_2} \cdot \xrightarrow{\alpha_4}$
3. $\xrightarrow{\alpha_1} \cdot \xrightarrow{\alpha_3} \cdot \xrightarrow{\alpha_4} \cdot \xrightarrow{\alpha_2}$
4. $\xrightarrow{\alpha_3} \cdot \xrightarrow{\alpha_1} \cdot \xrightarrow{\alpha_2} \cdot \xrightarrow{\alpha_4}$
5. $\xrightarrow{\alpha_3} \cdot \xrightarrow{\alpha_1} \cdot \xrightarrow{\alpha_4} \cdot \xrightarrow{\alpha_2}$
6. $\xrightarrow{\alpha_3} \cdot \xrightarrow{\alpha_4} \cdot \xrightarrow{\alpha_1} \cdot \xrightarrow{\alpha_2}$

Practical applications?

Question: is there any practical use of *this*?

Practical applications?

Question: is there any practical use of *this*?

$\Rightarrow$ we would argue yes, and it is about the *statistical* analysis of (concurrent) systems.

Practical applications?

Question: is there any practical use of *this*?

⇒ we would argue yes, and it is about the *statistical* analysis of (concurrent) systems.

- generate executions uniformly at random
- "navigate" the state-space wrt. the uniform distribution of executions, e.g. exploring the "less probable" parts of the system under study (skewing the uniform distribution)
- property-based (generative) testing
- statistical model-checking[1]

[1] cf. *Monte Carlo model checking*, R. Gosu and S. A. Smola, Tacas 2005.

Computing the semantic size?

Question: is it *difficult* to compute the semantic size of a process, i.e. to count its distinct executions ?

Computing the semantic size?

Question: is it *difficult* to compute the semantic size of a process, i.e. to count its distinct executions ? For some processes, it's "easy" ...

▷ e.g. tree-shaped processes[2] (scheduling problems):

$$\text{for a tree } T, \quad \frac{|T|!}{\prod_{S \text{ a subtree of } T} |S|}$$

(⇒ Hook-length formula, known since at least Knuth's TAOC but we had to find it)

▷ also series-parallel processes[3] (SP-posets): counting in $O(n)$

▷ also asynchronous structures[4] (promises): counting in $O(n^2)$

[2] *A Quantitative Study of Parallel Processes*, EJC Vol.13/1 (2016).
[3] *Entropic Uniform Sampling of Linear Extensions in Series-Parallel Posets.* CSR 2017
[4] *Beyond Series-Parallel Concurrent Systems: The Case of Arch Processes.* AofA 2018

Counting in general is hard

In the paper, we show:

- A *non-deadlocked* process expressed in the very simple *barrier synchronization calculus* (shown previously) has a control graph shaped after an *intransitive directed acyclic graph* (DAG)
- The correspondance is complete: any (intransitive) DAG can be expressed as a process (we did not pickup the syntax arbitrarily)
- The one-to-one correspondance conveys to *partially ordered sets*, a.k.a. Posets (the *covering* of a poset is an intransitive DAG, a.k.a its transitive reduction seen as a digraph)

Counting in general is hard

In the paper, we show:

- A *non-deadlocked* process expressed in the very simple *barrier synchronization calculus* (shown previously) has a control graph shaped after an *intransitive directed acyclic graph* (DAG)
- The correspondance is complete: any (intransitive) DAG can be expressed as a process (we did not pickup the syntax arbitrarily)
- The one-to-one correspondance conveys to *partially ordered sets*, a.k.a. Posets (the *covering* of a poset is an intransitive DAG, a.k.a its transitive reduction seen as a digraph)

Consequence: Process executions = Linear extensions (of arbitrary Posets)

Consequence^2: Counting process executions = Counting linear extensions (of arbitrary Posets)

Consequence^3: Counting process executions is $\sharp$-P complete (and that's not good)
$\Rightarrow$ cf. *Counting Linear Extensions* by G. Brightwell and P. Winkler. Order (1991)

Counting in general is hard

In the paper, we show:

