研究了一个名为 arch processes 的并发模型，导出了 run （执行可能性）的计数公式，给出了渐进结果，最后给了两个生成算法

作者1在AofA领域的发文很多，之前也研究了并发模型的计数，可以看参考文献

Beyond Series-Parallel Concurrent Systems: The Case of Arch Processes

Olivier Bodini
Laboratoire d'Informatique de Paris-Nord, CNRS UMR 7030 - Institut Galilée - Université Paris-Nord, 99, avenue Jean-Baptiste Clément, 93430 Villetaneuse, France.
Olivier.Bodini@lipn.univ-paris13.fr

Matthieu Dien
Institute of Statistical Science, Academia Sinica, Taipei 115, Taiwan.
dien@stat.sinica.edu.tw

Antoine Genitrini
Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6 -LIP6- UMR 7606, F-75005 Paris, France.
Antoine.Genitrini@lip6.fr

Alfredo Viola
Universidad de la República, Montevideo, Uruguay.
viola@fing.edu.uy

Abstract

In this paper we focus on concurrent processes built on *synchronization* by means of *futures*. This concept is an abstraction for processes based on a main execution thread but allowing to delay some computations. The structure of a general concurrent process is a directed acyclic graph (DAG). Since the quantitative study of increasingly labeled DAG (directly related to processes) seems out of reach (this is a ♯P-complete problem), we restrict ourselves to the study of *arch processes*, a simplistic model of processes with futures. They are based on two parameters related to their sizes and their numbers of arches. The increasingly labeled structures seems not to be specifiable in the classical sense of Analytic Combinatorics, but we manage to derive a recurrence equation for the enumeration.

有向无环图

For this model we first exhibit an *exact* and an *asymptotic* formula for the number of runs of a given process. The second main contribution is composed of a *uniform random sampler* algorithm and an *unranking* one that allow efficient generation and exhaustive enumeration of the runs of a given arch process.

精确计数，近似公式，均匀分布采样和生成

2012 ACM Subject Classification Mathematics of computing → Generating functions, Theory of computation → Generating random combinatorial structures, Theory of computation → Concurrency

Keywords and phrases Concurrency Theory, Future, Uniform Random Sampling, Unranking, Analytic Combinatorics

Digital Object Identifier 10.4230/LIPIcs.AofA.2018.14

Funding This research was partially supported by the ANR MetACOnc project ANR-15-CE40-0014.

29th International Conference on Probabilistic, Combinatorial and Asymptotic Methods for the Analysis of Algorithms (AofA 2018).
Editors: James Allen Fill and Mark Daniel Ward; Article No. 14; pp. 14:1–14:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

并发进程

Concurrent processes are logic units running independently in the same environment which share resources (processing time, file inputs or outputs, etc). To guarantee the good behavior of a concurrent program (a set of processes), a mechanism of *synchronization* has to be set up. For example, synchronization can be used to avoid a process to write in a file currently read by another process.

To deal with concurrent programs, researchers in the concurrency theory community formalize the programs in an abstract language called *process algebra*. Different formalisms exhibit different properties of concurrent programs. We are mainly interested in the so-called *Calculus of Communicating Systems* introduced in [18], because of its popularity in concurrency theory and the simplicity to reason about.

In this context, one of the main goals in concurrency is to check the good behavior of such programs. A very popular method to do the verification is the *model checking*: several logical properties (the specification of the program behavior) are checked for all the possible runs of the program (see [3] as a reference book).

A common problem in such a method is the *combinatorial explosion* phenomenon: the huge number of runs to check. To deal with that explosion, a statistical method has been introduced: the *Monte-Carlo model checking* (see [13]). Here, the idea is to check the specification only for few runs randomly sampled. Thus, the result of the method is not anymore a proof of good behavior but a statistical certificate.

In this paper we investigate this phenomenon from a combinatorial point of view. We consider concurrent programs as sets of *atomic* actions (executed computations) constrained by a partial order relation: some actions have to finish before others start their computations. Thus in this setting, a concurrent program can be modeled as a partial order (*a.k.a.* poset). Then, its runs (possible execution flows) are its linear extensions (i.e. the total orders compatible with the partial order) and the combinatorial explosion phenomenon (for a given family of posets) is the fast growth of the number linear extensions as its number of atomic actions increases.

The problem of counting the number of linear extensions of a poset is known to be ♯P-complete [10]. As a consequence, an analytic approach to study this counting problem for general posets seems out of reach. We thus limit the difficulty by considering restricted classes of partial orders. In previous works we dealt with tree-like processes [9], tree-like processes with non-deterministic choice [8] and Series-Parallel processes [5, 7]. In these papers, like in the present one, we take the point of view to model a partial order by its covering directed acyclic graph – DAG – (*a.k.a.* Hasse diagram). Then a linear extension becomes an increasing labeling of the covering DAG. Previously this consideration let us to use symbolic method to specify our models and so to use tools of Analytic Combinatorics (see [11]).

In the present work we focus on processes built on *synchronization* by means of *futures* or *promises* (see [4]). This concept is an abstraction for processes based on a main execution thread but allowing to delay some computations. These computations are run asynchronously and are represented as an object that can be queried in two ways: `finish?` to know if the computation has terminated and `get` to retrieve the result of the computation (and properly proceed the synchronization). This quite old principle aroses recently in many programming languages, especially in the very popular Javascript language (see [1]).

To emphasize these paradigm we consider *arch processes*: a simplistic model of processes with futures. An arch process is composed of a main *trunk* from which start several *arches*

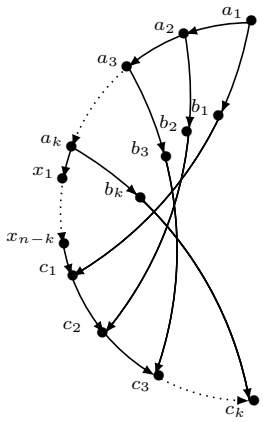

Figure 1 The (n, k)-arch process.

(modeling futures). The general shape of such a process is given in the Figure 1. Arch processes are based on two parameters related to their sizes and their numbers of arches. A combinatorial and recursive specification (as in [11]) for these increasing labeled structures seems out of the reach at the moment. As a consequence we present here a different approach to specify the problem.

