Statistical Analysis of Non-Deterministic Fork-Join Processes

> Martin Pépin Antoine Genitrini Frédéric Peschanski

December 3, 2020

Sorbonne Université — LIP6 — Paris

Concurrency

One computation unit shared by several processes:

- Possible dependencies between processes
- Scheduling

Concurrency

One computation unit shared by several processes:

- Possible dependencies between processes
- Scheduling

How to ensure that a program is correct regardless of the scheduling?

Concurrency

One computation unit shared by several processes:

- Possible dependencies between processes
- Scheduling

How to ensure that a program is correct regardless of the **scheduling**?

- Many possible schedulings: combinatorial explosion.

Concurrency

One computation unit shared by several processes:

- Possible dependencies between processes
- Scheduling

How to ensure that a program is correct regardless of the **scheduling**?

- Many possible schedulings: combinatorial explosion.
- Can we (efficiently) count them?
- Can we (efficiently) sample uniformly among them?

A negative result

[Brightwell & Winkler '91]

Counting the linear extensions of a partial order is a #-*P* complete problem.

I.e. it is as hard as counting the number of solutions in SAT.

So we cannot count efficiently...

A negative result

[Brightwell & Winkler '91]

Counting the linear extensions of a partial order is a #-*P* complete problem.

I.e. it is as hard as counting the number of solutions in SAT.

So we cannot count efficiently... in the **general** case.
But we can have some restrictions on the programs.

The long-term project

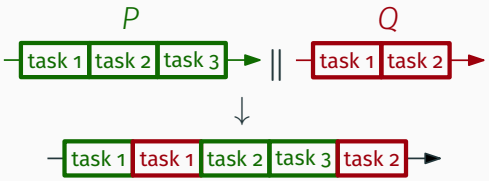
"Quantitative and algorithmic aspects of concurrency"

- Olivier Bodini, Matthieu Dien, Antoine Genitrini, Martin Pépin, Frédéric Peschanski, ...
- Identify **fundamental** components of concurrency and **interpret** them as combinatorial objects
- Algorithmic solutions for the **counting** and **sampling** problems
- Analytical results (when possible)

Outline

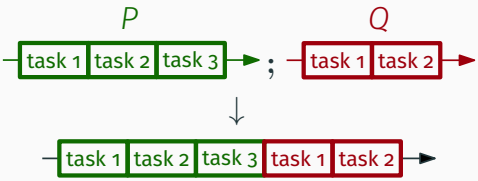
Fork-Join parallelism

Parallel composition



Execution = any interleaving of an execution of P and an execution of Q.

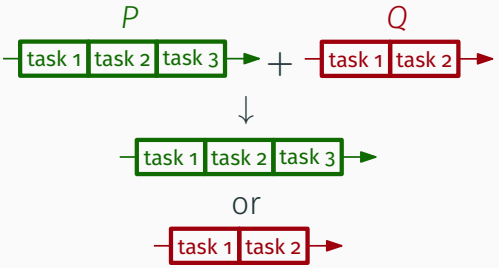
Sequential composition



Execution = an execution of P followed by an execution of Q.

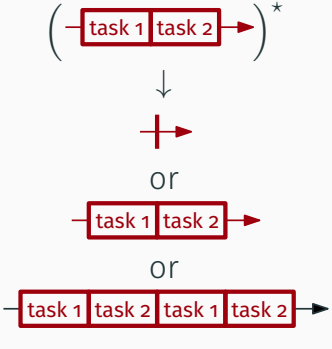
Non-determinism and loops

Non-deterministic choice



Execution = an execution of P or an execution of Q.

Loop



Execution = sequence of executions of Q

Non-deterministic Fork-Join programs (NFJ)

$$
\begin{array}{rcll}
P, Q & ::= & P \parallel Q & \text{(parallel composition)} \\
& | & P; Q & \text{(sequential composition)} \\
& | & P + Q & \text{(non-deterministic choice)} \\
& | & P^{\star} & \text{(loop)} \\
& | & a & \text{(atomic action)} \\
& | & 0 & \text{(empty program)}
\end{array}
$$

Non-deterministic Fork-Join programs (NFJ)

$$
\begin{array}{rcll}
P, Q & ::= & P \parallel Q & \text{(parallel composition)} \\
& | & P; Q & \text{(sequential composition)} \\
& | & P + Q & \text{(non-deterministic choice)} \\
& | & P^{\star} & \text{(loop)} \\
& | & a & \text{(atomic action)} \\
& | & 0 & \text{(empty program)}
\end{array}
$$

> All atomic actions accross a program are **distinct**.