- A *non-deadlocked* process expressed in the very simple *barrier synchronization calculus* (shown previously) has a control graph shaped after an *intransitive directed acyclic graph* (DAG)
- The correspondance is complete: any (intransitive) DAG can be expressed as a process (we did not pickup the syntax arbitrarily)
- The one-to-one correspondance conveys to *partially ordered sets*, a.k.a. Posets (the *covering* of a poset is an intransitive DAG, a.k.a its transitive reduction seen as a digraph)

Consequence: Process executions = Linear extensions (of arbitrary Posets)

Consequence^2: Counting process executions = Counting linear extensions (of arbitrary Posets)

Consequence^3: Counting process executions is $\sharp$-P complete (and that's not good)
$\Rightarrow$ cf. *Counting Linear Extensions* by G. Brightwell and P. Winkler. Order (1991)

... However there is a uniform random sampler available
(*Fast perfect sampling of linear extensions*. M. Huber. Discrete Mathematics (2006)).

Geometrical foundation: continuous embedding of a Poset

(This is classical combinatorics, but that does not make it easy to grasp...)

Idea[5]: Continuous embedding of a Poset into the unit hypercube.

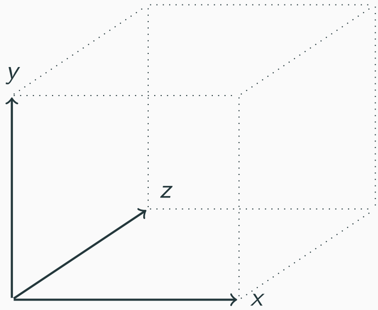
[5] *Two poset polytopes*. R. P. Stanley. Discrete & computational geometry (1986).

Geometrical foundation: continuous embedding of a Poset

(This is classical combinatorics, but that does not make it easy to grasp...)

Idea[5]: Continuous embedding of a Poset into the unit hypercube.

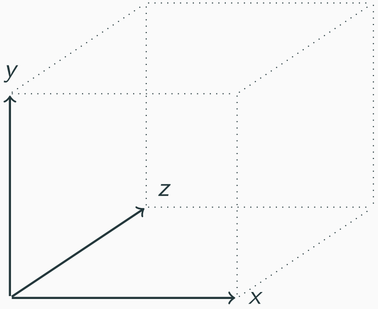
Example: embedding $\{x, y, z\}$ (size 3) into the ~~hyper~~cube (dimension 3)



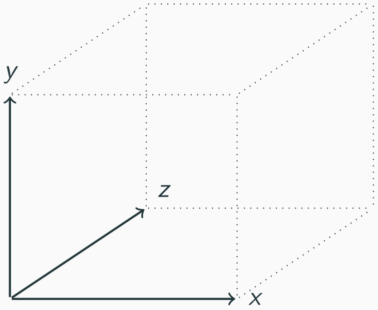
Remark: there is no constraint here, it's the unordered partial order.

[5] *Two poset polytopes*. R. P. Stanley. Discrete & computational geometry (1986).

Ordering constraint = slicing the hypercube

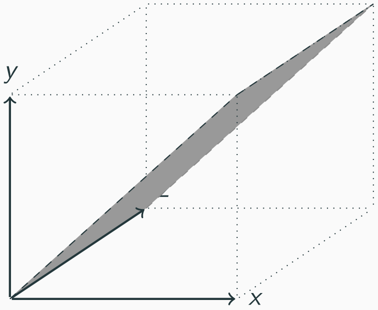


Ordering constraint = slicing the hypercube



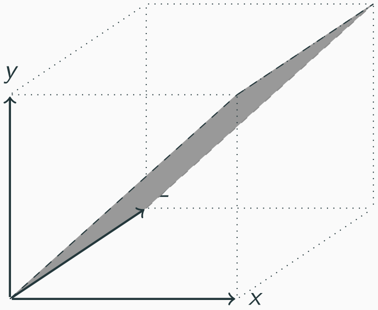
- Let's first "slice" the ~~hyper~~cube by an ~~hyper~~plane splitting the (x, y) face