For this model we exhibit *exact* and *asymptotic* formula for the number of increasing labelings.As a second main contribution is the design of two algorithms. The first one is an uniform random sampler for runs of a given arch process and the second one is an *unranking* algorithm which allows to obtain an exhaustive builder of runs. The design of these algorithms is motivated by the possible applications to (statistical) model checking.

The paper is organized as follows. The next section is devoted to the formal description of (n, k)-arch processes and gives the solution of the recurrence equation driving their numbers of runs. In Section 3 we prove the algebraicity of the bivariate generating function, we give a closed form formula for it and the asymptotic behaviors of the diagonal coefficients of the functions. Section 4 carefully describes both algorithms.

2 The arch processes and their runs

A concurrent program is seen as a *partially ordered set (poset)* of atomic actions where the order relation define the precedence constraints over the executions of the actions.
A run of a concurrent program is a *linear extension* of the corresponding poset: i.e. a total order compatible with the partial order relation.

Note that many other models of concurrent program exist but we have chosen to use this one because it is well-suited to study the combinatorial explosion phenomenon.
We introduce now the model of arch processes, a family of restricted concurrent program encoding synchronization by means of futures.

▶ **Definition 1.** Let n and k be two positive integers with $k \leq n+1$. The (n, k)-*arch process*, denoted by $A_{n,k}$, is built in the following way:

- the trunk of the process: a sequence of $(n + k)$ actions $a_1, \ldots, a_k, x_1, \ldots, x_{n-k}, c_1, \ldots, c_k$ and represented in Figure 1 on a semicircle;
- the k arches that correspond to the triplets, for all $i \in \{1, \ldots, k\}$, $a_i \to b_i \to c_i$.

Thus k is the number of arches in the process, and n is the length (along the trunk) between both extremities of each arch a_i and c_i (for all i). There are two extreme cases: when $k = n$,

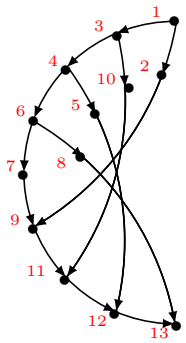

Figure 2 A run of the $(5, 4)$-arch process.

it corresponds to the arch processes that do not contain any node x_i in the trunk, and the case $k = n + 1$ that corresponds to the case where both the nodes a_k and c_1 are merged into a single node (and thus there is no node x_i).

In Figure 1 representing the (n, k)-arch process, the precedence constraints are encoded with the directed edges such that $a \to b$ means that the action a precedes b. We remark that the (n, k)-arch process contains exactly $(n + 2k)$ actions. Due to the intertwining of the arches, we immediately observe when k is larger than 1 then the arch processes are not Series-Parallel processes. Hence the results we exhibited in our papers [6, 7] cannot be applied in this context.

▶ **Definition 2.** An *increasing labeling* for a concurrent process containing ℓ actions is a bijection between the integers $\{1, \ldots, \ell\}$ and the actions of the process, satisfying the following constraint: if an action a precedes an action b then the label associated to a is smaller than the one related to b.

给每个节点标号，满足有向图定义的偏序

In Figure 2 we have represented an increasing labeling of the $(5, 4)$-arch process $A_{5,4}$ corresponding to the run $\langle a_1, b_1, a_2, a_3, b_3, a_4, x_1, b_4, c_1, b_2, c_2, c_3, c_4\rangle$. As one can see, every directed path (induced by the precedence relation) is increasingly labeled. Our quantitative goal is to calculate the number of runs for a given arch process.

▶ **Proposition 3.** *The number of runs of a concurrent process is the number of increasing labelings of the actions of the process.*

Thus, each increasing labeling is in bijection with a single linear extension.

While there is the classical hook-length formula for tree-processes [15, 9] and its generalization for Series-Parallel processes [6], to the best of our knowledge, no closed form formula is known for more general classes of processes. In the rest of the paper, for a given process A, we denote by $\sigma(A)$ its number of runs.

First, let us easily exhibit a lower bound and an upper bound (in the case $k < n + 1$) in order to obtain a first idea for the growth of the numbers of runs for the arch processes. We remark that a similar approach could be used for the case when $k = n+1$. We first enumerate the runs where all the b_i nodes are preceded by a_k, and all of them precede the node c_1. This imposes new precedence constraints for the process, and thus its number of runs is a lower bound for the total number of runs. In this case the b_i's permute without any constraint, i.e. $k!$ possibilities and then each permutation of the b_i's shuffles with the sequence $x_1, \ldots, x_{n-k}$. Thus we get the following lower bound for the number of runs of $A_{n,k}$:

平凡的上下界

$$\sigma(A_{n,k}) \geq k! \binom{k + n - k}{k} = \frac{n!}{(n - k)!}.$$

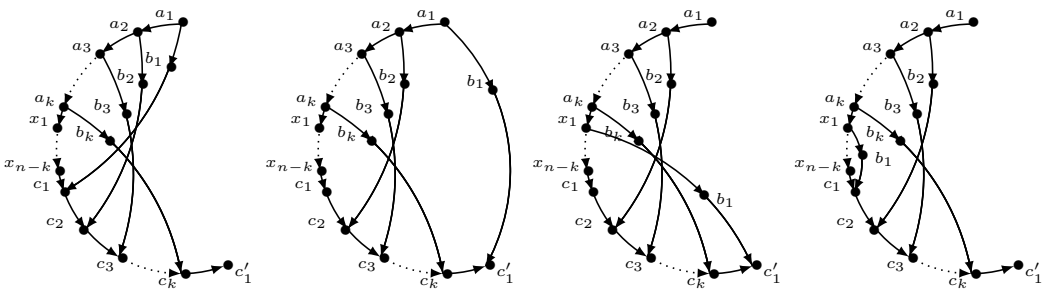

Figure 3 From left to right, the processes denoted $D_{n,k}$, $\overline{D}_{n,k}$, $\overline{D}^1_{n,k}$ and $\overline{D}^2_{n,k}$.