Combinatorial interpretation

Define the executions of P as a combinatorial class $[\![P]\!]$:

$$[\![0]\!] = \mathcal{E}$$
$$[\![a]\!] = \mathcal{Z}$$

Combinatorial interpretation

Define the executions of P as a combinatorial class $[\![P]\!]$:

$$
\begin{aligned}
[\![0]\!] &= \mathcal{E} \\
[\![a]\!] &= \mathcal{Z} \\
[\![P; Q]\!] &= [\![P]\!] \times [\![Q]\!] \\
[\![P \parallel Q]\!] &= [\![P]\!] \star [\![Q]\!]
\end{aligned}
$$

Combinatorial interpretation

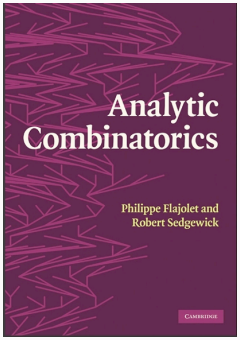
Define the executions of P as a combinatorial class $[\![P]\!]$:

$$
\begin{aligned}
[\![0]\!] &= \mathcal{E} \\
[\![a]\!] &= \mathcal{Z} \\
[\![P; Q]\!] &= [\![P]\!] \times [\![Q]\!] \\
[\![P \parallel Q]\!] &= [\![P]\!] \star [\![Q]\!] \\
[\![P + Q]\!]_{\neq 0} &= [\![P]\!]_{\neq 0} + [\![Q]\!]_{\neq 0} \\
[\![P^\star]\!] &= \textsc{Seq}\left([\![P]\!]_{\neq 0}\right)
\end{aligned}
$$

Combinatorial interpretation

Define the executions of P as a combinatorial class $[\![P]\!]$:

$$
\begin{aligned}
[\![0]\!] &= \mathcal{E} \\
[\![a]\!] &= \mathcal{Z} \\
[\![P; Q]\!] &= [\![P]\!] \times [\![Q]\!] \\
[\![P \parallel Q]\!] &= [\![P]\!] \star [\![Q]\!] \\
[\![P + Q]\!]_{\neq 0} &= [\![P]\!]_{\neq 0} + [\![Q]\!]_{\neq 0} \\
[\![P^\star]\!] &= \textsc{Seq}\left([\![P]\!]_{\neq 0}\right)
\end{aligned}
$$



"If you can specify it, you can analyse it"

Outline

Counting executions

Algorithm: $P \xrightarrow{\text{prev. slide}} [\![P]\!] \xrightarrow{\text{symbolic method}} GF \xrightarrow{[z^n]} \text{count}$

Counting executions

$$\textbf{Algorithm: } P \xrightarrow{\text{prev. slide}} [\![P]\!] \xrightarrow{\text{symbolic method}} GF \xrightarrow{[z^n]} \text{count}$$

Theorem

The counting algorithm performs $O(|P|M(n))$ arithmetic operations on big integers.

The coefficients of the polynomial have $O(n \ln n)$ bits.

- $|P|$ is the syntactic size of P.
- $M(n)$ is the cost of the multiplication of two polynomials of degree n.

Counting executions

$$\textbf{Algorithm: } P \xrightarrow{\text{prev. slide}} [\![P]\!] \xrightarrow{\text{symbolic method}} GF \xrightarrow{[z^n]} \text{count}$$

Theorem

The counting algorithm performs $O(|P|M(n))$ arithmetic operations on big integers.

The coefficients of the polynomial have $O(n \ln n)$ bits.

- $|P|$ is the syntactic size of P.
- $M(n)$ is the cost of the multiplication of two polynomials of degree n.

$$\Longrightarrow O(|P|M(n)M(n \ln n)) \text{ bit-complexity.}$$

Random sampling — the recursive method

Algorithm: $P \xrightarrow{\text{prev. slides}} [\![P]\!] \xrightarrow[[FZC'93]]{\text{recursive method}}$ uniform execution

Random sampling — the recursive method

Algorithm: $P \overset{\text{prev. slides}}{\longrightarrow} [\![P]\!] \overset{\text{recursive method}}{\underset{[FZC'93]}{\longrightarrow}}$ uniform execution

---

$$\textsc{sample}((a+b)^\star \parallel (c+(d;e)+(f;g)), 3)$$

Rule:

Random sampling — the recursive method

Algorithm: $P \xrightarrow{\text{prev. slides}} [\![P]\!] \xrightarrow[[FZC'93]]{\text{recursive method}}$ uniform execution