Ordering constraint = slicing the hypercube



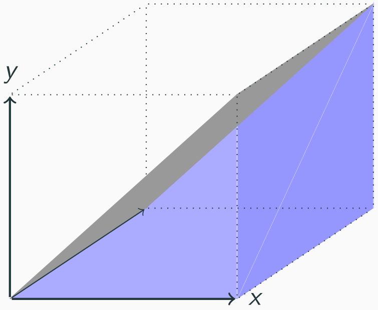
- Let's first "slice" the ~~hyper~~cube by an ~~hyper~~plane splitting the (x, y) face

Ordering constraint = slicing the hypercube



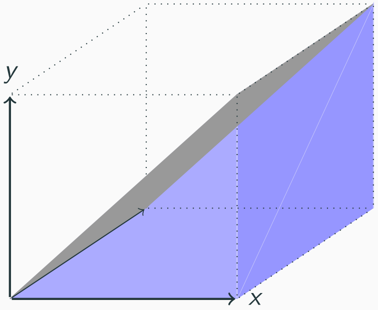
- Let's first "slice" the ~~hyper~~cube by an ~~hyper~~plane splitting the (x, y) face
- Then enforcing $x > y$ consists in taking the lower part of the slice

Ordering constraint = slicing the hypercube



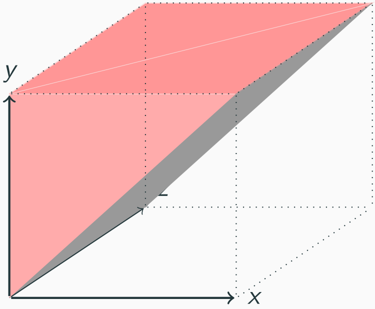
- Let's first "slice" the ~~hyper~~cube by an ~~hyper~~plane splitting the (x, y) face
- Then enforcing $x > y$ consists in taking the lower part of the slice

Ordering constraint = slicing the hypercube



- Let's first "slice" the ~~hyper~~cube by an ~~hyper~~plane splitting the (x, y) face
- Then enforcing $x > y$ consists in taking the lower part of the slice
- Conversely, enforcing $x < y$ consists in taking the upper part

Ordering constraint = slicing the hypercube



- Let's first "slice" the ~~hyper~~cube by an ~~hyper~~plane splitting the (x, y) face
- Then enforcing $x > y$ consists in taking the lower part of the slice
- Conversely, enforcing $x < y$ consists in taking the upper part

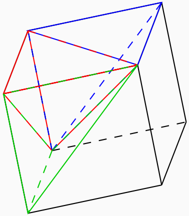
From slices to linear extensions

By successive slicing we can build a polytope C_P for an arbitrary poset P.
(note that the relative order of slices is arbitrary, this is just intersection)

From slices to linear extensions

By successive slicing we can build a polytope C_P for an arbitrary poset P.
(note that the relative order of slices is arbitrary, this is just intersection)

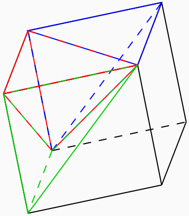
... and if we would slice further we would ultimately obtain a linear extension (as a simplex)



From slices to linear extensions

By successive slicing we can build a polytope C_P for an arbitrary poset P.
(note that the relative order of slices is arbitrary, this is just intersection)

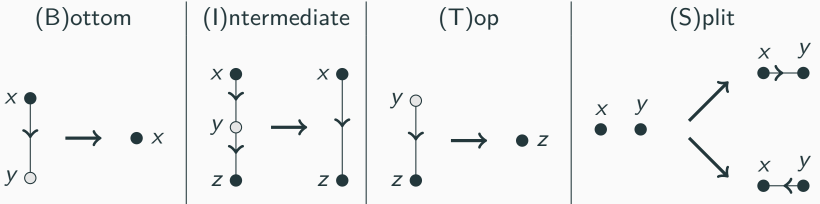
... and if we would slice further we would ultimately obtain a linear extension (as a simplex)