We now focus on an upper bound for the number of runs of $A_{n,k}$. Here again we suppose that all the permutations of the b_i's are possible, but we allow each b_i to appear everywhere between a_1 and c_k. This constraint is satisfied by all the runs, but some possibilities are not valid runs: thus we are computing an upper bound. Once the permutation of the b_i's is calculated, we shuffle it into the trunk (containing $n+k$ nodes):

$$\sigma(A_{n,k}) \leq k! \binom{k+n+k-1}{n+k-1} = \frac{(n+2k-1)!}{(n+k-1)!}.$$

A refinement of these ideas for the bounds computation allows to exhibit a recurrence formula for the value $\sigma(A_{n,k})$.

▶ **Theorem 4.** *Let n and k be two integers such that $0 \leq k \leq n+1$. The number $\sigma(A_{n,k})$ of runs of the process $A_{n,k}$ is equal to $t_{n,k}$ that satisfies:*

$$t_{n,k} = \frac{n+2k-1}{2} t_{n,k-1} + \frac{n-k}{2} t_{n+1,k-1} \quad \text{and} \quad t_{n,0} = 1. \tag{1}$$

In order to provide the proof, we first introduce the four processes in Figure 3. Notice that they are not arch processes. From left to right, the first process, denoted by $D_{n,k}$, is almost the process $A_{n,k}$. In fact, the single difference is that $D_{n,k}$ contains exactly one more action, denoted by c'_1, that is preceded by all the other actions. The second process $\overline{D}_{n,k}$ is related to $D_{n,k}$ in the following way: the precedence relation starting at b_1 is replaced, instead of having $b_1 \to c_1$, it is $b_1 \to c'_1$. Finally, for the two last processes $\overline{D}^1_{n,k}$ and $\overline{D}^2_{n,k}$, it is also the relations $a_1 \to b_1 \to c_1$ which are modified.

Proof. The extreme case $A_{n,0}$ corresponds to a process without any arch: just a trunk. Obviously it admits a single increasing labeling: it has a single run.

Suppose first that $k < n+1$. The number $\sigma(A_{n,k})$ is equal to the number of runs $\sigma(D_{n,k})$ because for all runs, the integer associated to c'_1 is inevitably the largest one: $2k+n+1$. Then, using a inclusion/exclusion principle, we obtain the following formula for the number $\sigma(D_{n,k})$:

$$\sigma(D_{n,k}) = \sigma(\overline{D}_{n,k}) - \left(\sigma(\overline{D}^1_{n,k}) - \sigma(\overline{D}^2_{n,k})\right). \tag{2}$$

In fact we are focusing on the action preceded by b_1. In $D_{n,k}$ it corresponds to c_1. By modifying it to c'_1 in $\overline{D}_{n,k}$ we allow runs where b_1 appears after c_1, thus that are not valid for $D_{n,k}$. We remove this number of non-valid runs with $\sigma(\overline{D}^1_{n,k}) - \sigma(\overline{D}^2_{n,k})$, by playing with both actions x_1 and c_1. To compute $\sigma(\overline{D}_{n,k})$, first omit the action b_1 (and its incoming

and outgoing edges) ; the remaining process is a $(n, k-1)$-arch process, up to renaming, with added top and bottom actions (a_1 and c'_1) which do not modify the number of runs of $A_{n,k-1}$. It remains to insert b_1 in this "almost" $A_{n,k-1}$, somewhere between a_1 and c'_1: there are $(2 \cdot (k-1) + n - 1) + 2 = 2k + n - 1$ possibilities. The term $(2 \cdot (k-1) + n - 1)$ are the cases where b_1 is put between a_2 and c_k and the term 2 corresponds to the cases where b_1 is either before a_2 or after c_k. The process $\overline{D}^1_{n,k}$ is similar to the arch process $A_{n,k}$, there is only an action a_1 that precedes it, so $\sigma(\overline{D}^1_{n,k}) = t_{n,k}$. Lastly, for the process $\sigma(\overline{D}^2_{n,k})$, forgetting b_1 we recognize $A_{n+1,k-1}$ up to renaming, so b_1 can be inserted between x_1 and c_1: there are $n - k$ possibilities. Finally we obtain the following equation

$$\sigma(A_{n,k}) = (n + 2k - 1) \cdot \sigma(A_{n,k-1}) - \sigma(A_{n,k}) + (n - k) \cdot \sigma(A_{n+1,k-1}).$$

Suppose now that $k = n+1$. Here there is no action x_i and both the nodes a_k and c_1 are merged into a single node. We can adapt equation (2) and obtain the same recurrence, but via a small difference in the computation: $\sigma(A_{k-1,k}) = 3k \cdot \sigma(A_{k-1,k-1}) - \sigma(A_{k-1,k}) - \sigma(A_{k,k-1})$. But since $k = n+1$, this recurrence is equal to equation (1) too. ◀

When $k \geq n+1$, one can think to the arch process $A_{n,k}$ as an arch process where the last $(k-n)$ actions a_{n-i} are merged with the first $(k-n)$ actions c_i. But the recursive formula (1) does not apply to such models: once $k > n+1$ the recurrence loses its combinatorial meaning.

The next result exhibits a closed form formula for the number of runs of the arch processes.