---

$$\textsc{sample}((a+b)^\star \parallel (c+(d;e)+(f;g)), 3)$$

Rule:

$$P_n = Q_0 R_n \binom{n}{0} + Q_1 R_{n-1} \binom{n}{1} + Q_2 R_{n-2} \binom{n}{2} + \cdots + Q_n R_0 \binom{n}{n}$$

Pick $k \in [\![0; n]\!]$ with probability $Q_k R_{n-k} \binom{n}{k} / P_n$

$$1 \cdot 0 \cdot \binom{3}{0} + 2 \cdot 2 \cdot \binom{3}{1} + 4 \cdot 1 \cdot \binom{3}{2} + 8 \cdot 0 \cdot \binom{3}{3} = 24 \cdot (0 + 1/2 + 1/2 + 0)$$

Random sampling — the recursive method

Algorithm: $P \xrightarrow{\text{prev. slides}} [\![P]\!] \xrightarrow[[FZC'93]]{\text{recursive method}}$ uniform execution

---

$\textsc{shuffle}(\textsc{sample}((a+b)^\star, 1), \textsc{sample}((c+(d;e)+(f;g)), 2))$

Rule:

$$1 \cdot 0 \cdot \binom{3}{0} + 2 \cdot 2 \cdot \binom{3}{1} + 4 \cdot 1 \cdot \binom{3}{2} + 8 \cdot 0 \cdot \binom{3}{3} = 24 \cdot (0 + 1/2 + 1/2 + 0)$$

Random sampling — the recursive method

Algorithm: $P \overset{\text{prev. slides}}{\longrightarrow} [\![P]\!] \overset{\text{recursive method}}{\underset{[FZC'93]}{\longrightarrow}}$ uniform execution

---

$\text{SHUFFLE}(\text{SAMPLE}((a+b)^\star, 1), \ldots)$

Rule:
$$P^\star \to 0 + P; P^\star$$

$$1 \cdot 0 \cdot \binom{3}{0} + 2 \cdot 2 \cdot \binom{3}{1} + 4 \cdot 1 \cdot \binom{3}{2} + 8 \cdot 0 \cdot \binom{3}{3} = 24 \cdot (0 + 1/2 + 1/2 + 0)$$

Random sampling — the recursive method

Algorithm: $P \overset{\text{prev. slides}}{\longrightarrow} [\![P]\!] \overset{\text{recursive method}}{\underset{[FZC'93]}{\longrightarrow}}$ uniform execution

---

$\text{SHUFFLE}(\text{SAMPLE}(0 + (a + b); (a + b)^\star, 1), \ldots)$

Rule:
$$P^\star \to 0 + P; P^\star$$

$$1 \cdot 0 \cdot \binom{3}{0} + 2 \cdot 2 \cdot \binom{3}{1} + 4 \cdot 1 \cdot \binom{3}{2} + 8 \cdot 0 \cdot \binom{3}{3} = 24 \cdot (0 + 1/2 + 1/2 + 0)$$

Random sampling — the recursive method

Algorithm: $P \overset{\text{prev. slides}}{\longrightarrow} [\![P]\!] \overset{\text{recursive method}}{\underset{[FZC'93]}{\longrightarrow}}$ uniform execution

---

$\text{SHUFFLE}(\text{SAMPLE}((a+b); (a+b)^\star, 1), \ldots)$

Rule:

$$1 \cdot 0 \cdot \binom{3}{0} + 2 \cdot 2 \cdot \binom{3}{1} + 4 \cdot 1 \cdot \binom{3}{2} + 8 \cdot 0 \cdot \binom{3}{3} = 24 \cdot (0 + 1/2 + 1/2 + 0)$$

Random sampling — the recursive method

Algorithm: $P \xrightarrow{\text{prev. slides}} [\![P]\!] \xrightarrow[[FZC'93]]{\text{recursive method}}$ uniform execution

---

$$\text{SHUFFLE}(\text{SAMPLE}((a+b); (a+b)^\star, 1), \dots)$$

Rule:

$$P_n = Q_0 R_n + Q_1 R_{n-1} + Q_2 R_{n-2} + \cdots + Q_n R_0$$

Pick $k \in [\![0; n]\!]$ with probability $Q_k R_{n-k} / P_n$

$$1 \cdot 0 \cdot \binom{3}{0} + 2 \cdot 2 \cdot \binom{3}{1} + 4 \cdot 1 \cdot \binom{3}{2} + 8 \cdot 0 \cdot \binom{3}{3} = 24 \cdot (0 + 1/2 + 1/2 + 0)$$

$$0 \cdot 1 + 2 \cdot 1 = 2 \cdot (0 + 1)$$

Random sampling — the recursive method

Algorithm: $P \xrightarrow{\text{prev. slides}} [\![P]\!] \xrightarrow[\text{[FZC'93]}]{\text{recursive method}}$ uniform execution