$\Rightarrow$ the number of linear extensions is then $|\ell| = n! \cdot \mathrm{Vol}(C_P)$
(with Vol "simply" a sum, i.e. an higher-dimensional integral)

Contribution 1: the BITS decomposition of DAGs/Posets/Barrier processes

Based on the hypercube embedding, this is "obvious" (isn't it?):



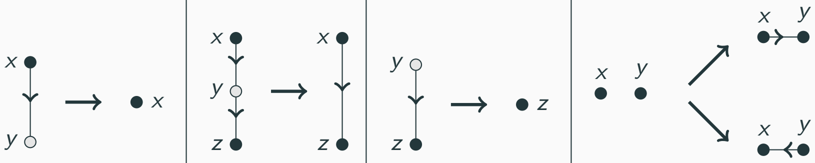
$\Psi' = \int_x^1 \Psi.dy$ | $\Psi' = \int_x^z \Psi.dy$ | $\Psi' = \int_0^z \Psi.dy$ | $\Psi' = \Psi_{x \prec y} + \Psi_{y \prec x}$

Remark: Ψ is your "current" polytope, Ψ' is the next one.

Contribution 1: the BITS decomposition of DAGs/Posets/Barrier processes

Based on the hypercube embedding, this is "obvious" (isn't it?):

(B)ottom	(I)ntermediate	(T)op	(S)plit



$\Psi' = \int_x^1 \Psi.dy$	$\Psi' = \int_x^z \Psi.dy$	$\Psi' = \int_0^z \Psi.dy$	$\Psi' = \Psi_{x \prec y} + \Psi_{y \prec x}$

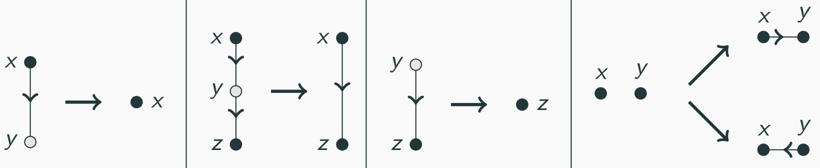
Remark: Ψ is your "current" polytope, Ψ' is the next one.

$\Rightarrow$ details (and example) in the paper of course!

Contribution 1: the BITS decomposition of DAGs/Posets/Barrier processes

Based on the hypercube embedding, this is "obvious" (isn't it?):

(B)ottom	(I)ntermediate	(T)op	(S)plit



$\Psi' = \int_x^1 \Psi . dy$	$\Psi' = \int_x^z \Psi . dy$	$\Psi' = \int_0^z \Psi . dy$	$\Psi' = \Psi_{x \prec y} + \Psi_{y \prec x}$

Remark: Ψ is your "current" polytope, Ψ' is the next one.

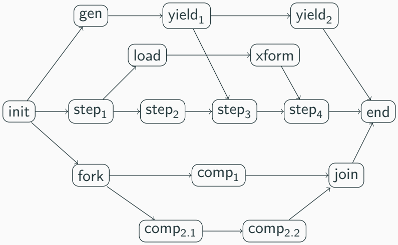
$\Rightarrow$ details (and example) in the paper of course!

Fact: the obtained formula is linear without the (S)plit rule

$\Rightarrow$ What can we do without it? What does it mean to need it?

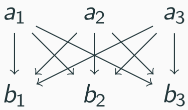
An example of a BIT-decomposable process