▶ **Theorem 5.** *Let n and k be integers such that $0 < k \leq n+1$. The number [1] of runs of the (n,k)-arch process is*

$$\sigma(A_{n,k}) = \frac{(2k+n-1)!!}{2^{k-1}} \sum_{s=0}^{k-1} \frac{(n+s)\ \mathit{par}(n,s)}{(n+s+1)!!} \sum_{1 \leq i_1 < \cdots < i_s \leq k} \prod_{j=1}^{s} (i_j + j + n - k - 1) \frac{\Gamma\left(\frac{2(k-i_j)+n+j+2}{2}\right)}{\Gamma\left(\frac{2(k-i_j)+n+j+3}{2}\right)},$$

$$\text{where } \mathit{par}(n,s) = \begin{cases} (2^{s/2})^{-1} & \text{if } s \text{ is even} \\ \sqrt{\pi}(2^{(s+1)/2})^{-1} & \text{if } s \text{ is odd and } n \text{ is even} \\ (2^{(s-1)/2}\sqrt{\pi})^{-1} & \text{if } s \text{ is odd and } n \text{ is odd.} \end{cases}$$

这种 closed form 有点复杂

Let us recall the double factorial notation: for $n \in \mathbb{N}$, $n!! = n \cdot (n-2)!!$ with $0!! = 1!! = 1$. We remark that the ratio of the two Γ-functions is related to the central binomial coefficient. The asymptotic behavior of the sequence does not seem immediate to obtain using this formula.

key-ideas. The formula for $\sigma(A_{n,k})$ is obtained by resolving the recurrence stated in equation (1). First remark that the calculation of $\sigma(A_{n,k})$ requires the values of $\sigma(A_{i,j})$ in the triangle such that $n \leq i \leq n+k$ and $0 \leq j \leq k-(i-n)$. The formula is computed by unrolling k times the recurrence. In particular, the index s in the formula corresponds to the number of times we have used the second term of equation (1), to reach the final term $\sigma(A_{n+s,0})$. The i_j values indicate in which iteration the second terms of equation (1) have been chosen. They describe the path from (n,k) to $(n+s,0)$. The brute formula obtained in this way is composed of a product of truncated double factorials that can be written as ratios

[1] In Theorem 5 we use the convention that the sum over the sequence of i_j's is equal to 1 when $s = 0$.

of double factorial numbers. Finally, by coupling the adequate numerators and denominators in the product we exhibit several Wallis's ratios [2] that are easily simplified by using the Γ-function: $\frac{(2n-1)!!}{(2n)!!} = \frac{1}{\sqrt{\pi}} \frac{\Gamma\left(n+\frac{1}{2}\right)}{\Gamma(n+1)}$. ◀

By using this closed form formula, or the bivariate recurrence (cf. equation (1)), we easily compute the first diagonals of the recurrence. The values of a given diagonal correspond to the class of arch processes with the same number of actions x_i in the trunk.

$$
\begin{aligned}
(\sigma(A_{k-1,k}))_{k\in\mathbb{N}\setminus\{0,1\}} &= (1, 12, 170, 2940, 60760, 1466640, 40566680, 1266064800, \dots)\\
(\sigma(A_{k,k}))_{k\in\mathbb{N}^*} &= (1, 5, 44, 550, 8890, 176120, 4130000, 111856360, \dots)\\
(\sigma(A_{k+1,k}))_{k\in\mathbb{N}^*} &= (2, 11, 100, 1270, 20720, 413000, 9726640, 264279400, \dots)\\
(\sigma(A_{k+2,k}))_{k\in\mathbb{N}^*} &= (3, 19, 186, 2474, 41670, 850240, 20386800, 561863960, \dots)
\end{aligned}
$$

We remark that the first terms of the sequence $(\sigma(A_{k+1,k}))_{k\in\mathbb{N}^*}$ coincide with the first terms of the sequence `A220433` (shifted by 2) in OEIS [2] . This sequence is related to a specific Alia algebra and is exhibited in the paper of Khoroshkin and Piontkovski [14]. In their paper, the exponential univariate generating function naturally appears as an algebraic function. This motivates us to study in detail the bivariate generating function for $(t_{n,k})$ and in particular its diagonals.

3 Algebraic generating functions

Let us associate to the bivariate sequence $(t_{n,k})_{n,k}$ the generating function, denoted by $A(z,u)$, exponential in u and ordinary in z:

$$A(z,u) = \sum_{n\geq 0, k\geq 0} \frac{t_{n,k}}{k!} z^n u^k.$$

Recall this series enumerates the increasing labelings of the arch processes, when $k \leq n+1$, but has no combinatorial meaning beyond this bound.

▶ **Proposition 6.** *The bivariate generating function $A(z,u)$ is holonomic and satisfies the following differential equation.*

$$(2zu - 2z - u)\frac{\partial}{\partial u}A(z,u) + (z-2)A(z,u) + z(z+1)\frac{\partial}{\partial z}A(z,u) + C(u) = 0.$$

得到偏微分方程

where $C(u)$ is an algebraic function determined by the initial conditions of the equation.

The differential equation can be exhibited since the recursive behavior of $(t_{n,k})$ is not disturbed beyond the bound $k > n+1$.

key-ideas. The differential equation is directly obtained from the recurrence equation (1). The function $C(u)$ encodes the initial conditions of the equation. The differential equation satisfied by $A(z,u)$ ensures its holonomicity (cf. [21, 11]). ◀

[2] OEIS corresponds to the On-line Encyclopedia of Integer Sequences: `http://oeis.org/`.

It is important to remark that $C(u)$ is holonomic. In fact we have $C(u) = u\frac{\partial}{\partial u}A(0,u) + 2A(0,u)$ and consequently $C(u)$ is holonomic as a specialization of a holonomic bivariate generating function. A direct computation for $C(u)$ exhibits the following differential equation

$$\begin{aligned}
&4\left(24u^2+3u+1\right)C(u) - 4u\left(84u^2-3u+1\right)\frac{\mathrm{d}}{\mathrm{d}u}C(u)\\
&-2u^2\left(216u^2-151u+13\right)\frac{\mathrm{d}^2}{\mathrm{d}u^2}C(u)\\
&-2u^2\left(58u^3-75u^2+33u-2\right)\frac{\mathrm{d}^3}{\mathrm{d}u^3}C(u)\\
&-u^3\left(8u^3-15u^2+12u-4\right)\frac{\mathrm{d}^4}{\mathrm{d}u^4}C(u) - 8\left(3u+1\right) = 0.
\end{aligned}$$

Note that we prove also that $C(u)$ is solution of an algebraic equation. This fact is really not obvious from a combinatorial point of view. But it is deduced through the fact that the function $A(0,u)$ is algebraic:

$$(8u^3-15u^2+12u-4)A(0,u)^3 + (12u^2-12u+6)A(0,u) - 2u^3 = 0. \tag{3}$$

The equation is obtained by a *guess and prove approach*. Once it has been guessed it remains to prove it by using the holonomic equation proven in Proposition 6. Thus we get

$$\begin{aligned}
&32\left(9u^2-12u+8\right)(u-1)^3\\
&+48\left(36u^6-120u^5+202u^4-199u^3+123u^2-44u+8\right)(u-1)^2\,C(u)\\
&+\left(8u^3-15u^2+12\,u-4\right)^3 C(u)^3 = 0.
\end{aligned}$$