---

$\text{SHUFFLE}(\text{SAMPLE}(a+b, 1), \ldots)$

Rule:

$1 \cdot 0 \cdot \binom{3}{0} + 2 \cdot 2 \cdot \binom{3}{1} + 4 \cdot 1 \cdot \binom{3}{2} + 8 \cdot 0 \cdot \binom{3}{3} = 24 \cdot (0 + 1/2 + 1/2 + 0)$

$0 \cdot 1 + 2 \cdot 1 = 2 \cdot (0 + 1)$

Random sampling — the recursive method

Algorithm: $P \xrightarrow{\text{prev. slides}} [\![P]\!] \xrightarrow[[FZC'93]]{\text{recursive method}}$ uniform execution

---

$$\text{SHUFFLE}(\text{SAMPLE}(a+b, 1), \ldots)$$

Rule:

$P_n = Q_n + R_n$

Choose Q with probability Q_n/P_n

$$1 \cdot 0 \cdot \binom{3}{0} + 2 \cdot 2 \cdot \binom{3}{1} + 4 \cdot 1 \cdot \binom{3}{2} + 8 \cdot 0 \cdot \binom{3}{3} = 24 \cdot (0 + 1/2 + 1/2 + 0)$$

$$0 \cdot 1 + 2 \cdot 1 = 2 \cdot (0 + 1)$$

$$1 + 1 = 2 \cdot (1/2 + 1/2)$$

Random sampling — the recursive method

Algorithm: $P \overset{\text{prev. slides}}{\longrightarrow} [\![P]\!] \overset{\text{recursive method}}{\underset{[FZC'93]}{\longrightarrow}}$ uniform execution

---

$\text{SHUFFLE}(a, \ldots)$

Rule:

$1 \cdot 0 \cdot \binom{3}{0} + 2 \cdot 2 \cdot \binom{3}{1} + 4 \cdot 1 \cdot \binom{3}{2} + 8 \cdot 0 \cdot \binom{3}{3} = 24 \cdot (0 + 1/2 + 1/2 + 0)$

$0 \cdot 1 + 2 \cdot 1 = 2 \cdot (0 + 1)$

$1 + 1 = 2 \cdot (1/2 + 1/2)$

Random sampling — the recursive method

Algorithm: $P \xrightarrow{\text{prev. slides}} [\![P]\!] \xrightarrow[[FZC'93]]{\text{recursive method}}$ uniform execution

---

$$\textsc{shuffle}(a, \textsc{sample}((c + (d; e) + (f; g)), 2))$$

Rule:

$$1 \cdot 0 \cdot \binom{3}{0} + 2 \cdot 2 \cdot \binom{3}{1} + 4 \cdot 1 \cdot \binom{3}{2} + 8 \cdot 0 \cdot \binom{3}{3} = 24 \cdot (0 + 1/2 + 1/2 + 0)$$

$$0 \cdot 1 + 2 \cdot 1 = 2 \cdot (0 + 1)$$

$$1 + 1 = 2 \cdot (1/2 + 1/2)$$

Random sampling — the recursive method

Algorithm: $P \xrightarrow{\text{prev. slides}} [\![P]\!] \xrightarrow[[FZC'93]]{\text{recursive method}}$ uniform execution

---

$\textsc{shuffle}(a, de)$

Rule:

$1 \cdot 0 \cdot \binom{3}{0} + 2 \cdot 2 \cdot \binom{3}{1} + 4 \cdot 1 \cdot \binom{3}{2} + 8 \cdot 0 \cdot \binom{3}{3} = 24 \cdot (0 + 1/2 + 1/2 + 0)$

$0 \cdot 1 + 2 \cdot 1 = 2 \cdot (0 + 1)$

$1 + 1 = 2 \cdot (1/2 + 1/2)$

Random sampling — the recursive method

Algorithm: $P \overset{\text{prev. slides}}{\longrightarrow} [\![P]\!] \overset{\text{recursive method}}{\underset{[FZC'93]}{\longrightarrow}}$ uniform execution

---

dae

Rule:

$1 \cdot 0 \cdot \binom{3}{0} + 2 \cdot 2 \cdot \binom{3}{1} + 4 \cdot 1 \cdot \binom{3}{2} + 8 \cdot 0 \cdot \binom{3}{3} = 24 \cdot (0 + 1/2 + 1/2 + 0)$

$0 \cdot 1 + 2 \cdot 1 = 2 \cdot (0 + 1)$

$1 + 1 = 2 \cdot (1/2 + 1/2)$

Random sampling — complexity

Theorem

Random sampling of executions has complexity $O(n \cdot \min(h(P), \ln n))$ where h denotes the "height" of P i.e. its maximum number of nested constructors.