$$Sys = \mathsf{init}.\nu(G_1, G_2, J_1).\left[\begin{array}{l} \mathsf{step}_1. \\ \quad \nu(IO)\left[\begin{array}{l} \mathsf{step}_2.\langle G_1\rangle\mathsf{step}_3. \\ \quad \langle IO\rangle\mathsf{step}_4.\langle G_2\rangle\langle J_1\rangle\mathsf{end} \\ \| \mathsf{load}.\mathsf{xform}.\langle IO\rangle 0 \end{array}\right] \\ \| \mathsf{gen}.\mathsf{yield}_1.(\langle G_1\rangle 0 \| \mathsf{yield}_2.\langle G_2\rangle 0) \\ \| \mathsf{fork}.\nu(J_2)\left[\begin{array}{l} \mathsf{comp}_1.\langle J_2\rangle 0 \\ \| \mathsf{comp}_{2.1}.\mathsf{comp}_{2.2}.\langle J_2\rangle 0 \\ \| \langle J_2\rangle\mathsf{join}\langle J_1\rangle 0) \end{array}\right] \end{array}\right]$$

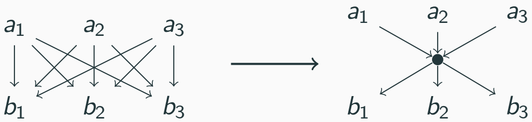


$\Rightarrow$ this process is BIT-decomposable, its has 1975974 distinct computations (Maxima computation)

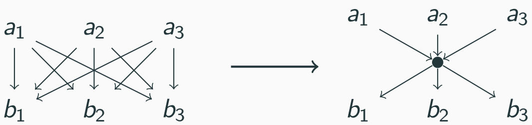
BIT-free processes ?

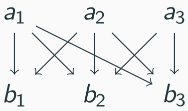


BIT-free processes ?

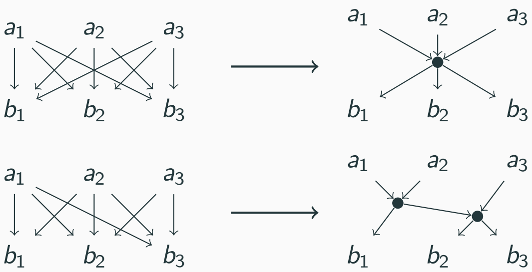


BIT-free processes ?

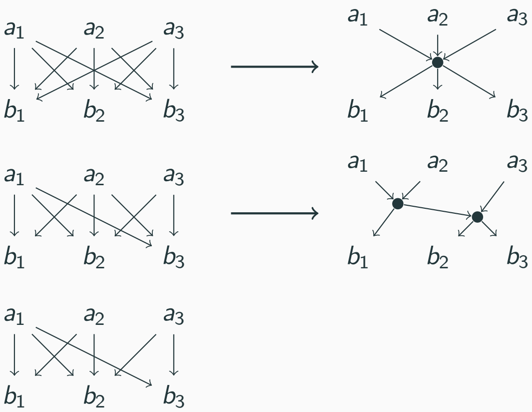




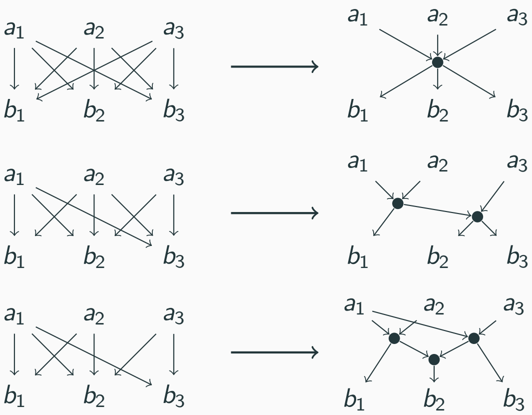
BIT-free processes ?



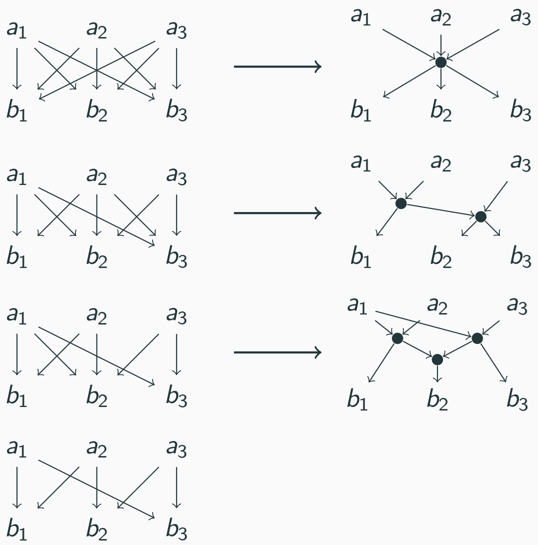
BIT-free processes ?