▶ **Theorem 7.** *The function $A(z,u)$ is an algebraic function in (z and u) whose annihilating polynomial has degree 3:*

$$\begin{aligned}
&2+6\left(12zu^3-18zu^2-2u^2+13zu+2u-3z-1\right)A(z,u)\\
&+6z^2\left(8u^3-15u^2+12u-4\right)A(z,u)^2\\
&+\left(8u^3-15u^2+12u-4\right)\left(z^3+6zu+3z^2-3z-1\right)A(z,u)^3 = 0.
\end{aligned}$$

还是代数方程！

Note that the choice to use a doubly exponential generating function (in u and z) for $(t_{n,k})$ would have made sense and would be holonomic too (closure property of Borel transform). But it would not be algebraic because of the inappropriate asymptotic expansion (cf. Theorem 9).

Proof. The fact that the initial conditions and a diagonal of $A(z,u)$ are algebraic suggests that it could also be algebraic as a function of z and u. Applying a bivariate guessing procedure, we observe that the bivariate function $H(z,u) = (u+1)(z^3+3z^2+6zu-3z-1)A(z,u)$ is such that $[z^n]H(z,u) = 0$ for $n > 2$. Furthermore $[z^j]H(z,u)$ is algebraic for $j = \{0,1,2\}$. So, let us calculate these z-extractions. First recall that $[z^0]A(z,u)$ satisfies the algebraic equation (3). In the same vein, $[z^1]A(z,u)$ satisfies the algebraic equation

$$\begin{aligned}
&\left(8u^3-15u^2+12u-4\right)f(u)^3 + 3\left(8u^3-15u^2+12u-4\right)f(u)^2\\
&\qquad +3\left(8u^3-15u^2+10u-2\right)f(u) + 8u^3-15u^2+6u = 0,
\end{aligned}$$

and finally $[z^2]A(z,u)$ satisfies the algebraic equation

$$\begin{aligned}
&\left(8u^3-15u^2+12u-4\right)f(u)^3 + \left(-24u^3+45u^2-36u+12\right)f(u)^2\\
&\qquad +\left(-72u^3+135u^2-84u+18\right)f(u) - 40u^3+75u^2-36u = 0.
\end{aligned}$$

Thus we obtain

$$\begin{aligned}
[z^0]H(z,u) &= -(1+u)A(0,u)\\
[z^1]H(z,u) &= -1+(u+1)\left((6u-3)A(0,u)-[z^1]A(z,u)\right)\\
[z^2]H(z,u) &= (u+1)\left((6u-3)[z^1]A(z,u)-[z^2]A(z,u)+3A(0,u)+(6u-4)\right).
\end{aligned}$$

Finally we get $A(z,u)=\dfrac{[z^{\leq 2}]H(z,u)}{(u+1)\left(z^3+3z^2+6uz-3z-1\right)}$. By using the elimination theory, we get a closed form algebraic equation for $A(z,u)$ of degree 27, that obviously cannot fit in the conference paper format. Nevertheless, this equation is not minimal. Simplifying it, we get a minimal polynomial of degree 3 which annihilates $A(z,u)$:

$$\begin{aligned}
&\left(8u^3-15u^2+12u-4\right)\left(z^3+3z^2+6zu-3z-1\right)A(z,u)^3\\
&+6z^2\left(8u^3-15u^2+12u-4\right)A(z,u)^2\\
&+6\left(12zu^3-18zu^2-2u^2+13zu+2u-3z-1\right)A(z,u)+2=0.
\end{aligned}$$

A direct proof by recurrence confirms the validity of this equation. ◀

We remark in the previous section that the diagonals of the function $A(z,u)$ are of particular interest because they define subclasses of arch processes with a fixed number of x_i actions covered by all the arches. In order to extract the generating functions of this subclass, we could use the Cauchy formula to compute $[u^0]A(z/u,u)$ and so on; we would keep the holonomicity property of the sequences but not their algebraicity. So, we prefer to define the generating function $B(z,u)=A(z/u,u)$. A similar proof as for the case $A(z,u)$ can be done to prove the algebraicity of $B(z,u)$. In particular, it exhibits the following algebraic equation satisfied by $B(z,u)$

$$\begin{aligned}
\left(9u^2+12u-4\right)\left(z^3+3z^2+6u-3z-1\right)B(z,u)^3+6z^2\left(9u^2+12u-4\right)B(z,u)^2\\
+6\left(18u^2z-18u^2+6uz+9u-3z-1\right)B(z,u)+2\left(6u-1\right)^2=0
\end{aligned}$$

In particular, $B(0,u)$ is associated to the sequence $(t_{k,k})_k$, $[z^1]B(z,u)$ corresponds to the sequence $(t_{k-1,k})_k$ and so on. By specializing $z=0$ in the latter algebraic equation then by resolving it through the Viète-Descartes approach for the resolution of cubic equation (detailed in the paper [19]), we obtain the following closed form formula corresponding to the branch that is analytic in 0:

$$B(0,u)=\sqrt{2}\sqrt{\frac{1-3u}{1-3u-\frac{9}{4}u^2}}\cos\left(\frac{1}{3}\arccos\left(\frac{6u-1}{\sqrt{2}(1-3u)}\sqrt{\frac{1-3u-\frac{9}{4}u^2}{1-3u}}\right)\right).$$

Even if the way we represented $B(0,u)$ could suggest a singularity when the argument of the arccos function is equal to 1, the function admits an analytic continuation up to its dominant singularity ρ : the solution of $1-3u-\frac{9}{4}u^2=0$, thus corresponding to $\rho=\dfrac{2}{3}\left(\sqrt{2}-1\right)$. 主奇点
Furthermore, by studying the global generating function $B(z,u)$, we obtain its singular expansion.