The $O(n \ln n)$ bound is an consequence of [FZC'93,Molinero'05].

The algorithms in numbers

Counting

$\lvert P\rvert$	n	# exec° $\leq n$	runtime
500	500	$1.058 \cdot 2^{1927}$	0.076s
500	1000	$1.081 \cdot 2^{3832}$	0.462s
500	3000	$1.341 \cdot 2^{11423}$	6.428s
1000	500	$1.473 \cdot 2^{2330}$	0.159s
1000	1000	$1.044 \cdot 2^{4712}$	0.874s
1000	3000	$1.092 \cdot 2^{14181}$	13.488s
2000	500	$1.768 \cdot 2^{2380}$	0.330s
2000	1000	$1.215 \cdot 2^{4746}$	1.870s
2000	3000	$1.699 \cdot 2^{14120}$	25.376s
5000	500	$1.607 \cdot 2^{2923}$	0.897s
5000	1000	$1.469 \cdot 2^{6016}$	5.434s
5000	3000	$1.226 \cdot 2^{18116}$	75.649s

Sampling

$\lvert P\rvert$	n	# exec° $= n$	UnifExec	IQR
500	500	$1.969 \cdot 2^{1926}$	0.218ms	5µs
500	1000	$1.004 \cdot 2^{3832}$	0.563ms	21µs
500	3000	$1.245 \cdot 2^{11423}$	3.718ms	203µs
1000	500	$1.420 \cdot 2^{2330}$	0.301ms	10µs
1000	1000	$1.005 \cdot 2^{4712}$	0.777ms	28µs
1000	3000	$1.051 \cdot 2^{14181}$	4.829ms	481µs
2000	500	$1.704 \cdot 2^{2380}$	0.308ms	14µs
2000	1000	$1.169 \cdot 2^{4746}$	1.021ms	51µs
2000	3000	$1.634 \cdot 2^{14120}$	7.291ms	1.2ms
5000	500	$1.589 \cdot 2^{2923}$	0.309ms	7µs
5000	1000	$1.448 \cdot 2^{6016}$	0.898ms	43µs
5000	3000	$1.208 \cdot 2^{18116}$	18.526ms	1.5ms

Outline

Conclusion

Take away:

- Applying methods from analytic combinatorics to concurrent problem can be effective

Conclusion

Take away:

- Applying methods from analytic combinatorics to concurrent problem can be effective

Implementation

- `https://gitlab.com/ParComb/libnfj`

Future work:

- Statistical model-checking
- Generalize the model

Thank you for your attention!

Execution prefixes

Program P	Prefix program $\mathrm{pref}(P)$	Specification of the prefixes $\langle P \rangle$
0	0	$\mathcal{E}$
a	$0 + a$	$\mathcal{E} + \mathcal{Z}$
$P \parallel Q$	$(\mathrm{pref}(P) \parallel \mathrm{pref}(Q))$	$\langle P \rangle \star \langle Q \rangle$
$P; Q$	$\mathrm{pref}(P) + (P; \mathrm{pref}(Q))$	$\langle P \rangle + S(P) \times (\langle Q \rangle \setminus \mathcal{E})$
$P + Q$	$\mathrm{pref}(P) + \mathrm{pref}(Q)$	$\langle P \rangle + (\langle Q \rangle \setminus \mathcal{E})$
$P^\star$	$P^\star; \mathrm{pref}(P)$	$\mathcal{E} + S(P^\star) \times (\langle P \rangle \setminus \mathcal{E})$

Prefix covering

Table 1: Expected number of prefixes to be sampled to discover 20% of the prefixes of a random program of size 25 with either the isotropic or the uniform method.

Prefix length	1	2	3	4	5
# prefixes	11	18	30	60	128
Isotropic	2.1	4.45	11.17	35.09	$1.28 \cdot 10^{14}$
Uniform	2.1	3.18	6.57	13.26	27.69
Gain	0%	40%	70%	165%	$4.61 \cdot 10^{14}$%