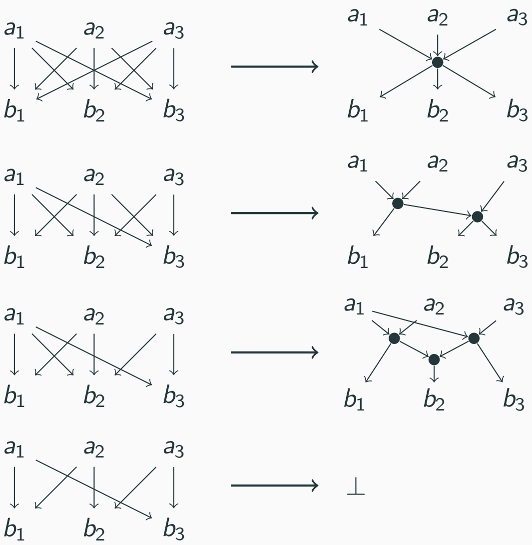
BIT-free processes ?



BIT-free processes ?



BIT-free processes ?



Contribution 2: A generic uniform random sampler

Based on the hypercube embedding, this is (less but still) "obvious" (isn't it?):

Algorithm 1 Uniform sampling of a simplex of the order polytope

function $\textsc{SamplePoint}(\mathcal{I} = \int_a^b f(y_i)\,\mathrm{d}y_i)$
  $C \leftarrow \mathsf{eval}(\mathcal{I})$; $U \leftarrow \textsc{Uniform}(a, b)$
  $Y_i \leftarrow$ the solution t of $\int_a^t \frac{1}{C} f(y_i)\,\mathrm{d}y_i = U$
  if f is not a symbolic constant **then**
    $\textsc{SamplePoint}(f\{y_i \leftarrow Y_i\})$
  else **return** the Y_i's

⇒ Complexity is linear in the number of integrals

⇒ details (and example) in the paper of course!

(Micro-) Benchmark

Alternative potential title: $217028 \times 2 \cdot 10^{292431}$ states and beyond !
(joke!)

FJ size	$\sharp\mathcal{LE}$	FJ gen	(count)	BIT gen	(count)	CFTP gen
10	19	1.10^{-5} s	(2.10^{-4} s)	6.10^{-4} s	(0.03 s)	0.04 s
30	10^9	2.10^{-5} s	0(2.10^{-4} s)	0.02 s	(0.03 s)	1.8 s
40	$6 \cdot 10^6$	4.10^{-5} s	(3.10^{-4} s)	3.5 s	(5.2 s)	5.6 s
63	$4 \cdot 10^{29}$	5.10^{-4} s	(0.03 s)	Mem. crash	(Crash)	55 s
217028	$2 \cdot 10^{292431}$	8.11 s	(3.34 s)	Mem. crash	(Crash)	Timeout

Arch size	$\sharp\mathcal{LE}$	ARCH gen	(count)	BIT gen	(count)	CFTP gen
10:2	43	2.10^{-5} s	(4.10^{-5} s)	0.002 s	6.10^{-6} s)	0.04 s
30:2	$9.8 \cdot 10^8$	0.003 s	(0.0009 s)	7.10^{-6} s	(0.0004 s)	1.5 s
30:4	$6.9 \cdot 10^{10}$	0.001 s	(0.005 s)	7.10^{-5} s	(0.004 s)	2.5 s
100:2	$1.3 \cdot 10^{32}$	0.75 s	(0.16 s)	Mem. crash	(Crash)	[6] 5.6 s
100:32	$1 \cdot 10^{53}$	2.7 s	(0.17 s)	Mem. crash	(Crash)	[6] 5.9 s
200:66	10^{130}	54 s	(31 s)	Mem. crash	(Crash)	Timeout

(Micro-) Benchmark

Alternative potential title: $217028 \times 2 \cdot 10^{292431}$ states and beyond !