▶ **Lemma 8.** *Near the singularity when u tends to ρ, the function $B(z,u)$ satisfies*

$$B(z,u)\underset{u\to\rho}{=}a(z)+\frac{b(z)}{\sqrt{\rho-u}}+o\left((\rho-u)^{-1/2}\right),$$

with $a(z)$ and $b(z)$ two functions independent from u.

By using this result we deduce the asymptotic behaviors of the diagonal coefficients of $A(z,u)$.

▶ **Theorem 9.** *Let i be a given integer greater than -1, and k tend to infinity:*

$$t_{k+i,k} \underset{k\to\infty}{\sim} \gamma_i \frac{\rho^{-k}}{\sqrt{k}}\, k! \quad \text{with } \gamma_0 = \frac{1}{2}\sqrt{\frac{3}{\sqrt{2}\pi}\left(\sqrt{2}-1\right)} \quad \text{and} \quad \gamma_i = \left(\frac{1}{\sqrt{2}-1}\right)^i \gamma_0.$$

渐进结果

This theorem is a direct consequence of Lemma 8. The $(\gamma_i)_i$ can be deduced by asymptotic matching (using an *Ansatz*).

Finally, by computing $[z^1]B(z,u)$ with the algebraic equation it satisfies, we prove that its second derivative is solution of the algebraic equation exhibited in OEIS `A220433`.

4 Uniform random generation of runs

We now introduce an algorithm to uniformly sample runs of a given arch process $A_{n,k}$. Our approach is based on the recursive equations (1) and (2) for the sequence $(t_{n,k})$. Here we deal with the cases $k \leq n$ and avoid the limit case $k = n+1$. Although the latter limit case satisfies this equation too, its proof is based on another combinatorial approach, and so the construction of a run cannot be directly deduced form the combinatorial approach proposed for the cases $k \leq n$. Of course, a simple adaptation of the algorithm presented below would allow to sample in $A_{k-1,k}$, but the lack of space prevent us to present it here.

Our algorithm is a *recursive generation algorithm*. But since the objects are not specified in a classical Analytic Combinatorics way, we cannot use the results of [12]. As usual for recursive generation, the first step consists in the computation and the memorization of the value $t_{n,k}$ and all the intermediate values $(t_{i,j})$ needed for the calculation of $t_{n,k}$.

▶ **Proposition 10.** *In order to compute the value $t_{n,k}$, it is sufficient to calculate the values in the bi-dimensional set $\{t_{i,j} \mid n \leq i \leq n+k \text{ and } 0 \leq j \leq k-(i-n)\}$. This computation is done with $\mathcal{O}\left(k^2\right)$ arithmetic operations.*

Recall that the coefficient computations are done only once for a given pair (n,k), and then many runs can be drawn uniformly for $A_{n,k}$ by using the recursive generation algorithm.

Let us present the way we exploit the recurrence equation (2) to design the sampling method. The main problem that we encounter is the presence of a minus sign in the recurrence equation. Let us rewrite it in a slightly different way: $\sigma(D_{n,k}) + \sigma(\overline{D}^1_{n,k}) = \sigma(\overline{D}_{n,k}) + \sigma(\overline{D}^2_{n,k})$.

Recall that the structures under consideration are depicted in Figure 3. We introduce the classes of increasingly labeled structures from $D_{n,k}, \overline{D}^1_{n,k}, \overline{D}_{n,k}$ and $\overline{D}^2_{n,k}$, respectively denoted by $I_{n,k}, \overline{I}^1_{n,k}, \overline{I}_{n,k}$ and $\overline{I}^2_{n,k}$. Remark that the number of runs of $A_{n,k}$ is equal to $|I_{n,k}|$, where the function $|\cdot|$ corresponds to the cardinality of the considered class. Obviously the equation on the cardinalities can be written directly on the classes $I_{n,k} \cup \overline{I}^1_{n,k} = \overline{I}_{n,k} \cup \overline{I}^2_{n,k}$ (since their intersections are empty: $I_{n,k}$ and $\overline{I}^1_{n,k}$ are distinct even if they are isomorphic). Thus, we consider the problem of sampling the class $I_{n,k} \cup \overline{I}^1_{n,k}$ where we bijectively replace the runs belonging to $\overline{I}^1_{n,k}$ by runs of $I_{n,k}$ (which can be performed recursively during the sampling procedure). The Algorithm Sampling(n,k) is based on the correspondence depicted in the Figure 3 and its adaptation presented above on the classes $I_{n,k} \cup \overline{I}^1_{n,k}$. In each case the algorithm completes a recursively drawn run and applies some renaming on the actions of that run. Then, it inserts the action b_1 according to the cases $\overline{I}_{n,k} \backslash \overline{I}^1_{n,k}$, $\overline{I}^1_{n,k}$ or $\overline{I}^2_{n,k}$. In the specific case $\overline{I}^1_{n,k}$, instead of b_1, it is the action b_k that is inserted and the renaming occurs in a similar fashion to obtain a run of $I_{n,k}$ from the one of $\overline{I}^1_{n,k}$.

Algorithm 1 Uniform random sample for $I_{n,k}$.

```
1:  function SAMPLING(n, k)
2:      if k = 0 then
3:          return ⟨x_1, x_2, ..., x_n⟩
4:      r := RAND_INT(0, 2 · t_{n,k} − 1)      ▷ a uniform integer between 0 and 2 · t_{n,k} − 1 in r
5:      if r < |Ī_{n,k}| then                  ▷ generation in Ī_{n,k}
6:          U := SAMPLING(n, k − 1)
7:          p_b := 1 + r//t_{n,k−1}            ▷ The position of the new b to insert
8:          if p_b > p_{x_1} then              ▷ generation in Ī¹_{n,k}
9:              Rename x_1 by a_k ; and each x_i with i > 1 by x_{i−1}
10:             Insert b_k at position p_b ; and c_k at the end of U
11:         else                               ▷ generation in Ī_{n,k}\Ī¹_{n,k}
12:             In U, rename each a_i (resp. c_i and b_i) by a_{i+1} (resp. c_{i+1} and b_{i+1})
13:             Rename x_{n−k+1} by c_1
14:             Insert b_1 at position p_b ; and a_1 at the head of U
15:     else                                   ▷ generation in Ī²_{n,k}
16:         U := SAMPLING(n + 1, k − 1)
17:         p_b := 2 + (r − (n + 2k − 1) · t_{n,k−1})//t_{n+1,k−1}
18:         Rename x_{p_b} by b_1 and x_{n−k+2} by c_1 ; and each x_i with i > p_b by x_{i−1}
19:         Insert a_1 at the head of U
20:     return U
```

Line 4 and 17 : the binary operator // denotes the Euclidean division.
The position of an action in a run is its arrival number (from 1 to the number of actions).