(joke!)

FJ size	$\sharp\mathcal{LE}$	FJ gen	(count)	BIT gen	(count)	CFTP gen
10	19	1.10^{-5} s	(2.10^{-4} s)	6.10^{-4} s	(0.03 s)	0.04 s
30	10^9	2.10^{-5} s	0(2.10^{-4} s)	0.02 s	(0.03 s)	1.8 s
40	$6 \cdot 10^6$	4.10^{-5} s	(3.10^{-4} s)	3.5 s	(5.2 s)	5.6 s
63	$4 \cdot 10^{29}$	5.10^{-4} s	(0.03 s)	Mem. crash	(Crash)	55 s
217028	$2 \cdot 10^{292431}$	8.11 s	(3.34 s)	Mem. crash	(Crash)	Timeout

Arch size	$\sharp\mathcal{LE}$	ARCH gen	(count)	BIT gen	(count)	CFTP gen
10:2	43	2.10^{-5} s	(4.10^{-5} s)	0.002 s	6.10^{-6} s)	0.04 s
30:2	$9.8 \cdot 10^8$	0.003 s	(0.0009 s)	7.10^{-6} s	(0.0004 s)	1.5 s
30:4	$6.9 \cdot 10^{10}$	0.001 s	(0.005 s)	7.10^{-5} s	(0.004 s)	2.5 s
100:2	$1.3 \cdot 10^{32}$	0.75 s	(0.16 s)	Mem. crash	(Crash)	6 5.6 s
100:32	$1 \cdot 10^{53}$	2.7 s	(0.17 s)	Mem. crash	(Crash)	6 5.9 s
200:66	10^{130}	54 s	(31 s)	Mem. crash	(Crash)	Timeout

$\Rightarrow$ All the (unoptimized Python) code available at

`https://gitlab.com/ParComb/combinatorics-barrier-synchro`

Conclusion (1)

The good parts

- The combinatorics tools are very sharp and characterize concurrency aspects in a very concrete way, the BIT-decomposition is IMHO a nice example of this.
- The geometrical interpretation (polytopes, etc.) is quite insightful, we only scratched the surface...
- The counting and random generation algorithms we propose apply directly on the control graphs or processes, there is no explicit construction of the state-space

Conclusion (2)

The bad parts

- The curse of expressivity: combinatorics tools are *so* sharp that they simply cannot apply on too complex structures
  (but you know when you cross the line)
  - Non-determinism *and* synchronization? (ongoing work)
    $\Rightarrow$ FSTCS'13: *The Combinatorics of non-determininism* (beautiful paper!)
  - Iteration? Recursion? (idea: unfolding of sizes, ...)

Conclusion (2)

The bad parts

- The curse of expressivity: combinatorics tools are *so* sharp that they simply cannot apply on too complex structures
  (but you know when you cross the line)
  - Non-determinism *and* synchronization? (ongoing work)
    $\Rightarrow$ FSTCS'13: *The Combinatorics of non-determininism* (beautiful paper!)
  - Iteration? Recursion? (idea: unfolding of sizes, ...)

$\Rightarrow$ someday we'll handle actual Petri Nets (at least some interesting subclasses), we'll tell you!

Conclusion (2)

The bad parts

- The curse of expressivity: combinatorics tools are *so* sharp that they simply cannot apply on too complex structures
  (but you know when you cross the line)
  - Non-determinism *and* synchronization? (ongoing work)
    $\Rightarrow$ FSTCS'13: *The Combinatorics of non-determininism* (beautiful paper!)
  - Iteration? Recursion? (idea: unfolding of sizes, ...)

$\Rightarrow$ someday we'll handle actual Petri Nets (at least some interesting subclasses), we'll tell you!

Thank you! Any question?