▶ **Theorem 11.** *The Algorithm* SAMPLING(n, k) *builds uniformly at random a run of* $A_{n,k}$ *in* k *recursive calls, once the coefficients computations and memorizations have been done.*

Since each object of $I_{n,k}$ is sampled in two distinct ways, the uniform sampling in $I_{n,k} \cup \bar{I}^1_{n,k}$ induces the uniform sampling of $I_{n,k}$.

Focus on the run of $A_{5,4}$ depicted in Figure 2: $\langle a_1, b_1, a_2, a_3, b_3, a_4, x_1, b_4, c_1, b_2, c_2, c_3, c_4\rangle$. It is either obtained from a (renamed) run of $\bar{I}^1_{5,4}$: $\langle a_1, b_1, a_2, a_3, b_3, x_1, x_2, c_1, b_2, c_2, c_3\rangle$ with $p_b = 8$ (Line 8 of the algorithm). Or it is built from $\langle a_1, a_2, b_2, a_3, x_1, b_3, x_2, b_1, c_1, c_2, c_3\rangle$ of $\bar{I}_{5,4}\backslash\bar{I}^1_{5,4}$, with $p_b = 1$ (Line 11). But it cannot be built from a run of $\bar{I}^2_{5,4}$.

In Figure 4, we have uniformly sampled 1000 runs for $A_{1000,1000}$ and we have represented in blue points every pair (k, n) corresponding to an increasing sub-structure from $A_{n,k}$ that has been built during the algorithm (k for abscissa and n for ordinate). Only around $4.78 \cdot 10^4$ sub-structures have been built among the $50 \cdot 10^4$ inside the red lines which are calculated for the value $t_{1000,1000}$. At the beginning $n \approx k$ and the **if** branch on Line 5 is preferred (instead of the **else** one on Line 15) because the number of x_i actions is too small. After some recursive calls, the number of x_i actions has increased and then both branches of the algorithm are taken with probabilities of the same order. Recall that the constants γ_i (cf. Theorem 9) are evolving with an exponential growth. Finally, we observe that only a small number of diagonals are necessary for the samplings. Since the diagonals (t_{n_i,k_i}) for increasing sequences $(n_i)_i$ and $(k_i)_i$ follow P-recurrences (cf. [16]), a lazy calculation of the terms of the necessary diagonals that envelop the blue points would allow to minimize the pre-computations of Proposition 10.

We close this section with the presentation of an *unranking* algorithm for the construction

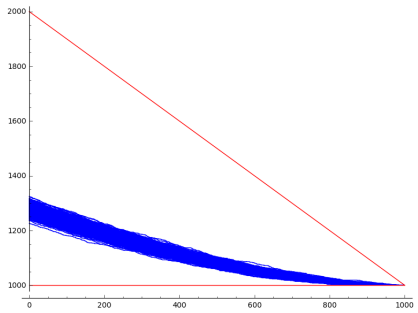

Figure 4 The terms $t_{i,j}$ needed for the sampling of 1000 runs of $A_{1000,1000}$.

of the runs of a given arch process $A_{n,k}$. This type of algorithm has been developed during the 70's by Nijenhuis and Wilf [20] and introduced in the context of Analytic Combinatorics by Martínez and Molinero [17]. Our algorithm is based on a bijection between the set of integers $\{0, \ldots, t_{n,k} - 1\}$ and the set of runs of $A_{n,k}$. Here again we restrict ourselves to the values $k \leq n$. As usual for unranking algorithms, the first step consists in the computation and the memorization of the values of a sequence. But compared to the uniform random sampling, here we need more information than the one given by the sequence $(t_{n,k})$.

To be able to reconstruct the run associated to a given rank, we need to know the position of the action x_1 in the recusively drawn run in order to decide if the action b_1 appears before or after it. First suppose $k < n$ and let $t_{n,k,\ell}$ be the number of runs in $\mathcal{A}_{n,k}$ whose action x_1 appears at position ℓ. Let us denote by $I_{n,k,\ell}$ the associated combinatorial class. We obtain directly a constructive recurrence for the sequence.

$$t_{n,k,\ell} = (\ell - 2)\, t_{n,k-1,\ell-2} + (n-k)\, t_{n+1,k-1,\ell-1} \quad \text{and} \quad t_{n,0,1} = 1;\ t_{n,0,\ell>1} = 0.$$

▶ **Proposition 12.** *The computation of $t_{n,k,\ell}$ is done with $\mathcal{O}\left(k^2\right)$ arithmetic operations.*

The Unranking algorithm computes a run given its rank in the following total order:

$$\alpha \preceq_{n,k} \beta \quad \text{iff.} \quad \begin{cases} \alpha \in I_{n,k,i_0} \text{ and } \beta \in I_{n,k,i_1} & \wedge \quad i_0 < i_1, \\ \text{or} \quad \alpha, \beta \in I_{n,k,i} & \wedge \quad \alpha \text{ is built recursively from } I_{n,k-1,i-2} \text{ and } \beta \text{ is built recursively from } I_{n+1,k-1,i-1} \\ \text{or} \quad \alpha, \beta \in I_{n,k,i} & \wedge \quad \alpha, \beta \in I_{n,k-1,i-2} \text{ (resp. } I_{n+1,k-1,i-1}) \text{ and } \alpha_0, \beta_0 \text{ inducing } \alpha, \beta \text{ satisfy } \alpha_0 \preceq_{n,k-1} \beta_0. \end{cases}$$

The run example of Figure 2 has rank 479 among the 1270 runs of $A_{5,4}$. Note that in the case $k = n$ (at the end there is no x_1) the algorithm is easily extended by considering the position of b_1 as the one of x_1.

▶ **Theorem 13.** *The Algorithm* Unranking(n, k, r) *builds the r-th run of $A_{n,k}$ in k recursive calls, once the coefficient memorizations $t_{n,k,\ell}$, for all ℓ such that $k + 1 \leq \ell \leq 2k + 1$ (and the necessary n and k), have been done.*

Note that the implementation of both algorithms can be much more efficient than the pseudocode exhibited above. Actually, only the absolute positions of the b_i actions are important in a run, because all other actions have their positions determined by the positions of the b_i actions. However, such implementations are much more cryptic to read, and so we preferred to present here easy-to-read algorithms.

Algorithm 2 Unranking for $I_{n,k}$.

1: **function** UNRANKING(n, k, r)
2: $\ell := k + 1$
3: **while** $r \geq 0$ **do**
4: $r := r - t_{n,k,l}$
5: $\ell := \ell + 1$
6: **return** CONS(n, k, ℓ, r)
7: **function** CONS(n, k, ℓ, r)
8: **if** $k = 0$ **then**
9: **return** $\langle x_1, x_2, \ldots, x_n \rangle$
10: **if** $r < (\ell - 2) \cdot t_{n,k-1,\ell-2}$ **then** ▷ generation in $I_{n,k-1,\ell-2}$
11: $rr := r \ \% \ t_{n,k-1,\ell-2}$
12: $U := \text{CONS}(n, k - 1, \ell - 2, rr)$
13: $p_b := 1 + r//t_{n,k-1,\ell-2}$ ▷ The position of the new b to insert
14: In U, rename each a_i (resp. c_i and b_i) by a_{i+1} (resp. c_{i+1} and b_{i+1})
15: Rename x_{n-k+1} by c_1
16: Insert b_1 at position p_b ; and a_1 at the head of U
17: **else** ▷ generation in $I_{n+1,k-1,\ell-1}$
18: $r' := r - (\ell - 2) \cdot t_{n,k-1,\ell-2}$
19: $rr := r' \ \% \ t_{n+1,k-1,\ell-1}$
20: $U := \text{CONS}(n + 1, k - 1, \ell - 1, rr)$
21: $p_b := 2 + r'//t_{n+1,k-1,\ell-1}$
22: Rename x_{p_b} by b_1 and x_{n-k+2} by c_1 ; and each x_i with $i > p_b$ by x_{i-1}
23: Insert a_1 at the head of U
24: **return** U

Line 11 and 19 : the binary operator % denotes the Euclidean division remainder.

References

1 The ecmascript 2015 language specification. Technical report, ECMA-262 6th Edition, 2015. URL: `http://www.ecma-international.org/ecma-262/6.0/index.html`.

2 M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, New York, ninth dover printing, tenth gpo printing edition, 1964.

3 C. Baier and J.-P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

4 H. C. Baker, Jr. and C. Hewitt. The incremental garbage collection of processes. In *Proceedings of the 1977 Symposium on Artificial Intelligence and Programming Languages*, pages 55–59. ACM, 1977.

5 O. Bodini, M. Dien, X. Fontaine, A. Genitrini, and H.-K. Hwang. Increasing diamonds. In *Latin American Symposium on Theoretical Informatics*, pages 207–219. Springer, Berlin, Heidelberg, 2016.

6 O. Bodini, M. Dien, A. Genitrini, and F. Peschanski. Entropic uniform sampling of linear extensions in series-parallel posets. In *12th International Computer Science Symposium in Russia (CSR)*, pages 71–84, 2017.

7 O. Bodini, M. Dien, A. Genitrini, and F. Peschanski. The Ordered and Colored Products in Analytic Combinatorics: Application to the Quantitative Study of Synchronizations in

Concurrent Processes. In *14th SIAM Meeting on Analytic Algorithmics and Combinatorics (ANALCO)*, pages 16–30, 2017.

8 O. Bodini, A. Genitrini, and F. Peschanski. The combinatorics of non-determinism. In *FSTTCS'13*, volume 24 of *LIPIcs*, pages 425–436. Schloss Dagstuhl, 2013.

9 O. Bodini, A. Genitrini, and F. Peschanski. A Quantitative Study of Pure Parallel Processes. *Electronic Journal of Combinatorics*, 23(1):P1.11, 39 pages, (electronic), 2016.

10 G. Brightwell and P. Winkler. Counting linear extensions is ♯P-Complete. In *STOC*, pages 175–181, 1991.

11 P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009. URL: `http://www.cambridge.org/uk/catalogue/catalogue.asp?isbn=9780521898065`.

12 P. Flajolet, P. Zimmermann, and B. Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 132(1-2):1–35, 1994.

13 R. Grosu and S. A. Smolka. Monte carlo model checking. In *TACAS'05*, volume 3440 of *LNCS*, pages 271–286. Springer, 2005.

14 A. Khoroshkin and D. Piontkovski. On generating series of finitely presented operads. *Journal of Algebra*, 426:377–429, 2015.

15 D. E. Knuth. *The art of computer programming, volume 3: (2nd ed.) sorting and searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

16 L. Lipshitz. The diagonal of a d-finite power series is d-finite. *Journal of Algebra*, 113(2):373–378, 1988.

17 C. Martínez and X. Molinero. Generic algorithms for the generation of combinatorial objects. In *MFCS'03*, pages 572–581. Springer Berlin Heidelberg, 2003.

18 R. Milner. *A Calculus of Communicating Systems*. Springer Verlag, 1980.

19 R.W.D. Nickalls. Viète, descartes and the cubic equation. *The Mathematical Gazette*, 90(518):203–208, 2006.

20 A. Nijenhuis and H.S. Wilf. *Combinatorial algorithms*. Computer science and applied mathematics. Academic Press, New York, NY, 1975.

21 R.P. Stanley. *Enumerative Combinatorics:*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2001.